Learning-Based Approaches for the Analysis and Optimization of Profile Extrusion Dies and Bioreactors

Lernbasierte Ansätze für die Analyse und Optimierung von Profilextrusionswerkzeugen und Bioreaktoren

Von der Fakultät für Maschinenwesen der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften genehmigte Dissertation

vorgelegt von

Daniel Wolff

Berichter/in: Univ.-Prof. Dr.-Ing. Stefanie Elgeti
Prof. Elie Hachem, PhD
Univ.-Prof. Marek Behr, Ph. D.

Tag der mündlichen Prüfung: 08. September 2023

Acknowledgements

This PhD thesis was written during my work as a research assistant at the Chair for Computational Analysis of Technical Systems at RWTH Aachen University.

The duration of my PhD was one of the most difficult times of my life so far. Neither during my high-school time nor during my Bachelor's and Master's studies I struggled as much as I did during these last three years. There were at least three times where I seriously considered quitting my PhD and looking for a different job. Yet, this document is the proof that I pulled through with it and now that I'm writing these lines I cannot express how glad and relieved I am that I did continue. However, I am convinced that I would not have managed to achieve this without the continuous support of my family, friends, and colleagues to whom I dedicate this section. The order in the following is arbitrary and if I forgot to mention someone, be assured that I'm truly sorry about it. Everyone who knows me can tell that I could easily write 10 pages full of acknowledgements, but for the sake of this document, I have to restrain myself.

First and foremost, I would like to thank my amazing colleague and best friend *Veronika Trávníková*. Not only did you cheer me up in countless tea breaks while I was preparing this document but you also always knew the right things to say to make me feel better during times when I was struggling. In both private and work-related problems, you always provided meaningful advice and softly pulled me back to the ground when I was freaking out. Your support means a lot to me and I will never take any of it for granted.

When speaking about struggling, there is no way around mentioning my friend and colleague *Jacques Zwar*. Since our times as partners-in-crime regarding the CAMPIGA development, I knew you were always only one Rocket.Chat message away to start a digital coffee break whenever one of us needed to talk. For sure I would not have pulled through with my PhD if it wasn't for our countless meetings and fruitful discussions.

Another person who was there for me and helped me a lot during my writing phase was *Stefan Wittschieber*. Although preparing your own doctoral defense during that time, you were always there to listen about my complaints or problems and always provided new viewpoints that helped me a lot to continue.

The same holds true for *Eugen Salzmann*. After I started at CATS, you quickly became one of my closest friends. When the results of Section 4.2 — three weeks before submission! — did not work out as expected, you helped me staying calm and took the time to stand behind me, looking over my shoulder, while trying to pin-point the problem in my code.

Another big thank you goes to my great friends *Leonardo Boledi*, *Jorge Guzman*, and *Laura Helleckes*. I am incredibly grateful that I got to know you through HDS-LEE and for all the memories we made together — either during numerous HDS-LEE events or outside of work. Moreover, I knew that I could always rely on you or ask for your advice whenever I needed it. It means a lot to me that I can count you to my friends and I am already looking forward to our future trips and activities.
Moreover, I would like to thank my friends — and best flatmates in the world — *Cailing Fu*, *Aron Zingler*, and *Mark Maslowski*. Our flat will always feel like home for me and I'm definitely going to miss our community once I have to look for a new job.

Regarding the contents of this thesis, none of it would have been possible without the excellent students whose theses I had the honor to supervise. Starting with Chapter 4, I would like to thank *Henri Lubjuhn*, who developed the first version of the `FrameXDE` framework in his seminar thesis and then implemented the first bioreactor test case in a succeeding Master's thesis. Building up on this work, I would like to thank *Nico Dirkes* who further developed the framework and managed to improve the model predictions significantly through the use of domain decomposition. The amount of work accomplished in your Master's thesis was truly impressive and you totally deserve the recognition you got for it. Regarding Chapter 5, the foundation was provided through the Master's thesis of *Michael Binder*. Although supervised completely remotely, none of the succeeding results would have been possible without your work. Following up on this, I would like to thank *Clemens Fricke* for his seminar thesis that resulted in the `releso` framework as well his outstanding Master's thesis, which provided us with more than enough material for two publications.
Besides, I would like to thank the students whose work did — unfortunately — not end up in this dissertation, namely *Benjamin Koch*, *Pauline Lekkas*, *Sven Warmuth*, and *Sofia Dubinvoskaia*. You still helped me learn and improve as a supervisor and made me realize how much I enjoy working with students.

Continuing with people that proof-read and provided incredibly helpful feedback for my thesis, I would like to thank *Michael Aichmüller*, *Nico Dirkes*, *Clemens Fricke*, *Anna Ranno*, *Veronika Trávníková*, *Stefan Wittschieber*, and *Jacques Zwar*.

Finally, I would also like to thank my supervisors *Stefanie Elgeti* and *Marek Behr*. You always had an open door for me and my questions, be it a physical one in Aachen or a digital one in Vienna. Thanks for enabling me to gather so many experiences during my PhD, ranging from countless conferences that I was able to attend, up to my research stays in Vienna and Austin. All of this contributed a lot to my development during this PhD and in consequence helped me finish this thesis. At this point I also want to mention *Norbert Hosters*. Although I was never a PhD student of your group, you still made me your responsibility and actively tried to improve my situation and state of mind, especially in the last months. Thank you for convincing me to pull through with my work at CATS!

Last but not least, I would like to thank my parents, *Christa and Franz-Josef Wolff*. For sure I would have never managed to achieve this academic degree without your care and support. Through your education and love, you contributed the most to forming the person I am today. You were there for me when I was struggling and I knew that I could always come to you and talk to you about my worries. I will always be thankful for that.

Aachen, October 2023 *Daniel Wolff*

Contents

Acronyms

A2C Advantage Actor Critic. 41, 78, 85, 86, 87, 94, 95, 96, 98, 103

AD Automatic Differentiation. 25, 33

ADAM Adaptive Moment Estimation. 24

AI Artificial Intelligence. XVII, 1, 7, 15, 16, 17, 18, 19

BC Boundary Condition. XIX, 11, 12, 33, 37, 39, 49, 50, 52, 53, 55, 57, 60, 62, 63, 66, 67, 68, 69, 71, 80, 101

BFGS Broyden Fletcher Goldfarb Shanno. 25

CAD Computer Aided Design. 79

CFD Computational Fluid Dynamics. 1, 4, 5, 9, 36, 39, 42, 43, 65, 75, 76, 77, 82, 92

CNN Convolutional Neural Network. 17, 39

cPINN Conservative Physics-Informed Neural Network. 33, 34, 69

CSG Constructive Solid Geometry. 44

CSTR Continuously-Stirred Tank Reactor. 4

DDPG Deep Deterministic Policy Gradient. 41, 76, 78, 84, 85, 86, 94, 95, 96, 103

DeepONet Deep Operator Network. 33, 34

DL Deep Learning. 1, 16, 17

DOF Degree of Freedom. XVII, XVIII, 76, 78, 79, 82, 83, 103

DQN Deep Q-Network. 41, 78, 86, 87, 96, 103

FEM Finite Element Method. 6, 9, 12, 13, 33, 47, 49, 50, 51, 57, 59, 60, 61, 64, 65, 66, 68, 72, 73, 80, 101, 103, 106

FFD Free Form Deformation. VI, XVII, 77, 78, 79, 80, 82, 83, 91

FFNN Feed-Forward Neural Network. V, VI, 7, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 31

GPU Graphics Processing Unit. 18

HPO Hyper-parameter Optimization. XVII, 29, 45, 46, 47, 65

IBC Initial and Boundary Condition. V, 6, 10, 11, 31, 36, 37, 44

IC Initial Condition. 11

IGA Isogeometric Analysis. 39

L-BFGS Limited Memory Broyden Fletcher Goldfarb Shanno. 25, 52, 66, 72

L-LAAF Layerwise Locally-Adaptive Activation Function. XVII, 48, 49

ML Machine Learning. 1, 5, 7, 16, 17, 18, 19, 22, 24, 30, 33, 40, 44, 45, 104

MLP Multi-Layer Perceptron. XVII, 17, 18, 20, 21

MOR Model-Order Reduction. V, 1, 2, 5

NN Neural Network. VI, 1, 7, 16, 17, 18, 21, 22, 24, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 48, 50, 52, 56, 65, 66, 67, 72, 102, 103

NURBS Non-Rational Uniform B-Splines. 79

ODE Ordinary Differential Equation. 69, 70, 102

PDE Partial Differential Equation. VI, XVII, XVIII, 5, 6, 9, 10, 12, 26, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 44, 45, 47, 48, 49, 52, 53, 55, 57, 58, 65, 67, 70, 77, 79, 80, 103, 108

PINN Physics-Informed Neural Network. VI, XVII, XVIII, XIX, 1, 2, 7, 17, 18, 25, 27, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 43, 44, 45, 46, 47, 48, 49, 50, 52, 53, 54, 56, 57, 58, 59, 60, 61, 62, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 101, 102, 103, 107, 110

PPO Proximal Policy Optimization. XVIII, 41, 76, 78, 81, 84, 85, 86, 87, 88, 89, 94, 95, 96, 98, 99, 103, 112, 113

PSPG pressure-stabilizing/Petrov-Galerkin. 13

RAR Residual-based Adaptive Refinement. XVII, 38, 47, 48

ReLU Rectified Linear Unit. 18, 20, 26, 29

RL Reinforcement Learning. VI, XVII, 1, 2, 7, 16, 17, 19, 40, 41, 42, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 86, 88, 89, 90, 92, 94, 96, 98, 102, 103, 104

RNN Recurrent Neural Network. 17

ROM Reduced Order Model. 1, 2, 4, 5, 6, 7, 43, 45, 49, 50, 53, 60, 62, 65, 69, 72, 73, 74, 101, 102, 103, 107

SAC Soft Actor Critic. 41, 78, 85, 86, 94, 95, 96, 103

SciML Scientific Machine Learning. V, VI, 1, 15, 16, 17, 18, 19, 20, 22, 24, 26, 28, 30, 32, 34, 35, 36, 38, 40, 42, 104

SDF Signed Distance Function. 39

SUPG Streamline-Upwind Petrov-Galerkin. 13

SVM Support Vector Machine. 18

TPE Tree-structured Parzen Estimator. 46

VPINN Variational Physics-Informed Neural Network. 33, 35

XPINN Extended Physics-Informed Neural Network. 33, 34, 69

Notation

Physical Quantities

Re	Reynolds number	–
t	Time	s
p	Pressure	$\mathrm{kg\,m^{-1}\,s^{-2}}$
ρ	Density	$\mathrm{kg\,m^{-3}}$
μ	Dynamic viscosity	$\mathrm{kg\,m^{-1}\,s^{-1}}$
A	Viscosity at zero shear (Carreau material law)	$\mathrm{kg\,m^{-1}\,s^{-1}}$
B	Reciprocal transition rate (Carreau material law)	s
C	Rate of change of the viscosity curve (Carreau material law)	–
$\dot{\gamma}$	Shear rate	$\mathrm{s^{-1}}$
$\dot{m}$	Mass flow	$\mathrm{kg\,s^{-1}}$
$\boldsymbol{x}$	Spatial position	m
$\boldsymbol{n}$	Outward-facing normal	m
$\boldsymbol{v}$	Velocity	$\mathrm{m\,s^{-1}}$
$\boldsymbol{b}$	Volumetric body force	$\mathrm{m\,s^{-2}}$
$\boldsymbol{\omega}$	Angular velocity	$\mathrm{rad\,s^{-1}}$
$\boldsymbol{\varepsilon}$	Rate-of-strain tensor	$\mathrm{s^{-1}}$
$\boldsymbol{\sigma}$	Cauchy stress tensor	$\mathrm{kg\,m^{-1}\,s^{-2}}$

Finite Element Method

Ω	Spatial domain
Γ	Boundary of the spatial domain
Ω^h	Computational mesh
N_n	Number of nodes in the computational mesh
N_el	Number of elements in the computational mesh
$\mathcal{L}^2$	Space of square-integrable functions
$\mathcal{H}^1$	First-order Sobolev space
$\mathcal{Q}$	Function space of pressure trial and test functions
$\mathcal{S}$	Function space of velocity trial functions
$\mathcal{V}$	Function space of velocity test functions
q	Test function for the pressure
$\boldsymbol{w}$	Test function for the velocity

(Physics-Informed) Neural Networks

$\mathcal{H}$	Hypothesis space
$\mathcal{X}$	Space of the neural network inputs
$\mathcal{Y}$	Space of the neural network outputs
Θ	Space of the neural network parameters
Λ	Space of physical parameters for parameterized problems
$\mathcal{D}$	Data set
$\mathcal{B}$	Mini-batch (subset of a data set)
$N_\mathcal{X}$	Number of the neural network input space dimensions
$N_\mathcal{Y}$	Number of the neural network output space dimensions
N_hl	Number of hidden layers
N_nhl	Number of neurons per hidden layer
N_epochs	Number of epochs
$N_\mathcal{D}$	Number of points in a data set
N_mb	Number of mini-batches

N_{PDE}	Number of evaluation points for partial differential equation residuals
N_{IBC}	Number of evaluation points for initial and boundary condition residuals
η	Learning rate
$\boldsymbol{b}$	Vector of a neural network biases
$\boldsymbol{W}$	Matrix of a neural network weights
$\boldsymbol{\theta}$	Vector containing all neural network parameters (weights and biases)
$\bar{\boldsymbol{x}}$	Generic input vector for a neural network
$\boldsymbol{z}$	Vector of intermediate neural network states
$\bar{\boldsymbol{y}}$	Generic output vector of a neural network
$\boldsymbol{u}$	Generic solution vector of a partial differential equation
$\boldsymbol{\lambda}$	Vector of physical parameters for parameterized problems
$\boldsymbol{\alpha}$	Vector comprising weights of the individual loss components
σ	Activation function
f_{FFNN}	Feed-forward neural network mapping
L	Loss function
$\mathcal{L}$	Expected risk
$\mathcal{L}_{\mathcal{D}}$	Empirical risk
$\mathcal{R}$	Residual (of a physical constraint)
ϱ	Regularization term

Reinforcement Learning

a	Action undertaken by the agent
r	Numerical reward
s	State of the environment
t	Step index
π	Policy the agent uses to determine its next action

List of Figures

List of Tables

Summary

This thesis investigates how methods from the field of machine learning can be incorporated into classical engineering workflows and thus provides another contribution toward bridging the gap between these two disciplines. Precisely, we examine how Machine Learning algorithms can enhance simulations, i.e., by reducing computing times, to enable challenging tasks like the optimization of even complex industrial applications. While being of interest in many engineering disciplines, e.g., the design of profile extrusion dies or bioreactors, repeatedly evaluating expensive high-fidelity (or full-order) simulation models is usually infeasible despite the advances in computational powers during the last decades. Machine Learning algorithms are promising candidates to overcome this drawback as they — once trained during an offline phase — allow for rapid evaluations online, e.g., within an optimization loop.

The first part of this thesis considers physics-informed neural networks to create reduced-order models for simulating the flows in bioreactors and profile extrusion dies. By incorporating knowledge about the problems' governing equations into the construction process of the reduced simulation model, superior accuracy and interpretability compared to classical black-box machine learning approaches are expected while simultaneously obliterating the need for data. We present two novel general strategies that enable physics-informed neural networks as reliable reduced models and demonstrate them on the considered application cases.

In the second part of this thesis, we focus on parametric shape optimization and study reinforcement learning: The use of reinforcement learning agents which make decisions based on (partial) observations of a system's state as an alternative to classical optimization algorithms allows for two approaches, i.e., incremental and direct shape optimization. On two different profile extrusion die geometries with differing design objectives, we compare both approaches and discuss the performance of different learning algorithms based on their classification. Finally, we explore vectorized environment training as one possibility of how learning the decision rule can be accelerated by parallelization.

Kurzfassung

Die vorliegende Arbeit untersucht, wie Methoden aus dem Bereich des maschinellen Lernens in klassische ingenieurwissenschaftliche Arbeitsabläufe integriert werden können und leistet damit einen weiteren Beitrag zur Überbrückung der Kluft zwischen diesen beiden Disziplinen. Konkret wird untersucht, wie Algorithmen aus dem Bereich des maschinellen Lernens Simulationen verbessern können, z.B. durch die Reduzierung von Rechenzeiten, um anspruchsvolle Aufgaben wie die Optimierung selbst komplexer industrieller Anwendungen zu ermöglichen. Die wiederholte Auswertung von teuren High-Fidelity-Simulationsmodellen ist trotz der in den letzten Jahrzehnten erzielten Fortschritte bei der Rechenleistung in der Regel nicht durchführbar, obwohl sie in vielen technischen Disziplinen von Interesse ist, z. B. beim Entwurf von Profilextrusionswerkzeugen oder Bioreaktoren. Algorithmen des Maschinellen Lernens sind vielversprechende Kandidaten, um diesen Nachteil zu überwinden, da sie — nach einem Offline-Training — schnelle Online-Evaluierungen, z. B. innerhalb einer Optimierungsschleife, ermöglichen.

Der erste Teil dieser Arbeit befasst sich mit physikalisch informierten neuronalen Netzen zur Erstellung von reduzierten Modellen für die Simulation von Strömungen in Bioreaktoren und Profilextrusionswerkzeugen. Durch die Einbeziehung der für die Probleme geltenden Gleichungen in den Erstellungsprozess des reduzierten Simulationsmodells wird eine höhere Genauigkeit und Interpretierbarkeit im Vergleich zu klassischen Black-Box-Maschinenlernansätzen erwartet, während gleichzeitig die Notwendigkeit von Daten entfällt. Wir stellen zwei neue allgemeine Strategien vor, die physikalisch informierte neuronale Netze als zuverlässige reduzierte Modelle ermöglichen, und demonstrieren diese an den betrachteten Anwendungsfällen.

Im zweiten Teil dieser Arbeit konzentrieren wir uns auf parametrische Formoptimierung und untersuchen bestärkendes Lernen: Der Einsatz von Agenten des bestärkenden Lernens, die Entscheidungen auf der Grundlage von (Teil-)Beobachtungen des Systemzustands treffen, als Alternative zu klassischen Optimierungsalgorithmen ermöglicht zwei Ansätze, nämlich die inkrementelle und die direkte Formoptimierung. Anhand von zwei verschiedenen Profilextrusionsgeometrien mit unterschiedlichen Konstruktionszielen vergleichen wir beide Ansätze und diskutieren die Leistung verschiedener Lernalgorithmen auf der Grundlage ihrer Klassifizierung. Zuletzt untersuchen wir das Training in einer vektorisierten Umgebung als eine Möglichkeit, wie das Lernen der Entscheidungsregel durch Parallelisierung beschleunigt werden kann.

Publications and Copyrights

Results of this thesis have been published in the following articles:

- Wolff, D., Fricke, C., Kemmerling, M., & Elgeti, S. (2023). Towards shape optimization of flow channels in profile extrusion dies using reinforcement learning. *Proceedings in Applied Mathematics and Mechanics* 22.1 http://dx.doi.org/10.1002/pamm.202200009
- Fricke, C., Wolff, D., Kemmerling, M., & Elgeti, S. (2023). Investigation of reinforcement learning for shape optimization of profile extrusion dies. *Advances in Computational Science and Engineering* 1.1 https://www.aimsciences.org/article/doi/10.3934/acse.2023001

1 Introduction

Artificial Intelligence (AI) has emerged as a discipline of computer science already in the last century [106]. Since then, research has been dedicated to developing novel technologies and methodologies to reproduce intelligent behavior with computer programs. Due to the evolution of computational resources in the last decades, which allows the application of computationally expensive approaches, such as Deep Learning (DL), the field of AI experiences ongoing momentum [78].

However, also other scientific fields have benefited from the increase in computational capabilities: The usage of computer simulations as a tool to examine problems arising from different scientific disciplines, including engineering science, has gained more attention and has by now become a state-of-the-art technique for obtaining better insights into technical scenarios. These advances have also encouraged the interest in transferring methods from the field of AI to the field of scientific computing, resulting in the interdisciplinary field of Scientific Machine Learning (SciML).

The Computational Fluid Dynamics (CFD) community, too, started experimenting with approaches combining standard Model-Order Reduction (MOR), like *Proper Orthogonal Decomposition*, with Machine Learning (ML) [12, 177]. In other efforts, fundamental principles such as frame-invariance or boundary conditions are incorporated into previously physics-agnostic data science approaches [91, 167]. Introducing such constraints a-priori into the structure of the Neural Network (NN) model is expected to boost the predictivity and the efficiency of ML tools.

The thesis at hand aims to build up upon previous achievements in the field of SciML and further study how classical engineering approaches can be cross-fertilized with ML methods. Specifically, we aim to exploit the synergy of simulations and AI-based techniques for two questions. The first part of this thesis is concerned with the methodology of Physics-Informed Neural Networks (PINNs), which recently emerged as another promising alternative for constructing Reduced Order Models (ROMs) aware of the conservation laws governing the problem under consideration. Although very prominent in the research community, this methodology has until now been primarily employed for problems with simple geometries and elementary constitutive laws, often already facing difficulties that hinder a straightforward application [26, 84, 175]. Thus, this thesis empirically investigates how PINNs can be improved to tackle more complex problems closer resembling industrial applications. The second part of this thesis then investigates Reinforcement Learning (RL)

as a learning-based alternative for solving shape optimization problems which are ubiquitous in engineering questions. In RL, an agent interacts with an environment, whilst trying to find a (optimal) strategy for solving a given problem (thereby mimicking the way how humans learn to accomplish a new task, namely by trial-and-error) [4]. While not necessarily superior to classical, e.g., gradient-based or evolutionary, optimization algorithms for one single problem, RL techniques are expected to perform notably well when similar optimization tasks are repeated since the learning algorithm finds a more general strategy for generating optimal shapes instead of concentrating only on the solution of one single problem [68].

Based on the exposition above, we can record the aim of this thesis in the following two research questions:

(RQ I) Do PINNs qualify as ROMs for engineering problems, i.e., can they provide accurate predictions for problems with intricate geometries or complex constitutive laws?

(RQ II) Can RL be employed to solve shape optimization problems, and how does the formulation of the learning problem influence the optimization?

Though yet quite general, we will substantiate each question in the dedicated chapters. Before diving into the technical details, in the following sections, we will first reinforce the motivation outlined above by briefly giving an overview of the context in which this thesis is embedded. Precisely, we will introduce the reader to the application cases of profile extrusion and bioreactors (Sections 1.1 and 1.2) as well as the topics of MOR and shape optimization (Sections 1.3 and 1.4).

1.1. Profile Extrusion

Profile extrusion is a widely used and — by now — well-established continuous production process for manufacturing plastic profiles with a constant cross-section. Its application ranges from fabricating plastic films, sealings, and cable channels to floor skirtings and window frames. The process can be described as follows [59, 131]: Plastic pellets are fed into a *hopper* at the beginning of the *extrusion line*. The hopper leads to a mechanical screw that rotates to transport the yet solid pellets forward into the *extruder*. The channel walls inside the extruder are heated by attached *heating bands*, causing the pellets to melt gradually to prevent overheating. As a result, a highly-viscous plastics melt is obtained, which can now be slowly cast into the desired shape. To this end, the plastics melt transitions from the screw into the *preforming zone* of the *extrusion die*, i.e., the shape-imparting part of the process. Here, the melt is transformed into the final profile by a cascade of consecutive transition zones until it finally leaves the die and enters the *calibrator*. In this process stage, the extruded profile is transported forward and cooled until it solidifies, residing in the desired shape. Afterward, the solidified profile is cut into pieces and dumped into storage.

From an engineering point of view, the die and the calibration unit decisively influence the quality of the manufactured profiles [59, Chapter 1]. Consequently, being able to model these components reliably is of utmost importance before tackling tasks like the optimization of the whole process. In this thesis, we are especially interested in the flow

of the molten plastic inside the profile extrusion die. An accurate model should reflect the plastics melt's rheological and thermodynamic properties. Moreover, depending on the type of plastic used in the manufacturing process, different material behavior must be considered in the model so that the computed flow state is in good agreement with the measurements during operation, rendering the modeling task challenging from a numerical point of view. Given a suitable material model, numerical simulations can be performed with the aim of identifying potential for improving the manufacturing process.

Despite its vast application, profile extrusion involves some challenges that need to be overcome to operate it efficiently. So far, the engineer in charge guides the design process of extrusion dies for new cross-section shapes mainly by their experience. However, designing flow channels to distribute the flow of the highly-viscous and non-Newtonian plastics melt inside a profile extrusion die is not a trivial task due to its strongly nonlinear rheological behavior [125]. Inhomogeneous velocity distributions at the die's outlet and remaining residual stresses inside the extruded, solidified part may result in significant deviations from the desired target shape, known as *warpage*. Consequently, homogeneous velocities at the outlet are considered the most relevant criterion for designing these flow channels [122, 59]. Moreover, to increase the overall efficiency and sustainability of the production process, it is of great interest to reduce material consumption during manufacturing, e.g., by reducing the amount of scrap. Here, the computational design of flow channels inside profile extrusion dies comes in handy and allows the numerical investigation of a quality criterion (e.g., a homogeneous velocity distribution at the extruder outlet) and thus generates a close-to-optimal design concerning this chosen design criterion.

Different design criteria can be considered to optimize the shape of the flow channels, e.g., a low-pressure drop, short residence times, or uniform die swells. Yet, the dominant quantity for the quality of the manufactured part is the flow homogeneity at the die exit, where a uniform velocity distribution is desired [59, Section 4.7]. The (shape) optimization of profile extrusion dies has been an active field of research for many years [39, 163, 183]. As pointed out in the review paper by Pittman [131], previous experiences were the driving factors of initial guidelines for design procedures, accompanied by pocket calculations and analytical formulas derived from reasonable simplifications of the governing conservation laws. However, with the increasing computational capabilities and the increasing complexity of cross-section geometries, computer-based design approaches have emerged that promise to be economically beneficial, further motivating research to tackle the computational optimization of flow channels inside extrusion dies.

1.2. Bioreactors

Bioreactors belong to the oldest tools for production processes. Their application ranges far back into ancient times, where they were already employed for producing consumer goods like beer and wine [101, Chapter 1]. Technological advances of the last decades enabled the break-through of bioreactors for cultivating food and pharmaceutical products [128], antibiotics [145], proteins and industrial enzymes [2], or vaccines [124]. Since bioreactors are a separate field of research, in the course of this thesis, we will restrict ourselves to the related engineering aspects and begin with the following brief definition: According to Mandenius [101], a bioreactor provides and maintains an environment in which biological

compounds react and interact with each other. More precisely, the goal is to provide the best possible conditions for these reactions.

There is a multitude of different reactor designs (an overview has , e.g., been comprised in [101, Figure 1.2]). This thesis only considers the Continuously-Stirred Tank Reactor (CSTR) — one of the most common choices [101, Chapter 1]. A CSTR consists of a cylindrical reactor volume, eventually equipped with *baffles*, i.e., thin vertical panels, mounted to the reactor wall and a central stirrer axis with multiple impellers. For operation, the biological compounds are added into the vessel volume and the mixture is agitated by a constant rotation of the stirrer. Some CSTRs also feature *spargers* which provide the cells inside the bioreactor with a constant supply of oxygen. Beside the different reactor types, even in the case of CSTRs, the vessel volumes can comprise up to thousands of liters with height-to-diameter aspect ratios ranging from 1 : 1 to 3 : 1 [127, Chapter 6]. This is complemented by different impeller designs, aeration concepts, and operation modes (i.e., bioreactors can be operated in batch, fed-batch, or continuous culture mode), resulting in a combinatorial variety of different designs. Consequently, selecting the best design for a specific application is far from being a trivial task.

As pointed out by Ozturk and Hu [127], "it is a time consuming and expensive task to undertake an experimental evaluation to compare the relative performance of competing culture systems for a specific application" and the paradigms for designing bioreactors have shifted from experience- or experimental-driven considerations to employing computers for performing more complex analyses [101, Chapter 1]. However, industrial practice so far often hesitated to employ classical mathematical modeling, and preferred statistical data analysis to optimize or improve bioreactor design or operation. The reason for this lies in the complexity of the physical phenomena that need to be taken into account, including multiple phases, i.e., gaseous, liquid, solid, corresponding to oxygen, nutrient solution and biological cells respectively, as well as unsteady turbulent flow patterns, e.g., introduced by the baffles for enhancing the mixing inside the bioreactor [101, Chapter 10]. Although the advances in computational sciences of the last years enable and support the design and analysis of bioreactor dynamics at different levels of granularity, solving the resulting mathematical models computationally usually requires large amounts of resources, thus rendering tasks like the optimization of these systems challenging, as they require repeated model evaluations. This motivates the need of reliable ROMs which provide sufficiently accurate predictions to tackle tasks like, e.g., the design of bioreactors.

The latter is a multi-objective process where different aspects need to be considered, such as optimizing the transport of media inside the reactor or ensuring the industrial operability of the performed reactions. A non-exhaustive list of different design criteria can, e.g., be found in [101, Table 1.2]. Examples for design criteria are mixing efficiency, rheology, or homogeneity of culture, just to name the ones that can be optimized with the help of CFD simulations. Actuating variables of the optimization can modify both the geometry or the operation. From a practical point of view, the mixing inside a bioreactor — with respect to both temperature and concentration gradients — is crucial [127, Chapter 6] to obtain proper cell growth and consequently a good yield. In industrial practice, one tries to operate a bioreactor in a regime that is as well-mixed as possible [101, Chapter 10]. The mixing inside a bioreactor is directly related to the mass transfer between the different phases. The mass transfer can, e.g., be controlled — in the design phase — by geometrical quantities like the impeller diameter, the number of impellers, their positioning, or the presence of baffles, while — on an operational level — it can be controlled by the stirrer

speed or the gas inlet flow rate. These design choices, however, are constrained by the fact that the employed biological cells are often sensitive to shear rates and can be damaged if the shear forces exerted onto the fluid are too high [127, Chapter 6]. In consequence, different objectives need to be balanced depending on the application and different designs might be chosen in industry as opposed to academia. Yet, academic research in the field of bioreactors has positively influenced the industrial application, e.g., by reducing process development times [101, Chapter 1], which highlights the benefits of further investigating possibilities for improvements by the means of computational methods.

1.3. Model-Order Reduction

As pointed out in the previous sections, the analysis of real-world production processes by the means of numerical computer simulations is often extremely time-consuming and memory-intensive. In scenarios where single fully resolved flow simulations already take many hours of run time on advanced supercomputers to complete, the repeated evaluation of these simulations becomes too expensive and multi-query applications intractable. Consequently, MOR is often the only option to include high-fidelity CFD simulations into the engineering practice and tackle challenges like, e.g., (shape/design) optimization, automatic control, inverse problems, uncertainty quantification, or real-time evaluation [58, Chapter 3]. In general, "MOR methods are able to extract the dominant behavior by reducing the size of the system to be solved" [10, Chapter 1], yielding ROMs that approximate the full-order model but at much lower computational costs.

In the past decades, a lot of research has been dedicated to develop different techniques for constructing reduced models, including the *Reduced Basis Method* [58] or the *Proper Generalized Decomposition* [25], just to name some. Further information can, e.g., be found in [9, 10]. While researchers began to employ MOR techniques to flow problems already in the last century, e.g., [32], they have by now become widely adopted in the CFD community [82], especially for large-scale industrial applications [149]. However, with the rise and various successes of ML methods, strategies have been devised to apply these techniques also in the context of MOR for CFD. These methods are very flexible and the large amount of available software packages allows to readily develop first prototypes for data-based ROMs. An overview of ML methodologies and their impact on fluid mechanics has recently been comprised in [19].

1.4. Parametric Shape Optimization

As motivated in the previous sections, shape optimization is one prominent example of a multi-query scenario that often occurs in engineering practice and is also of interest in this thesis. One can differentiate among different classes of shape optimization problems, namely, *parametric shape optimization*, *geometric shape optimization*, or *topological shape optimization* [102, Chapter 11], where we only consider the first type in this work. Generally, in classical shape optimization problems, the goal is to find a design that is optimal with respect to a distinct quantity of interest quantified by an objective function. In engineering applications, e.g., when considering the flow in a profile extrusion die or a bioreactor, this quantity of interest is usually constrained by Partial Differential Equations

(PDEs) [5]. In order to evaluate the objective function, the solution of the corresponding PDE problem is required. Most PDE problems cannot be solved analytically and numerical simulations are performed instead. In the case of a parametric shape optimization problem, the shape is parameterized by a set of design parameters, representing the design variables of the optimization problem. The typical solution procedure for solving a parametric shape optimization problem is visualized in Figure 1.1.

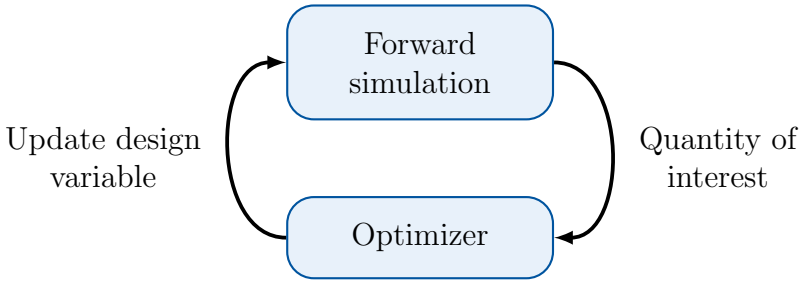


Figure 1.1.: Information flow in a parametric shape optimization problem. From the computational results of a forward simulation, the desired quantity of interest with respect to the goal of the optimization is evaluated and provided to an optimizer, which in turn computes the new design variables for the next iteration.

Starting from an initial guess for the design variables parameterizing the shape to be optimized, the quantity of interest is computed from the results of the forward simulation solving the underlying PDE problem. The respective value is then provided to an optimization algorithm, which will update the design variables accordingly. If the gradient of the objective function with respect to the design variables is available, gradient-based optimization algorithms like steepest-descent or variants thereof can be employed. However, since oftentimes gradients are not available or too costly to compute, gradient-free optimization methods, like genetic algorithms, are very popular, despite their limitations in problem size [81]. After the modification of the design variables by the optimizer, the forward simulation will be performed again. This iteration is continued until a sufficiently optimal geometry is obtained.

Since a more in-depth discussion of (shape) optimization problems and possible solution strategies is outside the scope of this work, we refer to the literature for further information [5, 102, 123].

1.5. Thesis Outline

After giving a motivation for the work conducted in this thesis and introducing the different application cases in the previous sections, the rest of this document is structured as follows:

In Chapter 2, we briefly explain the fundamentals of modeling incompressible fluid flows. These physical conservation laws form the basis for the derived ROMs in the subsequent chapters. We will comment on the closure equations that are relevant for this thesis and elaborate on the Initial and Boundary Conditions (IBCs) that are necessary for a well-posed problem formulation. Concluding, we will introduce the Finite Element Method (FEM) which will be used throughout this thesis for generating validation data for the ROMs. We explain the general idea and highlight shortcomings of the method as well as possible remedies.

Chapter 3 then focuses on the data science methods which are employed in this work. We begin by giving a brief taxonomy of the different fields of AI. This is followed by an introduction to NNs, in particular Feed-Forward Neural Networks (FFNNs), in Section 3.1. Here, we formalize the term NN mathematically, followed by a description of the training procedure, a simple framework for understanding the errors associated with NNs approximations, and a collection of heuristics giving advice for employing NN in practice. Building up on the general-purpose FFNN architecture, we investigate how knowledge about the underlying physical can be incorporated into the training of a NN with the aim to construct physics-informed ROMs. Especially interesting for this thesis are the PINNs to which Section 3.2 is dedicated. We discuss how the previously mentioned training process of classical FFNNs needs to be adapted to incorporate the physics, emphasize well-known caveats of this method but also advanced techniques for overcoming these difficulties, and close the section by mentioning some alternative approaches. Next, we move on to a different branch of ML, providing a brief overview of RL by introducing the classical interaction-loop between an agent and its environment. We will present a taxonomy of the different classes of RL agents, i.e., the different learning algorithms, and afterward outline one possibility for accelerating the learning process by letting a single agent interact with multiple environments at the same time. This concludes the theoretical parts of this thesis.

The following Chapters 4 and 5 document the numerical experiments performed to answer the previously posed Research Questions (RQ I) and (RQ II). Chapter 4 is dedicated to the investigation of Research Question (RQ I). After presenting the software framework developed in this thesis and used for carrying out the experiments in Section 4.1, Section 4.2 focuses on the construction of reduced simulation models for profile extrusion, while Section 4.3 deals with the application case of bioreactors. We remain in the same application but move on to the methodology of RL in Chapter 5 with the aim of answering Research Question (RQ II). We start again by giving a concise overview of the employed software setup in Section 5.1, before studying two test cases in Sections 5.2 and 5.3. The first test case is rather of academic nature and allows to assess the correctness of the optimized geometries. With the second test case, we move closer toward real-world applicability and the original goal of optimizing flow channels in profile extrusion dies by using a state-of-the-art objective for quantifying the flow homogeneity at the extruder outlet.

Finally, Chapter 6 recapitulates the research gaps pointed out above and emphasizes the contributions of this thesis to close them. Moreover, we will identify directions for future research, building up on the findings of this work.

2 Theory

This chapter provides a very brief introduction to the theoretical foundations of computational modeling. We will first introduce the governing equations and afterward outline the Finite Element Method (FEM) as a possible way to solve them. Since this is not the main focus of the work, only the most important aspects will be discussed with the aim of providing a general overview of the topic and highlighting some numerical difficulties that make classical Computational Fluid Dynamics (CFD) approaches challenging. All sections include references to the literature, in which more detailed information about the different topics can be found.

2.1. Conservation Laws of Fluid Flow

The flow of fluids can be described by conservation laws, i.e., the conservation of mass, momentum and energy. Generally, these laws form a system of non-linear Partial Differential Equations (PDEs). In an Eulerian frame-of-reference, the mass and momentum conservation equations on a spatio-temporal domain $\Omega \times (0, T] \subset \mathbb{R}^{n_{\text{sd}}} \times \mathbb{R}^{+}$ can be written in conservation form as

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \boldsymbol{v}) = 0 \qquad \text{in } \Omega \times (0, T] \,, \tag{2.1a}$$

$$\frac{\partial (\rho \boldsymbol{v})}{\partial t} - \boldsymbol{\nabla} \cdot (\boldsymbol{\sigma} - \rho \boldsymbol{v} \otimes \boldsymbol{v}) = \rho \boldsymbol{b} \qquad \text{in } \Omega \times (0, T] \,. \tag{2.1b}$$

The derivation of Equations (2.1a) and (2.1b) can, e.g., be found in [36, Section 1.4]. The equation system given in Equation (2.1) is not complete, as it requires additional closure equations to model the Cauchy stress tensor $\boldsymbol{\sigma}$. Only then is it possible to solve for the unknown density and velocity fields ρ and $\boldsymbol{v}$. The symbol $\boldsymbol{b}$ denotes a body or volume force field and can comprise, e.g., gravity forces.

The equations presented in (2.1) describe the flow of a compressible fluid. In this thesis, we consider only incompressible flows, i.e., the density ρ is a constant material property. Under this assumption, the governing Equations (2.1a) and (2.1b) reduce to

$$\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0 \qquad \text{in } \Omega \times (0, T] \,, \tag{2.2a}$$

$$\rho \left(\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \boldsymbol{\nabla}) \boldsymbol{v} \right) - \boldsymbol{\nabla} \cdot \boldsymbol{\sigma} = \rho \boldsymbol{b} \qquad \text{in } \Omega \times (0, T] \,. \tag{2.2b}$$

In the incompressible case, the pressure p replaces the density as an unknown quantity, that is, we solve Equations (2.2a) and (2.2b) for the velocity $\boldsymbol{v}$ and the pressure p. The next section will introduce the required closure equation for system (2.2).

2.2. Closure Equations

To obtain a closed system of equations, the Cauchy stress tensor needs to be modeled in terms of the unknowns $\boldsymbol{v}$ and p. By taking

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} - p\boldsymbol{I} \ ,$$

Equation (2.2b) becomes what is known as the *incompressible Navier-Stokes* equations, where $\boldsymbol{\varepsilon}$ denotes the rate-of-strain tensor

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\boldsymbol{\nabla v} + (\boldsymbol{\nabla v})^T\right) \ ,$$

and μ is the dynamic viscosity. The latter is a thermodynamic material property, which can depend on temperature and pressure. If the viscous part of the fluid's internal stresses, that is $2\mu\boldsymbol{\varepsilon}$, is linearly related to the velocity gradient $\boldsymbol{\nabla v}$, the fluid is said to be *Newtonian* [126]. Not making this assumption and modeling the viscosity μ instead as a function of the invariants of the rate-of-strain tensor $\boldsymbol{\varepsilon}$, gives rise to the class of *Generalized Newtonian* fluids [126]. *Shear-thinning* fluids are one example from this class, where the viscosity decreases with increasing shear rates [125]. Such shear-thinning behaviour is exhibited by many plastics melts [125] and various appropriate mathematical models exist [59]. In this work, we will only employ the semi-empirical relation found by Carreau [23]

$$\mu = \frac{A}{(1 + B\dot{\gamma})^C} \ , \tag{2.3}$$

which relates the viscosity to the shear-rate[1]

$$\dot{\gamma} = \sqrt{2\boldsymbol{\varepsilon} : \boldsymbol{\varepsilon}} \ .$$

This Carreau model is often used for describing the shear-thinning flow in profile extrusion dies due to its ability to accurately predict viscosities over a broad range of shear rates, including the special case of vanishing shear rates [59]. The parameters A, B, and C are material-specific and usually determined by measurements for different types of plastic. A corresponds to the viscosity at zero shear, B denotes the reciprocal transition rate and C the slope of the viscosity curve for $\dot{\gamma} \to \infty$.

2.3. Initial and Boundary Conditions for the Incompressible Navier-Stokes Equations

The PDEs as stated in Equation (2.2) form a parabolic initial-boundary value problem. To obtain a well-posed problem, it must be completed with proper Initial and Boundary Conditions (IBCs).

[1] Note that this quantity can be expressed in terms of the second invariant of the rate-of-strain tensor (see e.g. [100]) and thus satisfies the characteristic of a Generalized Newtonian fluid given by [126].

Initial Conditions (ICs) are only required for the unknown velocity field and are typically provided as

$$\boldsymbol{v}|_{t=0} = \boldsymbol{v}_0 \quad \text{in } \Omega \,, \tag{2.4}$$

where the initial velocity field $\boldsymbol{v}_0$ has to fulfill the mass conservation (2.2a), i.e.,

$$\boldsymbol{\nabla} \cdot \boldsymbol{v}_0 = 0 \quad \text{in } \Omega \,.$$

For the Boundary Conditions (BCs), different approaches have been investigated in the literature, e.g., in [134]. One common type of Boundary Conditions (BCs) are Dirichlet-type conditions, which explicitly define the value of an unknown quantity, e.g., the velocity, on (a part of) the spatial boundary $\Gamma_\mathrm{D} \subseteq \Gamma$

$$\boldsymbol{v} = \boldsymbol{v}_\mathrm{D} \quad \text{on } \Gamma_\mathrm{D} \times (0, T] \,. \tag{2.5}$$

Examples for these types of BCs are solid walls, where the fluid adheres to the wall, i.e., in the case of a non-moving wall, $\boldsymbol{v}_\mathrm{D} = \boldsymbol{0}$ is prescribed, or inflows, where the spatially and temporally varying velocity profile is imposed. Similar to the IC, these Dirichlet BCs need to obey the mass conservation (2.2a). Inserting Equation (2.5) into Equation (2.2a) and employing the Gauss theorem, this can be formulated as an integral constraint that has to hold on the whole spatial boundary for all time instances

$$\oint \boldsymbol{v}_\mathrm{D} \cdot \boldsymbol{n} \, \mathrm{d}\Gamma = 0 \quad \text{on } \Gamma \quad \forall\, t \in (0, T] \,. \tag{2.6}$$

In the case of transient problems, the above described IBCs Equations (2.4) and (2.5) need to be compatible, i.e.,

$$\boldsymbol{v}_0|_\Gamma = \boldsymbol{v}_\mathrm{D}|_{t=0} \,.$$

Another type of velocity BCs include Neumann-type conditions and are employed, e.g., on symmetry planes or on outflow boundaries.

BCs for the pressure are less common. Conditions for the pressure can be prescribed on boundaries with a Neumann-type condition for the velocity, while they should never be prescribed on boundaries with Dirichlet-type conditions as in Equation (2.5). Details can be found in [134, Section 5.7]. However, there exist natural BCs involving the pressure, e.g., when a force $\boldsymbol{f}$ is exerted onto the surface of the fluid. From Equation (2.2b) it follows that such a BC can be set by enforcing the relation [80, Chapter 2, §15]

$$-\boldsymbol{\sigma} \cdot \boldsymbol{n} = \boldsymbol{f} \quad \text{on } \Gamma \times (0, T] \,. \tag{2.7}$$

Care must be taken when only Dirichlet conditions for the velocity are prescribed on all parts of the boundary: In an incompressible flow, the pressure is only determined up to a constant. To uniquely define the pressure, one can fix its value at one point inside the domain or set its average value [36, Section 6.2.3].

2.4. Finite Element Discretization

There exist a variety of numerical methods for solving PDE systems like Equations (2.2a) and (2.2b). Among the most popular methods are Finite Differences, Finite Volumes and the FEM. In this thesis, we will only employ the latter for discretizing equation system (2.2). The FEM solves a weak integral formulation instead of the strong form of the PDE system. By doing so, the regularity requirements of the solution are reduced, which potentially allow the search for a solution in a larger space.

As the solution of problem (2.2) using the FEM is not the main focus of this thesis, we will only highlight the most important points and refer to the literature on FEM for a more extensive explanation [36, Section 1.5],[130, Chapter 5]. Here, we will introduce FEM by considering Equation (2.2) equipped with both Dirichlet and Neumann BCs according to Equations (2.5) and (2.7) on the Dirichlet Γ_D and Neumann Γ_N parts of the spatial boundary $\Gamma = \Gamma_\mathrm{D} \cup \Gamma_\mathrm{N}$. The weak form of Equation (2.2) then reads:

Find $(p, \boldsymbol{v}) \in \mathcal{Q} \times \mathcal{S}$ *such that for all* $t \in (0, T]$

$$\int q \boldsymbol{\nabla} \cdot \boldsymbol{v} \, \mathrm{d}\Omega + \int \left(\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \boldsymbol{\nabla}) \boldsymbol{v} \right) \cdot \boldsymbol{w} \, \mathrm{d}\Omega + \oint \boldsymbol{f} \cdot \boldsymbol{w} \, \mathrm{d}\Gamma_\mathrm{N} + \int \boldsymbol{\sigma} : \boldsymbol{\nabla} \boldsymbol{w} \, \mathrm{d}\Omega = \int \rho \boldsymbol{b} \cdot \boldsymbol{w} \, \mathrm{d}\Omega \,, \tag{2.8}$$

for all $(q, \boldsymbol{w}) \in \mathcal{Q} \times \mathcal{V}$.

The derivation can be found in Appendix A. We employ the same definition of the solution spaces as in [36, Section 6.4], i.e.,

$$\begin{aligned} \mathcal{S} &\coloneqq \left\{ \boldsymbol{v} \in (\mathcal{H}^1)^{n_\mathrm{sd}}(\Omega) \mid \boldsymbol{v} = \boldsymbol{v}_\mathrm{D} \quad \text{on } \Gamma_\mathrm{D} \right\} \,, \\ \mathcal{V} &\coloneqq \left\{ \boldsymbol{w} \in (\mathcal{H}^1)^{n_\mathrm{sd}}(\Omega) \mid \boldsymbol{w} = \boldsymbol{0} \quad \text{on } \Gamma_\mathrm{D} \right\} \,, \\ \mathcal{Q} &\coloneqq \mathcal{L}^2(\Omega) \,. \end{aligned}$$

Here, we can observe the previously mentioned reduced regularity requirements in practice: While the velocity needed to be twice and the pressure once continuously differentiable in the strong form of the incompressible Navier-Stokes equations, in the weak form, the velocity is only required to be weakly differentiable once, while square-integrability even suffices for the pressure.

The continuous weak form (2.8) is discretized on a computational mesh $\Omega^h \subset \Omega$ using a Galerkin FEM approach with linear interpolation functions velocity and pressure. This discretization however introduces numerical problems:

- The classical Galerkin approach is not well-suited for solving convection-dominated flows [36, Chapter 2], i.e., flows with a large Reynolds number $\mathrm{Re} \gg 1$[2]: A Galerkin discretization of the convective term $(\boldsymbol{v} \cdot \boldsymbol{\nabla}) \boldsymbol{v}$ introduces a truncation error in the form of negative diffusion – practically manifesting in node-to-node oscillations in the numerical solution.

[2]The Reynolds number Re is a dimensionless index describing the relation of inertia and viscous forces.

- An additional problem arises from the incompressibility assumption: In contrast to compressible flows, the mass and momentum conservation equations are only weakly coupled as the pressure appears only in the latter. The mass conservation equation reduces to an equality constraint that the solution needs to satisfy. Therefore, a numerical algorithm needs to compute a pressure field in such a way that the resulting velocity field satisfies the mass conservation. In FEM, this weak coupling results in a constraint on the chosen interpolation spaces for velocity and pressure and is known as *LBB condition*[3]. The stability of the numerical discretization can only be guaranteed for interpolation spaces that fulfill this condition. This is, however, not the case when linear functions are used for both velocity and pressure, resulting in, e.g., oscillating pressure modes.

The aforementioned problems are cured in an FEM framework by employing so-called stabilized FEMs, which is a separate field of research [42, 169]. The general idea of stabilized FEMs is to add mesh-dependent terms to the Galerkin formulation in a consistent way to provide sufficient control of the unknowns of the problem. In our FEM solver, we employ Streamline-Upwind Petrov-Galerkin (SUPG) [18] and pressure-stabilizing/Petrov-Galerkin (PSPG) [168] terms, which stabilize the presented formulation at moderately-large Reynolds numbers and enable equal-order interpolation of pressure and velocity.

[3] Also referred to as *inf-sup* condition, this theorem was named after the mathematicians Ladyzhenskaya, Babuška and Brezzi.

3

An Introduction to Scientific Machine Learning: Relevant Concepts

This chapter aims to provide a brief introduction to the complex topics of Artificial Intelligence (AI) and Scientific Machine Learning (SciML), with a particular focus on explaining the meaning of the most commonly used — and often confused — terms in this context. However, since an in-depth explanation of these terms is beyond the scope of this work, we will restrict the discussion to the most important concepts that are of immediate relevance for understanding the work presented in this thesis. We first introduce a simplified taxonomy, which is inspired by [50, Figure 1.4].

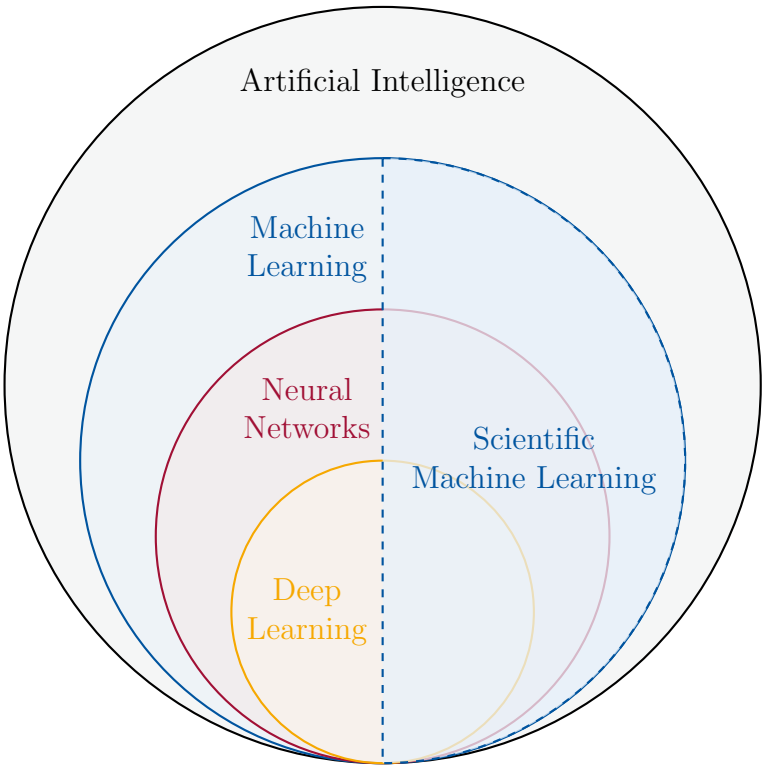


Figure 3.1.: A brief taxonomy of AI and selected subfields that are related to this thesis, inspired by [50, Figure 1.4].

As one can see in Figure 3.1, Machine Learning (ML) is a subfield of AI, which in itself is a branch of computer science and statistics. An important subfield of ML is Deep Learning (DL) which focuses on the design of Neural Networks (NNs), the driving force behind the increased interest in ML by the scientific community. With this thesis, we want to particularly focus on employing methods from the field of ML to scientific computing. This has by now become a research area of its own, referred to as SciML. In the following, we give brief definitions for each of these disciplines with respect to their relevance in this work.

Artificial Intelligence (AI) The field of AI first emerged as a discipline of computer science after a "summer research project" conducted at Dartmouth College in 1956 [106], held by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. In their proposal, published in 1955 prior to the workshop, they were convinced that "[...] every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it" and they set it as their goal "[...] to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves." While this is no official definition of the term AI, it describes the goals of this field well. For more details on AI, we refer to the literature, e.g., [144, 179, 78].

Machine Learning (ML) ML is a part of AI and presents one class of methods to accomplish the goals formulated by McCarthy et al. [106]. A formal definition of ML is given by Mitchell [111]:

> A computer program is said to learn from experience $\mathcal{E}$ with respect to some class of tasks $\mathcal{T}$ and performance measure $\mathcal{P}$, if its performance at tasks in $\mathcal{T}$, as measured by $\mathcal{P}$, improves with experience $\mathcal{E}$.

While this definition covers the common ground of all ML methods, one open question remains, namely, how exactly the "computer program" of Mitchell's definition is supposed to learn. Answering this question provides one possible way of further categorizing the variety of available ML algorithms, i.e., one typically distinguishes between supervised, unsupervised and Reinforcement Learning (RL):

- In **Supervised Learning** the experience $\mathcal{E}$ consists of both inputs $\bar{\boldsymbol{x}}$ and desired outputs (target values) $\bar{\boldsymbol{y}}$ of the computer program. In the simplest case, the performance measure $\mathcal{P}$ quantifies the difference between the output of the computer program and the target values. Possible tasks $\mathcal{T}$ are usually regression or classification tasks, depending on whether the outputs are continuous or discrete. For this thesis, we will only focus on the continuous case, i.e., regression.
- In **Unsupervised Learning**, the experience $\mathcal{E}$ only consists of inputs, but not the desired outputs. Possible related tasks $\mathcal{T}$ are clustering (i.e., identifying subgroups within the data that are similar) or dimensionality reduction (e.g., by projecting the data into a lower-dimensional space) [15]. As a consequence, the performance measure $\mathcal{P}$ now determines the distance between two samples of the provided experience (in the case of clustering) or the amount of information contained in the lower-dimensional representation (in the case of dimensionality reduction).

- The last class of methods is known as **Reinforcement Learning (RL)**. Here, the experience $\mathcal{E}$ is not provided to the computer program in the form of a data set from the onset, but instead is collected via a trial-and-error interaction: The computer program tries to solve a task $\mathcal{T}$ by undertaking actions following a learned strategy and, in doing so, is guided by a numerical reward signal judging the quality of the performed action with respect to the program's objective. A more in-depth introduction into this topic will be given in Section 3.3.

The above-presented taxonomy of ML methods is not strict, i.e., some methods cannot be assigned to just one of the mentioned categories unambiguously. This difficulty holds especially true for the distinction between supervised and unsupervised learning, as there are methods that learn in a semi-supervised way[1].

Scientific Machine Learning (SciML) SciML only emerged relatively recently as a subfield of ML aiming to bridge the gap between the ML and scientific computing communities. This field comprises enhancing classical ML methods based on a-priori-known physical principles like, e.g., conservation laws or symmetries, complementing existing numerical methods by ML techniques, e.g., learning constitutive equations from data, or tackling the complex task of providing decision support for real-world (industrial) applications by combining numerical simulations and experimental data [6].

Neural Networks (NNs) NN form one possible class of ML algorithms. By now, there exists a variety of architectures for different purposes, including Convolutional Neural Networks (CNNs) (e.g., used for image classification / segmentation tasks), Recurrent Neural Networks (RNNs) (e.g., used for speech recognition) or the classical Feed-Forward Neural Networks (FFNNs), which this thesis will focus on. They will be introduced in more detail in Section 3.1.

Deep Learning (DL) Finally, DL is the last field mentioned in Figure 3.1. According to the work of the same title by Goodfellow, Bengio, and Courville [50], DL is a part of ML aiming to represent relations hierarchically by learning different layers of abstractions that build upon each other. Examples for DL algorithms are the aforementioned CNNs and RNNs.

After introducing selected parts of AI above, we continue in the following with a brief and non-exhaustive overview of the most important events in the history and development of SciML. To avoid getting into too much detail, we only consider events after the field of AI was officially founded during the Dartmouth conference, already mentioned above.

In 1958, Rosenblatt [146] was the first scientist to introduce the *perceptron* as a mathematical model to describe how information is aggregated, stored, and used inside the human brain. Furthermore, he introduced the *perceptron learning* algorithm, which made it possible to employ this method for solving concrete problems. While the original perceptron only consisted of an input and an output layer between which information was propagated linearly, his work was soon extended to so-called *Multi-Layer Perceptrons (MLPs)*, which

[1] As we will argue in Section 3.2, Physics-Informed Neural Networks (PINNs) can be seen as both a supervised and unsupervised learning approach.

consisted of a hierarchical composition of multiple perceptrons. These MLPs were the predecessor of a class of NNs known today as FFNNs[2]. However, they were significantly harder to train and it took until 1967 that Ivakhnenko, Lapa, and McDonough [63] published the first working learning algorithm, which allowed to train MLPs for more realistic problems. Yet, only two years later, in 1969, Minsky and Papert [109] published their book *Perceptrons* in which they showed the caveats of the perceptron model proposed by Rosenblatt [146]. Precisely, they showed that a two-layer perceptron was incapable of usefully representing functions outside a very narrow and special class. While they merely argued that multi-layer perceptrons with only a single layer of adaptive weights (i.e., a special type of MLP) could not accurately represent arbitrary functions [14, Section 3.5.4], their proposition was commonly misunderstood to mean that FFNNs were not suited for real-world applications. Consequently, the interest in NNs faded until Rumelhart, Hinton, and Williams [151] invented the *back-propagation algorithm* in 1986 which presented an efficient way to train MLPs. This resurrected the interest in NNs, complemented by various achievements: For instance, in 1989 LeCun et al. [83] successfully applied the back-propagation algorithm to train a deep NN for recognizing hand-written zip codes. The overall training time of the network however still required three days. Aside from practical success, notable advances were made on the theory of ML as well: Still in 1989, Cybenko [29] and Hornik, Stinchcombe, and White [60] were able to prove that MLPs with a single hidden layer and sigmoidal activation functions are able to approximate any continuous function up to an arbitrary accuracy. What became known as the *universal function approximation theorem* layed the theoretical foundation for the success of NNs. A couple of years later, in 1993, Leshno et al. [85] extended this theorem for a wider class of activation functions, including the well-established Rectified Linear Unit (ReLU). After these years, the interest in NNs faded again and shifted towards different learning methods, e.g., Support Vector Machines (SVMs) [27], as they were easier to understand and train. Only in the late 2000s, the interest in NNs rose again due to the availability of more powerful Graphics Processing Units (GPUs) and large data sets [50]. Since then, ML and especially NNs have gained increasing interest in various disciplines: In the beginning, ML techniques were mostly employed for image classification [75], image segmentation [156], or speech recognition [114] and it took time until ML methods found their way into the field of scientific computing. Eventually, in 2019, the US Department of Energy published a report, in which the term SciML was used for the first time to describe the symbiosis of ML and scientific computing [6]. In this report, SciML is described as a "core component" of AI and — during the corresponding workshop conducted in 2018 — different research directions were identified to benefit from SciML techniques on a long-term basis [6].

The rest of this chapter will introduce the SciML concepts that form the core of this thesis. As we have seen in the previous paragraph, the development of the perceptron (and ultimately NNs) played a major role in the success story of ML. Therefore, we will introduce them mathematically in the upcoming Section 3.1. We will also formalize the learning process and discuss related strategies, e.g., like choosing a proper NN architecture. After introducing these basics, Section 3.2 will deal with PINNs that only recently emerged

[2]Within this chapter, we will use the term MLP for historical reasons when introducing the mathematical basis, while for the remainder of this thesis, rather the term NN will be used synonymously for FFNNs, as these are the only type of NNs considered in this work.

as a natural class of methods that tries to bridge the gap between physical modeling and ML. We will investigate the implications of introducing physical laws into the learning process and discuss advantages and drawbacks compared to classical FFNNs. This will form the theoretical basis for understanding the results presented in Chapter 4. Finally, Section 3.3 concludes this chapter by providing an introduction to RL. We will familiarize the reader with the general terminology that slightly differs from classical ML and comment on the relation to the previously introduced FFNNs by explaining how an algorithm can gather experience from trial-and-error interactions.

3.1. An Introduction to Feed-Forward Neural Networks

After giving a broad but superficial overview of AI and SciML, this section now specifically deals with Feed-Forward Neural Networks (FFNNs). We will first explain some general terminology before continuing with the (mathematical) details of this method.

FFNNs are usually employed for classification or regression tasks where they process vectorial input data of a fixed size. In this thesis, we only employ them for regression tasks. The general regression task can be stated as follows:

> Given a data set $\mathcal{D}$ of inputs (or *features*) $\bar{\boldsymbol{x}} \in \mathcal{X} \subset \mathbb{R}^{N_\mathcal{X}}$ and outputs (or *observations*) $\bar{\boldsymbol{y}} \in \mathcal{Y} \subset \mathbb{R}^{N_\mathcal{Y}}$, find the function (or *model*) $f : \mathcal{X} \to \mathcal{Y}$ from a fixed model class (or *hypothesis space*) $\mathcal{H}$ that best describes the structure in the data.

Here, $N_\mathcal{X}, N_\mathcal{Y} \in \mathbb{N}^+$ denote the dimension of the input and output space respectively. In ML terminology, this process of finding the best model given the provided data set is called *learning* or *training*. The choice of the hypothesis space is crucial for the learning process, as it poses a restriction on which structures in the data can be captured by the model. One well-known example of regression is linear regression, where the hypothesis space $\mathcal{H}$ is chosen as the set of linear functions. FFNNs belong to a class of functions capable of approximating even highly nonlinear relations inherent in the data.

3.1.1. Mathematical Basics of Feed-Forward Neural Networks

We begin our mathematical introduction with the perceptron model proposed by Rosenblatt [146]. Following the terminology in neuroscience and the brain's structure, a perceptron is nowadays also referred-to as *neuron*. For simplicity, our perceptron maps multiple inputs onto a single output. Let $\bar{\boldsymbol{x}} \in \mathcal{X} \subseteq \mathbb{R}^{N_\mathcal{X}}$ denote the real-valued input vector and let $\bar{y} \in \mathbb{R}$ be the corresponding response. The perceptron model then reads

$$\bar{y} = \sigma\left(b + \sum_{j=1}^{N_\mathcal{X}} w_j \bar{x}_j\right) \ ,$$

which can be written in compact form as

$$\bar{y} = \sigma\left(b + \boldsymbol{w}^T \bar{\boldsymbol{x}}\right) \ , \tag{3.1}$$

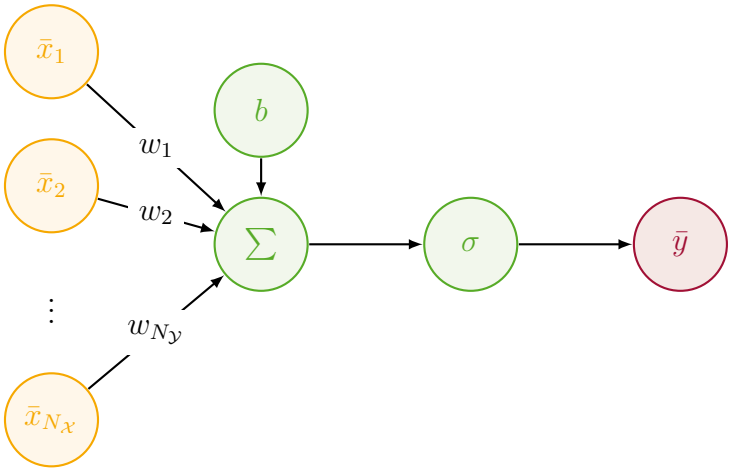


Figure 3.2.: Visualization of a perceptron (a single neuron) for a scalar-valued output.

where $\sigma(\,\cdot\,)$ denotes the (non-linear) activation function, $b \in \mathbb{R}$ the bias and $\boldsymbol{w} \in \mathbb{R}^{N_\mathcal{X}}$ the vector of weights. Figure 3.2 visualizes a perceptron as defined by Equation (3.1).

While b and $\boldsymbol{w}$ are determined during the model training (which will be described in more detail in Section 3.1.2), the activation function needs to be selected before the model is trained, making it a so-called *hyper-parameter*. There exists a variety of activation functions to be used in regression tasks, among which the most well known are the ReLU, sigmoid, and hyperbolic tangent activation functions. For more detailed information, we refer to the recent review paper [65].

Now let us assume that we do not have a scalar-valued output, but instead a vector-valued output $\bar{\boldsymbol{y}} \in \mathcal{Y} \subseteq \mathbb{R}^{N_\mathcal{Y}}$. Equation (3.1) then describes how to compute a single component of the vector-valued output and we can generalize the notation as follows:

$$\bar{\boldsymbol{y}} = \sigma\left(\boldsymbol{b} + \boldsymbol{W}^T \bar{\boldsymbol{x}}\right) \ . \tag{3.2}$$

Here, the bias has become a vector $\boldsymbol{b} \in \mathbb{R}^{N_\mathcal{Y}}$ and the matrix $\boldsymbol{W} \in \mathbb{R}^{N_\mathcal{X} \times N_\mathcal{Y}}$ now contains the weight vectors from Equation (3.1) as columns. The activation function is applied component-wise.

Originally used for classification tasks, the perceptron Equation (3.2) corresponds to a linear discriminant with a non-linear activation function [15]. In general, methods of this type can only find a solution to a problem if the decision boundary, i.e., the boundary separating the different classes, is linear. Although this can — in some cases — be overcome by adding a fixed transformation of the input features [147], another problem evolved, namely, finding feature transformations that make the data linearly separable. Moreover, these transformations needed to be fixed before training the perceptron, i.e., they could not be determined automatically for arbitrary data sets. Consequently, to apply perceptrons to more complex problems, either an increasing amount of input features or more intricate feature transformations were required [14], rendering this method less attractive.

A natural extension of the perceptron model is learning the required feature transformations. Mathematically speaking, the perceptron model is extended by additional $N_\text{hl} \in \mathbb{N}$ *hidden layers*, creating a so-called MLP, also referred to as FFNN. We denote the total

number of layers by N_l. Since the output layer does not count as hidden layer, the following relation holds $N_\mathrm{l} = N_\mathrm{hl} + 1$. Then, an MLP can be recursively defined as

$$\boldsymbol{z}^{(1)} = \bar{\boldsymbol{x}} \tag{3.3a}$$

$$\boldsymbol{z}^{(l+1)} = f_\mathrm{P}^{(l)}\left(\boldsymbol{z}^{(l)}\right) \qquad \forall\, l \in \{1, ..., N_\mathrm{hl}\} \tag{3.3b}$$

$$\bar{\boldsymbol{y}} = f_\mathrm{P}^{(N_\mathrm{l})}\left(\boldsymbol{z}^{(N_\mathrm{l})}\right) \ , \tag{3.3c}$$

where

$$f_\mathrm{P}^{(i)}(\boldsymbol{z}) = \sigma^{(i)}\left(\boldsymbol{b}^{(i)} + \left(\boldsymbol{W}^{(i)}\right)^T \boldsymbol{z}\right) \ . \tag{3.3d}$$

An MLP as defined by the recursive relation of Equation (3.3) is depicted in Figure 3.3.

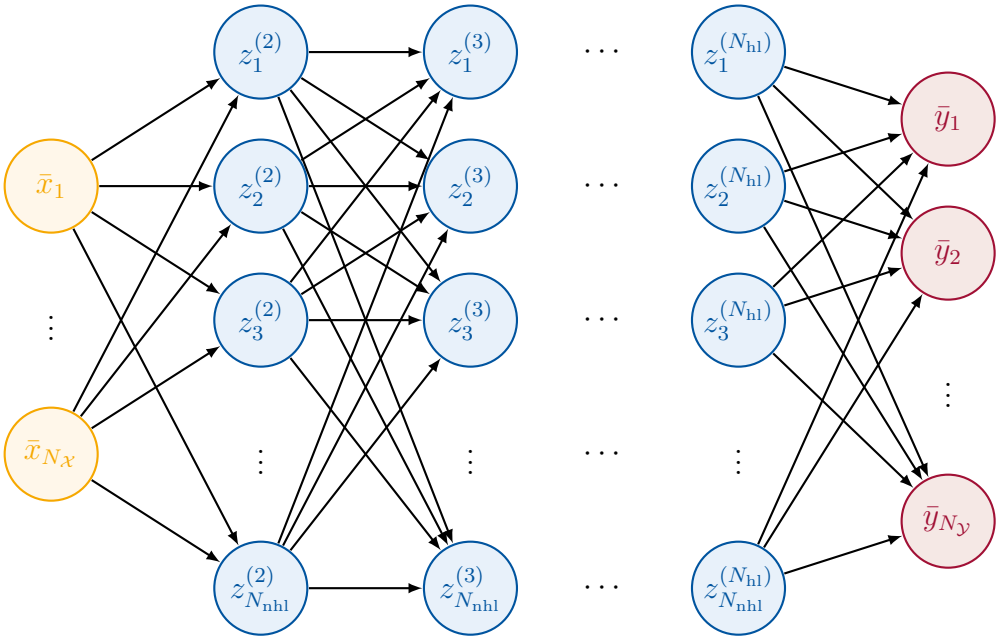


Figure 3.3.: Visualization of an MLP for the general case of a vector-valued output.

Equation (3.3) can also be understood as a composition of multiple perceptrons in the form given by Equation (3.2), which allows to rewrite it in a more compact notation:

$$f_\mathrm{FFNN} : \mathcal{X} \times \Theta \to \mathcal{Y}, (\bar{\boldsymbol{x}}, \boldsymbol{\theta}) \mapsto \bar{\boldsymbol{y}} = f_\mathrm{FFNN}\left(\bar{\boldsymbol{x}}; \boldsymbol{\theta}\right) = \left(f_\mathrm{P}^{(N_\mathrm{l})} \circ \ldots \circ f_\mathrm{P}^{(1)}\right)(\bar{\boldsymbol{x}}) \ . \tag{3.4}$$

Here, all parameters $\left\{(\boldsymbol{b}^{(l)}, \boldsymbol{W}^{(l)})\right\}_{l=1}^{N_\mathrm{l}}$ of Equation (3.3) have been combined into one single parameter vector $\boldsymbol{\theta} \in \Theta$ of the parameter space Θ. The main difference to the classical perceptron is the presence of the hidden layers, which enables the MLP to learn the required feature transformations. However, it also introduces additional hyper-parameters that need to be carefully selected, namely the number of (hidden) layers N_l (N_hl), as well as the width $N_\mathrm{nhl}^{(l)}$ of each hidden layer l, corresponding to the number of neurons in said layer. These hyper-parameters mentioned so far are also referred to as the *architecture* of a NN. We will briefly discuss the topic of finding a suitable architecture in Section 3.1.5.

For the remainder of this thesis, we will make the following simplifying assumptions about the NN architecture:

1. All hidden layers have the same number of neurons, denoted by N_{nhl}.

2. All hidden layers have the same activation function, i.e.,
   $\sigma^{(l)}(x) = \sigma(x) \quad \forall\, l \in \{1, ..., N_{\text{hl}}\}$

3. The activation function of the output layer is the identity, i.e., $\sigma^{(N_{\text{l}})}(x) = x$.

3.1.2. Training Feed-Forward Neural Networks: Learning from Data

After introducing the concept of FFNNs mathematically in Section 3.1.1, this section will address the question of how to determine the NN's parameters for a given application. NNs are generally trained in supervised manner. Therefore, a *data set* $\mathcal{D}$ is required which contains inputs and their corresponding outputs

$$\mathcal{D} \coloneqq \left\{ \left(\bar{\boldsymbol{x}}^{(1)}, \bar{\boldsymbol{y}}^{(2)} \right), \ldots, \left(\bar{\boldsymbol{x}}^{(N_{\mathcal{D}})}, \bar{\boldsymbol{y}}^{(N_{\mathcal{D}})} \right) \right\} \subset \mathcal{X} \times \mathcal{Y} \, .$$

Such a data set is assumed to be generated from an underlying – though unknown – *data generation process*, with the corresponding joint probability distribution $\mathbb{P}^{\text{data}}_{(\bar{\boldsymbol{X}}, \bar{\boldsymbol{Y}})}$, i.e., $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \sim \mathbb{P}^{\text{data}}_{(\bar{\boldsymbol{X}}, \bar{\boldsymbol{Y}})}$ and probability density $p^{\text{data}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. In practice, this data generation process can, e.g., be a computer simulation (in case of synthetically generated data) or a real world process from which measurements are taken.

The idea of the training is to determine the parameters of the FFNN Equation (3.4) such that the resulting model is the best approximation given the chosen hypothesis space and data set $\mathcal{D}$. Consequently, a measure is needed that specifies how close the current model prediction $\bar{\boldsymbol{y}}_{\text{pred}} = f_{\text{FFNN}}(\bar{\boldsymbol{x}}; \boldsymbol{\theta})$ is to its respective target $\bar{\boldsymbol{y}}$. This metric is known as *loss function* in the ML community. Different types of loss functions have been proposed in the literature, including the *absolute error* (also referred to as ℓ_1-*loss*) or the *squared error* (also referred to as ℓ_2-*loss*). The choice of this loss function L directly influences the final model, e.g., the squared error penalizes outliers more heavily than the absolute error. Throughout this thesis, we will always employ the squared error as loss function, i.e.,

$$L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^{+}, (\bar{\boldsymbol{y}}_{\text{pred}}, \bar{\boldsymbol{y}}) \mapsto L(\bar{\boldsymbol{y}}_{\text{pred}}, \bar{\boldsymbol{y}}) = \|\bar{\boldsymbol{y}}_{\text{pred}} - \bar{\boldsymbol{y}}\|_2^2 \, . \tag{3.5}$$

While the loss function measures only the closeness of individual data points, we need to define a measure for the total error of the approximation made by the NN f_{FFNN}. This is often referred to as the *expected risk* and can be computed as

$$\begin{aligned} \mathcal{L}^{\text{data}} &= \mathbb{E}_{(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \sim \mathbb{P}^{\text{data}}_{(\bar{\boldsymbol{X}}, \bar{\boldsymbol{Y}})}} \left[L\left(f_{\text{FFNN}}(\bar{\boldsymbol{x}}; \boldsymbol{\theta}), \bar{\boldsymbol{y}} \right) \right] \\ &\coloneqq \int_{\mathcal{X} \times \mathcal{Y}} L\left(f_{\text{FFNN}}(\bar{\boldsymbol{x}}; \boldsymbol{\theta}), \bar{\boldsymbol{y}} \right) p^{\text{data}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \, \mathrm{d}\bar{\boldsymbol{x}} \, \mathrm{d}\bar{\boldsymbol{y}} \, . \end{aligned} \tag{3.6}$$

This can be understood as the average of the loss function L over all possible data points $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ that can be generated by the data generation process $\mathbb{P}^{\text{data}}_{(\bar{\boldsymbol{X}}, \bar{\boldsymbol{Y}})}$. However, since the data generation process is generally unknown (as is its probability density), the expected risk cannot be computed in practice, but is instead approximated by a (discrete) estimator.

This estimator is evaluated on the previously mentioned data set $\mathcal{D}$ and is known as the *empirical risk* or *average loss*[3]

$$\mathcal{L}_{\mathcal{D}}^{\text{data}}(\boldsymbol{\theta}) \coloneqq \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} L\left(f_{\text{FFNN}}\left(\bar{\boldsymbol{x}}^{(i)}; \boldsymbol{\theta}\right), \bar{\boldsymbol{y}}^{(i)}\right) . \tag{3.7}$$

Here, the expectation is consistently estimated by the arithmetic sample mean.

The learning task is reduced to an optimization problem, namely, finding the model $f_{\mathcal{H},\mathcal{D}}^{\star}$ which minimizes the average loss on the hypothesis space $\mathcal{H}$ at hand

$$f_{\mathcal{H},\mathcal{D}}^{\star} = \underset{f_{\text{FFNN}} \in \mathcal{H}}{\arg\min}\, \mathcal{L}_{\mathcal{D}}^{\text{data}}(\boldsymbol{\theta}) .$$

However, instead of performing an optimization with respect to an unstructured function space, we leverage having parameterized the estimation model by optimizing over only said parameter space

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta} \in \Theta}{\arg\min}\, \mathcal{L}_{\mathcal{D}}^{\text{data}}(\boldsymbol{\theta}) . \tag{3.8}$$

Once the optimal parameters $\boldsymbol{\theta}^{\star}$ are found, the optimal model can be obtained according to

$$f_{\mathcal{H},\mathcal{D}}^{\star}(\bar{\boldsymbol{x}}) \coloneqq f_{\text{FFNN}}\left(\bar{\boldsymbol{x}}; \boldsymbol{\theta}^{\star}\right) .$$

Equation (3.8) reveals the natural connection between learning and optimization, thereby highlighting once again the importance of choosing a suitable loss function. For example, a differentiable loss function allows the use of gradient-based optimization algorithms, which can be expected to have better performance with regards to computational effort and accuracy [81]. In the following section we will shortly outline different methods to solve Equation (3.8).

3.1.3. Numerical Optimization of Loss Functions

Assuming a continuously differentiable loss function and activation function, gradient-based optimization algorithms can be used to solve the learning task Equation (3.8). These assumptions are satisfied for the previously described FFNNs. However, computing the gradient of an objective function like Equation (3.7) with respect to the model's parameters naively can be computationally expensive. Yet, for FFNNs, the back-propagation algorithm [151] allows to efficiently compute these gradients successively by applying the chain rule of differential calculus to every layer starting from the output layer. First, we will introduce algorithms that only require first-order derivative information before mentioning one example that uses second-order derivative information.

[3] In the literature, the terms *loss* or *loss function* are often used synonymously to *average loss* and the meaning is implied by the context.

Gradient Descent Variants One of the simplest – and most popular – possibilities for gradient-based optimization algorithms are gradient-descent methods, such as *steepest descent* [150]. Steepest descent — also referred-to as *batch gradient descent* [50, Section 8.1.3] — utilizes that a function's gradient always points into the direction of maximum change (i.e., positive gradient) and consequently updates the parameters proportional to the negative gradient

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}^{\text{data}}\left(\boldsymbol{\theta}\right) \ . \tag{3.9}$$

Here, η is the *learning rate*, determining the amount of change of the model parameters: A large learning rate corresponds to large parameter changes, while a small learning rate only results in minor adaptions. While this algorithm allows to perform an update of the network's parameters in a single step, the associated computational costs scale linearly with the size of the whole data set $\mathcal{D}$. As this data set can be huge, this method quickly becomes prohibitively expensive. Here, we can leverage the fact that the empirical risk $\mathcal{L}_{\mathcal{D}}^{\text{data}}$ is in fact an estimate, i.e., an expectation, of the expected risk. We can approximately evaluate the gradient of this estimate in turn by averaging over a small number of randomly generated samples, also referred to as *mini-batch* $\mathcal{B}^{(j)} \subset \mathcal{D}$. This way, the overall computational effort is still lower than in the batch case. This method is known as *(mini-batch) stochastic gradient descent* [50, Section 8.1.3]:

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}^{(j)}}^{\text{data}}\left(\boldsymbol{\theta}\right) \ . \tag{3.10}$$

Here, N_{mb} mini-batches $\mathcal{B}^{(j)}$ are chosen such that $\mathcal{D} = \cup_{j=1}^{N_{\text{mb}}} \mathcal{B}^{(j)}$. In the most extreme case, we perform the update of the parameters with respect to only one sample at a time. The approximation of the gradient by randomly generated samples, however, results in noisy parameter updates. The amount of noise — or the variance — in the parameter updates can be controlled by the number of samples contained in each mini-batch. Practically, the number of points chosen to estimate the gradient is influenced by numerous factors, including the required accuracy and available hardware [50, Section 8.1.3]. In the literature, it is generally suggested to employ small batches (i.e., in the range of 50 to 500 samples [70, 150]. Larger batch sizes result in a worse model performance in terms of the model's ability to generalize to unseen data. Research suggests that with larger batch sizes the optimization is more likely to converge to sharp minima, where the increased sensitivity of the NN's loss landscape negatively impacts the generalization [70].

The possibility of not using the complete data set, but only parts of it, introduces a slightly different terminology in ML, where often the term *epochs* is used in addition to *iterations*. Iteration refers to performing one single update step as reported in Equations (3.9) and (3.10), while the term epoch means performing these updates until the whole data set has been processed once. Consequently, the number of epochs and iterations are identical for the classical batch steepest descent method Equation (3.9), but differ for its stochastic variant Equation (3.10), where one epoch involves multiple iterations.

Variants of the mini-batch stochastic descent presented above are state-of-the-art algorithms for training NNs. These variants aim to circumvent some of the drawbacks of the vanilla approaches, which, e.g., can converge to undesirable local minima or saddle-point-like plateaus.

ADAM The probably most popular among these gradient descent variants is the Adaptive Moment Estimation (ADAM) algorithm [72]. ADAM does not update all parameters

identically, i.e., with a single learning rate, but instead determines different learning rates for different parameters. The motivation is to perform rather large updates on parameters for which only little information is available. In contrast, parameters for which information is less scarce undergo smaller updates with a higher updating frequency. Additionally, Adam's learning rate varies over the course of the algorithm's iterations. The aim is to accelerate the descent into promising directions, while simultaneously reducing oscillations (also known as *zigzagging*) that steepest descent methods are prone to [123]. To overcome this, Adam computes exponentially decaying moving averages of the mean and the (uncentered) variance of the loss function's gradient. For more details, we refer to the original publication of the authors [72] or to the explanation in the review paper [150], which does not only cover Adam, but also earlier variants like Adagrad or RMSProp.

L-BFGS Aside from algorithms that only use first-order derivative information, there are methods, called *Newton methods*, utilizing second-order derivative information. These can be derived from applying Newton's method for solving non-linear equation systems to the first-order necessary condition for (local) optimality. The main difference to the algorithms introduced in the previous sections is that the descent direction now involves the second derivative of the loss function. The motivation for employing Newton methods is their superior convergence rate, i.e., it exhibits quadratic convergence in the vicinity of the local optimum [123, Section 3.3]. However, providing second-order derivatives generally scales at least quadratically in the number of parameters, rendering it infeasible for applications with even a moderate number of parameters. Even with modern approaches like Automatic Differentiation (AD), the computation of second-order derivatives is generally associated with high computational efforts. Therefore, *quasi-Newton methods* have been developed, which also only require first-order derivatives, but approximate the second-order derivatives through incremental updates based on gradient information from previous iterations [123, Chapter 6]. The most popular example from this class is the Broyden Fletcher Goldfarb Shanno (BFGS) method which iteratively constructs an approximation of the inverse Hessian of the loss function [123, Section 6.1] based on the computed gradients. When training PINNs in Chapter 4 we will employ a variant of this algorithm, namely the Limited Memory Broyden Fletcher Goldfarb Shanno (L-BFGS) method [92], which requires less memory. While the BFGS method still exhibits super-linear convergence [123, Section 6.4], only R-linear convergence behavior can be proven for L-BFGS [92]. For further details, we refer to the original publication [92] or the book [123].

3.1.4. Quantifying the Error of Feed-Forward Neural Networks

Now that we have established the mathematical framework for using FFNNs in the previous sections, this section will briefly discuss their regression errors. To better understand the approximation made by FFNNs, the regression error can be decomposed into different components as already discussed in the literature, e.g., in [16, 93, 121].

Our goal is to approximate a certain function $f^\star \in \mathcal{F}$ from some function space $\mathcal{F}$. Therefore, we want to employ a FFNN $f_{\text{FFNN}} \in \mathcal{H}$ from our chosen hypothesis space $\mathcal{H}$ of the form given in Equation (3.4). This introduces the so-called *approximation error*

The distance of this hypothesis space to the target function is referred to as *approximation error*

$$\mathcal{E}_{\text{app}} = \|f^\star - f^\star_{\mathcal{H}}\| \ ,$$

where $f^\star_{\mathcal{H}}$ denotes the best approximation of the target function $f^\star$ inside our hypothesis space $\mathcal{H}$. In order to find this best approximation exactly, we would need to measure the discrepancy between our approximation and the target function on the whole domain. However, this is computationally infeasible. Instead, we only approximate this discrepancy by evaluating the error on a finite-dimensional data set $\mathcal{D}$ by computing the average loss according to Equation (3.7). Consequently, the FFNN only learns to approximate $f^\star$ on $\mathcal{D}$ but not on the rest of the domain. This introduces the so-called *estimation error*:

$$\mathcal{E}_{\text{est}} = \left\|f^\star_{\mathcal{H}} - f^\star_{\mathcal{H},\mathcal{D}}\right\| \ .$$

Here, $f^\star_{\mathcal{H},\mathcal{D}}$ represents the exact solution obtained from the minimization of the average loss Equation (3.8). A final layer of error is introduced due to the fact that the optimization is not performed analytically but numerically, introducing the optimization error

$$\mathcal{E}_{\text{opt}} = \left\|f^\star_{\mathcal{H},\mathcal{D}} - f_{\text{FFNN}}\right\| \ .$$

The total error made by employing a FFNN to approximate a function $f^\star$ can now be shown to be bounded by the three components introduced above, i.e.,

$$\begin{aligned}\mathcal{E}_{\text{tot}} &= \|f^\star - f_{\text{FFNN}}\| \\ &\leq \|f^\star - f^\star_{\mathcal{H}}\| + \left\|f^\star_{\mathcal{H}} - f^\star_{\mathcal{H},\mathcal{D}}\right\| + \left\|f^\star_{\mathcal{H},\mathcal{D}} - f_{\text{FFNN}}\right\| \\ &= \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} \ .\end{aligned}$$

Consequently, by minimizing $\mathcal{E}_{\text{app}}$, $\mathcal{E}_{\text{est}}$, and $\mathcal{E}_{\text{opt}}$, we can minimize the total regression error with respect to $f^\star$. In the following, we briefly discuss the individual error components.

Approximation Error As stated in the introduction of Chapter 3, the universal approximation theorem states that an FFNN with two layers $N_{\text{l}} = 2$ (i.e., only a single hidden layer $N_{\text{hl}} = 1$) can approximate any continuous function up to an arbitrary accuracy [29, 60, 85]. This guarantees that there is a fixed and finite number N_{nhl} of neurons in the hidden layer such that the approximation error becomes arbitrarily small, i.e., $\mathcal{E}_{\text{app}} < \epsilon$ for any $\epsilon > 0$. While this proves that FFNNs form a suitable hypothesis space for approximating continuous functions, there is no guarantee about the convergence rate or a ratio between the required number of neurons N_{nhl} and the desired accuracy ϵ. As a consequence, it could be that $N_{\text{nhl}} \to \infty$ as $\epsilon \to 0$.

One actuating variable for controlling the approximation error is the size of the hypothesis space: Choosing a larger hypothesis space generally provides a possibility to reduce the approximation error. However, this approach also makes the training process more expensive due to the increased number of parameters.

For some classes of functions and special NN architectures, researchers have quantified the approximation rates in terms of the width and depth of the network. In [157], the authors derive bounds when approximating (Hölder) continuous functions with deep networks employing ReLU activation functions while varying both the network's depth as well as its width. Siegel [164] considers instead ReLU-activated networks of fixed width and varying depth and derives bounds for the approximation of functions from Sobolev spaces which are, e.g., relevant for solving Partial Differential Equations (PDEs).

Estimation Error One way to control the estimation error is to increase the size of the data set. However, this is generally difficult in engineering problems, where data is typically scarce as it is retrieved, e.g., from computationally intensive numerical high-fidelity simulations. Moreover, the estimation error to some extent counterbalances the approximation error, as choosing a larger hypothesis space (thus reducing the approximation error) may result in an increased estimation error [16].

Optimization Error The optimization error can be controlled by specifying a predefined tolerance. However, the authors of [16] argue that it is beneficial to terminate the optimization before fulfilling this convergence criterion. The reason is that one optimizes the empirical risk, which is only a discrete representation of the mismatch between the chosen approximation and the true target function. Since we already optimize an approximated quantity, the error introduced by the numerical optimization is negligible. In practice, we perform this optimization only for a predefined number of epochs. Moreover, early termination of the optimization is also beneficial as it helps to prevent an increase of the generalization error due to overfitting [15, Section 5.5.2], which will be discussed in more detail in Section 3.1.6. For selected gradient-based optimization algorithms, asymptotic results for the optimization error are reported in [16].

Generalization Error Another often-mentioned error in the context of NNs is the so-called *generalization error*[4]. It is used as a performance metric for estimating how well a trained NN generalizes to data not seen during training. It is defined as

$$\begin{aligned}\mathcal{E}_{\text{gen}} &= \left\| f^\star - f^\star_{\mathcal{H},\mathcal{D}} \right\| \\ &\leq \left\| f^\star - f^\star_{\mathcal{H}} \right\| + \left\| f^\star_{\mathcal{H}} - f^\star_{\mathcal{H},\mathcal{D}} \right\| \\ &= \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} \ .\end{aligned}$$

A small generalization error is desirable, as it indicates that the trained model has actually extracted features from the data and does not overfit them. One way to decrease the generalization error is to enhance the learning process with additional a-priori knowledge, i.e., in the form of known physical relations, motivating the use of Physics-Informed Neural Networks (PINNs) which will be introduced in Section 3.2. Quantifying the generalization error is complicated in practice, yet it can be approximated, e.g., by evaluating the trained model empirically on a separate test data set [50, Section 5.2]. While it allows to compare and rank the performance of trained models, it only provides an a-posteriori estimate, i.e., it cannot be used to determine a suitable model architecture.

3.1.5. Deeper vs. Wider: A Short Discourse on Choosing a Proper Feed-Forward Neural Network Architecture

According to the discussion in Section 3.1.4, by fixing the architecture of a NN, one also determines the hypothesis space and thus directly limits the approximation capabilities of the resulting optimized model. While this highlights the importance of a NN's architecture, the question remains, how to choose an appropriate architecture for solving the considered

[4] In the literature, there exist multiple definitions for the generalization error [55, 93, 110, 159]. Throughout this thesis, we understand the generalization error as defined by Niyogi and Girosi [121]

problem? This question has been extensively discussed in the literature, see e.g., the discussion in [90, 120, 174].

The number of hidden layers N_{hl} as well as the width of each hidden layer N_{nhl} have the largest influence on the architecture (see Section 3.1.1). The network's approximation capabilities can be improved by increasing the values of both parameters (as this enlarges the size of the hypothesis space). Increasing both parameters simultaneously, however, increases the computational costs significantly due to the rising amount of learnable parameters, which — for complex applications — makes the learning process infeasible in terms of computational effort and training time. This leaves two possibilities: Either increase the width of a network, while keeping the depth fixed or vice versa.

Nowadays, it is generally favored to make the networks deeper instead of wider. This has multiple reasons. First of all, deep NNs are able to approximate certain classes of functions more efficiently [33], as they require less neurons per hidden layer, resulting in an overall fewer number of parameters [50, Section 6.4]. Here, the term *efficiency* refers to the number of computational units inside the network: A fewer number of hidden units means that less memory is required to store the resulting model, while the model can also be evaluated faster, i.e., with less floating point operations. Additionally, Delalleau and Bengio [33] argue that deep architectures generalize better when both the size of the available training data set and the computational resources are constrained. This argument is also supported by Bengio and Lecun [8], Hastie, Tibshirani, and Friedman [55], and Lin, Tegmark, and Rolnick [90] who attribute the better generalization to the fact that the additional layers allow deep NNs to learn in a hierarchical way, where the multiple layers correspond to different levels of abstraction (similarly to the human brain), i.e., earlier layers extract lower-level information which is subsequently combined into higher-level features. Another argument in favor of deep NNs is given in [55, Section 11.5.4]: Here, the authors suggest to use rather more than less hidden layers and incorporate additional regularization terms into the loss functions to be able to better capture possible nonlinearities in the data. Equipped with the additional regularization term, weights, i.e., connections, that are not needed to represent the data, can be reduced to zero, essentially obliterating the additional layers.

Despite the vast arguments for promoting deep architectures, in practice, there also exist abundant examples where shallow architectures outperform deep ones [8, 120]. One advantage of shallow architectures lies in the ability to still use convex loss functions [8], while training deep NNs is inherently difficult due to the non-convexity of the loss landscape. Another problem often encountered in deep networks is the so-called *vanishing or exploding gradient* problem [50, Section 8.2.5] which additionally hinders the optimization of the loss function through too small or too large parameter updates. With respect to the application for solving problems with underlying physics, Lin, Tegmark, and Rolnick [90] further attribute the success of shallow architectures to a special structure in the data. This structure often exhibits locality or symmetry properties, which can efficiently be exploited using low-order approximations. Heuristics for choosing the number of neurons inside the hidden layers of a shallow FFNNs have been proposed in the literature, where many establish a connection between the number of neurons per hidden layer and the dimensions of the input and output spaces [174]. Huang [62] follows a different approach and derives a heuristic for networks with only two hidden layers. This heuristic depends on the size of the available data set.

Concluding this discussion, there exists no best architecture for all purposes. Instead, a proper architecture is problem-dependent and differs depending on the available data and training resources, i.e., computing power and time. In practice, finding a suitable architecture is an iterative task, which is guided by experience and experimentation. One indicator for the performance of an individual architecture is the generalization error, which can be estimated for each architecture and then compared to identify the most promising one [50, Section 6.4]. While this has been a manual and time-consuming process for a long time, by now, there is a variety of computational frameworks specifically dedicated to automatizing the search for suitable hyper-parameters, a field which is known as Hyper-parameter Optimization (HPO) [50, Section 11.4]. These frameworks usually compare different architectures with respect to an estimate for the network's performance (e.g., an estimate of the generalization error) and report the best model in the end. This comparison can either be performed by evaluating a fixed number of (randomly) selected model architectures or by introducing an additional level of optimization over the chosen performance metric. We will briefly discuss one of these frameworks in Section 4.1.2.

3.1.6. Some Remarks on Common Issues When Training Neural Networks

Though NNs have become a well-established and powerful choice for regression tasks, their training process involves some challenges that need to be tackled. The list presented in the following is non-exhaustive. For more details, we refer, e.g., to [50, Section 8.2], [55, Section 11.5], and the references therein.

Firstly, as discussed in Section 3.1.2, training a NN reduces to an optimization of the empirical risk. As the empirical risk is generally a non-linear and non-convex function, the same problems arising in classical optimization also apply here, i.e., the conditions on the existence of a global optimum. Furthermore, the algorithms that are typically employed for training NNs (cf. Section 3.1.3) are local optimizers, i.e., there is no guarantee that the minimum they find is globally optimal. Although classical optimization theory provides algorithms which can find global minima, these are usually too computationally expensive to be employed in the context of NNs. Due to the non-convexity of the empirical risk, the loss landscape of NNs often features multiple local optima and — especially in higher dimensions — increasingly often saddle points [30], hindering the progress of state-of-the-art optimizers in finding a sufficiently accurate approximation of Equation (3.8).

A second problem, which also arises in classical optimization, concerns the initialization of the NN's parameters. As pointed out in [55, Section 11.5.1], the parameters cannot be simply initialized to zero, as this would result in zero gradients, i.e., a gradient-based algorithm would not make any progress. Instead, it is suggested to initialize the weights randomly with zero mean [117, 55]. In the literature, different methods have been proposed depending on the type of activation function used. An early guideline for automatically initializing the weights, given a normalized training data set and sigmoid activation functions, suggested to initialize the weights such that the output layer has zero mean and a standard deviation of one [117]. In the meantime, the *Glorot initialization* [49] has become state-of-the-art for hyperbolic tangent activation functions, while the *He initialization* [56] is used for ReLU activations.

The final common problem that we want to address here is *overfitting*: Overfitting occurs if the NN is trained for too long and starts to learn the training data set in a tabular manner. Consequently, the error on the training data set continuously decreases, while the generalization error slowly starts to increase simultaneously [15, Section 5.5.2]. One possibility to detect (and bypass) this behavior is to not only monitor the progress of the training error (i.e., the value of the empirical risk) over the number of optimization iterations, but additionally test the model repeatedly on a separate *validation data set*. Tracking the validation error provides a simple way to estimate the generalization error and the training can be terminated as soon as the validation error starts to increase [55, Section 11.5.2]. Moreover, one can employ *regularization* methods to prevent a NN from overfitting [50, Section 5.2.2]. There exists a variety of different regularization techniques [50, Chapter 7], among which parameter norm regularization methods belong to the simplest. Here, an additional term ϱ is added to the empirical risk which poses a restriction on the size of the learned NN parameters, i.e., the weights and biases. The empirical risk then becomes

$$\tilde{\mathcal{L}}_{\mathcal{D}}^{\text{data}}(\boldsymbol{\theta}) = \mathcal{L}_{\mathcal{D}}^{\text{data}}(\boldsymbol{\theta}) + \alpha_{\varrho}\varrho(\boldsymbol{\theta}) \ .$$

Popular choices for the regularization term include ℓ_1-regularization, i.e., $\varrho^{\ell_1}(\boldsymbol{\theta}) \coloneqq \|\boldsymbol{\theta}\|_1$, or ℓ_2-regularization, that is, $\varrho^{\ell_2}(\boldsymbol{\theta}) \coloneqq \frac{1}{2}\|\boldsymbol{\theta}\|_2^2$. The parameter $\alpha_{\varrho} \in [0, \infty)$ serves as an additional hyper-parameter for controlling the strength of the regularization, i.e., $\alpha_{\varrho} \to \infty$ will add more regularization and promote smaller NN parameters. As a final note, we want to mention that regularization is mostly applied to the NN's weights which are considered to be more impactful with respect to the network's generalization [50, Section 7.1].

3.2. An Introduction to Physics-Informed Neural Networks

When thinking about SciML, one method seems like the natural combination between ML and scientific computing and has in the last years gained massive popularity all around the globe: Physics-Informed Neural Networks (PINNs). Although originally proposed in the last century in works by Psichogios and Ungar [132] and Lagaris, Likas, and Fotiadis [77], this method only achieved its break-through after the reinvention by Raissi, Perdikaris, and Karniadakis [138] in 2017[5]. The idea of PINNs is to *learn* the solution to a physical problem, i.e., one that is governed by (partial) differential equations. However, instead of requiring (only) input and output data pairs as in a classical supervised ML approach, the learning process is guided by the underlying governing equations. We will explain this idea as well as some variants thereof in more detail in the next section. Afterward, we will comment on the convergence and generalization capabilities of PINNs before investigating the concept from a practical perspective, shedding light on the difficulties of this method as well as discussing possible solutions. Finally, we will mention different competing approaches for solving Partial Differential Equations (PDEs) with NNs.

[5] Although their paper was only published in the Journal of Computational Physics in 2019, the preprints were already available on Arxiv in 2017 [139, 140].

3.2.1. Mathematical Basics of Physics-Informed Neural Networks

To introduce the framework for training, we first recall that we are now considering (scientific) problems, which are described by physical laws in the form of (partial) differential equations, e.g., the ones presented in Section 2.1. To make the introduction easier, we rewrite them in a generic (residual) form as follows

$$\mathcal{R}_{\text{PDE}}\left[\boldsymbol{u}\right] = \mathbf{0} \quad \text{on } \Omega \times (0, T] \tag{3.11a}$$
$$\mathcal{R}_{\text{IBC}}\left[\boldsymbol{u}\right] = \mathbf{0} \quad \text{on } \Gamma \times \{0, T\}\ , \tag{3.11b}$$

where $\mathcal{R}_{\text{PDE}}\left[\,\cdot\,\right]$ denotes the residual form of the PDEs and $\mathcal{R}_{\text{IBC}}\left[\,\cdot\,\right]$ Initial and Boundary Conditions (IBCs), formulated as a residual too. $\boldsymbol{u}$ symbolizes the PDEs' (unknown) solutions that we want to find. The idea is to approximate this spatially and temporally varying and (potentially) parameterized unknown solution vector with a FFNN as introduced in Section 3.1.1, that is

$$\boldsymbol{u}(\boldsymbol{x}, t; \boldsymbol{\lambda}) \approx f_{\text{FFNN}}\left(\bar{\boldsymbol{x}}; \boldsymbol{\theta}\right)\ ,$$

where we have grouped all independent variables of the unknown solution into the input vector of the FFNN $\bar{\boldsymbol{x}} = (\boldsymbol{x}, t, \boldsymbol{\lambda})$. Here, $\boldsymbol{\lambda} \in \Lambda$ corresponds to the vector of (physical) parameters, which parameterize the PDEs. With that, we obtain an approximation that — once learned — can be evaluated fast and cheaply for different inputs, thus being a promising candidate for a reduced simulation model. In order to learn such a NN approximation, we need to first define a proper loss function. Here, the underlying governing equations come into play: Unlike in Section 3.1.2, the employed loss function only depends on a residual, e.g., the deviation of the prediction from the target value, that is

$$L : \mathcal{Y} \to (\mathbb{R}^+)^{N_{\mathcal{Y}}}, \bar{\boldsymbol{r}} \mapsto L\left(\bar{\boldsymbol{r}}\right) = \bar{\boldsymbol{r}} \odot \bar{\boldsymbol{r}}\ , \tag{3.12}$$

where $\odot$ denotes the component-wise multiplication. Note, that in contrast to Equation (3.5) this function computes the component-wise squared error, i.e., it returns a vector. We will elaborate why we make this distinction at the end of this section. For training PINNs, instead of (only) leveraging data, we additionally employ the underlying governing equations as stated in Equation (3.11). In fact, we evaluate the strong form of the PDE residual Equation (3.11) using our NN approximation f_{FFNN}, compute the component-wise loss according to Equation (3.12) and average over all possible data points. This leaves us with the following definition of the expected risk

$$\begin{aligned}\mathcal{L}^{\text{PDE}} &= \mathbb{E}_{\bar{\boldsymbol{x}} \sim \mathbb{P}_{\bar{\boldsymbol{X}}}^{\text{PDE}}}\left[L\left(\mathcal{R}_{\text{PDE}}\left[f_{\text{FFNN}}\left(\bar{\boldsymbol{x}}; \boldsymbol{\theta}\right)\right]\right)\right] \\ &\coloneqq \int_{\mathcal{X}} L\left(\mathcal{R}_{\text{PDE}}\left[f_{\text{FFNN}}\left(\bar{\boldsymbol{x}}; \boldsymbol{\theta}\right)\right]\right) p^{\text{PDE}}\left(\bar{\boldsymbol{x}}\right)\ \mathrm{d}\bar{\boldsymbol{x}}\ .\end{aligned} \tag{3.13}$$

One apparent difference to Equation (3.6) is that we now only require points from the input space $\mathcal{X}$ to evaluate all components of the expected risk. Consequently, we only evaluate the expectation with respect to a single random (input) variable $\bar{\boldsymbol{X}}$ instead of considering a joint probability of inputs and outputs. Additionally, Equation (3.13) needs

to be understood component-wise, as in case of vector-valued outputs, corresponding to PDE systems, the loss function returns a vector instead of a scalar. Analogously, we have

$$\begin{aligned}\mathcal{L}^{\mathrm{IBC}} &= \mathbb{E}_{\bar{\boldsymbol{x}}\sim\mathbb{P}^{\mathrm{IBC}}_{\bar{\boldsymbol{X}}}}\left[L\left(\mathcal{R}_{\mathrm{IBC}}\left[f_{\mathrm{FFNN}}\left(\bar{\boldsymbol{x}};\boldsymbol{\theta}\right)\right]\right)\right] \\ &:= \int_{\mathcal{X}} L\left(\mathcal{R}_{\mathrm{IBC}}\left[f_{\mathrm{FFNN}}\left(\bar{\boldsymbol{x}};\boldsymbol{\theta}\right)\right]\right) p^{\mathrm{IBC}}\left(\bar{\boldsymbol{x}}\right)\, \mathrm{d}\bar{\boldsymbol{x}}\ .\end{aligned}$$

Similar to the previously discussed purely data-driven case, we usually cannot evaluate the expected risks and therefore once again introduce estimates, for the actual minimization, which read

$$\mathcal{L}^{\mathrm{PDE}}_{\mathcal{D}_{\mathrm{PDE}}}(\boldsymbol{\theta}) = \frac{1}{N_{\mathrm{PDE}}}\sum_{i=1}^{N_{\mathrm{PDE}}} L\left(\mathcal{R}_{\mathrm{PDE}}\left[f_{\mathrm{FFNN}}\left(\bar{\boldsymbol{x}}^{(i)}_{\mathrm{PDE}};\boldsymbol{\theta}\right)\right]\right) \tag{3.14}$$

$$\mathcal{L}^{\mathrm{IBC}}_{\mathcal{D}_{\mathrm{IBC}}}(\boldsymbol{\theta}) = \frac{1}{N_{\mathrm{IBC}}}\sum_{i=1}^{N_{\mathrm{IBC}}} L\left(\mathcal{R}_{\mathrm{IBC}}\left[f_{\mathrm{FFNN}}\left(\bar{\boldsymbol{x}}^{(i)}_{\mathrm{IBC}};\boldsymbol{\theta}\right)\right]\right)\ . \tag{3.15}$$

Note that we employ two different data sets, one including points inside the domain and on the boundary $\mathcal{D}_{\mathrm{PDE}} \subset (\Omega \times (0,T]) \cup (\Gamma \times \{0,T\})$ and one which only contains only points on the boundary $\mathcal{D}_{\mathrm{IBC}} \subset (\Gamma \times \{0,T\})$. N_{PDE} and N_{IBC} respectively represent the number of points in the individual datasets, i.e., $N_{\mathrm{PDE}} = |\mathcal{D}_{\mathrm{PDE}}|$ and $N_{\mathrm{IBC}} = |\mathcal{D}_{\mathrm{IBC}}|$.

With these components, we can formulate the most general case of an empirical risk for a PINN that will then be minimized during training as described in Section 3.1.2

$$\mathcal{L}^{\mathrm{PINN}}_{\mathcal{D}}(\boldsymbol{\theta}) = \boldsymbol{\alpha}_{\mathrm{PDE}} \cdot \mathcal{L}^{\mathrm{PDE}}_{\mathcal{D}_{\mathrm{PDE}}}(\boldsymbol{\theta}) + \boldsymbol{\alpha}_{\mathrm{IBC}} \cdot \mathcal{L}^{\mathrm{IBC}}_{\mathcal{D}_{\mathrm{IBC}}}(\boldsymbol{\theta}) + \alpha_{\mathrm{data}}\mathcal{L}^{\mathrm{data}}_{\mathcal{D}_{\mathrm{data}}}(\boldsymbol{\theta})\ . \tag{3.16}$$

Here, we have introduced weights for each loss function component, denoted by $\alpha_{(\,\cdot\,)}$, which allow to place different emphasis on the individual components. Let us conclude this section with some remarks about Equation (3.16):

1. In our definition, there is a inherent difference between the loss components $\mathcal{L}^{\mathrm{PDE}}_{\mathcal{D}}, \mathcal{L}^{\mathrm{IBC}}_{\mathcal{D}}$ and $\mathcal{L}^{\mathrm{data}}_{\mathcal{D}}$: While the latter is considered a scalar quantity (according to its definition in Equation (3.7)), the former two are vector-valued quantities, following from the (generally) vector-valued character of Equation (3.11). As a direct consequence, vector-valued weights are introduced to allow a different weighting for the different equation components. This can be crucial if different equations have different orders of magnitude as it is the case, e.g., for the incompressible Navier-Stokes equations presented in Equation (2.2).

2. The empirical risk for PINNs defined above covers a wide range of applications, regardless of whether data in form of measurements from a real-world process or snapshots from a high-fidelity simulation is available. In that sense, PINNs can be counted to the field of supervised learning algorithms. By omitting the last term $\mathcal{L}^{\mathrm{data}}_{\mathcal{D}}$ of Equation (3.16), the training no longer depends on data, thus forming an unsupervised learning approach. The only data needed, are samples from the input space of the NNs, also called *collocation points* in the literature, at which the physics-informed loss components $\mathcal{L}^{\mathrm{PDE}}_{\mathcal{D}}, \mathcal{L}^{\mathrm{IBC}}_{\mathcal{D}}$ are evaluated. Thus, a PINN can be trained on any geometry, which is accessible through sampling and can be considered as a mesh-free approach for solving PDEs [69].

3. Though not completely obvious at first glance, the evaluation of $\mathcal{L}_{\mathcal{D}}^{\text{PDE}}$ (and in case of, e.g., Neumann-type Boundary Conditions (BCs) also of $\mathcal{L}_{\mathcal{D}}^{\text{IBC}}$) requires the computation of the NN's derivatives with respect to (parts of) its input vector $\bar{\boldsymbol{x}}$. These derivatives are neither computed by hand (which would be infeasible for large NNs) nor approximated by finite differences. Instead, Automatic Differentiation (AD) is employed, through which analytically exact derivatives can be obtained algorithmically [105]. Since this technique is nowadays implemented in most ML packages [7], it does not pose an additional restriction on the applicability of the method. For a more detailed explanation of AD, we refer to the literature [51, 119].

As outlined above, the core idea of PINNs is simple, yet effective, as proven by applications in several fields [24, 34, 98, 103, 143, 152, 184]. A part of the reason, why PINNs have become so popular, may also be attributed to the abundance of (open-source) software packages developed for that purpose, including, e.g., `deepxde` [93], `NVIDIA SimNet`, resp. `Modulus`[6] [57], or `SciANN` [54], just to name a few. More detailed, yet still non-exhaustive, lists can be found in [69] as well as in this post[7] recently compiled by Holger Marschall. However, so far, we have only introduced the most simple version of PINNs. By now, there exists a variety of extensions which further develop this idea [22, 69]. We will introduce some of these in the next section.

3.2.2. Beyond Vanilla Physics-Informed Neural Networks: Advanced Architectures

Within this section only the most important variants of PINNs are introduced. These include Variational Physics-Informed Neural Networks (VPINNs), Deep Operator Networks (DeepONets), as well as distributed architectures like Conservative Physics-Informed Neural Networks (cPINNs) [67] and Extended Physics-Informed Neural Networks (XPINNs) [64].

VPINNs [71] The idea of Variational Physics-Informed Neural Networks (VPINNs) is similar to classical FEM: Instead of considering the strong form of the PDE residual as in the classical PINN method, a variational weak form is considered. Precisely, a Petrov-Galerkin-type approach is chosen, where different function spaces are employed for the trial and test function spaces associated with the variational form. For VPINNs, the trial space corresponds to the space of NNs, while Legendre polynomials are used to construct the test space. The NN approximation is then trained to solve this variational form. Due to the variational form, similar to the derivation of a weak formulation in FEM, higher-order derivatives can be reduced by partial integration, consequently lowering regularity requirements. Furthermore, since the derivatives are computed using AD, this helps to reduce the computational costs during training. However, moving to variational forms of PDEs instead of the strong forms, introduces the necessity to numerically compute integrals involving NNs.

[6] Called `SimNet` in the original publication, the development of this framework continues under the new name `Modulus`.
[7] Last accessed: 25th March 2023

DeepONets [94] Another architecture building up upon the idea of PINNs that has been gaining increasing popularity are Deep Operator Networks (DeepONets). Instead of learning only the (parameterized) solution to a (parameterized) PDE, these models are trained to represent general (non-)linear operators. The idea for the network architecture stems from the *universal operator approximation theorem*, which is less well-known than the universal function approximation theorem, but essentially states that any continuous (nonlinear) operator, mapping a continuous function over some Banach-space onto another continuous function, can be approximated by a NN of a certain architecture up to any desired accuracy. Practical applications have shown that DeepONets exhibit a low generalization error, while being more expensive to train than vanilla PINNs. While DeepONets do not need to be employed for solving PDEs, remarkable success was achieved for learning differential operators [21, 89, 94, 104, 162] which makes them an interesting candidate for constructing reduced simulation models.

cPINNs [67] and XPINNs [64] These architectures extend the PINN concept to problems involving multiple subdomains. The idea of Conservative Physics-Informed Neural Networks (cPINNs) is to decompose the computational domain into multiple subdomains and couple these by enforcing the continuity of fluxes across the domain boundaries. This is done by incorporating additional terms into the classical PINN loss function introduced in Equation (3.16). A separate network is then trained in each subdomain. Along the shared interface of multiple domains, the continuity of the solution is enforced additionally, while in the case of hyperbolic conservation laws which may involve discontinuous solutions the predictions of each subdomain's network are averaged. While — as the name implies — the cPINN approach mainly targets PDEs in conservative form, the Extended Physics-Informed Neural Network (XPINN) approach generalizes this to arbitrary PDEs. Instead of requiring the continuity of fluxes across the interfaces, they ensure the continuity of the solution as well as the continuity of the PDE residuals. However, depending on the problem at hand, additional interface conditions can be imposed. As pointed out by Shukla, Jagtap, and Karniadakis [161], both architectures allow for parallelization, thus show potential for making the training process more efficient. Moreover, as we will see later in Section 4.3, the possibility of having different NNs in each subdomain also allows to solve different equations, i.e., if simplifications of more complex PDEs apply in certain subdomains, which can help to reduce the model cost and improve prediction capabilities. Of course, one drawback of cPINNs and XPINNs lies in the choice of the subdomains. Hu et al. [61] investigate under which conditions XPINNs outperform classical vanilla PINNs with respect to the generalization error and come to the conclusion that one has to balance the simplicity arising from the decomposition of the (complex) unknown solution and the networks' tendency to overfit due to less training data within the individual subdomains.

3.2.3. On the Error of Physics-Informed Neural Networks: Convergence and Generalization

After the discussion of advanced PINN-like network architectures, this section will now briefly comment on the convergence and generalization of vanilla PINNs. Generally, the same considerations made in Section 3.1.4 about the decomposition of the error for the

application of NNs hold true for PINNs as well. Yet, there is an important difference to classical NN approaches: Incorporating additional knowledge in the form of the governing equations into the learning process can be regarded as a special type of physics-driven regularization [118]. As a consequence, physics-informed models can be expected to exhibit a smaller generalization error, while simultaneously increasing the interpretability of the results [118, 69].

Some work has been dedicated in the literature to prove that PINNs actually converge to the solution of the governing PDE as well as deriving estimates of the generalization error $\mathcal{E}_{\text{gen}}$ for PINNs. Already in 2020, Shin, Zhang, and Karniadakis [160] presented a framework for analyzing the loss functions of PINNs and derived a-priori and a-posteriori error estimates for the special case of linear PDEs. In their work, they consider both continuous and discrete loss functions, as well as variational formulations as they are, e.g., employed in training VPINNs. The derived bounds however are limited by the fact that some estimations are only available for very special types of NNs, e.g., NNs with only two layers. Shin, Darbon, and Karniadakis [159] were the first to provide an abstract framework for the convergence theory of PINNs with respect to the generalization error. They show that under certain assumptions, the expected loss is bounded by the empirical loss. Based on that, they derived convergence proofs for special PDE cases, i.e., linear elliptic and parabolic PDEs, minimizing a regularized version of the empirical loss. In the same year, Mishra and Molinaro [110] derived two estimates of the total error[8] of PINNs for arbitrary non-linear PDEs. These estimates only differ in how the collocation points are chosen, i.e., according to a (Gaussian) quadrature rule or random sampling. With assumptions on the uniqueness of the solution and the stability of the differential operator, the authors show that in both cases the total error of a PINN is bounded by terms including the training error as well as the number of sample points. This relation of the total error and the training error is important, as it proves that minimizing the loss function during training actually reduces the generalization error, which is the main goal. Moreover, it shows that the generalization error can be kept small if the training error is small and the number of sample points large.

3.2.4. Are Physics-Informed Neural Networks the Solution for Scientific Machine Learning? — A Practical Discussion

As mentioned before, after their rediscovery, PINNs quickly became one of the most popular methods in SciML. This section aims to provide a pragmatic view on PINNs and report some difficulties which need to be tackled to employ the method in practice. Afterward, we will introduce some possible solutions to these challenges. Some of the aspects mentioned in this section will be brought up when discussing the results obtained with PINNs in this thesis in Chapter 4.

[8] Note that the authors employ a different definition of the generalization error, which corresponds to our definition of the total error.

The Dark Side of the Moon: Shortcomings of Physics-Informed Neural Networks

We will start with some remarks related to the mathematical theory of PINNs as introduced above. When inspecting the usually-employed loss function Equation (3.16), the following can be observed:

1. In contrast to the classical supervised case (Equation (3.7)), the loss function is made up of multiple contributions, namely components related to the underlying PDE, the IBCs, and (optionally) data. It is reasonable to assume that these loss components have different magnitudes in practice. Consequently, they need to be balanced in a way that the resulting loss function does not concentrate on minimizing a single component but respects all components equally. Therefore, the selection of the weights $\alpha_{(\cdot)}$ is important to obtain a NN that reasonably approximates the desired PDE solution.

2. From classical PDE theory it is known that the choice of IBCs is crucial for the well-posedness of a PDE problem, i.e., equipping a PDE with insufficient initial or boundary constraints can result in infinitely many solutions [40, 153]. However, in a classical PINN, IBCs are only imposed weakly by adding Equation (3.15) to the loss function. That way, these constraints will most likely not be fulfilled exactly, even if the corresponding loss weights $\boldsymbol{\alpha}_{\text{IBC}}$ are increased (resulting in a stronger penalization in case of a violation of these conditions).

3. Another remaining question concerns the selection of the collocation points: How does one properly select the domain and boundary samples for evaluating the loss function? Drawing an analogy to classical solution approaches for PDEs, one typically refines the computational mesh of the problem domain in the vicinity of intricate geometric features or in regions, where the solution is expected to exhibit complex phenomena (e.g., vortices in Computational Fluid Dynamics (CFD)). These requirements however cannot be met by randomly sampling points from the domain and the boundary.

Beside these implications, other drawbacks have been discovered and discussed in the literature. We begin with practical experience reports: Chuang and Barba [26] present two two-dimensional test cases, namely the Taylor-Green vortex flow as well as a flow around a cylinder, for which they compare the predictions of the trained PINN to that of a classical CFD solver, both with respect to accuracy and training time. Their findings show severe deficiencies of PINNs. Training a PINN to provide "acceptable" results on the Taylor-Green vortex case required 32 h, while the classical CFD simulation was finished in roughly 20 s. In the case of the flow around the cylinder, the PINN was not even able to produce a meaningful approximation, as the solution did not exhibit the expected vortex-shedding behavior. A similar result has also been reported by Rao, Sun, and Liu [142].

On the theoretical side, work has been dedicated to explain, why classical PINNs fail to accurately approximate the unknown PDEs solution in some cases. Wang, Teng, and Perdikaris [175] consider two examples, namely the Helmholtz equation and the Laplace equation, to illustrate a pathology with respect to the gradient of the loss function: The magnitudes of the gradients of the loss components $\mathcal{L}_{\mathcal{D}}^{\text{PDE}}$ and $\mathcal{L}_{\mathcal{D}}^{\text{IBC}}$ with respect to the NN's weights differ significantly. Thus, the learning process of the NN is biased towards

fulfilling the loss component with the larger gradient, e.g., the PDE residual, while paying less attention the other components, e.g., the IBCs. This behavior is caused by many large eigenvalues of the loss function's Hessian, rendering the employed gradient-based optimization scheme unable to further decrease the loss, although being given the correct descent direction. In a follow-up work of the same group, the learning dynamics were further investigated, revealing a connection between the convergence rate of the training error and the weights of the loss components $\alpha_{(\,.\,)}$ as well as the sizes of the mini-batches, i.e., the number of points N_{PDE} and N_{IBC} used to approximate the loss function's gradient [176]. Though yet only available for NNs with a single hidden layer, this result highlights the importance of properly choosing the hyper-parameters of the training set-up.

Finally, Mojgani, Balajewicz, and Hassanzadeh [115] examine a couple of convection-dominated problems on which PINNs perform poorly, i.e., fail to find an approximate solution. The authors explain this by establishing an analogy to the so-called *Kolmogorov n-width*, a measure for the approximability of a function class by linear subspaces [108]. In linear approximation theory it can be shown that the accuracy of such an approximation is related to the deterioration of a snapshot matrix's singular values. Since there is no similar result for the non-linear case, the authors examine numerically if the same argument can be applied there. They show that all the investigated problems exhibit a slow decrease of singular values, corresponding to a large Kolmogorov n-width, and consequently conclude that PINNs generally only perform well on problems with a small Kolmogorov n-width.

Bringing Light into the Darkness: Strategies to Overcome Practical Challenges

After enumerating a couple of challenges in the previous section, we present possible solution approaches in the following. We start with the mentioned weak imposition of the IBCs. In contrast to adding $\mathcal{L}_{\mathcal{D}}^{\text{IBC}}$ to the loss function, it is also possible to strongly enforce these conditions by modifying the PINN prediction in a post-processing step. Let us consider the case of Dirichlet-type BCs, i.e., $\bar{\boldsymbol{y}} = g(\bar{\boldsymbol{x}}) \quad \text{on } \Gamma$. By choosing

$$\bar{\boldsymbol{y}} = \tilde{f}_{\text{FFNN}}\left(\bar{\boldsymbol{x}}; \boldsymbol{\theta}\right) \coloneqq g(\bar{\boldsymbol{x}}) + l(\boldsymbol{x}) f_{\text{FFNN}}\left(\bar{\boldsymbol{x}}; \boldsymbol{\theta}\right) \ ,$$

where

$$\begin{cases} l(\boldsymbol{x}) = 0 & \text{on } \Gamma \\ l(\boldsymbol{x}) > 0 & \text{in } \Omega \setminus \Gamma \end{cases}$$

is a function that vanishes on the boundary of the computational domain, we obtain a NN approximation that satisfies the BCs analytically [95]. The function l which is referred to as *length-factor function* [158] in the literature is generally difficult to find for arbitrary geometries. Lagari et al. [76] present an approach for determining such a function analytically in rectangular domains, while Sheng and Yang [158] present a numerical approach based on interpolation of samples from the domain boundary, which allows the construction of such a function for arbitrary geometries at the fixed costs of solving a linear equation system once as a pre-processing step before starting the NN training.

Another difficulty of the PINN method identified in the previous section was the choice of the loss weights. Here, a lot of work has been dedicated to find solutions: Based on

their insights into the gradient pathologies of PINNs, Wang, Teng, and Perdikaris [175] propose an empirical scheme for automatically adapting the loss weights based on the magnitudes of the loss components' gradients. A different adaptive scheme is proposed in their succeeding work [176] to tackle the degenerated convergence behavior. For the special case of linear and well-posed PDEs, Meer, Oosterlee, and Borovykh [107] derive a method to optimally choose the loss weights, however based on the analytical solution of the problem under consideration. Since this is typically not available, they also provide a way to compute them heuristically. Lastly, Li and Feng [87] propose a bi-level optimization algorithm inspired by a min-max formulation of the loss function, where the loss weights are updated by gradient-descent steps depending on the evolution of the individual loss components. They apply the method to a forward and an inverse problem involving the two-dimensional Navier-Stokes equations and show that the obtained results exhibit a smaller relative error compared to the classical PINN with constant loss weights.

The fact that PINNs seem unable to capture the vortex-shedding behind a cylinder mentioned earlier has been solved by using a mixed-variable formulation, i.e., learning not to predict the actual unknown solutions of the incompressible Navier-Stokes equations velocity and pressure, but instead predicting the stream function, the Cauchy stress tensor as well as the pressure [142]. This choice of variables has the advantage that the continuity equation is automatically satisfied by the definition of the stream function [80, §10]. This observation is interesting as it shows that the formulation of the governing PDEs can have a significant influence on the prediction quality of the PINN approximation.

Additionally, the problem of properly choosing sampling points within the domain has been tackled by introducing so-called adaptive sampling techniques. Meer, Oosterlee, and Borovykh [107] propose the use of two different, disjunct sets of points for training and validation: Whenever the loss value obtained on the validation set exceeds the loss value on the training set by a certain factor, new point sets are generated, doubling the total number of points in each set. While this strategy allows to adapt the total number of collocation points, it does not take geometric features or the behavior of the solution into account. A first method to tackle this challenge has already been introduced with the concept of Residual-based Adaptive Refinement (RAR) in [93] and we will come back to that later in Chapter 4. An extensive comparison of different random, resampling, and adaptive sampling techniques has recently been conducted by Wu et al. [181]. Their findings suggest that adaptive refinement methods utilizing a probability density function based on the residual can help to decrease the error significantly in comparison to non-adaptive methods. As a compromise, the required collocation points can be resampled randomly in cases where it is difficult to sample points from a constructed probability distribution. This however introduces the resampling rate as an additional hyper-parameter, which needs to be chosen appropriately.

Lastly, we come back to the problem that PINNs only seem to work well on problems which exhibit a small Kolmogorov n-width. To overcome this issue, Mojgani, Balajewicz, and Hassanzadeh [115] proposed a two-fold solution: For once, the governing PDEs are formulated with respect to a Lagrangian frame-of-reference instead of the usual Eulerian one, and secondly, a special PINN architecture is developed. The latter consists of two branches, where one branch involves a shallow NN that only solves for the evolution of the characteristic curves, while the second branch involves a NN approximating the solution on the characteristics. While the authors show that with these modifications PINNs are able to learn reasonable solutions to the considered convection-dominated problems,

it also introduces new problems, e.g., mesh generation or the the additional necessity of interpolating the Lagrangian solution back into the Eulerian frame-of-reference.

With this practical discussion, we conclude the theoretical introduction of PINNs. In the next section, we quickly review different approaches, how NNs can be leveraged to predict solutions to PDEs.

3.2.5. A Look Outside the Box: Alternative Approaches for Solving Partial Differential Equations with Neural Networks

Though a prominent subject of current research, PINNs are not the only NN-based possibility for solving PDEs. In the following, we will introduce some alternative concepts, starting off with CNN-based concepts.

Already in 2016, Guo, Li, and Iorio [52] proposed to use CNNs for predicting steady laminar flow fields with the aim to perform this task in real-time. After computing a discrete representation of the geometry using a Signed Distance Function (SDF), it is used as an input to a CNN which is trained in a classical purely supervised fashion, using only data obtained from numerical CFD simulations. The employed CNN consists of an encoder-decoder architecture featuring a shared encoding part for extracting an abstract geometry representation which is used by multiple decoding parts, each corresponding to one solution component. Recently, building up upon this work, Eichinger, Heinlein, and Klawonn [38] investigated and compared different CNN architectures, showing that very small prediction errors can be obtained, when using a *U-Net* architecture. Moreover, they extend the work to a transfer-learning approach, allowing to make predictions of the flow field in completely new geometries with little additional training effort.

Gao, Sun, and Wang [46] also employ CNNs for predicting (parameterized) PDE solutions on different geometries, proposing a physics-informed architecture which does not require data for training. To handle arbitrary, i.e., non-rectangular, geometries, they introduce a coordinate transformation, mapping the PDE into a rectangular reference domain first, before learning the solution. The BCs are strongly imposed by adding padding to the respective convolutional filters. In order to discretize the PDEs, special convolutions are used which correspond to the Finite Difference approximations of the derivatives in the reference domain. According to the authors, the method performs well on different test cases which are compared to results obtained from numerical simulations.

Another approach for solving PDEs with NNs is the *Deep Ritz Method* proposed by Weinan and Yu [178]. It targets a special class of variational (minimization) problems, as which many PDE problems can be expressed. However, instead of minimizing over a function space, a NN with a special architecture is used to convert the minimization problem into an optimization problem with respect to the NN's parameters. Similar to the idea of PINNs, the objective function is evaluated at randomly generated domain points.

Finally, as a last concept, we want to mention the work by Möller, Toshniwal, and Ruiten [116] on *IgaNets* aiming to combine Isogeometric Analysis (IGA) and NNs: Here, a NN is trained to predict the solution coefficients of the unknown solution's expansion in terms of the spline basis used for the discretization. The inputs of the NN are a set of parametric coordinates, the coefficients of eventual source terms and BCs (discretized with

the same spline basis), as well as the control points of the computational domain. The NN is then trained in a physics-informed way by minimizing the strong form of the governing PDE similar to the description in Section 3.2.1.

The above presented list of alternatives to PINNs is surely non-exhaustive but serves as a conclusion of this theoretical part. In the next section, we will now focus on another ML method, namely RL.

3.3. An Introduction to Reinforcement Learning

Reinforcement Learning (RL) is a sub-field of machine learning concerned with sequential decision-making. To solve such decision-making problems, an *agent* interacts with an *environment*, which encodes the problem to be solved. This interaction loop is visualized in Figure 3.4. The environment is in a distinct *state* s at any given time, which the agent can observe and, based on the observation, perform an *action* a, to exert influence on the environment. This combination of observing and performing an action is typically called a *step*. Solving the given problem involves taking a series of sequential steps, called an *episode*, to achieve some predefined goal. The agent's behavior, i.e., the choice of which action to choose in a given state, is determined by a *policy* $\pi : s \mapsto a$.

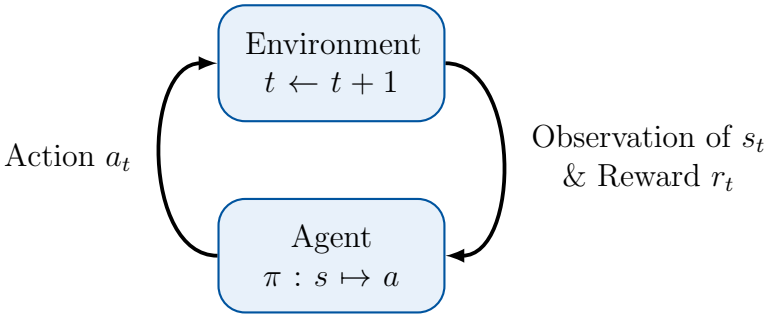


Figure 3.4.: Interaction loop between an agent and an environment during training. In each interaction/training step t, the agent selects an action a_t according to a policy π based on observations of the current environment's state s_t and a numerical reward signal r_t. This changes the state of the environment and generates the new observation and the new reward for the next step.

This policy is learned during a training phase, where the agent interacts with the environment and receives *rewards* r depending on how well it is solving the task at hand. The collected experience consisting of states, actions, and rewards is used to iteratively improve the policy. During training, actions are selected either through the current best estimate of what a good policy would look like or through some exploration mechanism.

While RL algorithms generally follow the framework described above to learn a policy, they differ substantially in their inner functioning and can be characterized based on a number of different properties. We explore these properties in the following Section 3.3.1 and investigate which subset of algorithms is especially well-suited for shape optimization problems in profile extrusion in the remainder of this work.

Since training RL agents is dependent on large amounts of collected experience from the environment, the training process can be time-consuming. To address this issue, we further identify a suitable approach to reduce training times in Section 3.3.2.

3.3.1. Agents - The Reinforcement Learning Learning Algorithms

A major goal of our study is to investigate the suitability of different RL algorithms to shape optimization problems in profile extrusion. Since different algorithms feature different characteristics, specific algorithms or subgroups of algorithms may emerge as especially suited in terms of training speed, stability, and final performance of the trained agents.

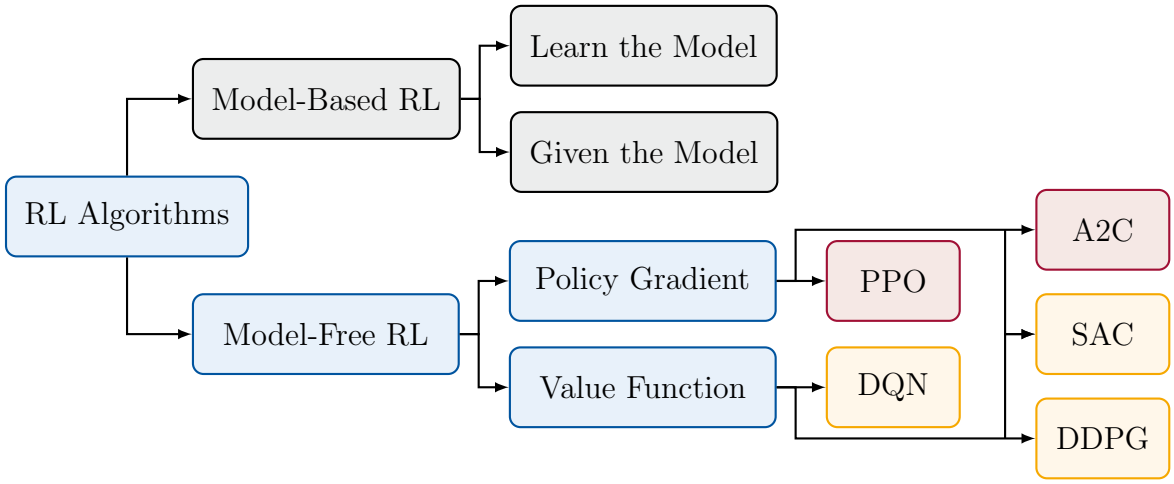


Figure 3.5.: Limited taxonomy of RL agents. Blue boxes represent categories of agents which are relevant for this thesis. Orange boxes symbolize agents trained with an off-policy method, while agents from red boxes are trained with the on-policy method. The categories listed in the gray boxes are not considered in this thesis.

In Figure 3.5, a taxonomy of RL agents is given, showing a range of different agents from diverse agent categories. Since we only study model-free agents in this thesis, no model-based examples are listed explicitly, but the category is depicted for completeness. Model-based RL agents utilize a predictive model of the environment either during training or at decision time. This model of the environment can either be learned during training (Learn the Model) or given before starting the RL training process (Given the Model).

Model-free agents learn a strategy for solving their task without such an explicit predictive model. There are two main methods for learning this strategy. Policy gradient-based agents directly learn the policy, determining which actions to take in which states. In contrast, value function-based approaches learn a function predicting the expected return or reward of the possible actions. This value function is then utilized in the policy to decide which actions to perform. Examples for both approaches are given in the taxonomy, with Proximal Policy Optimization (PPO) [154] belonging to the class of gradient-based methods and Deep Q-Network (DQN) [112] being a value function-based approach. The idea of policy gradient and value function-based methods can also be combined. These combined approaches commonly use an actor-critic structure, consisting of two parts. The critic learns a value function, which is then used to update the actor's policy function. Agents with this training approach are, e.g., Advantage Actor Critic (A2C) [113], Soft Actor Critic (SAC) [53], and Deep Deterministic Policy Gradient (DDPG) [88].

This combined approach alleviates some problems with pure policy gradient algorithms, which can have stability issues during training, but are generally faster to train than pure value function-based methods [74].
Agents can be additionally classified based on the data they use for training. Here, one differentiates between *on-policy* and *off-policy* agents. On-policy algorithms can only use training samples generated with the agent's current policy and need to be discarded after each policy update. Consequently, on-policy methods can only use each training sample once, i.e., in the same training step in which it was generated. In contrast, off-policy methods have no restrictions on the usability of training samples concerning the state of the agent's policy. They can reuse already collected samples even if they are generated with now outdated policy states. Thus, agents with this training method are more sample efficient [166].

3.3.2. Vectorized Environment Training — Accelerating Training Times

While RL agents, once trained, can solve given problems very quickly, the training process itself tends to be time- and resource-intensive. This is especially problematic during the initial development phase, where multiple parameters are typically iteratively refined until a configuration is found that successfully solves the problem. The longer the time required to evaluate the effects of changes to the system, the slower the development progresses. Hence it is beneficial to explore approaches for reducing training times.

One possibility to reduce training times is parallelization. As discussed in [135], the speedup achievable by increasing the parallelization of the computations carried out within the environment (i.e., a Computational Fluid Dynamics (CFD) simulation in their case) is typically limited. This is especially true for small calculations: Here, the startup costs and the communication overhead weigh more severe and are not necessarily dominated by the time required to perform the actual computations. Instead, Rabault and Kuhnle [135] proposed a different parallelization approach, which does not provide the CFD simulation with more computational resources, but creates multiple environments simultaneously, each running separate CFD simulations. A single agent can then collect experiences from all of these environments in parallel and consequently collect a larger amount of experience in a fixed amount of time compared to the usual single environment setup. These environments are independent of each other, i.e., they do not communicate, so that communication only happens between the agent and each environment. In this configuration, the agent needs to be enabled to provide (different) actions to all environments in each training step and gather the training data consisting of observations, rewards, and actions. This method is known as *vectorized environment training* in the field of RL.

In our work, we follow the same strategy as outlined above to accelerate the training times of our agents. Our framework for RL-based shape optimization – which we introduce in more detail in the next section – employs the open-source python package `stable-baselines3` [137] for the implementation of the RL algorithms as well as realizing the vectorized environment training.

4 Physics-Informed Neural Networks as Reduced Simulation Models

Within this chapter, we present our findings on the application of Physics-Informed Neural Networks (PINNs) to construct ROMs for numerical simulations as formulated in Research Question (RQ I). Directly at the beginning, we want to make the point that we do not claim or believe that PINNs will ever replace classical numerical (Computational Fluid Dynamicss (CFDs)) simulations, but they might be used to enhance classical simulations, e.g., by solving parts of the problem in complex multi-physics applications or for providing a cheap-to-evaluate surrogate model when tackling tasks like the optimization or (real-time) control of physical systems. In particular, we aim to answer the following research questions in the context of this chapter:

(I) (a) Can PINNs learn the complex rheological behavior of plastic melts in profile extrusion dies?

(b) Can PINNs accurately capture the flow features in intricate bioreactor geometries?

Thus, this chapter is structured as follows: In Section 4.1, we briefly introduce the software framework `FrameXDE` developed in this thesis and used for obtaining the results for the two application cases, which we introduce subsequently. We will highlight its most important features with respect to their impact on generating reasonably well-performing Reduced Order Models (ROMs).

Section 4.2 is dedicated to the continuous manufacturing process of profile extrusion. Here, the difficulty of constructing a proper ROM mainly arises from the inherent non-linearity of the material laws involved to model the shear-thinning flow of molten plastic. We show that training a PINN to predict the solution of a simple two-dimensional channel flow is already challenging and that it does not learn a meaningful solution when all loss components are weighted equally. Therefore, we introduce a heuristic for choosing the loss weights and discuss the improvement with respect to the PINN's prediction capabilities.

The second application case, presented in Section 4.3, deals with the prediction of the flow field in bioreactors. We will consider two different types of ROMs, namely one parameterized model which can predict the flow field for various Reynolds numbers, and a second model which is trained for a fixed Reynolds number but involves more elaborate techniques inspired by insights from high-fidelity modeling for obtaining a much better prediction accuracy.

4.1. The FrameXDE Framework

To facilitate the use of PINNs for different applications, the FrameXDE package [96] was developed, building up on the publicly-available python package deepxde by Lu et al. [93] and further improved in two following theses [35, 97]. While deepxde generally supports many different Machine Learning (ML) libraries, when beginning to develop FrameXDE, the tensorflow package [1] was by far the most supported, so we chose it for all experiments reported in this thesis. In the following, we will first briefly introduce the general structure of the package, before going into detail about some selected features, which may help to improve the prediction accuracy of PINN models.

4.1.1. Architecture and Model Creation Workflow

The FrameXDE package comprises a couple of modules which aim to make the training and the evaluation of PINN models as easy as possible for inexperienced users, while simultaneously providing advanced possibilities for researchers who are more familiar with the topic. While the network architecture as well as most of the hyper-parameters (e.g., the optimization algorithm, the learning rate, ...) can be configured via a simple .json file, additional information such as the governing Partial Differential Equations (PDEs), Initial and Boundary Conditions (IBCs) or the geometry need to be directly defined in the source code.

The following subsections will shortly introduce the most important modules of the package, i.e., introducing different available geometries as well as the error measures used to compare the PINN predictions to numerical high-fidelity simulation data.

The geometry Module: Enable Flexible Geometry Creation

Generally, PINNs can be trained on arbitrary computational domains Ω, as long as it is possible to generate (random) points inside the domain as well on its boundary Γ. deepxde uses Constructive Solid Geometry (CSG) to create complex geometries, i.e., boolean operations can be performed to combine simpler shapes [93]. While this works well for the simple geometries usually employed in academic examples, it is rather not useful when dealing with more realistic, e.g., industrial, geometries. Consequently FrameXDE features additional geometry classes to alleviate this drawback.

Using Computational Meshes as Discrete Geometry Representation Although one advantage of PINNs is that they can be seen as a mesh-free solution approach for PDEs [22], if a computational mesh of a certain geometry is available, we can exploit it as a discrete geometry representation. Assume we have been given a computational mesh $\Omega^h = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N_\mathrm{n})}\} \subset \Omega$ of a computational domain $\Omega \cup \Gamma$. Then we only need to identify the points that belong to the boundary Γ and can define the required data sets as

$$\begin{aligned} \mathcal{D}_\mathrm{PDE} &\coloneqq \Omega^h \\ \mathcal{D}_\mathrm{IBC} &\coloneqq \left\{ \boldsymbol{x} \in \Omega^h \mid \boldsymbol{x} \in \Gamma \right\} . \end{aligned}$$

Now we can (randomly) generate points from these data sets to train the PINN. This approach however might introduce a bias in the training data in the case of locally refined

meshes: The reason therefore is that a higher point density automatically corresponds to a higher probability of generating points from this region, eventually resulting in a local overfitting of the PINN.

From hypercube to Arbitrary Shapes with Spline-based Geometries Another idea for dealing with more complex geometries, is to define them directly via surface or volume splines. This greatly simplifies the generation of points inside the domain and on the boundary, since the spline's parameter space is always a hypercube, i.e., – without restriction of generality – we can even assume a unit hypercube. Consequently, we only need to generate points on the boundary as well the inner part of a unit hypercube and then subsequently transfer them into physical space by evaluating the spline. This functionality has been implemented in `FrameXDE` with the help of the open-source package `gustaf`[1]. This approach however, has two caveats: For once, the distribution of points in the physical space (i.e., the distribution of collocation points used for training the PINN) does not necessarily correspond to the distribution of points in the parameter space but instead depends on the spacing of the knot vector (i.e., equidistant points in the parameter space will not be equidistant in the physical space). Moreover, during the training of a PINN one needs to check if certain points lie within the geometry or on its boundary respectively. When using a spline representation of the geometry, in principle, a non-linear equation system needs to be solved to determine that, increasing the computational complexity of this approach.

4.1.2. Selected Features

In this section, we specifically introduce some features that we found especially useful in improving the accuracy of PINNs as ROMs. Throughout the section, we will illustrate the respective features on a simple test case, where we train a PINN to predict the solution to a Poisson equation with constant unit source term and homogeneous boundary conditions on an L-shaped geometry. That is, we have

$$\begin{aligned} \mathcal{R}_{\text{PDE}}[u] &= -\Delta u - 1 \,, \\ \mathcal{R}_{\text{IBC}}[u] &= u \,, \end{aligned}$$

where u denotes the unknown PDEs solution.

HPO Hyper-parameter Optimization (HPO) is not only restricted to PINNs but generally applicable to many ML models, which usually come with a couple of parameters requiring (optimal) tuning on a given problem to achieve a good performance [50, Section 11.4]. In principle, the hyper-parameters can either be chosen by manual trial-and-error or by automatically trying out different combinations until the best-performing model is found. The first approach usually requires a profound understanding of the involved model parameters as well as their interplay [50, Section 11.4]. In the case of the second approach there are different algorithms to explore the parameter space automatically, which can be generally divided into model-free and model-based methods.

In model-free methods, the search space is explored by evaluating the model at distinct points from the search space and choosing the best model after a fixed number of

[1] Available on https://github.com/tataratat/gustaf (last accessed: 25th March 2023)

evaluations. Different methods arise depending on how the points are generated from the parameter space. When performing a *grid search*, for each hyper-parameter a discretized interval needs to be provided and different models are trained for all possible combinations of the discrete hyper-parameter values, resulting in an exponential complexity in the number of hyper-parameters. In contrast, when performing *random search*, a probability distribution is provided for each hyper-parameter and the evaluation points of the model are generated by drawing samples from these distributions. Since for every evaluation, all hyper-parameter values are drawn from their distributions, this method often performs better than a grid search in the presence of parameters that only have a minor influence on the performance measure.

Model-based methods instead iteratively construct and improve a model of the employed performance measure and use optimization techniques to find the best-performing hyper-parameter values. Many state-of-the-art algorithms use Bayesian regression models to construct the surrogate.

Within `FrameXDE`, we employ the `optuna` python package by Akiba et al. [3]. `optuna` allows to dynamically construct a parameter space, i.e., not all parameters need to be defined immediately and features different sampling methods for finding good hyper-parameters within the parameter space. By default, a model-based approach is employed, using a Tree-structured Parzen Estimator (TPE) algorithm [11]. As performance estimate, we take the empirical risk as defined in Equation (3.16) evaluated at a separate test data set not seen during training.

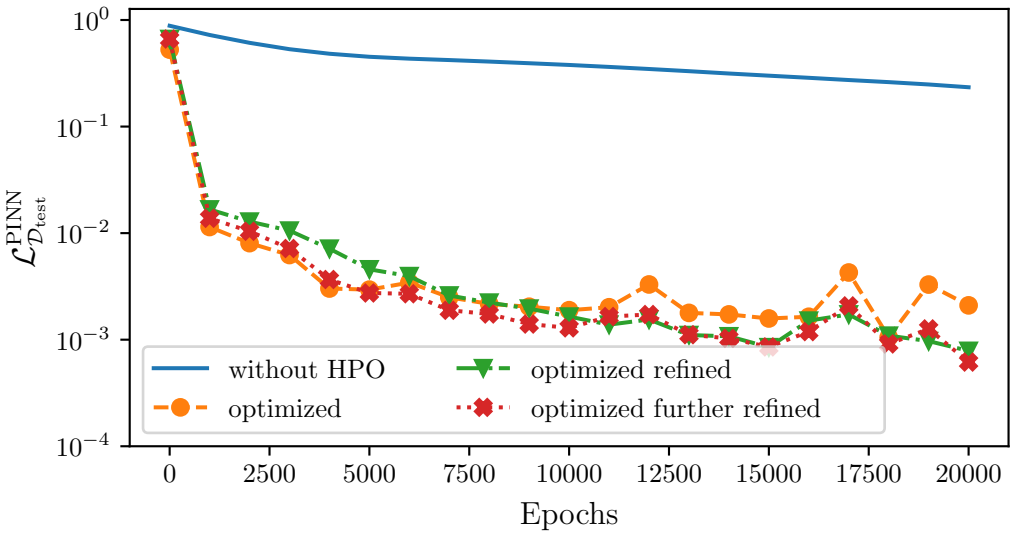


Figure 4.1.: Comparison of the convergence of PINNs with and without HPO. As it can be seen, performing HPO can significantly improve the prediction accuracy (orange curve), while there is no guarantee of finding the best model. The performance can still be improved by further manual tuning (green and red curves).

Figure 4.1 shows the evolution of the loss function for the Poisson problem, evaluated on a separate test data set $\mathcal{D}_{\mathrm{test}}$ not seen during training, thus being a measure for the generalization error. One can see that significant improvement can be achieved by performing a HPO (e.g., comparing the blue and orange curves). Yet, there is no guarantee

that the best model architecture is found by the HPO as manual fine-tuning of the optimized architecture resulted in even better predictions (green and red curves). This can of course be attributed to the limited amount of trials allowed during the HPO which is computationally very expensive.

RAR As already discussed in Section 3.2.4, it sometimes makes sense to not only sample collocation points randomly but instead refine the sampling in certain regions of the computational domain. We can draw an analogy to mesh refinement / adaptation in the classical Finite Element Methods (FEMs), where usually an indicator function (e.g., an estimate of the generalization error) is employed to determine regions of interest. One rather simple approach to accomplish this for PINNs is Residual-based Adaptive Refinements (RARs) [93, 181]. Here, the empirical risk (3.16) is used as indicator function, as it informs about regions in the computational domain where the strong form of the PDE or boundary / initial conditions are only poorly fulfilled. This function is then evaluated on an additional, fine, but randomly sampled set of points and afterward a subset of the points with the highest value of the residual are added to the data sets used to evaluate the empirical risk in the next iteration. This procedure can be repeated after a predefined number of iterations.

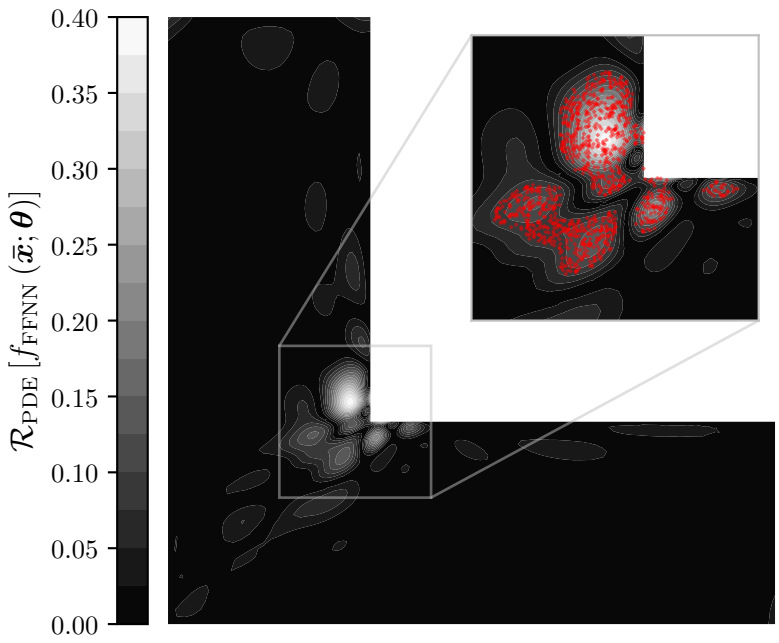


Figure 4.2.: Point-wise PDE residual obtained from the evaluation of a trained PINN. The residual is highest in the vicinity of the kink in the geometry. Here, RAR adds additional points to improve the PINN prediction (visualized as red crosses in the zoomed-in plot in the upper-right corner).

In Figure 4.2, we visualize the PDE residual after training a PINN on our example problem. The residuals increase close to the sharp kink in the L-shaped geometry. The

upper right corner of the plot shows a zoomed-in version of that location. Here, the points that are added after performing one RAR iteration are highlighted in red. One can see that they are exactly generated where the PDE residual is highest. Although it works well in this example, RAR also introduces additional parameters, e.g., the number of points on which the residual is evaluated, the number of points with the highest residual that will be added, and the number of RAR iterations or an acceptance threshold respectively, determining how many RAR steps are supposed to be performed.

L-LAAFs One common criticism of PINNs is that they are hard to train, i.e., exhibit long training times due to poor convergence [26, 175]. To overcome this drawback, Jagtap, Kawaguchi, and Em Karniadakis [66] proposed the use of locally-adaptive activation functions, i.e., activation functions whose slopes can be controlled by the learning algorithm. There are different ways to define this parameter, e.g., globally, layer-wise or neuron-wise. Here, we only consider the layer-wise case, which is referred-to as Layerwise Locally-Adaptive Activation Function (L-LAAF). Therefore, the evaluation in the hidden layers introduced in Equation (3.3d) is rewritten as

$$f_{\mathrm{P}}^{(l)}(\boldsymbol{z}) = \sigma^{(l)}\left(cd^{(l)}\left[\boldsymbol{b}^{(l)} + \left(\boldsymbol{W}^{(l)}\right)^T \boldsymbol{z}\right]\right) \qquad \forall\, l \in \{1, ..., N_{\mathrm{hl}}\}\ ,$$

where $c \geq 1$ is a fixed constant (i.e., an additional hyper-parameter that needs to be chosen appropriately for every problem) and $d^{(l)} \in \mathbb{R}$ are additional learnable parameters which are initialized such that $cd^{(l)} = 1 \quad \forall\, l \in \{1, ..., N_{\mathrm{hl}}\}$ holds true. Consequently, the vector of trainable Neural Network (NN) parameters now also includes the slope parameters $\boldsymbol{d} = \left(d^{(1)}, \ldots, d^{(N_{\mathrm{hl}})}\right)^T$, i.e., $\tilde{\boldsymbol{\theta}} = (\boldsymbol{\theta}, \boldsymbol{d})^T$. Note that the introduction of the additional parameters is negligible in the layer-wise case, as we only add one additional parameter per hidden layer.

By adding the additional parameters for controlling the activation functions' slopes, the aim is to increase the magnitude of the back-propagated gradients to prevent them from vanishing [66] which is especially important the longer the training continues. This can be improved by adding an additional so-called *slope-recovery* term to the PINN's loss function, namely

$$\mathcal{S}(\boldsymbol{d}) = \frac{1}{\frac{1}{N_{\mathrm{nhl}}} \sum_{l=1}^{N_{\mathrm{nhl}}} d^{(l)}}\ .$$

Consequently, when applying L-LAAF, the resulting loss function becomes

$$\tilde{\mathcal{L}}_{\mathcal{D}}^{\mathrm{PINN}}(\tilde{\boldsymbol{\theta}}) = \mathcal{L}_{\mathcal{D}}^{\mathrm{PINN}}(\boldsymbol{\theta}) + \alpha_d \mathcal{S}(\boldsymbol{d})\ ,$$

with $\mathcal{L}_{\mathcal{D}}^{\mathrm{PINN}}$ defined according to Equation (3.16) and α_d an additional weight for the slope-recovery term. Jagtap, Kawaguchi, and Em Karniadakis [66] prove that – equipped with this loss function and certain assumptions on the learning rate of the employed steepest-descent algorithm – the PINN training will converge to at least a critical point (if not a local minimum), which is not sub-optimal.

Figure 4.3 depicts the evolution of the empirical risk with and without the additional slope recovery term $\mathcal{S}(\boldsymbol{d})$. One can see that the empirical risk is smaller in each iteration if the slope recovery term is included, i.e., it converges faster (see, e.g., the orange and green curves). However, one can also see that the performance greatly varies with the additional hyper-parameter c (see red curve), indicating that a proper tuning of this parameter is required to obtain good convergence.

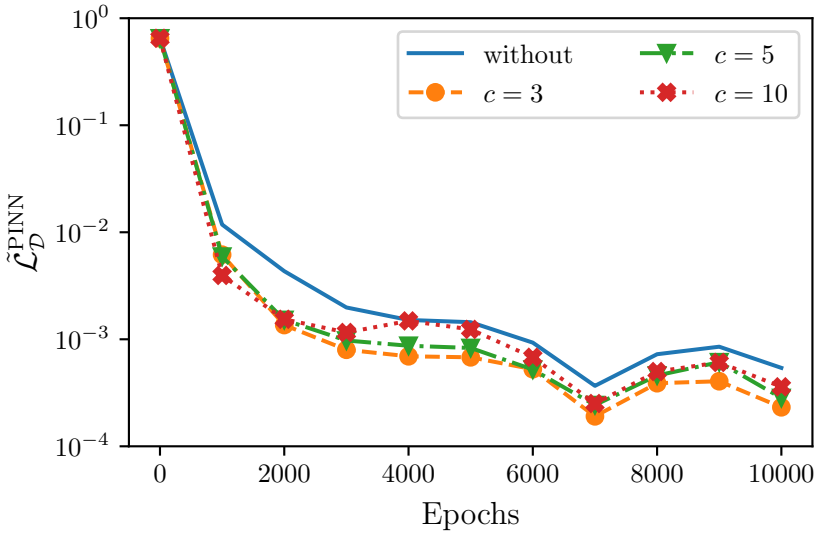


Figure 4.3.: Convergence of the loss function with and without the slope recovery term. While the convergence is generally better, i.e., faster, if L-LAAFs are used, the behaviour greatly depends on the value of the additional hyper-parameter c.

4.2. Application to Profile Extrusion

This section will demonstrate some of the difficulties one might face when employing PINNs as ROMs. The application case is inspired by the manufacturing process of profile extrusion, i.e., we consider the flow of molten plastic through an idealized two-dimensional profile extrusion die geometry. The behavior of the plastics melt inside the flow channel will be modeled by a shear-thinning law according to Carreau as introduced in Equation (2.3). This material model introduces a non-linearity into the governing equations, which needs to be correctly captured by the PINN model, making this test case challenging. To begin with, we will report a negative result in Section 4.2.1 by demonstrating that the naive application of PINNs to this problem fails, if all loss function components are weighted equally. Afterward, we will propose a heuristic for choosing the loss weights based on a non-dimensionalization of the governing PDEs in Section 4.2.2 and revisit the test case. We will see that with the proposed heuristic we are able to obtain predictions which are in very good agreement with high-fidelity data from a numerical FEM simulation.

First, we revisit the model equations that govern the flow of molten plastic inside profile extrusion dies. In our model, we assume a stationary, incompressible and isothermal flow. Since the Reynolds number characterizing the flow in profile extrusion dies is typically very small, i.e., $\text{Re} \ll 1$, we neglect convection. Under these assumptions, Equation (2.2) simplify and the loss function components corresponding to the underlying conservation laws read

$$\mathcal{R}_{\text{PDE}}\left[\boldsymbol{u}\right] = \begin{pmatrix} \boldsymbol{\nabla} \cdot \boldsymbol{v} \\ -\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} \end{pmatrix} . \tag{4.1}$$

For this test case, the flow channel is resembled by a rectangle of length l and height h, i.e., $\Omega = [0, l] \times \left[-\frac{h}{2}, \frac{h}{2}\right]$. Regarding Boundary Conditions (BCs), we prescribe Dirichlet

conditions for the velocity vector at the inflow, no-slip conditions on the wall, while we restrict the y-component of the velocity as well as the pressure at the outflow. This results in the following BC residuals

$$\mathcal{R}_{\text{inflow}}\left[\boldsymbol{u}\right] = \begin{pmatrix} v_{\text{x}} - v_{\text{x,max}}\left(1 - 4\left(\frac{y}{h}\right)^2\right) \\ v_{\text{y}} \end{pmatrix}, \tag{4.2a}$$

$$\mathcal{R}_{\text{wall}}\left[\boldsymbol{u}\right] = \begin{pmatrix} v_{\text{x}} \\ v_{\text{y}} \end{pmatrix}, \tag{4.2b}$$

$$\mathcal{R}_{\text{outflow}}\left[\boldsymbol{u}\right] = \begin{pmatrix} v_{\text{y}} \\ p \end{pmatrix}. \tag{4.2c}$$

The parabolic velocity profile prescribed at the inflow for the x-component of the velocity, v_x, corresponds to the analytical solution of this problem if a Newtonian fluid is considered instead, i.e., $\mu = A = \text{const.}$. The employed geometry and material parameters are reported in Tables 4.1 and 4.2.

Table 4.1.: Geometry and BC-related parameters of the profile extrusion test case.

Parameter	Value	Unit
l	0.2	m
h	0.02	m
$v_{\text{x,max}}$	0.115	$\text{m}\,\text{s}^{-1}$

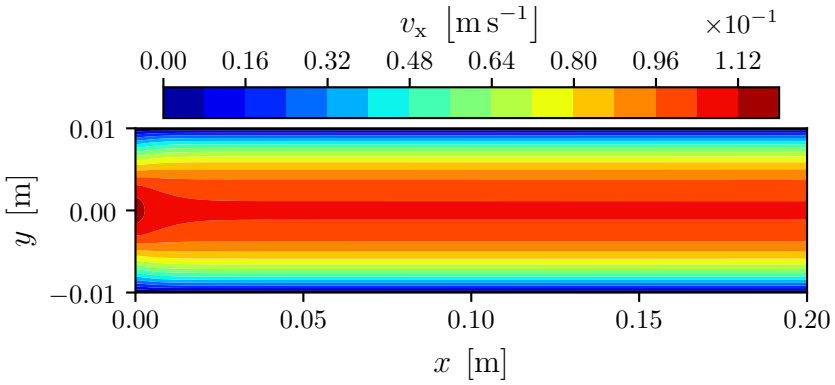
Table 4.2.: Material parameters of the profile extrusion test case.

Parameter	Value	Unit
ρ	1070	$\text{kg}\,\text{m}^{-3}$
A	6589	$\text{kg}\,\text{m}^{-1}\,\text{s}^{-1}$
B	0.138	s
C	0.725	–

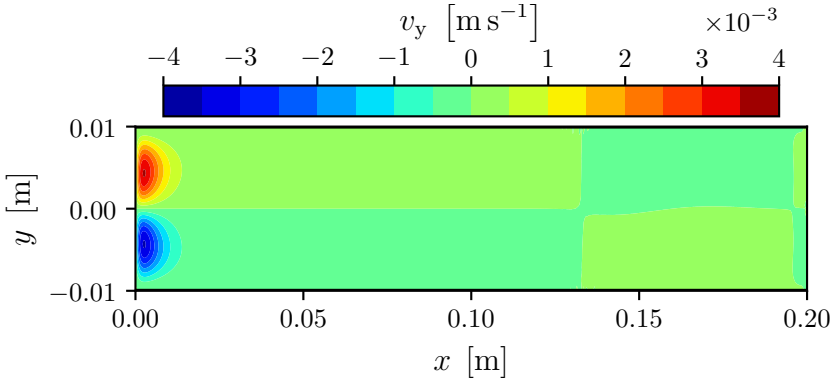
For reference, we have computed a high-fidelity FEM solution on a structured grid consisting of $N_{\text{n}} = 37500$ nodes and $N_{\text{el}} = 36926$ quadrilateral elements. The solution fields for the velocity in x-direction v_x and the pressure p are depicted in Figure 4.4. Since the BC prescribed on the inflow boundary stems from the analytical solution obtained for a Newtonian fluid, immediately downstream of the inflow one can observe an inlet zone ($x < 0.025\,\text{m}$) in which the flow adapts according to the different material behavior (cf. Figure 4.4a). In this region, the $y-$velocities are also relevant and not negligible (cf. Figure 4.4b) as they would be in the Newtonian case. In terms of the pressure field, we obtain a linear profile which is once again similar to the Newtonian case (cf. Figure 4.4c). In the following section, we will discuss the construction of PINN-based ROMs for this application.

4.2.1. Unit Weighting of the Loss Function Components

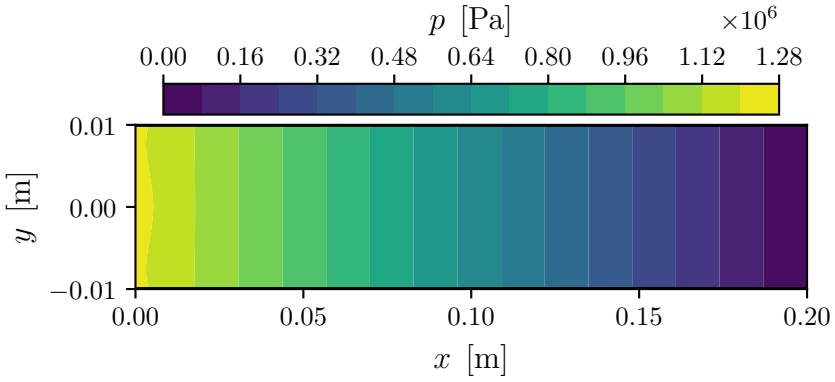
As pointed out in Section 3.2.1, the individual contributions to a PINN's loss function can be weighted individually. As already alluded to in Section 3.2.4, no general rule exists, on how these weights should be determined for an arbitrary problem. In this section, we consider the naive approach of choosing unit weights. Equipped with the governing equations and the BCs reported above, we train a PINN to predict the solution. The NN architecture as well as the learning parameters of the training are reported in Tables 4.3 and 4.4.



(a) x-component of velocity field.



(b) y-component of velocity field.



(c) Pressure field.

Figure 4.4.: Numerical high-fidelity solution obtained from an FEM simulation.

Table 4.3.: NN architecture.

Parameter	Value
N_{hl}	2
N_{nhl}	32
σ	tanh

Table 4.4.: Hyper-parameters related to the training of the NN.

Parameter	Value
Optimizer	L-BFGS
N_{epochs}	$5 \cdot 10^4$
$\|\mathcal{B}_\Omega\|$	1024
$\|\mathcal{B}_\Gamma\|$	512
Resampling rate	10^3
$\boldsymbol{\alpha}_{\text{PDE}} = (\alpha_{\text{momentum,x}}, \alpha_{\text{momentum,y}}, \alpha_{\text{mass}})^T$	$(1,1,1)^T$
$\boldsymbol{\alpha}_{\text{BC}} = (\alpha_{\text{inflow}}, \alpha_{\text{wall}}, \alpha_{\text{outflow}})^T$	$(1,1,1)^T$

It is known, that NNs perform best, if the inputs to the network and every layer are normalized [55, Section 11.5.3][117, Section 1.4.3]. Thus, we also employ input and output scaling to our PINN by introducing additional transformation layers. That is, the NN model becomes

$$f_{\text{PINN}} = \left(f_{\text{post}} \circ f_{\text{FFNN}} \circ f_{\text{pre}}\right)(\bar{\boldsymbol{x}}) \,, \tag{4.3}$$

where

$$f_{\text{pre}} : \mathcal{X} \to [-1,1]^{N_\mathcal{X}} \,, \bar{\boldsymbol{x}} \mapsto \bar{\boldsymbol{x}}^\star \,,$$
$$f_{\text{post}} : [-1,1]^{N_\mathcal{Y}} \to \mathcal{Y}, \bar{\boldsymbol{y}}^\star \mapsto \bar{\boldsymbol{y}} \,,$$

account for scaling the inputs and outputs into appropriate ranges.

Figure 4.5 depicts the evolution of the different loss components over the number of trained epochs. One can observe that 4 out of 6 loss components are L-shaped, i.e., they decay fast within the first roughly 2000 epochs before continuing to decay at a much smaller rate and eventually leveling out after a reduction of roughly 12 orders of magnitude. Interestingly, two loss components, i.e., the ones corresponding to the inflow and wall BCs do not depict any decrease at all but in fact stay nearly constant over the whole training time. This already indicates that the PINN might have had problems to learn the boundary conditions. In fact, if one looks at the PINN's predictions for the unknown solution fields of the PDE system shown in Figure 4.6, it can be constituted that – despite the seemingly good convergence behavior and automatic premature termination of the training – the PINN completely fails to provide an accurate prediction. While these findings on the one hand motivate the need for a more elaborate way of choosing the loss function weights, it also reveals that a good convergence of the loss function does not necessarily yield good predictions. One reason for this can be attributed to the fact that the loss function only measures how well the PINN prediction satisfies the governing physical constraints but

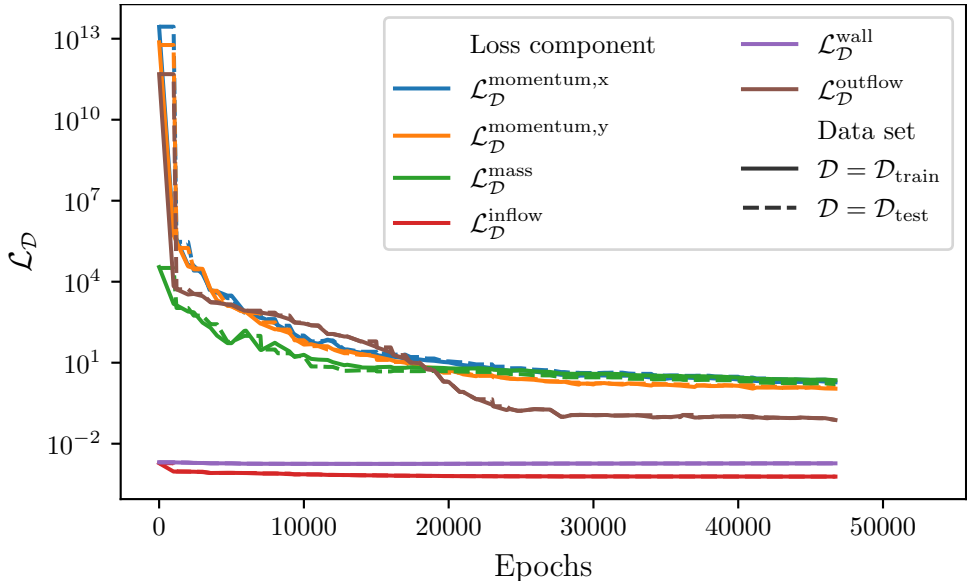


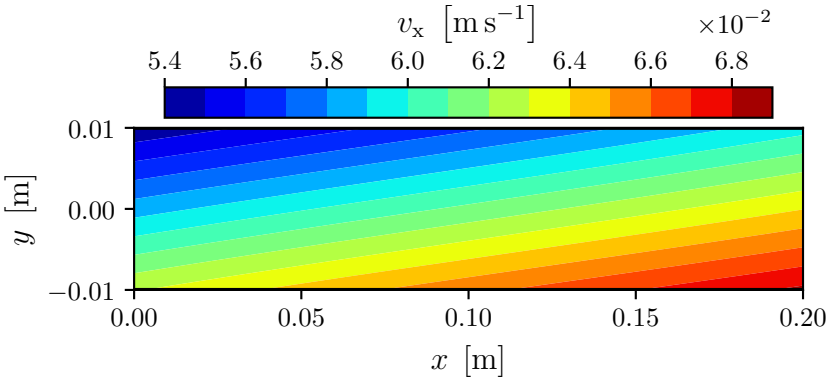
Figure 4.5.: Convergence of the loss function components with respect to the number of trained epochs. The training was automatically terminated before the maximum amount of 50000 epochs was reached since the training parameters did not change anymore.

not how close it actually is to the true solution. Figure 4.6 shows that the PINN under-predicts every single solution field, i.e., it does not even manage to predict the correct order of magnitude for the pressure field (cf. Figure 4.6c). Moreover, the predictions for the velocity components Figures 4.6a and 4.6b also do not match that of the numerical solution qualitatively. The under-prediction of all solution fields and the missing improvement in the losses associated with the BCs hint that the PINN learns a trivial solution to the PDE problem. This explains the low final loss values and the good fulfillment of the strong PDE residual throughout the domain, which is visualized in Figure B.1.

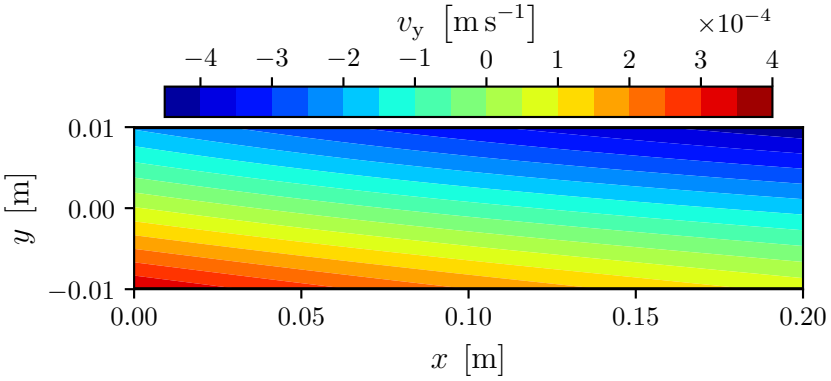
4.2.2. A Heuristic Approach for Choosing the Loss Weights

As we have seen in the previous section, choosing the weights arbitrarily results in a poor prediction quality. Consequently, the choice of appropriate weights is crucial to obtain a meaningful ROM. Ideally, we would like to have a method that allows to choose the weights without a significant computational overhead, while simultaneously being easily applicable to different problems without much tuning. In the following, we will briefly introduce such a heuristic.

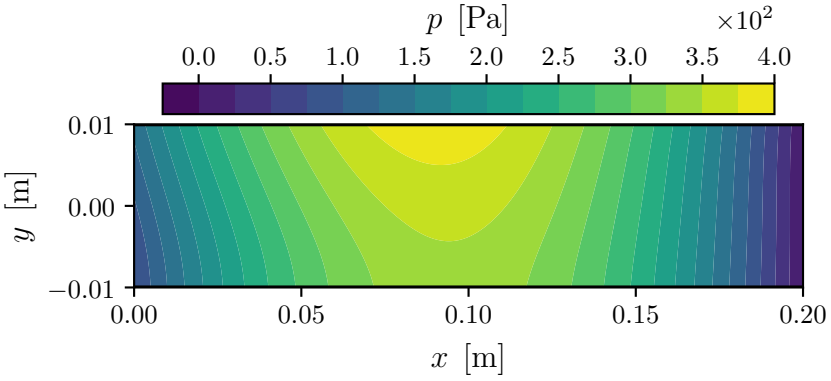
From a physical point of view, it is clear that all components involved in the PINN loss (3.16) have different meanings. As a consequence, their orders of magnitude can differ significantly. With respect to the combined loss function, this means that some terms dominate over others due to their different magnitudes. This motivates the idea to choose the loss weights such that the individual loss components are balanced with respect to their magnitudes. Mathematically, this can be achieved, by non-dimensionalizing all involved equations.



(a) x-component of velocity field.



(b) y-component of velocity field.



(c) Pressure field.

Figure 4.6.: PINN predictions for the solution fields after training for 50000 epochs with unit loss function weights.

We will illustrate this procedure for the equations governing the flow inside a profile extrusion die, i.e., Equation (4.1). Stating the conservation laws component-wise, we obtain

$$\begin{aligned} \frac{\partial v_\mathrm{x}}{\partial x} + \frac{\partial v_\mathrm{y}}{\partial y} &= 0 \,, \\ -\mu \left(\frac{\partial^2 v_\mathrm{x}}{\partial x^2} + \frac{\partial^2 v_\mathrm{x}}{\partial y^2} \right) + \frac{\partial p}{\partial x} &= 0 \,, \\ -\mu \left(\frac{\partial^2 v_\mathrm{y}}{\partial x^2} + \frac{\partial^2 v_\mathrm{y}}{\partial y^2} \right) + \frac{\partial p}{\partial y} &= 0 \,. \end{aligned}$$

These equations can be non-dimensionalized by introducing reference quantities for each involved variable (in the following denoted by the index $_\mathrm{ref}$) which provide reasonable estimates for the magnitude of these quantities within the computational domain. The PDE itself is then formulated with respect to dimensionless quantities (denoted by a superscript $\star$), which have an order of $\mathcal{O}(1)$. In front of each derivative, we obtain a (constant) coefficient composed of multiple reference quantities and material parameters. For this application case, the non-dimensional PDEs read

$$\begin{aligned} \frac{v_\mathrm{x,ref}}{x_\mathrm{ref}} \frac{\partial v_\mathrm{x}^\star}{\partial x^\star} + \frac{v_\mathrm{y,ref}}{y_\mathrm{ref}} \frac{\partial v_\mathrm{y}^\star}{\partial y^\star} &= 0 \,, \\ -\mu \frac{v_\mathrm{x,ref}}{x_\mathrm{ref}^2} \frac{\partial^2 v_\mathrm{x}^\star}{\partial (x^\star)^2} - \mu \frac{v_\mathrm{x,ref}}{y_\mathrm{ref}^2} \frac{\partial^2 v_\mathrm{x}^\star}{\partial (y^\star)^2} + \frac{p_\mathrm{ref}}{x_\mathrm{ref}} \frac{\partial p^\star}{\partial x^\star} &= 0 \,, \\ -\mu \frac{v_\mathrm{y,ref}}{x_\mathrm{ref}^2} \frac{\partial^2 v_\mathrm{y}^\star}{\partial (x^\star)^2} - \mu \frac{v_\mathrm{y,ref}}{y_\mathrm{ref}^2} \frac{\partial^2 v_\mathrm{y}^\star}{\partial (y^\star)^2} + \frac{p_\mathrm{ref}}{y_\mathrm{ref}} \frac{\partial p^\star}{\partial y^\star} &= 0 \,. \end{aligned}$$

The coefficients in front of each equations' summands provide an estimate for their magnitude. More precisely, the magnitude $\mathcal{M}$ of an equation is determined by the magnitude of its largest coefficient, yielding the following equation orders:

$$\mathcal{M}_\mathrm{mass} = \max \left\{ \frac{v_\mathrm{x,ref}}{x_\mathrm{ref}}, \frac{v_\mathrm{y,ref}}{y_\mathrm{ref}} \right\} \,, \tag{4.4a}$$

$$\mathcal{M}_\mathrm{momentum,x} = \max \left\{ \mu \frac{v_\mathrm{x,ref}}{x_\mathrm{ref}^2}, \mu \frac{v_\mathrm{x,ref}}{y_\mathrm{ref}^2}, \frac{p_\mathrm{ref}}{x_\mathrm{ref}} \right\} \,, \tag{4.4b}$$

$$\mathcal{M}_\mathrm{momentum,y} = \max \left\{ \mu \frac{v_\mathrm{y,ref}}{x_\mathrm{ref}^2}, \mu \frac{v_\mathrm{y,ref}}{y_\mathrm{ref}^2}, \frac{p_\mathrm{ref}}{y_\mathrm{ref}} \right\} \,. \tag{4.4c}$$

Similarly, we can proceed for the BCs, where especially in the case of Dirichlet BCs, the non-dimensionalization is straightforward. For this test case, we obtain the following expressions from Equation (4.2)

$$\mathcal{M}_\mathrm{inflow} = \max \left\{ v_\mathrm{x,ref}, v_\mathrm{y,ref} \right\} \,, \tag{4.4d}$$

$$\mathcal{M}_\mathrm{wall} = \max \left\{ v_\mathrm{x,ref}, v_\mathrm{y,ref} \right\} \,, \tag{4.4e}$$

$$\mathcal{M}_\mathrm{outflow} = \max \left\{ v_\mathrm{y,ref}, p_\mathrm{ref} \right\} \,. \tag{4.4f}$$

After identifying the dominating coefficients in all physical constraints, we can use them to determine the weights such that all contributions are weighted equally. We need to recall

that all residuals are passed through a loss function measuring the error, where for our case a mean-squared error is employed (cf. Section 3.1.2). This results in the following heuristic to choose the loss weights:

$$\boldsymbol{\alpha} = \left(\mathcal{M}^{-2}_{\text{mass}}, \mathcal{M}^{-2}_{\text{momentum,x}}, \mathcal{M}^{-2}_{\text{momentum,y}}, \mathcal{M}^{-2}_{\text{inflow}}, \mathcal{M}^{-2}_{\text{wall}}, \mathcal{M}^{-2}_{\text{outflow}}\right)^T . \tag{4.5}$$

This heuristic for choosing the loss weights provides an easy-to-evaluate formula that can moreover be easily extended to other problems as it only requires a non-dimensionalization of all involved equations as well as reference values for all variables which can generally be determined for most problems. However, since the core idea of this approach is to balance all loss contributions, the chosen reference values have to be representative for the magnitude of the considered quantities. We will demonstrate this for the pressure in our application for which a reference value is usually hard to specify in the incompressible case as the pressure gradient is rather characteristic for a problem instead of its actual value. It is thus common to compute the reference pressure p_{ref} based on other reference quantities. Here, expressions like $p_{\text{ref}} = \rho v^2_{\text{ref}}$ or $p_{\text{ref}} = \frac{\mu v_{\text{ref}}}{l_{\text{ref}}}$ are used in the literature [41], which we however found to under-predict the pressure in our application. Therefore, we propose a different term for the reference pressure which is derived from an analytical expression for the pressure gradient in the case of a Newtonian channel flow, i.e.,

$$p_{\text{ref}} = \frac{8\mu v_{\text{x,max}} x_{\text{ref}}}{y^2_{\text{ref}}} . \tag{4.6}$$

The derivation can be found in Appendix B.2.

Next, we will investigate the performance of the proposed heuristic by applying it to determine loss weights that will result in meaningful PINN predictions for this application case. The architecture of the NN as well as the hyper-parameters related to the training process are the same as in Section 4.2.1, except for the loss weights which are chosen according to Equations (4.4) and (4.5). The used reference values are reported in Table 4.5.

Table 4.5.: Reference quantities chosen for non-dimensionalization of the involved equations.

Reference quantity	Expression
x_{ref}	l
y_{ref}	h
$v_{\text{x,ref}}$	v_{max}
$v_{\text{y,ref}}$	$0.01\,\text{m}\,\text{s}^{-1}$
p_{ref}	Equation (4.6)

Figure 4.7 depicts the evolution of the loss function components over the number of trained epochs. Note that this plot depicts only the loss components, i.e., before multiplication with the heuristically chosen weights, to allow a better comparison with the case of unit weights depicted in Figure 4.5. Firstly, it can be observed that all losses exhibit significantly higher magnitudes than in the previous section, especially after the training is completed. On average, all loss components have been reduced by roughly 3 orders of magnitude. Secondly, we can constitute that the loss curves corresponding to the

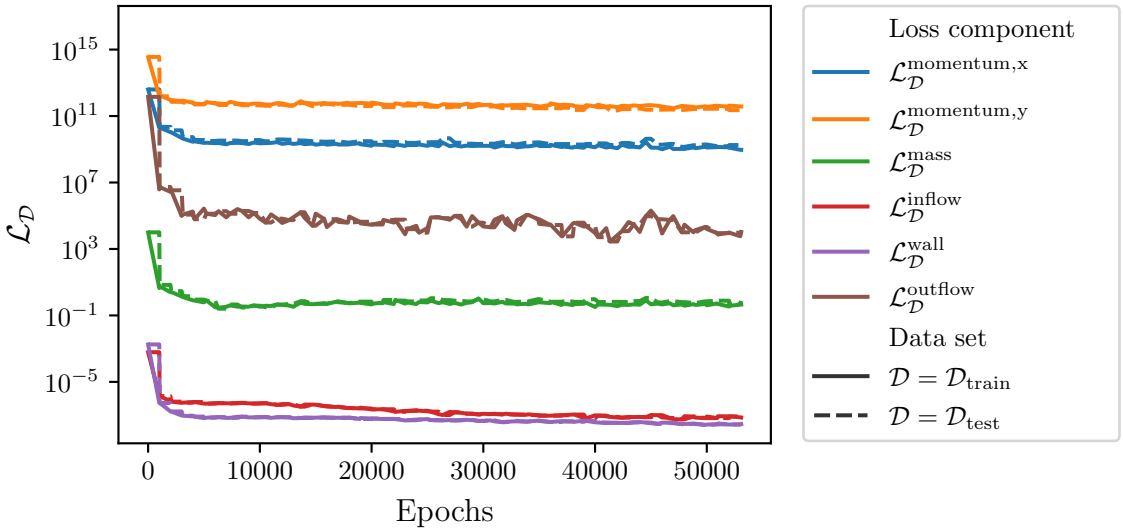
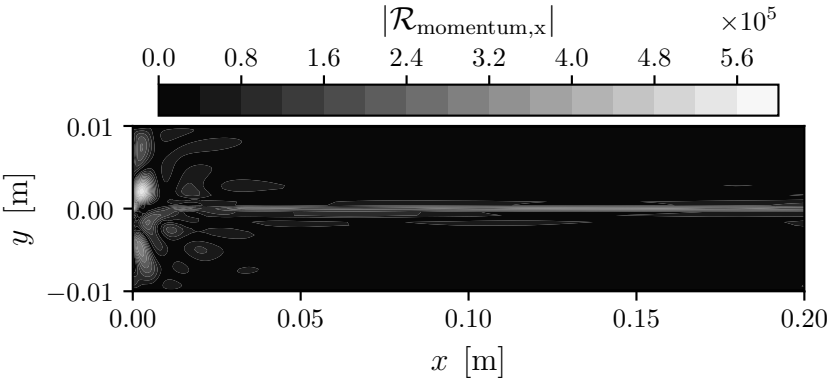


Figure 4.7.: Convergence of the unweighted loss function components with respect to the number of trained epochs. After a rapid initial decay, the loss curves quickly flatten out, indicating convergence.

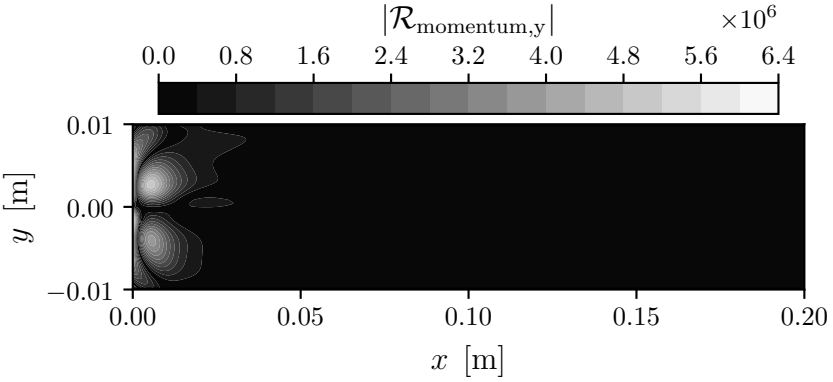
boundary conditions at the inflow and on wall now also feature a reduction in contrast to the previously presented convergence plot. An explanation for the high loss values can be found in the distribution of the strong PDE residuals of the trained PINN model inside the domain. Their absolute values are depicted in Figure 4.8. The PDE residuals are low in the majority of the domain, while the regions with large residuals are confined to the vicinity of the inflow. This can be attributed to the prescribed BC (cf. Equation (4.2a)) which corresponds to the velocity profile of a Newtonian fluid, i.e., it does not fulfill the conservation equations in the case of shear-thinning behavior. As mentioned in Section 3.1.2, we employ a mean-squared error loss in our test cases. This function heavily penalizes outliers which explains the huge values of the loss components observed in Figure 4.7. We are sure that the absolute magnitude of the strong PDE residuals can be significantly reduced if a different BC is applied that already fulfills the conservation equations, e.g., by prescribing a velocity profile obtained from a numerical high-fidelity solution. This however will not be discussed in this thesis.

A plot of the weighted loss components can be found in Figure B.2 in Appendix B and confirms the general idea of the proposed heuristic, namely to balance all loss components equally, as there the weighted losses start from and converge to similar values.

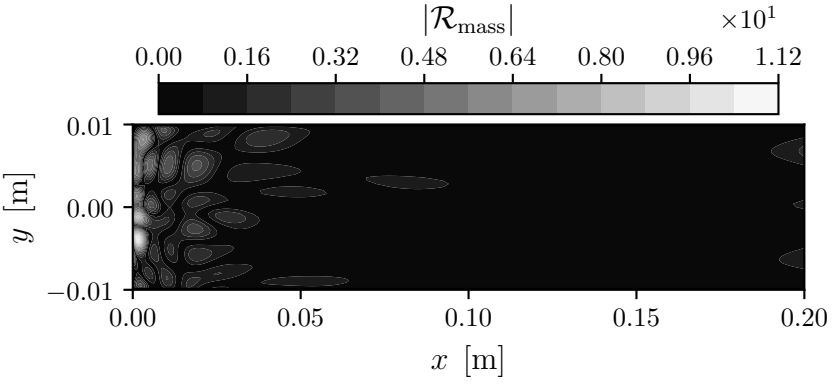
Next, we will take a look at the pointwise deviation of the PINN prediction with respect to the FEM high-fidelity solution. Surface plots of the absolute error are shown in Figure 4.9. Generally, these errors are very small across most parts of the domain. Similar to the plots of the PDE residuals, the regions with the highest errors are found close to the inflow region. These results confirm the observations of the previous section: Although the (unweighted) residuals are very high, the error is rather low, i.e., high loss function value does not necessarily result in a poor PINN prediction and vice versa. This finding requires a more careful examination, especially with respect to its transferability to different application cases.



(a) x-momentum residual.

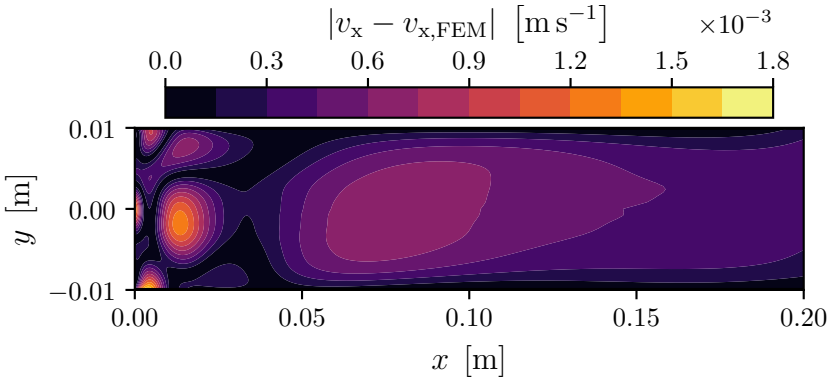


(b) y-momentum residual.

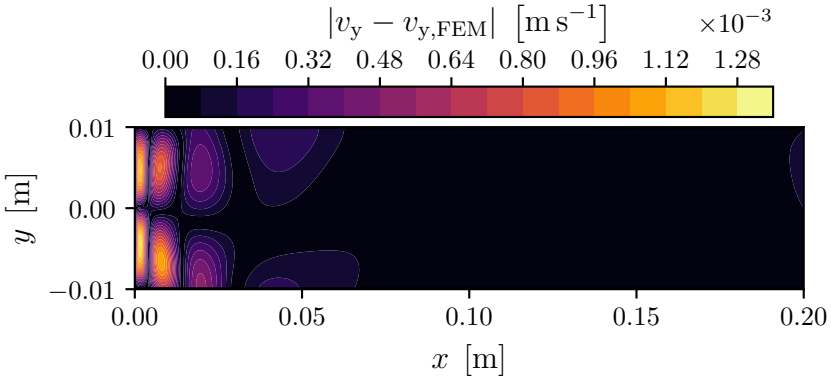


(c) Mass residual.

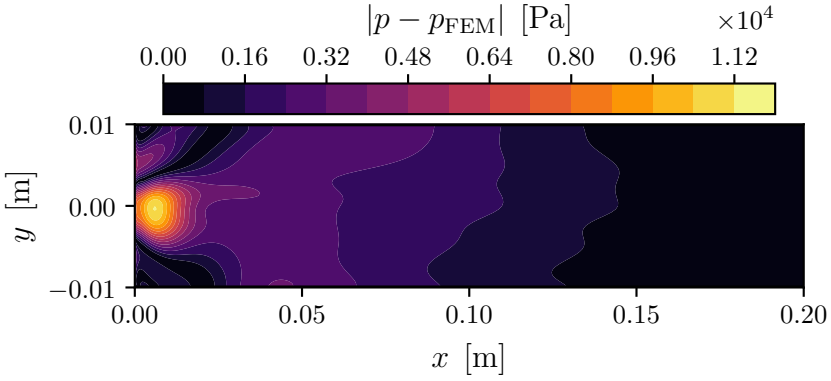
Figure 4.8.: Absolute value of the strong PDE residuals for the trained model presented in Section 4.2.2.



(a) x-component of velocity field.



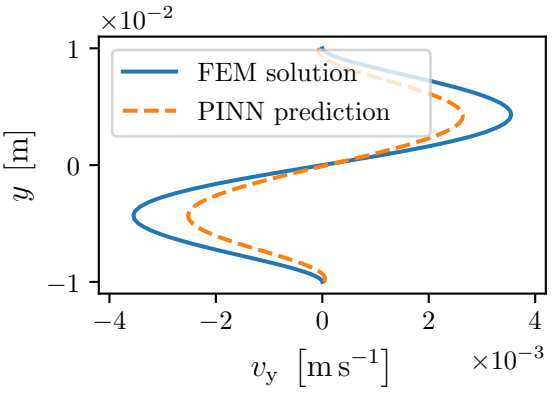
(b) y-component of velocity field.



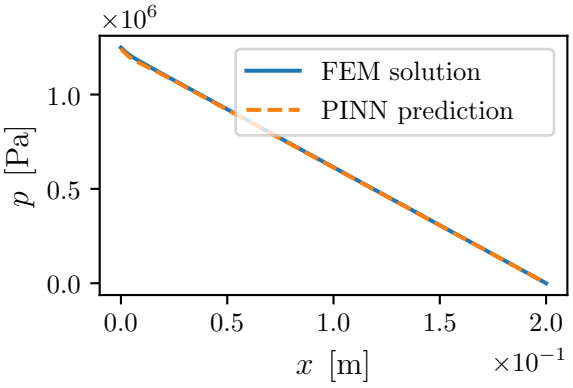
(c) Pressure field.

Figure 4.9.: Pointwise absolute error between the PINN prediction and a numerical high-fidelity solution obtained from an FEM simulation.

To further assess the quality of the PINN prediction, we compare velocity and pressure profiles of the PINN to the FEM solution at distinct locations in the domain. The results are depicted in Figures 4.10 and 4.11.



(a) Velocity profiles (y-component) at $x = 0.0025\,\mathrm{m}$.



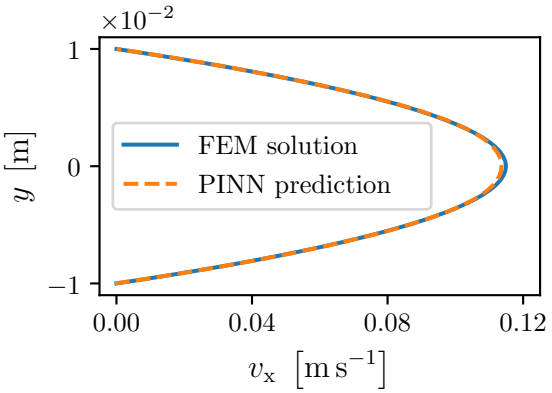
(b) Pressure characteristic along $y = 0.0\,\mathrm{m}$.

Figure 4.10.: Comparison of y-velocity and pressure profiles at different locations in the computational domain.

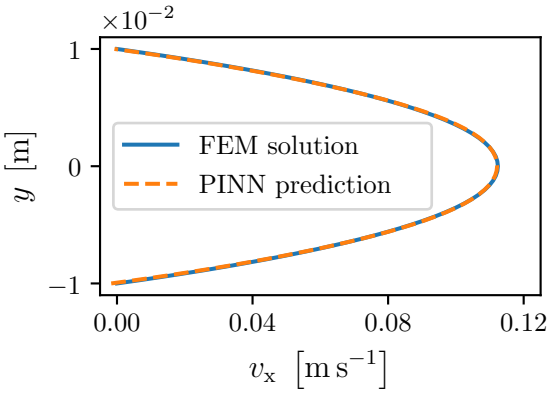
Generally, we can constitute a very good agreement of the PINN prediction and the high-fidelity solution. The most obvious difference is found in the y-velocity profile (cf. Figure 4.10a) which however is very difficult to predict as the absolute values are rather small while it is simultaneously only an artifact caused by the inappropriate inflow BC. Still, the profile predicted by the PINN qualitatively agrees with the FEM solution. The pressure characteristics along the flow channel (cf. Figure 4.10b) also shows very good agreement, especially towards the end of the flow channel. Similar to the y-velocity profile, it is however under-predicted in the transition area close to the inflow. The x-velocity profiles depicted in Figure 4.11 very well capture the shear-thinning behavior of the fluid: Starting from the prescribed parabolic velocity profile at the inflow, corresponding to a Newtonian fluid (cf. Figure 4.11a), the peak velocity slowly decreases, while the profile becomes broader (cf. Figures 4.11b and 4.11c) until the flow is fully developed (cf. Figure 4.11d) and does not undergo any further significant changes towards the end of the channel (cf. Figures 4.11e and 4.11f). The evaluation directly at the inflow $x = 0.0\,\mathrm{m}$ (Figure 4.11a) shows the largest discrepancy between the PINN-based ROM and the high-fidelity FEM solution: In FEM, the Dirichlet inflow BC is met exactly, while it is only weakly imposed for the PINN and consequently also needs to be learned, explaining the deviation.

Surface plots of the PINN prediction for the different solution fields can be found in Figure B.3.

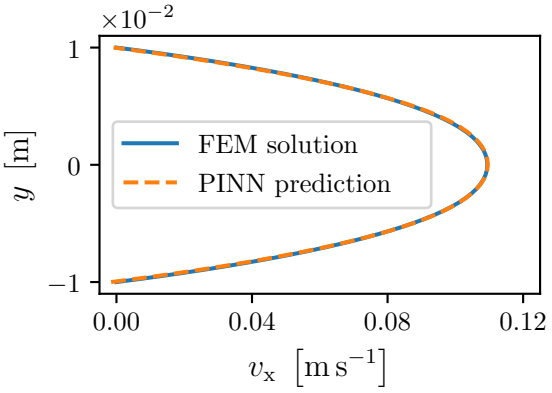
With that, we conclude the application of PINNs to the process of profile extrusion for the rest of this thesis. We will revisit the application of profile extrusion in Chapter 5, but for now continue with the methodology and investigate the prediction of flow fields in bioreactors.



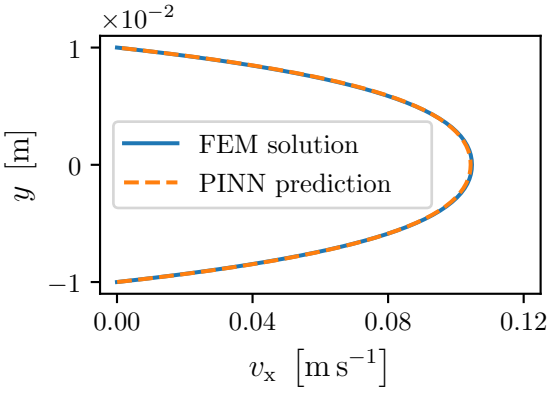
(a) Velocity profiles (x-component) at $x = 0.0\,\mathrm{m}$.



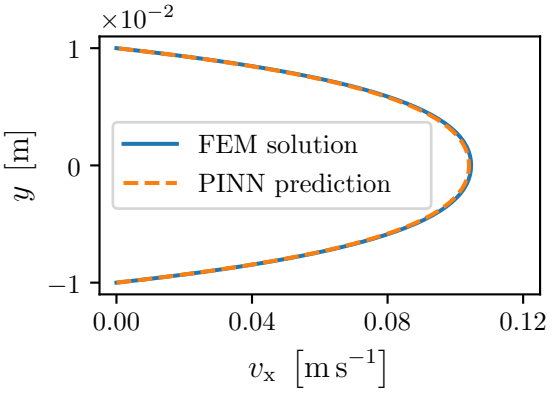
(b) Velocity profiles (x-component) at $x = 0.0025\,\mathrm{m}$.



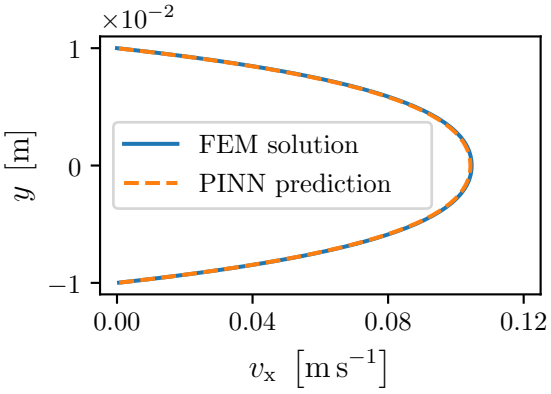
(c) Velocity profiles (x-component) at $x = 0.005\,\mathrm{m}$.



(d) Velocity profiles (x-component) at $x = 0.025\,\mathrm{m}$.



(e) Velocity profiles (x-component) at $x = 0.1\,\mathrm{m}$.



(f) Velocity profiles (x-component) at $x = 0.2\,\mathrm{m}$.

Figure 4.11.: Comparison of the PINN velocity profiles (x-component) and the FEM solution at different locations in the computational domain.

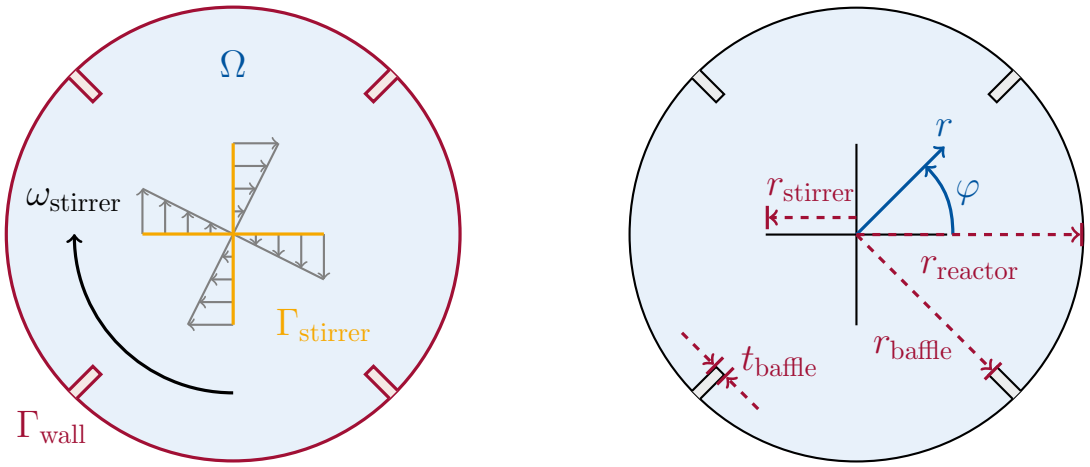
4.3. Application to Bioreactors

In this section, we employ PINNs for constructing reduced simulation models for a simplified bioreactor test case. In the following, we will first introduce the test case and then show the results obtained for two different ROMs. In Section 4.3.1, we train the first model to predict a flow field that has been parameterized in the angular velocity of the bioreactor's stirrer. Since the resulting predictions exhibit a significant error when compared to numerical high-fidelity results, we train the second model for a single stirrer frequency and focus on possibilities for increasing the prediction accuracy in Section 4.3.2.

We begin our investigation of PINNs for bioreactors by making some assumptions to simplify the governing equations. As mentioned in Section 1.2, modeling the flow in bioreactors is very challenging from a numerical point of view. The same still holds true if we want to learn a numerical solution using PINNs instead of computing it using conventional approaches, especially if we train the PINN without data from a high-fidelity solution. To this end, we assume a stationary, incompressible, Newtonian, single-phase flow. Further, we do not incorporate a turbulence model. With that, the incompressible Navier-Stokes equations introduced in Equation (2.2) can be simplified, resulting in the following residuals

$$\mathcal{R}_{\text{PDE}}\left[\boldsymbol{u}\right] = \begin{pmatrix} \boldsymbol{\nabla} \cdot \boldsymbol{v} \\ \rho \left(\boldsymbol{v} \cdot \boldsymbol{\nabla}\right) \boldsymbol{v} - \boldsymbol{\nabla} \cdot \boldsymbol{\sigma} \end{pmatrix} . \tag{4.7}$$

Moreover, we will only consider the flow in a two-dimensional cross section through the stirrer plane. The resulting computational domain is depicted in Figure 4.12a. In order



(a) Schematic of the bioreactor test case considered in this thesis. The velocity profiles resulting from BC Equation (4.9b) is visualized on top of the stirrer blades.

(b) Dimensions of the bioreactor geometry. The values of the indicated quantities are reported in Table 4.6.

Figure 4.12.: Overview of the model for bioreactor test cases. **(a)** shows the fixed, i.e., non-rotating, computational domain and its boundaries. **(b)** illustrates the dimensions of the bioreactor geometry.

to avoid dealing with moving-domain methods due to the rotating stirrer, our model is

formulated such that we consider the flow field at a specific instant in time, where the stirrer blades are exactly in the position visualized in Figure 4.12a. No force term is applied to the momentum conservation equation, so the flow is expected to be driven by the chosen BCs. The BCs consist of two parts, namely the reactor wall Γ_{wall} and the stirrer blades Γ_{stirrer}, i.e., $\Gamma = \Gamma_{\text{stirrer}} \cup \Gamma_{\text{wall}}$. On both Γ_{stirrer} and Γ_{wall}, the fluid will stick to the boundary, moving with the same velocity (corresponding to a no-slip BC):

$$\boldsymbol{v} = \boldsymbol{v}_{\text{wall}} \qquad \text{on } \Gamma_{\text{wall}} \,, \tag{4.8a}$$

$$\boldsymbol{v} = \boldsymbol{v}_{\text{stirrer}} \qquad \text{on } \Gamma_{\text{stirrer}} \,. \tag{4.8b}$$

While the velocity of the wall is zero, i.e., $\boldsymbol{v}_{\text{wall}} = \boldsymbol{0}$, the stirrer is supposed to move in clock-wise direction with a constant angular velocity of $\boldsymbol{\omega}_{\text{stirrer}}$. According to the physical law of uniform circular motion, the velocity of a single stirrer blade can be calculated as

$$\boldsymbol{v}_{\text{stirrer}} = \boldsymbol{\omega}_{\text{stirrer}} \times \boldsymbol{r} \,.$$

Here, $\boldsymbol{r}$ denotes a radial vector, pointing away from the coordinate system's origin. In the case of a two-dimensional cartesian coordinate system aligned with the stirrer axes, this simplifies to

$$\boldsymbol{v}_{\text{stirrer}} = \begin{pmatrix} \omega_{\text{stirrer}} y \\ -\omega_{\text{stirrer}} x \end{pmatrix} \,,$$

where we have assumed a clock-wise rotation of the impeller and x and y denote the spatial coordinates. This allows to rewrite the BCs Equation (4.8) into the following residual form

$$\mathcal{R}_{\text{wall}}\left[\boldsymbol{u}\right] = \begin{pmatrix} v_x \\ v_y \end{pmatrix} \,, \tag{4.9a}$$

$$\mathcal{R}_{\text{stirrer}}\left[\boldsymbol{u}\right] = \begin{pmatrix} v_x - \omega_{\text{stirrer}} y \\ v_y + \omega_{\text{stirrer}} x \end{pmatrix} \,. \tag{4.9b}$$

The BC on Γ_{stirrer} is supposed to yield a linear velocity profile on each stirrer blade as shown in Figure 4.12a.

An important non-dimensional quantity for determining the flow regime inside bioreactors is the Reynolds number Re. Here, we employ the definition of the Reynolds number according to Rosseburg et al. [148] as

$$\text{Re} = \frac{\rho \omega_{\text{stirrer}} (2 r_{\text{stirrer}})^2}{\mu} \,. \tag{4.10}$$

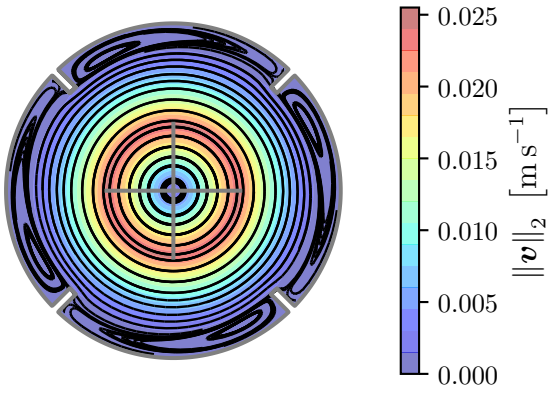
We will assume the density and viscosity to be constant material parameters and only vary the stirrer velocity in Section 4.3.1 to modify the Reynolds number. The material properties of the fluid are reported in Table 4.7. Unless mentioned otherwise, we consider an operating point at $\text{Re} = 4000$, corresponding to a stirrer velocity of $\omega_{\text{stirrer}} = 0.625\,\text{s}^{-1}$.

Table 4.6.: Dimensions of the bioreactor geometry depicted in Figure 4.12b.

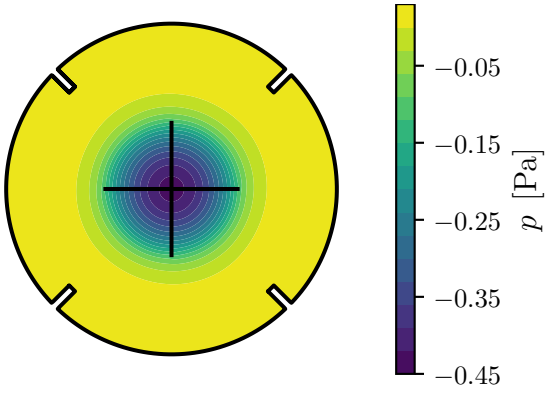
Parameter	Value	Unit
r_{stirrer}	0.040	m
r_{reactor}	0.100	m
r_{baffle}	0.085	m
t_{baffle}	0.005	m

Table 4.7.: Constant material properties used for the simulations and PINN training.

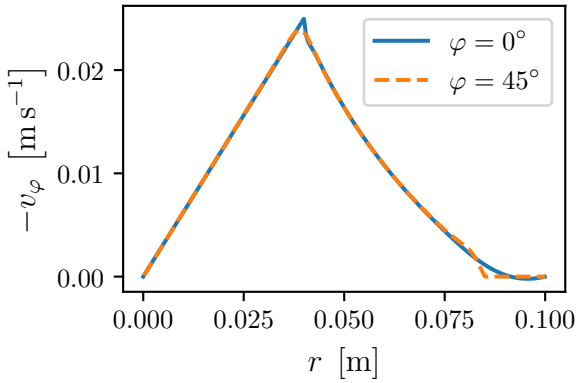
Parameter	Value	Unit
ρ	1000	$\mathrm{kg\,m^{-3}}$
μ	0.001	$\mathrm{kg\,m^{-1}\,s^{-1}}$



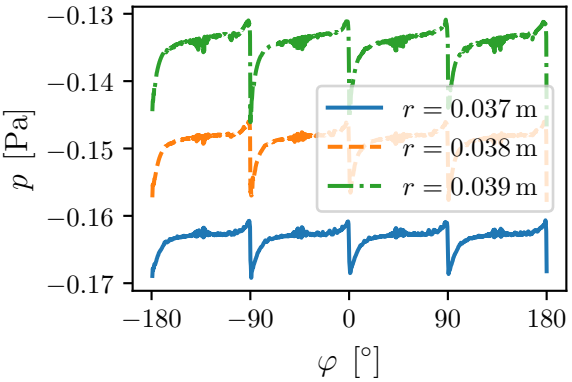
(a) Velocity field and streamlines. Vortices form behind the baffles in the direction of flow.



(b) Pressure field.



(c) Profiles of the circumferential velocity component along two selected lines through the geometry.



(d) Profiles of the pressure in circumferential direction at three different radii.

Figure 4.13.: Numerical high-fidelity solution obtained from an FEM simulation at Re = 4000.

In Figure 4.13, we show the results obtained from computing a high-fidelity FEM solution to the presented model problem. The simulations were performed on a fine computational mesh Ω^h consisting of $N_{\text{n}} = 292692$ nodes and $N_{\text{el}} = 290554$ triangular elements. Due to the high mesh resolution, we were able to capture the vortex formation behind the baffles in the numerical velocity field (cf. Figure 4.13a). We expect the PINN to have difficulties resolving these vortices as it is only repeatedly provided with a comparatively small number of randomly drawn points. However, these features are essential for the mixing capabilities

of the reactor. For the sake of completeness, the pressure field is displayed in Figure 4.13b. Yet, in the following sections, we will restrict the scope of this work to the velocity field. The pressure field features jumps across the stirrer blades, which are captured in the FEM simulation by introducing duplicated nodes on the internal stirrer boundary (cf. Figure 4.13d). We expect the PINN to have difficulties in reproducing this discontinuous behavior [99]. In Figure 4.13c, profiles of the circumferential velocity component v_φ along two selected radial lines through the geometry are depicted. These give us insights into the solution's properties, which will become important later in Section 4.3.2.

In the following sections, we discuss different approaches for constructing PINN-based ROMs.

4.3.1. Predicting Flow Fields For Varying Stirrer Velocities

In this section, we investigate PINNs to predict solutions for the simplified bioreactor model presented in the beginning of this chapter (Equations (4.7) and (4.9)). More precisely, we are interested in predicting solutions to the parameterized PDE problem for varying stirrer velocities ω_{stirrer}, respectively Reynolds numbers Re. Having such a ROM enables a rapid evaluation at multiple different parameter values (once the training is completed, the evaluation only consists of simple floating point operations) as opposed to classical CFD simulations that have to be rerun for every new parameter value. We summarize the most important results in the following. For a more detailed discussion of the presented findings, we refer to [97].

For now, our aim is to have a model capable of predicting the flow field for Reynolds numbers in the interval $\text{Re} \in [1000, 10000]$. According to Equation (4.10), this corresponds to stirrer velocities in the interval $\omega_{\text{stirrer}} \in [0.156\,25\,\text{s}^{-1}, 1.5625\,\text{s}^{-1}]$. The input space of the PINN now consists of the cartesian product of the spatial domain and the additional (one-dimensional) parameter space, i.e., $\mathcal{X} = \Omega \times \Lambda$. For the parameter space, we assume that the stirrer velocities are uniformly distributed (with distribution $\mathcal{U}$) and consequently generate random samples during training, i.e., $\omega_{\text{stirrer}} \sim \mathcal{U}_{[0.15625,1.5625]}$.

In order to find a suitable network architecture, a Hyper-parameter Optimization (HPO) was performed and manually tuned afterward. The resulting architectural parameters are reported in Table 4.8, while the hyper-parameters related to the training are given in Table 4.9. Similar to the description in the previous section, the inputs to the network are normalized and the outputs rescaled (cf. Equation (4.3)) so that the network itself only sees normalized values. For more details on the employed normalization, we refer to [97, Section 5.1].

Table 4.8.: Final NN architecture after manually fine tuning the results obtained from the HPO.

Parameter	Value
N_{hl}	2
N_{nhl}	40
σ	tanh

Figure 4.14 depicts the evolution of the loss function components over the training epochs. All components converge to a value on the order of 10^{-5} while we shall note that

Table 4.9.: Hyper-parameters related to the training of the NN.

Parameter	Value
Optimizer	L-BFGS
N_{epochs}	10^5
$\|\mathcal{B}\|$	256
Resampling rate	10^3
$\boldsymbol{\alpha}_{\text{PDE}} = (\alpha_{\text{momentum,x}}, \alpha_{\text{momentum,y}}, \alpha_{\text{mass}})^T$	$(1, 1, 1)^T$
$\boldsymbol{\alpha}_{\text{BC}} = (\alpha_{\text{stirrer}}, \alpha_{\text{wall}})^T$	$(1, 100)^T$

the loss of the mass conservation strongly oscillates around this value. This indicates a higher sensitivity with respect to the randomly-drawn mini-batches than the other loss components. At the end of the training, we observe a single sharp peak in the loss components related to both BCs as well as the mass residual. This can be interpreted as an overshoot: Since all loss components are already converged, further optimizing the total loss will not result in any additional improvements.

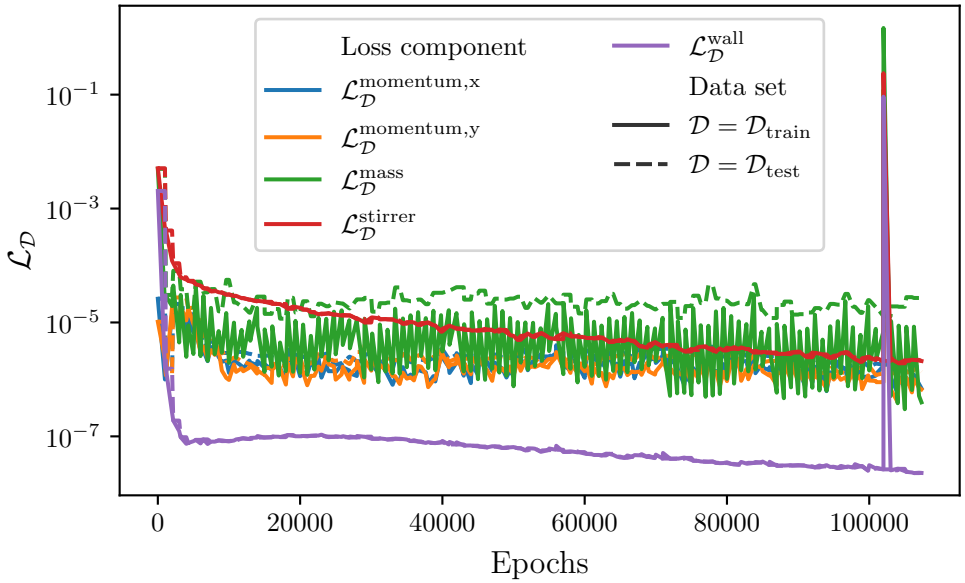


Figure 4.14.: Convergence of the loss function components with respect to the number of trained epochs. All components converge to a reasonably low value.

For testing the trained model's performance, we have performed numerical high-fidelity simulations for distinct stirrer velocities (resp. Reynolds numbers). Afterward, we evaluated the trained PINN model for each stirrer velocity at the nodes of the computational mesh. The pointwise error between the velocity predictions of the trained PINN and the results obtained from the FEM has been computed and normalized with the theoretical maximum velocity at the stirrer tip $v_{\text{tip}} = \omega_{\text{stirrer}} r_{\text{stirrer}}$. In Figure 4.15, we show a comparison of the resulting error distributions for each value of the investigated Reynolds number.

At first glance, the computed errors are generally large, with errors amounting up to roughly 16.2 %. This shows that despite the seemingly good convergence, the solution

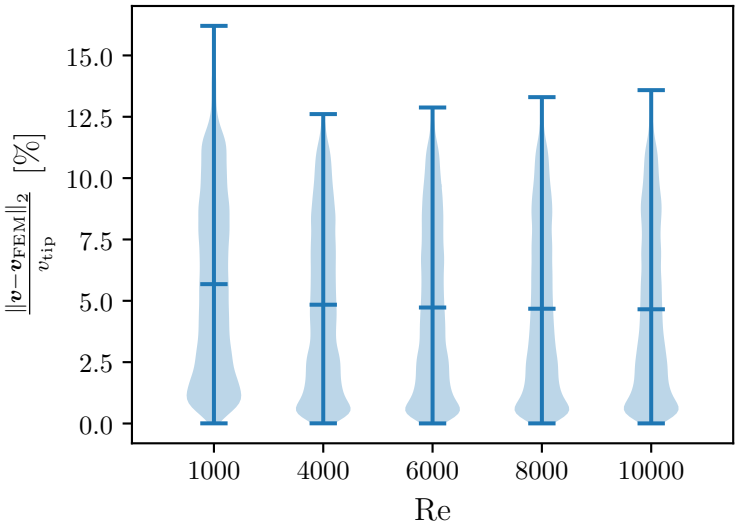


Figure 4.15.: Violin plots indicating the distribution of the pointwise normalized absolute errors with respect to a numerical high-fidelity simulation.

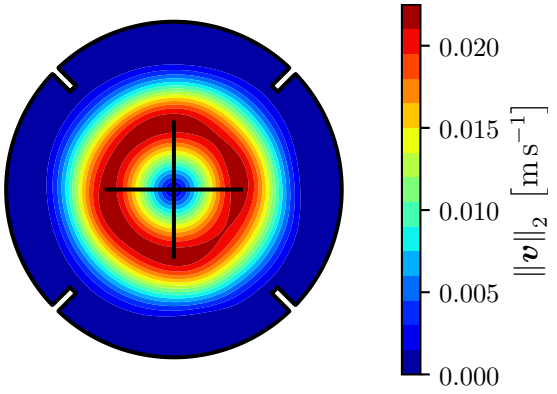
found by the PINN still exhibits systematic errors with respect to the high-fidelity solution. The general tendency of the error distributions, however, corresponds to the expectations: With increasing Reynolds number, the maximum errors increase as well. This can be explained with the inherent nonlinearity of the governing PDEs that becomes more pronounced at higher Reynolds numbers, manifesting in more complex flow patterns behind the baffles. Interestingly, the highest errors can be observed in the case of the smallest Reynolds number (Re = 1000). We explain this behavior with the small (absolute) values of the velocity on the order of $\mathcal{O}(10^{-3})$ in the low Reynolds number regime: As we have already seen in the previous section (cf.Section 4.2.1), such small solution values can result in small PDE residuals. Consequently, the NN's learning process concentrates on reducing different loss components (e.g., the ones related to the BCs). While this might reduce the error on the boundaries, the errors inside the computational domain remain. Although the maximum errors of this model are rather large, the error distributions (shaded areas in light blue) show that at the majority of the tested points the errors are significantly lower (indicated by the thickness of the columns, where an increased thickness corresponds to a larger number of points). As a result, the mean errors amount to roughly 5 %. For better reference, the minimum, mean, and maximum errors for the different cases are reported in Table 4.10.

While Figure 4.15 gives us a general idea about the magnitude of the errors and consequently the quality of the approximation, it does not give information about the spatial distribution of the error inside the computational domain. To investigate this, we consider the case of a fixed Reynolds number, i.e., Re = 4000, and examine the PINNs's prediction as well as the pointwise error as depicted in Figure 4.16.

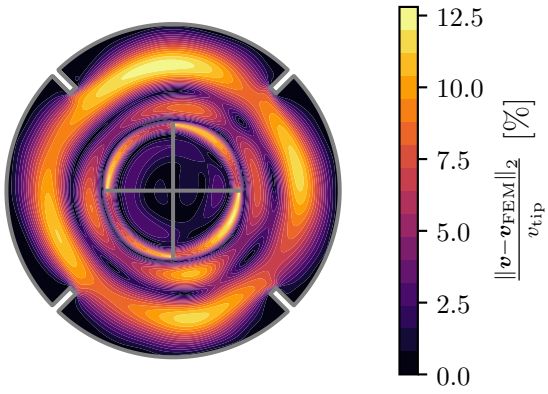
Starting with the PINN prediction shown in Figure 4.16a, one can observe that the flow field is asymmetric, although the geometry is perfectly symmetric. The reason for this lies in the inherent stochasticity of the training process: After 1000 epochs of training, a new batch of points is randomly generated inside the domain. Yet, there is no guarantee

Table 4.10.: Comparison of the minimum, mean and maximum normalized absolute errors, indicated by the horizontal bars in Figure 4.15.

Re	$\frac{\lVert \boldsymbol{v}-\boldsymbol{v}_{\mathrm{FEM}} \rVert_2}{v_{\mathrm{tip}}}$ [%]		
	min.	mean	max.
1000	$6.46 \cdot 10^{-3}$	5.68	16.20
4000	$7.08 \cdot 10^{-3}$	4.84	12.61
6000	$3.37 \cdot 10^{-3}$	4.73	12.88
8000	$4.72 \cdot 10^{-3}$	4.68	13.30
10000	$4.57 \cdot 10^{-3}$	4.66	13.59



(a) PINN prediction of the velocity field at Re = 4000.



(b) Pointwise relative deviation from the high-fidelity FEM solution with respect to the tip velocity at Re = 4000.

Figure 4.16.: Evaluation of the trained PINN model for a fixed Reynolds number of Re = 4000. **(a)** depicts the PINN's prediction evaluated at the nodes of the computational mesh, while **(b)** shows the pointwise absolute error compared to the FEM solution.

that these points are distributed symmetrically — in fact, it is more likely that they are not. Consequently, the PINN's prediction can be expected to differ depending on the distribution of points seen during training, explaining the asymmetric velocity field. In terms of BCs, the no-slip condition at the reactor walls as well as the movement of the stirrer seem to be captured well. However, the flow is only in rotation in the vicinity of the stirrer and rapidly decays to nearly zero away from the impeller tips. The pointwise relative deviation between the prediction and the high-fidelity solution with respect to the tip velocity visualized in Figure 4.16b confirms this observation: While the PINN captures the BCs and the flow around the stirrer well, the highest errors occur in the transition region behind the stirrer and towards the baffles. Furthermore, the PINN is not able to capture the vortices that form in the regions behind the baffles. Together with the underestimation of the velocity magnitude, this results in comparatively large deviations.

Based on the results presented above, we can constitute that PINNs are able to capture general structures in the flow field, e.g., the rotating flow, but — at least if employed naively — fail to predict more refined subtleties in the solution, e.g., the vortex formation behind the baffles. This behavior is not restricted to a single Reynolds number, but a general

problem so far, as the similar error distributions for varying Reynolds numbers indicate. Consequently, we need to improve the prediction of the model at a single Reynolds number before trying to parameterize with respect to the stirrer velocity. Here, the observation that the PINN learns an asymmetric prediction already gives rise to one idea how to further improve the model quality, i.e., by accounting for the problem's symmetry. This should further help to reduce the necessary computational costs. Moreover, one could think about reformulating the problem in cylindrical coordinates, as these seem more natural than cartesian coordinates given the circular geometry. The next section will now focus on improving the prediction accuracy for a single Reynolds number, leveraging some of the aforementioned ideas.

4.3.2. Improving the Prediction Quality with a Domain Decomposition Approach

After obtaining the results presented in the previous section, the research focus shifted from creating a PINN-based ROM for multi-query scenarios towards creating a ROM with sufficient approximation accuracy in the first place. Therefore, we once again consider the flow field inside the bioreactor at the fixed Reynolds number of $\mathrm{Re} = 4000$. Many different approaches for improving the predictions were investigated, including a reformulation of the problem in cylindrical coordinates, the additional incorporation of high-fidelity data samples, strong BC imposition, and domain decomposition. In the following, we only present the final model, which yielded the best results. For an in-depth report of all performed experiments, we refer to [35].

When inspecting the numerical high-fidelity solution depicted in Figure 4.13, we observe that both the velocity and pressure do not change significantly in circumferential direction in an inner region around the stirrer, denoted by Ω_{inner}. Moreover, the magnitude of the circumferential velocity dominates the radial velocity, i.e.,

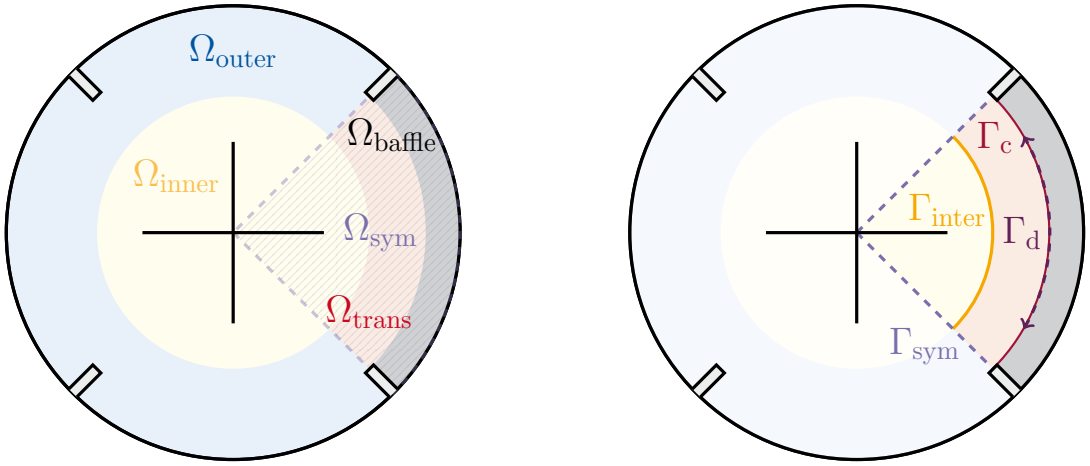
$$\frac{\partial(\,\cdot\,)}{\partial\varphi} = 0 \qquad \text{in } \Omega_{\text{inner}}\,, \tag{4.11a}$$

$$v_\varphi \gg v_r \qquad \text{in } \Omega_{\text{inner}}\,. \tag{4.11b}$$

These findings allow us to reduce the governing equations Equation (4.7) to a system of Ordinary Differential Equations (ODEs) when formulated in cylindrical coordinates, where the only unknowns are the circumferential velocity v_φ and the pressure p (cf [35, Section 3.2.3]). This motivates the idea of employing a domain decomposition approach, where multiple PINNs are trained simultaneously to predict the solution in each subdomain. At the domain interfaces, different coupling conditions can be defined, e.g., based on fluxes, as in the case of Conservative Physics-Informed Neural Networks (cPINNs) [67], or based on more general terms, as in the case of Extended Physics-Informed Neural Networks (XPINNs) [64]. An algorithm for the parallel training of multiple PINNs has been proposed by Shukla, Jagtap, and Karniadakis [161]. We will follow this approach for constructing a PINN model with improved prediction quality.

Aside from the already identified two subdomains Ω_{inner} and $\Omega_{\text{outer}} \coloneqq \Omega \setminus \Omega_{\text{inner}}$, we will add an additional subdivision in the outer region. By analyzing the velocity profiles of the numerical high-fidelity solution depicted in Figure 4.13c, we observe a sharp kink in the circumferential velocity located at the position of the baffle, where it reaches zero due

to the no-slip condition on the wall. In order to capture this behavior, we introduce a third subdomain that separates the outer region Ω_{outer} into a transition region Ω_{trans} and the region exactly between the baffles Ω_{baffle}, i.e., $\Omega_{\text{outer}} = \Omega_{\text{trans}} \cup \Omega_{\text{baffle}}$. By subdividing the outer domain in that way, we obtain a new interface. There, we prescribe coupling conditions for the unknown quantities. Precisely, we can design the conditions such that they allow for the behavior observed in Figure 4.13c. The complete domain decomposition is visualized in Figure 4.17. In the following, we will discuss each part of the model in detail, summarizing key features of each subdomain.



(a) Overview of the different subdomains. Only a quarter of the original domain Ω_{sym} is considered due to the problem's symmetry. In Ω_{inner}, the governing PDEs are simplified to ODEs. In Ω_{trans} and Ω_{baffle}, the full Navier-Stokes equations are learned.

(b) Interfaces connecting the different subdomains in Ω_{sym}. Special coupling conditions relating quantities of interests and their derivatives are imposed on the interfaces Γ_{inter} and Γ_{d}, whereas on Γ_{c} only the continuity of the circumferential velocity is enforced to allow for the kink observed in the high-fidelity solution.

Figure 4.17.: Decomposition of the computational domain Ω into three subdomains. **(a)** visualizes the different domains, while **(b)** depicts all involved interfaces.

We begin our discussion with general remarks on the model: In all subdomains, cylindrical coordinates are used to describe the flow. This means that instead of solving for the velocities in x- and y-direction v_x, v_y as in the cartesian case presented in Section 4.3.1, we now solve for the circumferential and radial velocities v_φ, v_r. Moreover, we consider the symmetry of the problem and only train a PINN to predict the solution in a quarter of the computational domain Ω_{sym}. The location of the symmetric domain Ω_{sym} has been chosen such that its boundaries are aligned with the reactor baffles, as shown in Figure 4.17.

Next, we focus on the inner domain around the stirrer Ω_{inner}. In [35], different locations for the boundary of the inner domain were investigated. The best performing model was found if the interface was placed at $r = 0.08\,\text{m}$. Within this region, we train a PINN to solve the simplified ODE system for the circumferential velocity component and the

unknown pressure, i.e., the unknown solution vector reduces to $\boldsymbol{u} = (v_\varphi, p)^T$, which is determined by the following residual

$$\mathcal{R}_{\text{PDE}}^{\text{inner}}[\boldsymbol{u}] = \begin{pmatrix} \frac{\mathrm{d}^2 v_\varphi}{\mathrm{d}r^2} + \frac{1}{r}\frac{\mathrm{d}v_\varphi}{\mathrm{d}r} - \frac{v_\varphi}{r^2} \\ -\frac{v_\varphi}{r^2} + \frac{1}{\rho}\frac{\mathrm{d}p}{\mathrm{d}r} \end{pmatrix} .$$

The radial velocity component v_r is assumed to be zero. The BCs on the stirrer have been strongly imposed following the approach mentioned in Section 3.2.4.

In the outer part of the symmetric domain Ω_{outer}, the simplifications assumed for Ω_{inner} in Equation (4.11) are no longer valid and a separate PINN is trained to minimize the full Navier-Stokes residual, as reported in Equation (4.7). While the predictions of the radial velocity component and the pressure are shared for Ω_{baffle} and Ω_{trans} (and thus will be denoted by v_r^{outer} and p^{outer}), different circumferential velocity components are predicted for each subdomain, denoted by v_φ^{trans} and $v_\varphi^{\text{baffle}}$, to improve the accuracy of the PINN prediction with respect to the numerical high-fidelity solution. The no-slip BC on the reactor wall Γ_{wall} is still weakly imposed as in the model presented in Section 4.3.1.

After briefly discussing all involved subdomains, we will now introduce the coupling conditions required on all involved interfaces. Since we only consider a subset Ω_{sym} of the complete computational domain Ω, we need to account for the symmetry with special BCs on Γ_{sym}, which essentially read

$$\boldsymbol{u}|_{\boldsymbol{x}^{(-)}} = \boldsymbol{u}|_{\boldsymbol{x}^{(+)}} \qquad \text{on } \Gamma_{\text{sym}} .$$

Here, $\boldsymbol{x}^{(-)}$ and $\boldsymbol{x}^{(+)}$ denote the left and right limiting values as the spatial coordinates approach the symmetry interface Γ_{sym}.

On Γ_{inter}, the following conditions have to hold:

$$v_\varphi^{\text{inner}} = v_\varphi^{\text{trans}} \qquad \text{on } \Gamma_{\text{inter}} , \tag{4.12a}$$

$$0 = v_r^{\text{outer}} \qquad \text{on } \Gamma_{\text{inter}} , \tag{4.12b}$$

$$p^{\text{inner}} = p^{\text{outer}} \qquad \text{on } \Gamma_{\text{inter}} , \tag{4.12c}$$

$$\frac{\mathrm{d}v_\varphi^{\text{inner}}}{\mathrm{d}r} = \frac{\partial v_\varphi^{\text{trans}}}{\partial r} \qquad \text{on } \Gamma_{\text{inter}} , \tag{4.12d}$$

which ensure the continuity of the networks' predictions across the different subdomains as well as the continuity of the derivatives in radial direction to avoid kinks. The continuity of derivatives in circumferential direction is included in Equations (4.12a) to (4.12c) due to the alignment of Γ_{inter} with the circumferential direction.

The interface between Ω_{trans} and Ω_{baffle} has been further split up into two interfaces Γ_{c} and Γ_{d}, on which only conditions for the circumferential velocity need to be described, as the other solution components are shared between both subdomains. Precisely, the continuity of the circumferential velocity is enforced on Γ_{c}, while the continuity of the derivative is prescribed on $\Gamma_{\text{d}} \subset \Gamma_{\text{c}}$, that is

$$v_\varphi^{\text{trans}} = v_\varphi^{\text{baffle}} \qquad \text{on } \Gamma_{\text{c}} ,$$

$$\frac{\partial v_\varphi^{\text{trans}}}{\partial r} = \frac{\partial v_\varphi^{\text{baffle}}}{\partial r} \qquad \text{on } \Gamma_{\text{d}} .$$

After discussing the model in detail, we will now investigate the obtained improvements in terms of the resulting PINN's prediction accuracy. In Table 4.11, the architectural parameters for the employed PINNs are reported, while Table 4.12 lists most of the other hyper-parameters related to the PINN training. Once again, all involved networks feature affine transformation layers at the input and output and we refer to [35, Section 3.4] for the details.

Table 4.11.: NN architectures for predicting the solution in the different subdomains.

(a) NN architecture for predicting $\boldsymbol{u} = (v_\varphi^{\text{inner}}, p^{\text{inner}})^T$ in Ω_{inner}.

Parameter	Value
N_{hl}	2
N_{nhl}	25
σ	tanh

(b) NN architecture for predicting $\boldsymbol{u} = (v_\varphi^{\text{trans}}, v_\varphi^{\text{baffle}}, v_r^{\text{outer}}, p^{\text{outer}})^T$ in Ω_{outer}.

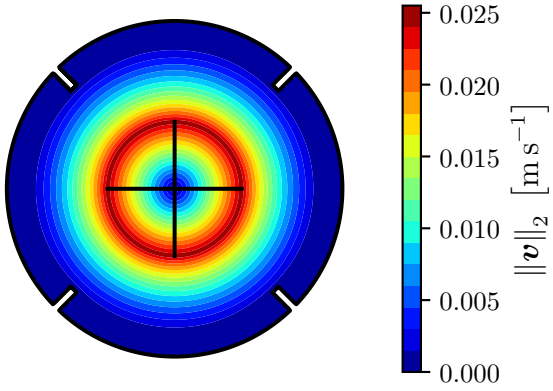
Parameter	Value
N_{hl}	2
N_{nhl}	100
σ	tanh

Table 4.12.: Hyper-parameters related to the training of the NN. The loss weights have been omitted due to the numerous loss function components and we refer to [35, Table 6] instead.

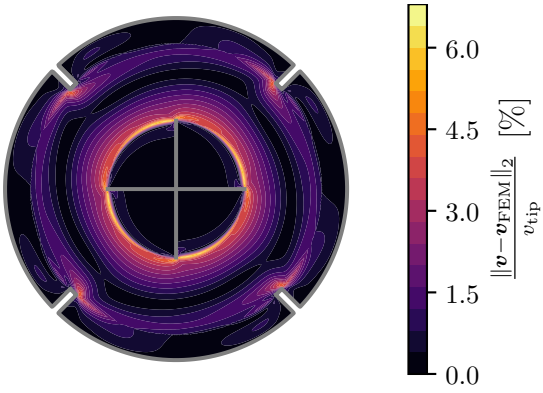
Parameter	Value
Optimizer	L-BFGS
N_{epochs}	$5 \cdot 10^4$
$\|\mathcal{B}_\Omega\|$	2048
$\|\mathcal{B}_{\Gamma_i}\|$	512
Resampling rate	10^3
$\varrho(\boldsymbol{\theta})$	$\varrho^{\ell_1}(\boldsymbol{\theta}) + \varrho^{\ell_2}(\boldsymbol{\theta})$

When looking at the PINN prediction in Figure 4.18a, we immediately observe that the model employing the domain decomposition approach was able to capture the velocity field significantly better than the vanilla model presented in the previous section. The magnitude of the velocity does not decay so fast in the direction of the wall and the shape of the vortices behind the baffles matches that of the high-fidelity solution sufficiently well. Investigating the relative deviation depicted in Figure 4.18b, we see that it has been significantly decreased in a vast majority of the computational domain. The regions of highest errors are now confined to the wake of the stirrer as well as the immediate vicinity of the baffle edges. The maximum value of the relative deviation has been reduced from roughly 12.61 % to now 6.45 %, while the average error amounts to less than 1 %. Already here we can conclude that the increased accuracy through the domain decomposition approach makes the PINN suitable for use as ROM. As a last result, we will investigate the improvement of the PINN with respect to the profile of the circumferential velocity through the domain. A comparison with the numerical FEM solution is presented in Figure 4.19.

One can see from Figure 4.19a that the prediction of the circumferential velocity is now in very good agreement with the numerical solution if we consider a straight line from the

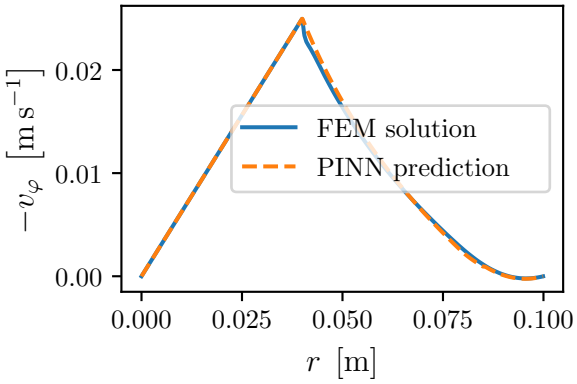


(a) PINN prediction of the velocity field at $\mathrm{Re} = 4000$.

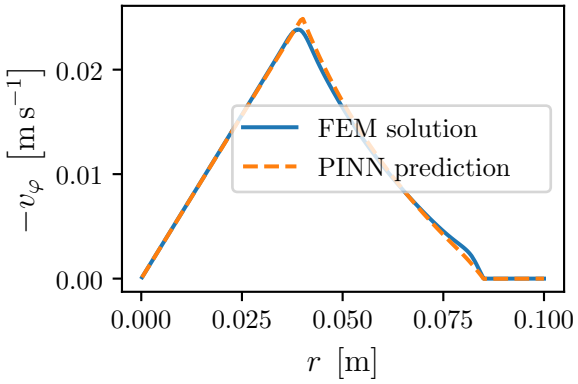


(b) Pointwise relative deviation from the high-fidelity FEM solution with respect to the tip velocity at $\mathrm{Re} = 4000$.

Figure 4.18.: Evaluation of the PINN model trained with the presented domain decomposition approach. **(a)** depicts the PINN's prediction evaluated at the nodes of the computational mesh, while **(b)** shows the pointwise absolute error compared to the FEM solution.



(a) Circumferential velocity profiles along the radial line at $\varphi = 0°$.



(b) Circumferential velocity profiles along the radial line at $\varphi = 45°$.

Figure 4.19.: Comparison of the domain decomposition PINN and the numerical high-fidelity solution with respect to the profiles of the circumferential velocity component v_φ along two selected lines through the domain. **(a)** visualizes the profiles along the radial line from the central stirrer axis to the wall, while **(b)** shows the profile along the the line from the center toward the upper right baffle.

stirrer's central axis to the wall ($\varphi = 0°$). Moreover, as evidenced by Figure 4.19b, the velocity profile now also exhibits a kink when considering a line from the stirrer's central axis towards the baffle ($\varphi = 45°$). This proves the benefit of introducing the additional separation of the outer region into Ω_{trans} and Ω_{baffle} and especially only enforcing the continuity of values on Γ_{c} instead of requiring the continuity of derivatives as well.

With these results, we conclude the chapter about PINNs as ROMs for profile extrusion and bioreactor modeling. In two different application cases we have seen that PINNs can

serve as ROMs in the little- to no-data regime, providing reasonably accurate predictions. However, this did not work straightforward for the investigated model problems. Instead, additional effort in the form of model construction and tuning was required. For the profile extrusion application case, a proper scaling of the individual loss function components provided the breakthrough. For bioreactors, an even more elaborate model involving domain decomposition turned out to be necessary for obtaining reasonable results. These findings now form the basis for further developments of the presented models to obtain even more realistic ROMs of the considered applications.

5 Reinforcement Learning for Shape Optimization

This chapter is dedicated to the investigation of Reinforcement Learning (RL) for shape optimization to provide an answer to Research Question (RQ II), which can be further substantiated concerning the following aspects:

(II) (a) How can shape optimization of profile extrusion dies be realized by the means of RL and are there obvious differences between these approaches?

(b) How do the available learning algorithms differ with respect to the convergence and reproducibility of the training process?

(c) How can the training time of the RL agents be accelerated?

The presented results are based on on the work conducted in [13, 43, 44] and closely follow the discussion in [45]. After giving a brief overview of RL-based shape optimization in the context of Computational Fluid Dynamics (CFD) we will explain how we have applied the concept of RL to our shape optimization problem. We present the developed software framework used for generating the results in Section 5.1, introducing the employed geometry parameterization, the governing equations of our model problem, as well as the interaction of the RL agents with their environment. To answer Research Questions (II a) to (II c), we investigate two test cases, which we present in more detail in Sections 5.2 and 5.3. The first test case (Section 5.2) considers a T-shaped geometry and optimizes it with respect to the mass-flow ratio between its outflows and can be seen as a proof-of-concept test case, as it allows to study the feasibility and correctness of the optimized geometries by visual inspection. The second test case (Section 5.3) is more complex and aims towards a more realistic scenario by considering the flow homogeneity at the outlet of a converging channel geometry as the design criterion.

At the latest after its breakthrough in playing Atari games [112], RL has become an established method for solving tasks where a valid solution strategy can be learned from repeated interactions, e.g., like playing computer games [170] or tabletop games [165]. Yet, the method has also gained attention in fields like robotics [73], control theory [20], and engineering design automation [37].

Recently, RL has been applied to solve problems from the field of CFD. The review by Rabault et al. [136] highlighted the developments with respect to optimal design and control tasks, where they believe the empirical strategies found by RL algorithms to be

beneficial compared to traditional approaches. Two subsequent reviews by Garnier et al. [47] and its updated version by Viquerat, Meliga, and Hachem [171] also investigated the impact of RL onto the CFD community, extending the application areas to not only numerical, but also experimental applications, considering works on heat transfer, drag reduction, microfluidics, and swimming. In the following, we focus on the advances of RL for shape optimization, paying special attention to the employed learning algorithms as well as the geometry parameterization.

A first step has already been taken in 2008 by Lampton, Niksch, and Valasek [79], who utilized a Q-learning algorithm for optimizing the shape of an airfoil with respect to lift, drag as well as momentum about the tip. The geometry of the airfoil was parameterized by four scalar quantities, which were modified by the RL agent. In contrast to the approaches presented in the following, a Q-learning algorithm is restricted to a discrete action space, i.e., the set of admissible actions needs to be finite. This is because Q-learning algorithms estimate the value of each individual action for a given state and then derive a policy from these value estimates. In their review, Viquerat, Meliga, and Hachem [171] titled approaches with discrete action spaces *incremental shape optimization*. Different RL algorithms have also been applied to perform incremental shape optimization in further publications: Yan et al. [182] employed a Deep Deterministic Policy Gradient (DDPG) algorithm to optimize the placement and geometry of a missile fin with respect to the missile's lift-to-drag ratio under additional geometric and aerodynamic constraints. The geometry was updated directly, employing a mesh-update method based on radial-basis functions for running the CFD simulations on the modified geometries. The same RL algorithm was leveraged in the work by Qin et al. [133] to optimize compressor cascade blade profiles under multiple objectives, including a low total pressure loss, a large laminar flow area as well as constraints on the outlet flow angle. In order to obtain a flexible geometry parameterization, the authors perturbed an initial blade profile with Hicks-Henne bump functions. The latter have also been used by Li, Zhang, and Chen [86] to optimize the shape of supercritical airfoils under transonic flow conditions; however, only for additional local geometry modifications, while the overall airfoil shape has been described by a high-order Bernstein polynomial. The optimization itself was performed by a Proximal Policy Optimization (PPO) algorithm.

As an alternative to the incremental shape optimization approach, in *direct shape optimization*, the design variables are not restricted to discrete values, meaning the agent can choose from a continuous action space. This approach has been used, e.g., by Viquerat et al. [172] to optimize two-dimensional shapes with respect to their lift-to-drag ratio. The shapes were parameterized by Bézier-curves and three different configurations with a different number of Degrees of Freedom (DOFs) (up to 12 in total) were examined. The optimization was once again performed by a PPO algorithm. Concluding, we want to mention the work of Ghraieb et al. [48], who recently introduced a framework for RL in CFD, using a Policy Based Optimization algorithm [173] to optimize both two and three-dimensional airfoil shapes with respect to their lift-to-drag ratio under moderately turbulent flow conditions. The airfoil shapes were represented once again by Bézier-curves and afterward immersed in a computational mesh, which was anisotropically adapted to align with the changing geometries.

5.1. The releso Framework

In this section, we cover the key aspects to leverage RL for the shape optimization of profile extrusion die flow channels. Therefore, relevant RL components, e.g., the environment as well as the agent's action and observation spaces, have to be defined accordingly. Figure 5.1 depicts the RL interaction loop between the agent and our custom environment, which is defined in our framework `releso` [43], following the RL environment interface provided by the state-of-the-art package `OpenAI Gym` [17]. For our application, this environment comprises the parameterization of the extrusion die flow channel's geometry and a CFD simulation of the plastics melt inside the die.

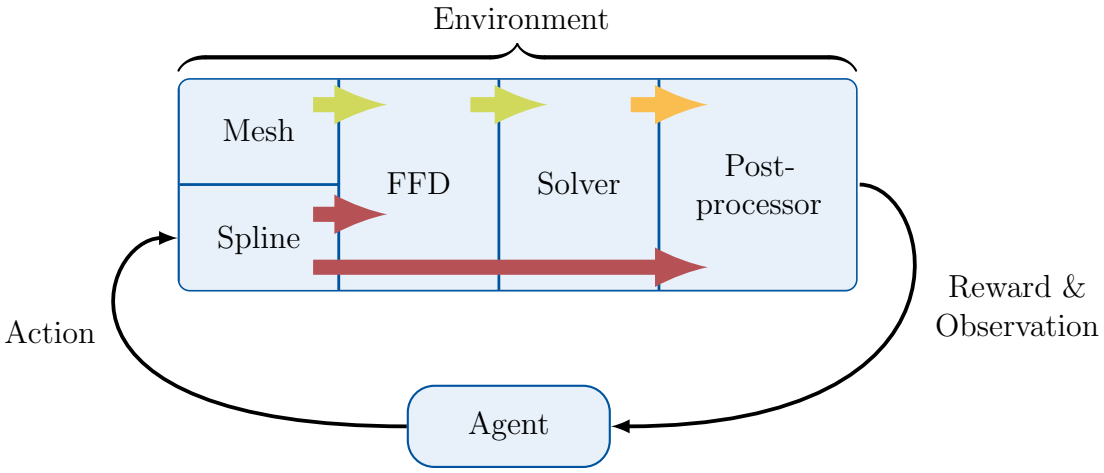


Figure 5.1.: Visualization of the RL interaction between an agent and the environment in our `releso` framework. The environment has been customized to our shape optimization problem and comprises the base mesh and the deformation spline used for the geometry parameterization, a FFD module deforming the mesh, the solver computing the governing PDE problem, and a component for postprocessing the simulation results to determine the reward and an observation of the CFD environment for the agent. The arrows correspond to the information flows between the different components: Green arrows represent meshes, red arrows spline parameterizations, and the yellow arrow symbolizes the simulation results. Based on the provided reward and observation, the agent chooses an action to modify the deformation spline of the FFD.

The remainder of this section loosely follows the flow of information indicated by the arrows in Figure 5.1: We begin with discussing different possibilities to define an agent's action space in Section 5.1.1. This definition is closely related to the two RL-based shape optimization approaches mentioned in the introduction. We explain how the agent can modify a geometry through its actions to enable RL-based shape optimization for both cases. Section 5.1.2 then introduces the concept of Free Form Deformation (FFD), which we employ to parameterize our geometries. We conclude this section by stating the governing equations of plastics melt flow, which constrain our shape optimization problem, and comment on their numerical solution in Section 5.1.3.

5.1.1. Actions: Shape Optimization Methods

As already mentioned in the introduction of this chapter, two RL-based shape optimization methods are described in [171], namely direct and incremental shape optimization. Which method can be used to optimize a geometry, depends on the definition of the agent's action space. Generally, RL agents can either work with continuous or discrete action spaces, although there exist some algorithms that can work with both. In the case of continuous action spaces, the agent can choose action values from a continuous interval, while for discrete action spaces, it selects a value from a discrete set.

When applied to shape optimization problems, the agent needs to be able to modify the geometry with its actions. Its actions are thus defined such that they change the design variables, i.e., the DOFs of the geometry parameterization. Discrete actions are used in the incremental shape optimization approach. In each step of the RL-loop, a single DOF is incremented or decremented by a predefined amount inside the action's value interval. This provides the agent with two actions per DOF in the problem definition. In contrast, the direct shape optimization approach uses continuous actions. Here, each DOF corresponds to a single action which is only constrained by the provided interval. While in the incremental approach only a single DOF can either be incremented a decremented at a time, direct shape optimization allows changing all DOFs at once in each step.

The choice of the action space consequently impacts the way how a shape is optimized: The incremental approach slowly moves toward the optimal geometry, modifying only a single design variable in each iteration, while the direct approach proposes the optimal geometry directly, i.e., ideally, in a single step. Therefore, we expect the training of agents for a direct optimization approach to be more challenging, both with respect to the number of training steps required and the design of the reward function. We will discuss the latter for each approach in the sections of the respective examples.

Most agents introduced in Section 3.3.1 do not support both shape optimization approaches described in this section. For the agent implementations of `stable-baselines3` used in this thesis, Table 5.1 provides an overview of their compatibility with the respective approach.

Table 5.1.: Compatibility of the agents implemented in `stable-baselines3` with regard to the direct and incremental shape optimization method.

Agent	Incremental	Direct
PPO	✓	✓
DQN	✓	-
SAC	-	✓
A2C	✓	✓
DDPG	-	✓

5.1.2. Free Form Deformation: Parameterizing the Geometry

We have seen in the previous section how an agent can modify a given geometry parameterization through its actions. A crucial task in shape optimization however, is to first find a suitable parameterization. Since we want to employ the proposed RL-based shape

optimization method to a variety of extrusion-die flow-channel geometries in the future, the chosen parameterization should be generally applicable and not exploit any features that are restricted to a special geometry. Moreover, we want to control the resulting DOFs of the parameterization, i.e., ideally it should allow to modify even intricate 3D geometries with relatively few parameters.

One possibility would be to directly modify the Computer Aided Design (CAD) model of the flow channel geometry and generate a new mesh in every iteration. However, generating a mesh from a CAD representation can be a challenging and computationally intensive task, which – especially for realistic geometries – cannot be easily automated [28]. Normally, a mesh is only generated once for a geometry and thus the meshing costs also occur only once, which is not the case for shape optimization, where the geometry is gradually modified. Yet, the geometry only undergoes small changes between two succeeding design iterations, motivating the use of a less computationally expensive method.

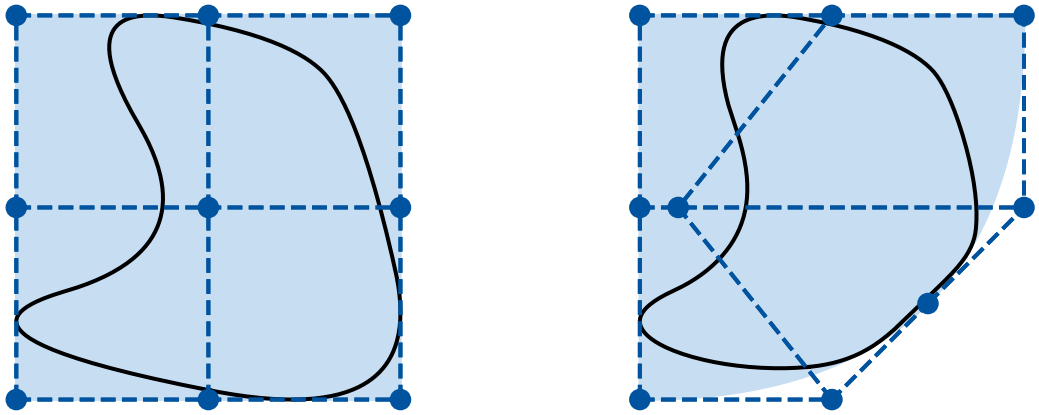
To this end, we propose the use of Free Form Deformation (FFD) [155]. Here, an initial geometry is meshed once and this base mesh is then deformed by a transformation function to represent geometric changes. This means that a mesh of the modified geometry can be generated without the need of first adapting the CAD representation and then re-meshing it in each iteration. Splines (e.g., B-splines or Non-Rational Uniform B-Splines (NURBS) [129]) are a common choice for these transformation functions since they result in smooth deformations. Moreover, they allow the representation of deformations with a comparatively small amount of parameters. This is of great interest in both optimization and RL as it helps to keep the dimensions of the design variables / action space low. To adapt the mesh, the position of each vertex in the mesh is transformed according to the transformation spline. This is visualized in Figure 5.2. Figure 5.2a shows the initial geometry, embedded into the parameter space of the undeformed transformation spline depicted in light-blue. The control points of the transformation spline (the dark-blue colored dots) are the DOFs of this geometry parameterization. Once they have been modified (e.g., through actions chosen by the RL agent), the initial geometry can be deformed accordingly, as shown in Figure 5.2b. To perform the FFD within our environment, the open source python package `gustaf`[1] is used.

5.1.3. Solver: The Partial Differential Equations Governing the Flow of Molten Plastic

In this work, we want to optimize flow channels inside profile extrusion dies with respect to different quantities of interest. As this is done iteratively, the optimization objective needs to be re-evaluated after each geometry modification. Therefore, the underlying equations governing the flow of the highly-viscous plastics melt inside the extrusion die geometry Ω need to be solved. We model the plastics melt based on the same assumptions as already stated in Section 4.2, i.e., the governing Partial Differential Equations (PDEs) of Equation (2.2) simplify to

$$\begin{aligned} \boldsymbol{\nabla} \cdot \boldsymbol{v} &= 0 \quad \text{in } \Omega \,, \\ -\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} &= \mathbf{0} \quad \text{in } \Omega \,. \end{aligned}$$

[1] Available on https://github.com/tataratat/gustaf (last accessed: 25th March 2023)



(a) Initial geometry and undeformed transformation spline.

(b) Deformed spline and resulting deformation of the geometry.

Figure 5.2.: Visualization of the key idea of FFD: An initial geometry **(a)** is transformed by modifying the control points (dark-blue dots) of a transformation spline (highlighted in light-blue) to obtain a deformed geometry **(b)**.

Again, we employ the Carreau material law introduced in Equation (2.3) to model the shear-thinning behavior. The material parameters chosen in our test cases are given in Table 5.2.

Table 5.2.: Material properties of the shear-thinning material law for all test cases.

Parameter	Value	Unit
A	10935	$\mathrm{kg\,m^{-1}\,s^{-1}}$
B	0.433	s
C	0.699	-

Concerning BCs, we impose no-slip BC on the walls of the profile extrusion die as well as a traction-free condition at the outflow, i.e.,

$$\boldsymbol{v} = \boldsymbol{0} \quad \text{on } \Gamma_{\text{wall}} \,, \tag{5.1a}$$

$$\boldsymbol{\sigma} \cdot \boldsymbol{n} = \boldsymbol{0} \quad \text{on } \Gamma_{\text{out}} \,. \tag{5.1b}$$

The inflow BC depends on the considered geometry and will thus be reported for each test case in Sections 5.2 and 5.3. We solve the resulting PDE problem with a stabilized FEM approach as explained in Section 2.4, implemented in our in-house simulation code. Linear basis functions are used to discretize the unknown pressure and velocity fields. The numerical solution of the model equations is integrated into our custom environment. In every RL step, i.e., after each geometry modification, our simulation code is invoked to compute new solutions for the velocity and pressure fields. Based on the solution fields, the numerical design criterion can be evaluated.

So far, we have only introduced the components of our RL framework, which are test case-independent. However, some components like, e.g., the reward function or the agent's observation space depend on considered problem. Thus, they are defined in the following

Sections 5.2 and 5.3, where two selected application cases are presented, for which we want to answer Research Questions (II a) to (II c) as posed in the introduction.

5.2. Optimizing the Mass-Flow Ratio between the Outflows of a T-shaped Geometry

As a first test case, we consider a simple two-dimensional T-shaped geometry, serving as an introductory example to assess the capabilities of the proposed RL-based shape optimization framework. Already introduced in a prior work by the authors [180] to investigate the applicability and reproducibility of the incremental shape optimization approach learned by a Proximal Policy Optimization (PPO) agent, we extend this application here to a direct shape optimization approach and compare the performance of different agents.

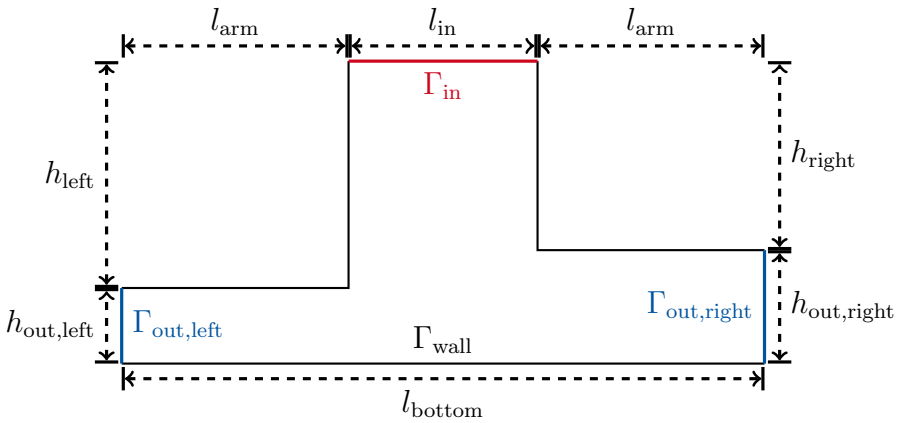


Figure 5.3.: Geometry and boundaries of the initial T-shaped domain.

The geometry to be optimized is depicted in Figure 5.3, the corresponding dimensional parameters are given in Table 5.3. The general learning task is stated as follows: Optimize the geometry Ω such that a desired mass-flow ratio $\varrho^\star$ between the mass flows through the left and right outflow boundaries is achieved. While this simple objective does not resemble the overall goal to optimize an extrusion die geometry with respect to the flow homogeneity at the outflow, it allows us to intuitively assess the correctness of the optimized geometries.

Table 5.3.: Dimensions of the T-shaped geometry.

Parameter	Value	Unit
l_{arm}	0.03	m
l_{bottom}	0.085	m
l_{in}	0.025	m
h_{left}	0.03	m
$h_{\text{out,left}}$	0.01	m
$h_{\text{out,right}}$	0.015	m
h_{right}	0.025	m

To make the learning process more general, we do not train the agents to optimize the geometry for a single target mass-flow ratio, but instead, randomly select a new one from a predefined interval according to a uniform distribution, i.e., $\varrho^\star \sim \mathcal{U}_{[0.1,2]}$, at the beginning of each episode. Thus, after training an agent, it should be able to generate optimal shapes for a variety of mass-flow ratios.

In the rest of this section, we will first introduce the parameterization of the test case's geometry, i.e., the agent's action space. We continue with a description of how the optimization objective can be determined from the results of the CFD simulation. This is followed by a section on reward shaping, which is crucial as the reward signal guides the agent toward learning a good strategy. For the results, we train different agents with both an incremental and a direct strategy and compare the learning processes with respect to their convergence behavior. We will furthermore compare the general behavior of both approaches. To wrap this section up, we investigate the vectorized environment training with regard to a potential decrease in the real-world training time (wall time) of the agent.

5.2.1. Geometry Parameterization

The T-shaped geometry depicted in Figure 5.3 represents a flow separator with one inflow and two outflows. The boundary conditions on the wall Γ_{wall} and at the outflow $\Gamma_{\text{out}} = \Gamma_{\text{out,left}} \cup \Gamma_{\text{out,right}}$ are given by Equation (5.1). The boundary condition on Γ_{in} is given by

$$\boldsymbol{v} = \begin{pmatrix} 0 \\ -0.45 \end{pmatrix} \quad \text{on } \Gamma_{\text{in}} \, .$$

The deformation spline is a two-dimensional B-spline with second order basis functions in both parametric dimensions. It features nine control points in total, arranged as shown in Figure 5.4. This gives the agent 18 DOFs, two for each control point, as the control points can move in x and y direction. Agents following an incremental shape optimization approach can choose from 36 discrete actions while agents utilizing the direct optimization approach operate with an 18-dimensional action space.

The base mesh for the FFD consists of 4759 triangular elements and 2515 nodes.

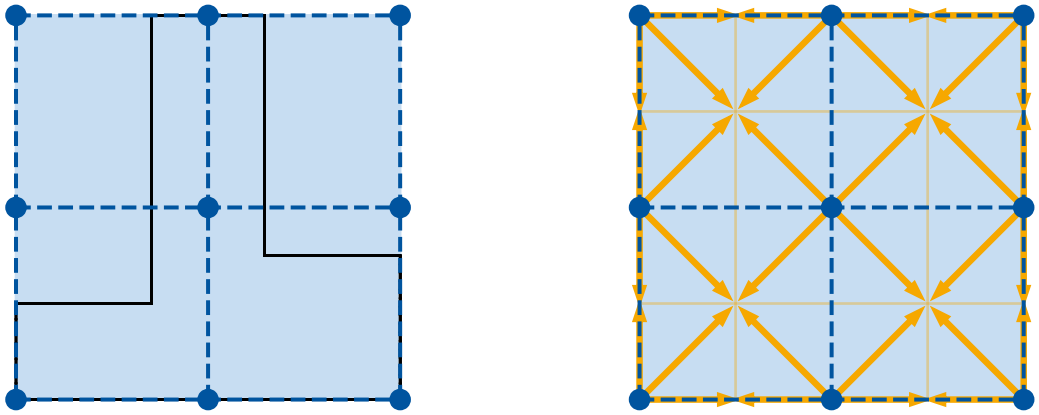
5.2.2. Observations

Since we want to optimize the T-shaped geometry with respect to desired mass-flow ratio $\varrho^\star$, the agent needs to be provided with information about the mass-flow ratio for a given geometry. Therefore, we can compute the mass-flow ratio from the results of the CFD simulation. The mass-flow ratio between the left and right outflow at a certain step t is defined as

$$\varrho_t = \frac{(\dot{m}_{\text{out,left}})_t}{(\dot{m}_{\text{out,right}})_t} \, , \tag{5.2}$$

where the mass flows over each outflow boundary $\Gamma_{\text{out,left}}$ and $\Gamma_{\text{out,right}}$ can be computed by integrating the velocity field $\boldsymbol{v}$ over each boundary

$$\dot{m}_{(\,\cdot\,)} = \rho \oint \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}\Gamma_{(\,\cdot\,)} \, . \tag{5.3}$$



(a) Geometry embedded into the deformation spline of the FFD.

(b) DOFs of the parameterization.

Figure 5.4.: **(a)** shows the deformation spline used for the parameterization of the T-shaped geometry. To make the FFD more generic, the actual geometry is scaled to the parametric space of the transformation spline before applying geometric modifications. Additionally, the possible movement of the control points is illustrated by the orange arrows in **(b)**.

Here, ρ denotes the density of the molten plastic, which is constant due to the incompressibility assumption. However, as long as we are only considering mass-flow ratios according to Equation (5.2), the density is irrelevant since it cancels out.

In each step t, the agent is now provided with the mass-flow ratio ϱ_t, the target mass-flow ratio $\varrho^\star$, and the control points of the transformation spline.

5.2.3. Reward Shaping

Reward shaping is one of the most crucial parts in RL, as the reward is responsible for guiding the agent toward a good strategy for solving the problem. However, this is not a straightforward task, as it greatly depends on the problem under consideration as well as the shape optimization strategy utilized. In the following sections, we present reward functions for both shape optimization strategies.

Direct Approach For the direct approach, we first define the deviation of the current step's mass-flow ratio from the desired target ratio according to

$$e_{(\,\cdot\,)} = \left|\varrho_{(\,\cdot\,)} - \varrho^\star\right| .$$

This quantity is then used with a Boolean value indicating the success of the simulation to shape the following reward function

$$r_t = \begin{cases} -10 & \text{simulation failed} \\ -e_t & e_t \geq \epsilon \\ 5 & e_t < \epsilon \end{cases} . \tag{5.4}$$

This reward function is non-sparse, i.e., the agent receives a reward signal in every step and not only after it accomplishes its task. A failed simulation (which should only occur in

the case of a tangled mesh) is severely penalized by a fixed value of -10. In contrast, the agent receives a constant reward of 5 if the deviation from the goal ratio falls below the given acceptance threshold ϵ. Since we ideally want the agent to reach the goal in a single step in the direct approach, a negative reward is given for every step in which the agent does not reach the goal. To provide the agent with some information on how close the current shape is to the optimal shape, this negative reward corresponds to the deviation from the goal ratio instead of, e.g., a constant negative reward.

Incremental Approach For this test case, the incremental approach has already been investigated in [180] and we employ the same reward function here. It depends on the current target mass-flow ratio $\varrho^\star$, the mass-flow ratios ϱ_t and ϱ_{t-1}, as well as the deviations from the target mass-flow ratio e_t and e_{t-1} at steps t and $t-1$

$$r_t = \begin{cases} -10 & \text{simulation failed} \\ -0.5 & e_t > e_{t-1} \\ -0.2 & e_t = e_{t-1} \\ \frac{|\varrho_t - \varrho_{t-1}|}{e_{t-1}} & e_t < e_{t-1} \wedge e_t \geq \epsilon \\ 5 & e_t < \epsilon \end{cases}, \tag{5.5}$$

Similar to the reward function in the direct approach Equation (5.4), a reward of -10 is only generated, if the episode is considered unsuccessful and terminated prematurely. Likewise, a reward of 5 corresponds to a successful solution of the task. With the incremental approach, the agent is expected to optimize the geometry in multiple subsequent steps. Here, we can distinguish between three possible scenarios: If a modification of the geometry results in a bigger deviation from the desired target, i.e., $e_t > e_{t-1}$, the agent is penalized with a negative reward of -0.5. If the agent did not make progress, i.e., $e_t = e_{t-1}$, it also receives a penalty, amounting to -0.2. In case of improvement concerning the desired target mass-flow ratio, the agent is rewarded proportional to the relative improvement compared to the previous step. This asymmetric definition of the reward has been chosen to prevent the agent from undertaking sequences of actions that would hinder it in increasing the accumulated reward. Instead, it obtains direct feedback that provides knowledge about the quality of the previous action regarding the desired goal.

5.2.4. Agent Comparison

In this section, we compare different learning algorithms following both direct and incremental optimization strategies to investigate their suitability and behavior with respect to training effort and convergence. Within the whole section, each experiment is performed twice to show consistency among repeated training runs.

Direct Approach In Figure 5.5, the progression of the agents' episode reward over the trained steps is shown. The algorithms differ significantly with respect to the convergence of the episode reward. First, it can be noticed that DDPG and PPO manage to steadily increase their reward, albeit on different scales. Especially in the early training phase ($t \leq 20\,\text{k}$), the DDPG agents show more volatility compared to the PPO agents. However, the DDPG agents converge faster to the optimal reward value of 5 (see Equation (5.4)). The

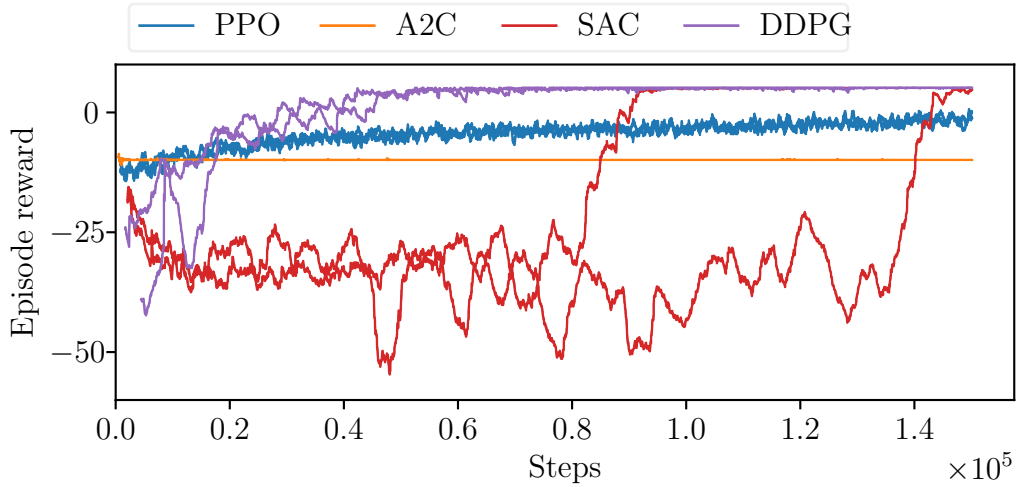


Figure 5.5.: Comparison of different algorithms trained to optimize the T-shaped geometry following a direct strategy with respect to the episode reward over the trained steps. Each run was repeated twice as indicated using the same color.

reward curves corresponding to the two SAC agents exhibit three distinct phases: Initially, the rewards stay more or less constant for roughly 40 k steps. Afterward, the rewards start to oscillate, culminating in sharp jumps that increase the episode rewards to their optimal value of 5. After the jumps, the episode rewards even out at this value. Noteworthy is that the sharp jumps occur after a totally different amount of training steps. This suggests that the SAC agent is more susceptible to random variations during the training. The training runs conducted with the A2C agent show no learning progression at all. While at first sight, this seems to suggest an inability of this algorithm to learn a valid strategy for this problem, it may also stem from the fact that the reward function shaped in Equation (5.4) does not work well for this particular algorithm.
In terms of consistency, of all agents that successfully learned a strategy on this test case, the PPO agents show the smallest variation among repeated runs. In contrast, the biggest variations can be observed for the SAC agents, as already discussed above.

To answer Research Question (II b) properly, we also need to compare the different algorithms with respect to their total training time. The wall-clock times are reported in Table 5.4. One can directly spot differences in the efficiency, as the different algorithms

Table 5.4.: Wall-clock times of the agents trained with the direct optimization method to optimize the T-shaped geometry.

Agent	Max. training time
PPO	26.1 h
A2C	46.5 h
SAC	26.0 h
DDPG	33.5 h

require different amounts of time to complete the demanded 150 k steps. As before, A2C performs worst when measured in this metric. Of all algorithms exhibiting convergent

behavior with respect to the episode reward, the DDPG agent takes the longest to perform the 150 k steps. However, it would be possible to stop the training much earlier since it is already converged after roughly 45 k training steps as seen in Figure 5.5. The wall-clock times of SAC and PPO are equally long. However, the inconsistent behavior evident in Figure 5.5 could also be observed with respect to the training times of the two runs. Both PPO runs consistently took roughly 26 h, but it needs to be mentioned that this algorithm did not fully converge, i.e., the optimal reward was not yet reached within the fixed amount of 150 k training steps.

This investigation shows that it is important to not only focus on the training time alone, but also take the reward into account: In terms of training time, both PPO runs outperform the DDPG agents. However, the DDPG agents accumulate their reward faster, thus amortizing the increased duration of the individual training steps to some extent.

Incremental Approach For the incremental optimization strategy, the employed RL package `stable-baselines3` provides three algorithms that can operate on discrete action spaces (see Table 5.1), namely PPO, A2C, and DQN. These will be compared in the following.

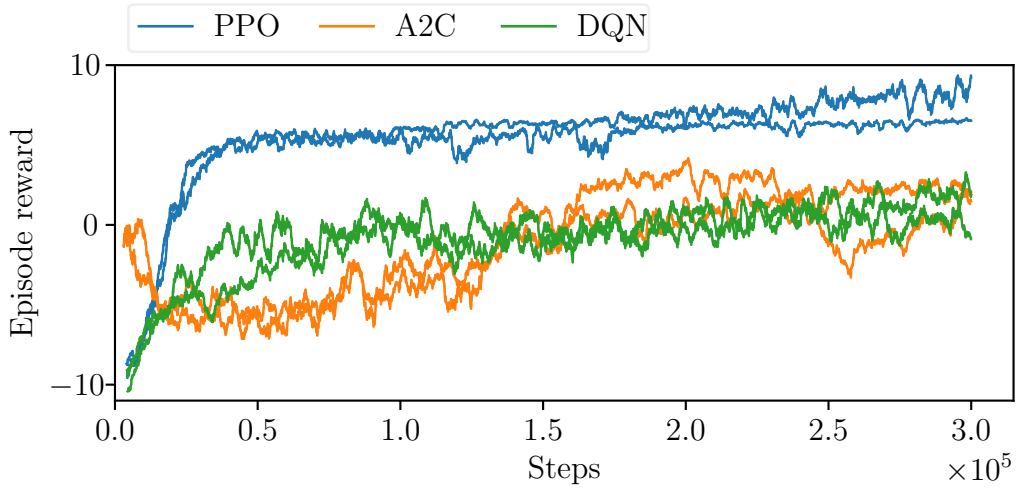


Figure 5.6.: Comparison of different algorithms trained to optimize the T-shaped geometry following an incremental strategy with respect to the episode reward over the trained steps. Each run was repeated twice as indicated using the same color.

Figure 5.6 again depicts the episode reward over the trained steps, for up to 300 k steps. The PPO agent exhibits a steep learning curve, especially during the early training phase ($t \leq 50\,\text{k}$) and quickly converges to the optimal reward of 5. Afterward, the curve of the episode reward flattens and oscillates around this value. Both PPO runs show very similar behavior. The other two algorithms do not converge in the allotted time; nevertheless their episode reward curves slowly increase over the depicted training steps. It is worth noting that here, the A2C agent manages to learn a strategy for increasing its episode reward – in contrast to its application with the direct optimization strategy. A possible reason behind this behavior as well as an in-depth comparison of the different agent types will be briefly discussed in the next paragraph. One interesting anomaly can be observed from

250 k steps onward: One of the PPO training runs diverges and finds a way to increase its episode reward again. This can be explained with a non-optimal reward function as the agent evidently finds a way to exploit Equation (5.5) to increase the reward beyond the optimal value of 5. Exploitation of that sort can be prevented in two possible ways: Either the reward function needs to be improved or the training of the agent needs to be stopped as soon as it is converged.

Table 5.5.: Wall-clock times of the agents trained with the incremental optimization method for the use case with the T-shaped geometry.

Agent	Max. training time
PPO	49.9 h
A2C	45.1 h
DQN	45.0 h

In contrast to the results with the direct optimization strategy, the wall time comparisons for the incremental approach (see Table 5.5) reveal less pronounced differences between the different algorithms. The only outlier is one of the PPO agent runs, which takes about 5 h longer than the other. This can be attributed to the additional computational overhead associated with the time required to reset an episode: The PPO training run, which does not diverge, needs to be reset more often, since it needs fewer steps per episode to accomplish its task than its counterpart, which just tries to exploit the reward function to increase its reward even further.

Examples of optimized geometries obtained from a PPO agent following an incremental shape optimization approach are included in Figure C.1.

A Remark on the Learning Behavior of A2C Although the A2C agent did not manage to learn a good policy for optimizing the T-shaped geometry with a direct strategy, it did when following an incremental approach. Even more surprisingly – as we will show in Section 5.3.4 – the A2C agent also managed to learn a strategy for optimizing the geometry of the second test case with the direct shape optimization method. While performing further experiments not contained in this thesis, it was found that the A2C agent manages to find and exploit a local optimum in the reward function Equation (5.4). This local optimum always provides a constant penalty of -10 if the agent generates a tangled mesh, no matter in which training step. So if the agent generates a tangled mesh as fast as possible, i.e., in the first step, without accumulating additional negative rewards before, the episode reward converges to a value of -10. After this discovery, we adapted the reward function to resolve this issue. In an additional experiment with the adapted reward function, an A2C agent following a direct shape optimization approach was also successfully trained for the T-shaped geometry. The results are not published in this thesis since the change in the reward function would have required reevaluating all other experiments to make the results comparable. Nevertheless, we considered this finding worth mentioning here.

5.2.5. Vectorized Environment Training

As we have seen in the previous sections, training an RL agent can require a significant amount of time, depending on the employed optimization strategy as well as the learning algorithm. In this section, we want to address Research Question (II c) and see if the wall training time can be reduced by letting the agent interact with multiple environments. We will investigate this research question here exemplarily for the incremental optimization approach.
The previous results showed that the PPO agent trained with an incremental optimization strategy has a stable and reproducible learning curve if the training is stopped after the algorithm is converged. Therefore, this combination was chosen to investigate the vectorized environment training. Even though only results for the PPO agent are shown here, the other algorithms were also tested and behaved similarly. The conducted experiments include a comparison of training runs using one, two, four, and eight vectorized environments. Each agent is trained for a maximum number of 100 k steps.

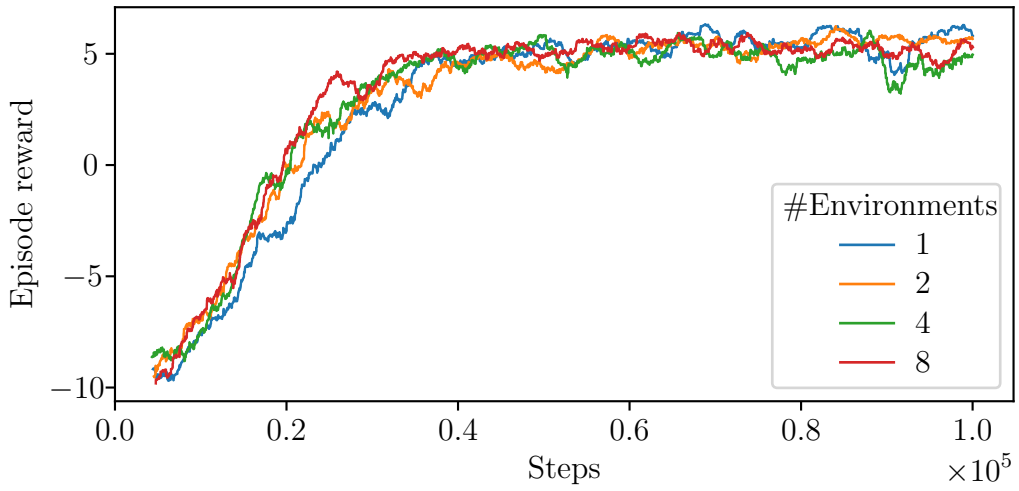


Figure 5.7.: Episode reward over training steps of a PPO agent following an incremental strategy for optimizing the T-shaped geometry when interacting with different numbers of environments.

Figure 5.7 visualizes the agent's episode reward over the trained steps. It shows, that the training behavior does not significantly change if the number of environments is increased.

This however changes completely, if the episode reward is plotted over the wall time instead, as depicted in Figure 5.8. Here, a significant decrease in the total training time can be observed. While not scaling perfectly, i.e., halving the training time with every doubling of the amount of environments, an overall reduction from about 15.33 h with one environment to 3.33 h with eight environments can be observed. This corresponds to a wall time reduction of about 78 %, which answers Research Question (II c) for this application case.

The next section introduces a more realistic example with respect to the optimization of flow channels in profile extrusion dies.

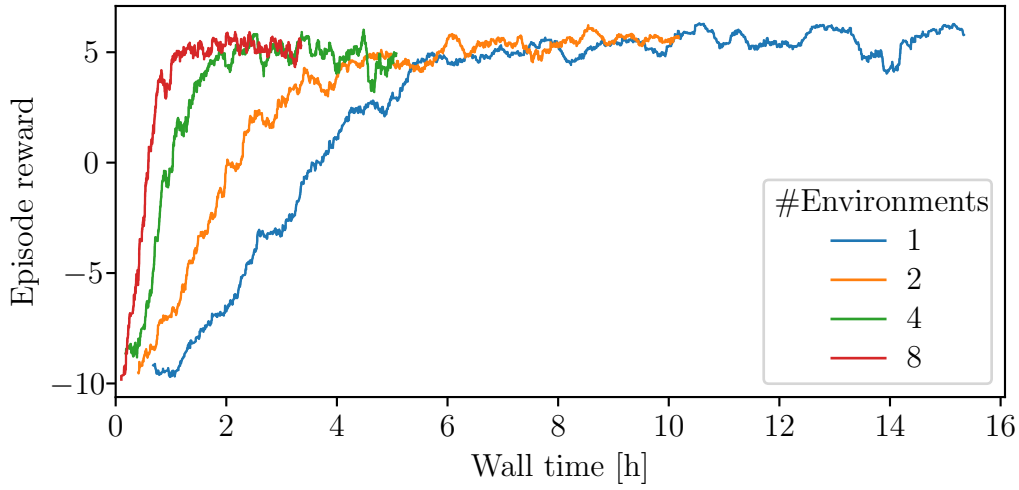


Figure 5.8.: Episode reward over wall time of a PPO agent following an incremental strategy for optimizing the T-shaped geometry when interacting with different numbers of environments.

5.3. Optimizing a Converging Channel with Respect to the Flow Homogeneity

In this test case, an agent is trained to optimize a two-dimensional converging channel geometry with respect to the homogeneity of the flow at the outlet. The initial geometry is depicted in Figure 5.9, the corresponding dimensions are reported in Table 5.6. In real profile extrusion lines, this quantity has the largest influence on the quality of the manufactured part [59]. Since we want to apply the RL-based shape optimization method to realistic flow channel geometries in future research, with this test case, we want to investigate the feasibility of this method when employed with a state-of-the-art design criterion.

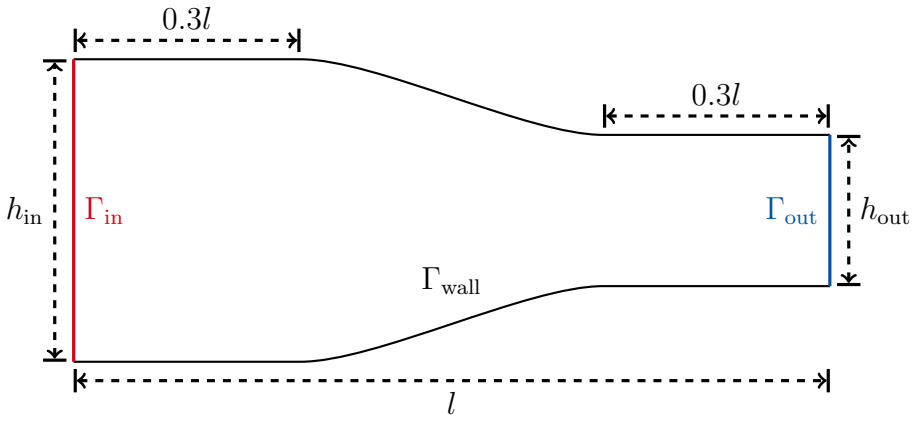


Figure 5.9.: Geometry and boundaries of the initial converging channel domain.

Compared to the previous one, this test case is especially challenging for two reasons, i.e., the generalization to a whole class of geometries and the inability to reach the optimum.

Table 5.6.: Dimensions of the converging channel geometry.

Parameter	Value	Unit
l	1.0	m
h_{in}	0.4	m
h_{out}	0.2	m

Generalizing the task of optimizing the flow homogeneity would require creating many diverse geometries. Otherwise, the agent could not learn a strategy applicable to all. If only a single geometry is presented during training, the agent only learns to solve this single problem. However, this would defeat the main attraction of this method, namely to train an agent for a more generalized task and then be able to solve specific tasks inside the generalized task's envelope faster than conventional optimization methods.
The inability to reach the optimum stems from the fact that the velocity profile with optimal flow homogeneity would be a block profile. However, such a velocity profile cannot be achieved in real profile extrusion lines since the plastics melt adheres to the channel walls. This, in turn, presents a problem for our RL approach: Our experiments showed that the agent learns best if it tries to reach a discrete goal within each episode and if the learning is terminated upon achievement. If the episode is not ended (i.e., when the optimum cannot be reached), the agent will continue performing actions. At least with the incremental strategy, these additional actions will eventually move the state away from an already good solution, incurring negative rewards. One possible remedy would be to switch to sparse rewards, i.e., only generate a reward if the state reached the optimal flow homogeneity. In order to end the episode at or close to the optimal geometry, the best flow homogeneity achievable needs to be determined with the used geometry parameterization, creating a cyclic dependency. To overcome this issue, in this thesis, we used a preliminary training run to empirically determine the best achievable value of the quality criterion. Of course, this does not guarantee the optimality of this value, but possibly a sufficiently good approximation since more than 100 k designs have been evaluated. Following this approach, we found that the numerical value of the optimal homogeneity quality criterion $q^\star = 0.358$ is very close to the quality criterion of the initial channel geometry.

Therefore, the learning task for this test case can be stated as: Optimize the geometry Ω such that the flow homogeneity at the outflow boundary amounts to $q^\star$. To increase the generalizability of this test case, the initial geometry depicted in Figure 5.9 is deformed at the beginning of each episode by a randomly chosen parameterization of the same spline which is used by the agent. By doing this, the agent solves the previously stated learning task on a different geometry in each episode, thus learning a more general strategy. At the same time, this also circumvents the issue that the original geometry already provides a quantitatively good homogeneity, as far as our numerical experiments were able to show.

The rest of this section largely mimics the structure of Section 5.2. For the results, however, an additional section is included, where we compare the direct and incremental shape optimization approach for this test case.

5.3.1. Geometry Parameterization

The converging channel geometry is depicted in Figure 5.9. The boundary conditions on Γ_{wall} and Γ_{out} are once again chosen according to Equation (5.1) and we also employ a block profile on Γ_{inflow} similar to Section 5.2, except that we rotate it by 90° to fit the changed orientation of the inflow boundary

$$\boldsymbol{v} = \begin{pmatrix} 0.45 \\ 0 \end{pmatrix} \quad \text{on } \Gamma_{\text{in}} \ .$$

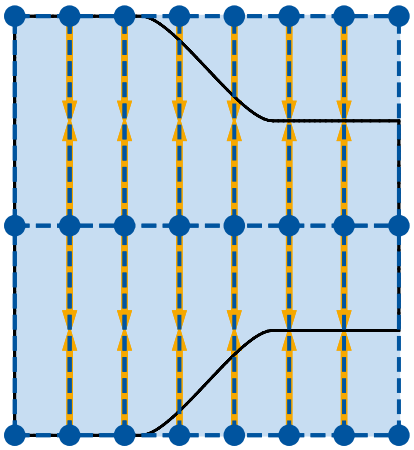


Figure 5.10.: Deformation spline used for the parameterization of the channel geometry. Additionally, the possible movement of the control points is illustrated by the orange arrows.

The geometry is parameterized by a two-dimensional B-spline with second-order basis functions in both parametric directions. Its 24 control points are arranged as shown in Figure 5.10. Here, we restrict the movement of the control points and only allow changes perpendicular to the main flow direction. Moreover, we fix the position of those control points, which would modify the position of the inflow or outflow. In total, we end up with 18 control point coordinates that can move in y-direction. Agents following an incremental shape optimization approach thus choose from a set of 36 discrete actions, while agents pursuing a direct optimization approach have an 18-dimensional action space available.

The base mesh for the FFD consists of 3886 triangular elements and 2040 nodes.

5.3.2. Observations

The optimization objective is to achieve a flow that is as homogeneous as possible at the outflow. Different methods to quantify the flow homogeneity have been studied in the literature [39, 141]. They all have in common that they divide the outflow boundary into non-overlapping so-called *patches* (highlighted in different colors in Figure 5.11) and evaluate a quality criterion on each patch. This patch-wise quality criterion usually relates the average velocity on these patches

$$v_{\text{out},i} = \frac{\dot{m}_{\text{out},i}}{\rho A_{\text{out},i}} \tag{5.6}$$

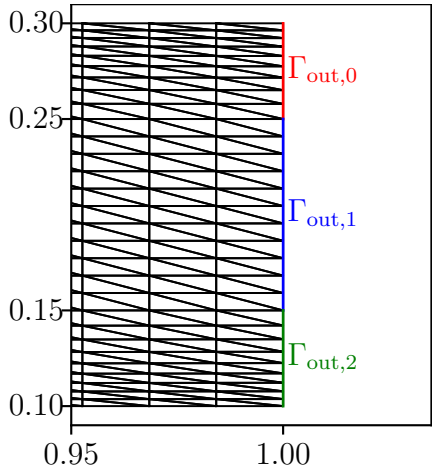


Figure 5.11.: Outflow boundary of the converging channel geometry. The outflow is divided into three patches $\Gamma_{\text{out},i}$, for which the flow-homogeneity criterion is evaluated.

to the average velocity across the whole outflow boundary

$$v_{\text{avg}} = \frac{\dot{m}_{\text{out}}}{\rho A_{\text{out}}} \,. \tag{5.7}$$

Therein, the mass flows $\dot{m}_{(\,.\,)}$ are computed according to Equation (5.3), while $A_{(\,.\,)}$ denotes the area of the respective boundary segment $\Gamma_{(\,.\,)}$. The flow homogeneity is then quantified by the means of the patch-wise ratio between these velocities as

$$\omega_{\text{out},i} = \frac{v_{\text{out},i}}{v_{\text{avg}}} \,. \tag{5.8}$$

Building on Equations (5.6) to (5.8), we employ the quality criterion proposed by Rajkumar et al. [141] in this work, which can be calculated on each patch according to

$$q_{\text{out},i} = \frac{\omega_{\text{out},i} - 1}{\max(\omega_{\text{out},i}, 1)} \,. \tag{5.9}$$

Equations (5.8) and (5.9) confirm the previous argument that a block profile would be optimal with respect to the flow-homogeneity criterion: In case of a block profile, the average velocity on each patch would be identical to the average velocity of the whole outflow boundary, i.e., $v_{\text{out},i} = v_{\text{avg}}$, which in turn implies $\omega_{\text{out},i} = 1$. With this, we obtain the theoretically optimal value of $q_{\text{out},i} = 0$.

The patch-wise quality criteria $q_{\text{out},i}$ are computed from the results of the CFD simulation and then provided to the agent as observations. Moreover, they are used to determine the reward. The reward functions for both incremental and direct RL-based shape optimizations are presented in the following section.

5.3.3. Reward Shaping

Similar to the test case with the T-shaped geometry presented in Section 5.2, we also shaped two reward functions for the direct and incremental approach. Since we used three patches in our geometry (see Figure 5.11) to compute the flow homogeneity at the outflow,

we need to combine them into one single value q_t, which is done by employing a sum of squares

$$q_t = \sum_i \left(q_{\text{out},i}\right)_t^2 . \tag{5.10}$$

This single value is now used to determine the reward. Note that the block profile is also still optimal when considering Equation (5.10), since $q_{\text{out},i} = 0$ also implies $q_t = 0$.

Direct Approach The reward function for the direct optimization strategy is defined similar to the previous test case presented in Equation (5.4)

$$r_t = \begin{cases} -10 & \text{simulation failed} \\ -q_t & q_t \geq q^\star \\ 5 & q_t < q^\star \end{cases} .$$

Unless the agent reaches the goal or the solver fails, the reward is proportional to the quality criterion and negative since the optimal quality criterion is 0. As the direct optimization method's goal is to adapt the shape in a single step, each step where the agent fails to solve its task is undesirable. To be guided towards an optimal geometry, the agent receives a less severe penalty the closer the objective is to its goal.

Incremental Approach In the incremental approach, the reward is defined based on the improvement between two succeeding steps $t-1$ and t which reads

$$I_t = q_{t-1} - q_t .$$

Based on that, the reward function is defined as

$$r_t = \begin{cases} -10 & \text{simulation failed} \\ 2I_t & I_t < 0 \\ I_t & I_t \geq 0 \\ 5 & q_t < q^\star \end{cases} .$$

Here, the improvement I_t is used as the reward if the improvement is greater than zero, meaning that the homogeneity improved compared to the last step. When the homogeneity deteriorates, I_t is negative and again used as reward, but this time multiplied by a factor of two. This hinders the agent in exploiting the reward function and discourages it from undertaking actions that decrease the flow homogeneity. Once again, the reward for achieving the goal is 5 and for tangling the mesh is -10.

5.3.4. Agent Comparison

Similar to the previous test case, we start our investigation by comparing the different algorithms and examining their impact on training progression and stability. The experiments are shown for a limited number of steps, which is chosen such that all necessary information is contained in the selected interval. As before, all experiments in this section have been conducted twice. We first compare the agents trained with the direct shape optimization method, before comparing agents following an incremental shape optimization approach.

Direct Approach Figure 5.12 depicts the agents' episode rewards over the total amount of trained steps. All results are shown for up to 350 k steps and at most 100 k episodes. As

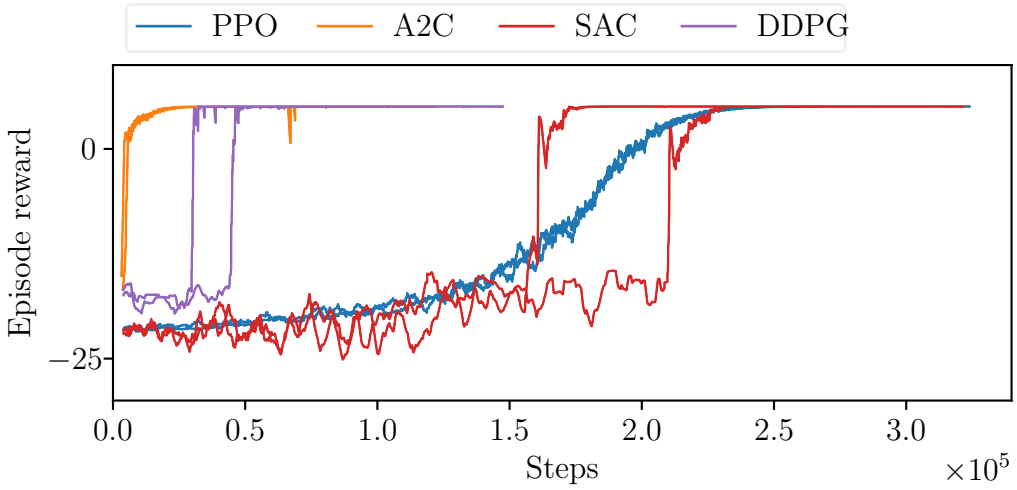


Figure 5.12.: Comparison of different agents trained to optimize the converging channel geometry following a direct strategy with respect to the episode reward over the trained steps. Each run was repeated twice as indicated using the same color.

a first observation, we note that for this test case, all agents manage to learn a strategy, as all reward curves converge to the optimal reward of 5. This is in contrast to the previously presented optimization of the T-shaped geometry, where the A2C agent did not learn anything. Secondly, we discover that the general shape of the agents' learning curves is similar between different runs, i.e., the algorithms seem to find the same strategies. However, only the learning curves of PPO and A2C are in very good agreement in terms of reproducibility. The sudden jump in the episode reward of the SAC agent that we observed in Figure 5.5 is also present in this test case. Yet, we note an identical behavior for the DDPG agent, which did not show this behavior before. While converging quickly and monotonously to the maximal reward limit in the previous example, the DDPG now behaves differently, displaying a behavior similar to the SAC agent. However, the sharp jumps occur roughly 100 k steps earlier. The authors cannot definitely say, why the behaviour of this algorithm differs so much between the two examples. One explanation might be that the preceding test case was much simpler to learn as can be seen by the fewer steps required to learn a solution strategy. Therefore, the first phase of roughly constant episode reward, visible in Figure 5.12 for $t \leq 30\,\mathrm{k}$, is simply not present in Figure 5.5, as the agent directly found a way to increase the reward without first exploring the action space. One can also see that the A2C and DDPG agents are not trained for the whole 300 k steps. Instead, their training is stopped earlier when they reach the episode limit of 100 k episodes. This limit is reached so early since the agents are trained with a direct optimization approach: After being fully converged (i.e., $t \geq 50\,\mathrm{k}$), they only need a very small number of steps (often only one) per episode to accomplish their task. This can be seen in Figure 5.14, where the steps per episode are plotted over the total number of training steps.

Next, we investigate the performance of the different algorithms with respect to overall training time. Their wall-clock times differ widely as shown in Table 5.7. They range from approximately 44 h for the A2C and DDPG agent to approximately 70 h for the SAC

Table 5.7.: Wall-clock times of the agents trained with the direct optimization method for the converging channel use case.

Agent	Training time
PPO	64.2 h
A2C	44.0 h
SAC	70.0 h
DDPG	45.0 h

agent. One reason for these huge deviations is that the A2C and DDPG agents performed far fewer steps than the PPO and SAC agents as they managed to learn a strategy faster (as will later on be seen in Figure 5.14). The difference between the 64 h for the PPO agent and the 70 h for the SAC agent shows that the former is more time efficient during the training. A more in-depth analysis of the differences with regard to the training time is complicated for the direct approach: Before starting a new episode, the environment needs to be reset, which involves a constant computational overhead that does not count toward the actual training time. However, as soon as the agent finds a good strategy, the number of steps per episode decreases significantly, in turn proportionally increasing the number of episode resets. When the agents manage to optimize the geometry in a single step, the time used for training is basically identical to the time for resetting the environment, complicating the task of determining the actual learning time.

Incremental Approach Next, we compare the algorithms that are compatible with an incremental shape optimization approach. The achieved episode reward of the agents is shown for up to 200 k steps in Figure 5.13. In contrast to the results seen for the T-shaped

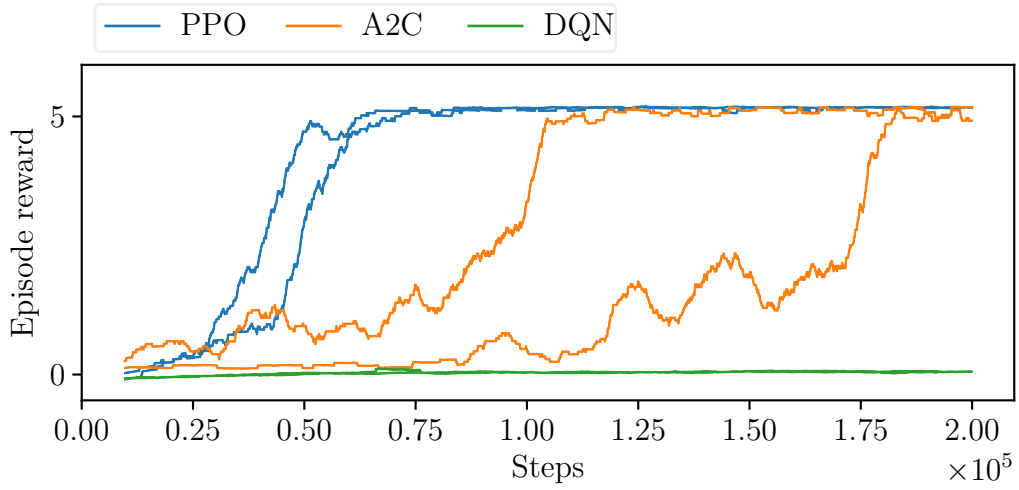


Figure 5.13.: Comparison of different agents trained to optimize the converging channel geometry following an incremental strategy with respect to the episode reward over the trained steps. Each run was repeated twice as indicated using the same color.

geometry in Figure 5.6, the A2C agent manages to fully converge within the depicted

number of steps, while the DQN agent does not converge at all. It seems to initially make progress during the first 100 k steps but stagnates after that. The authors are not sure about the reason for this stagnation. It might be attributed to a not-optimal reward function, which seems to only work well for the PPO agent as this algorithm repeatedly converges within $t \leq 75\,\text{k}$ steps. However, different explanations are possible, e.g., the training progress of the DQN agent might be generally much slower compared to the other algorithms (as it also only converges slowly when optimizing the T-shaped geometry as seen in Figure 5.6) or the problem might simply be too complex for the agent to learn, given the employed combination of reward and observations. In terms of consistency between repeated training runs, A2C now exhibits significant deviations. Clearly, further experiments are necessary to explain the behavior of the agents.

Table 5.8.: Wall-clock times of the agents trained with the incremental optimization method for the converging channel use case.

Agent	Training time
PPO	42.2 h
A2C	43.1 h
DQN	44.7 h

The learning curves of the PPO and A2C agents increase smoother than the learning curves seen for the SAC and DDPG agents in the previous section, but less gradually than the PPO agent trained on the T-shaped geometry following the same strategy.

As reported in Table 5.8, the wall-clock times of the different algorithms are rather similar as opposed to the training times of the previous experiment. The difference between the DQN agent and the PPO agent only amounts to roughly 2.5 h. This can partly be attributed to the fact that – in the case of incremental shape optimization – all agents train for the same number of steps and that they are not yet fully converged, as we will discuss in the next Section 5.3.5.

Examples of optimized geometries obtained from a PPO agent following an incremental shape optimization approach are included in Figure C.2.

5.3.5. Comparison of the Optimization Approaches: Direct vs. Incremental Shape Optimization

After investigating both RL-based shape optimization approaches as introduced in Section 5.1.1 on two different examples, we want to use this section to compare both methods with the aim of providing an answer to Research Question (II a). We have seen that different agents following either strategy can be successfully trained for both test cases. To validate that the strategies actually do what they promise, specifically that the direct optimization method is able to find an optimal geometry in only a single step, the number of training steps per episode (corresponding to one optimization each) has been analyzed and is depicted in Figures 5.14 and 5.15 for both approaches. The difference between the results is apparent by comparing the y-axes and corresponds to our prior expectation. Once converged, agents following a direct optimization strategy decrease the required steps per episode significantly to – on average – one step per episode. In contrast, agents pursuing

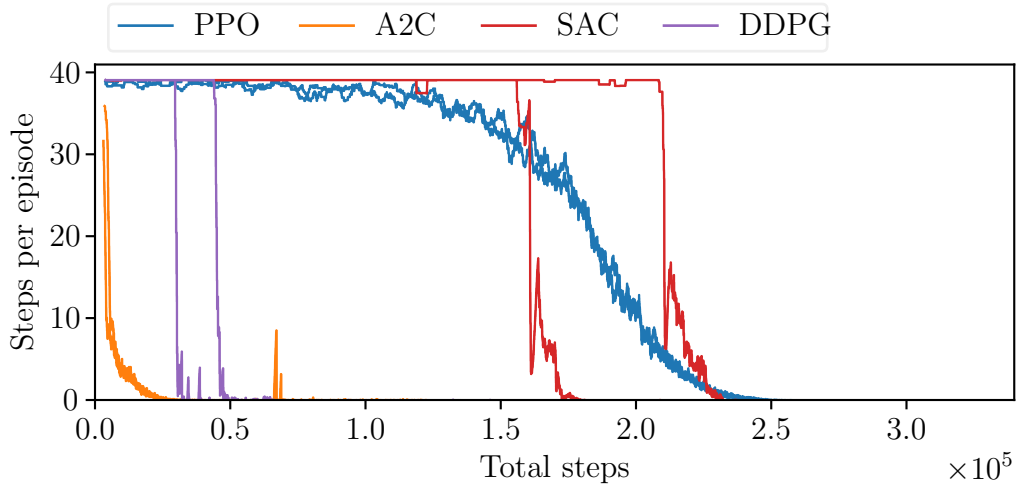


Figure 5.14.: Comparison of different agents trained to optimize the converging channel geometry following a direct strategy with respect to the steps per episode over the trained steps. Each run was repeated twice as indicated using the same color.

an incremental approach still need more than 30 steps per episode, however, starting from initially 100 steps, which were selected as a preliminary termination criterion. In the plot

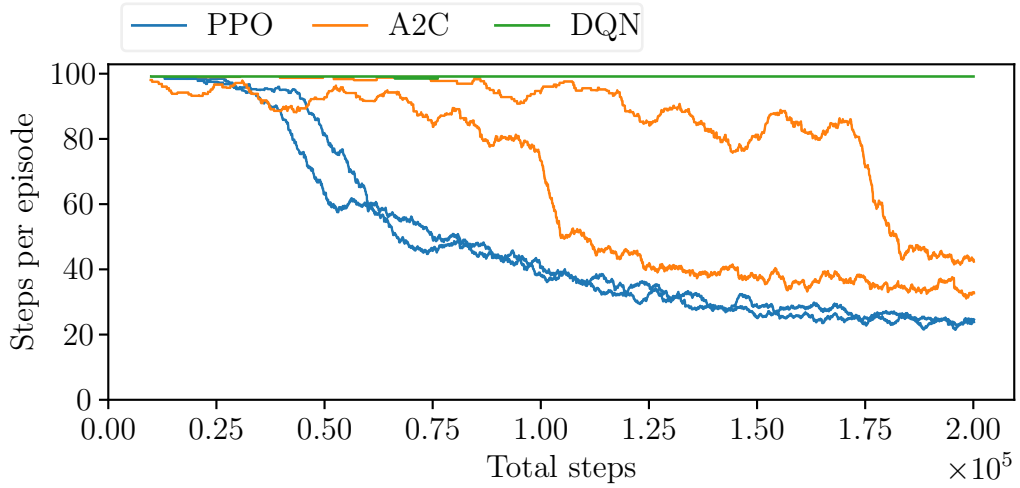


Figure 5.15.: Comparison of different agents trained to optimize the converging channel geometry following an incremental strategy with respect to the steps per episode over the trained steps. Each run was repeated twice as indicated using the same color.

for the incremental shape optimization strategy, we can also see that the algorithms are indeed not yet fully converged since they are still gradually decreasing the required steps per episode.

This shows that both strategies can be successfully employed to solve the posed shape optimization problems. However, a clear statement of which optimization approach is generally preferable for our application cannot be made despite the extensive numerical experiments performed in this thesis. This becomes especially obvious when comparing

the PPO and A2C agent: When using a PPO agent, it generally converges faster with the incremental strategy than with the direct optimization approach. This, however, does not hold true for the A2C agent, where the behavior greatly varies depending on the problem under consideration.

5.3.6. Vectorized Environment Training

After investigating the benefit of utilizing vectorized environments on the T-shaped geometry, we also test it here for the converging channel test case. As seen in the previous sections, the PPO agent trained with an incremental strategy once again proves to be relatively stable during training and is therefore again chosen for conducting these experiments. Every run trains a PPO agent for 125 k steps since it is assumed that the PPO agent should be converged in that interval (which seems a reasonable assumption, given the learning curve in Figure 5.13). The results of the vectorized environment training for

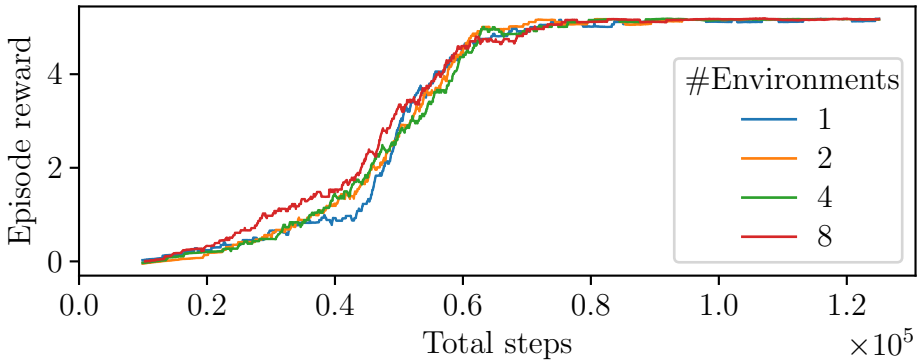


Figure 5.16.: Episode reward over training steps of a PPO agent following an incremental strategy for optimizing the converging channel geometry when interacting with different numbers of environments.

the converging channel test case are presented in Figure 5.16 and generally show the same tendency as seen for the T-shaped geometry in Figure 5.7. The training progress made per step is once again very similar between the different experiments. When comparing the wall times of the different training runs, we observe a significant reduction when utilizing multiple vectorized environments as shown in Figure 5.17. Using eight environments, the wall time was reduced from roughly 28 h to roughly 5 h, corresponding to a reduction of roughly 82 %. With this, we can conclude that the utilization of vectorized environments is use case independent and, thus, an important tool in the fast iterative development of new application cases.

It should be noted that the vectorized environment training did not scale completely linearly for our application. This means that despite a significant decrease in wall time, the used core hours[2] increase disproportionately with the utilized environments. This can partly be attributed to the increased overhead as well as to communication bottlenecks, i.e., some environments might need to wait for new instructions from the agent while it communicates with another environment or updates its policy. In addition, the scaling

[2] accumulated time consumed by all processor cores utilized in the computation

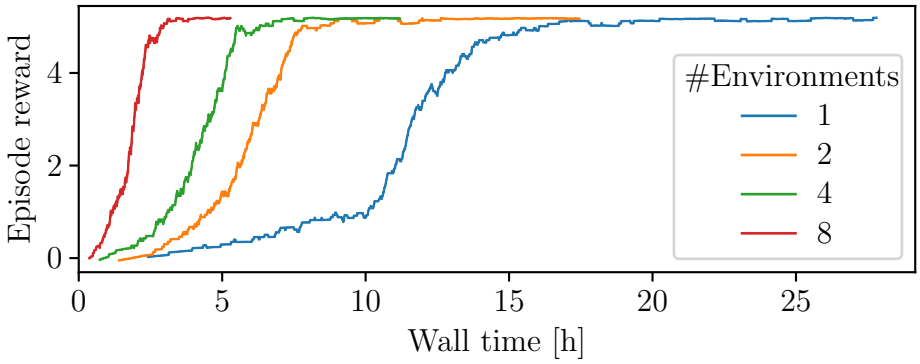


Figure 5.17.: Episode reward over wall time of a PPO agent following an incremental strategy for optimizing the converging channel geometry when interacting with different numbers of environments.

between both experiments with eight vectorized environments, i.e., comparing the scaling obtained for the T-shaped geometry to the scaling of the converging channel, differs. Therefore, Research Question (II c) can be answered qualitatively, while a quantitative answer needs further investigation.

6 Conclusion and Outlook

This chapter concludes the thesis by summarizing its most important contributions in the view of the research questions posed in Chapter 1. Afterward, we will close this chapter by pointing out directions for further continuing the research performed in this work.

The first part of this thesis was focused on the application of Physics-Informed Neural Networks (PINNs) as Reduced Order Models (ROMs), which was also formulated in Research Question (RQ I). This research question was substantiated for two distinct application cases. First, we were concerned with the applicability of PINNs as ROM in the presence of complex constitutive laws (Research Question (I a)). Although we only considered the shear-thinning flow in a rectangular two-dimensional flow channel, we were able to show the deficiencies of a naive application of PINNs to this simplified problem. In fact, no meaningful predictions were obtained, when all contributions to the PINN's loss functions were weighted equally. This behavior was attributed to the significantly different orders of magnitude inherent to the different loss components in real-world applications. As a consequence, the optimization algorithm proposed parameter updates that were biased towards reducing the components with the largest magnitude (i.e., the momentum residuals), while completely neglecting components of smaller magnitude (i.e., the inflow and wall Boundary Conditions (BCs)). Subsequently to this discovery, a heuristic to overcome this problem was proposed by scaling the losses based on a non-dimensionalization of the governing conservation laws such that they are initially roughly equal. Using this heuristic, reasonably good predictions were obtained, i.e., the absolute pointwise deviation with respect to a numerical high-fidelity solution obtained from an Finite Element Method (FEM) simulation was below $0.0006\,\mathrm{m\,s^{-1}}$ in the vast majority of the computational domain. The comparison of velocity profiles at different locations along the flow channel as well as the pressure gradient in flow direction turned out to be in very good agreement with the high-fidelity data. The biggest deviations occurred in the vicinity of the inflow, which can be explained with the fact that the prescribed inflow velocity profile corresponded to a Newtonian fluid and does not fulfill the momentum equations in the case of a shear-thinning constitutive model. Thus, we believe that the errors can even be reduced further if a better inflow BC is chosen. To answer Research Question (I a), it can be constituted that PINNs can serve as ROM for profile extrusion applications if the training process is designed such that all physical constraints are equally respected.

Second, we investigated PINNs for the prediction of flow fields in bioreactors, which is an especially challenging task due to the development of complex flow patterns in the vicinity of intricate geometric features (Research Question (I b)). Since one is usually interested in constructing a ROM for multi-query scenarios, i.e., parameterized problems, we aimed to train a PINN to predict the velocity and pressure inside a two-dimensional bioreactor geometry over a range of different Reynolds numbers. While — in contrast to the previous application — the PINN was able to capture the flow field qualitatively, the deviations were quantitatively quite large and amounted up to more than 16 % relative to the impeller's tip speed. The largest errors arose in the regions behind the baffles, where vortices are expected to form. As mentioned in Section 1.2, these are essential to the mixing capabilities of a reactor, rendering the trained Neural Network (NN) inappropriate as reliable ROM. Moreover, the first model was trained on the full geometry without exploiting its symmetry, resulting in asymmetric predictions due to the stochasticity of the learning process. Building up on these findings, we restricted ourselves on predicting the flow field for a single Reynolds number with the aim of significantly improving the prediction accuracy for that case. Using insights obtained from the numerical high-fidelity solution, a domain decomposition approach was employed, where the governing equations were reduced to a system of Ordinary Differential Equations in a region around the impeller and the full Navier-Stokes equations were only learned in the outer part close to the baffles. Moreover, the equations were formulated in cylindrical coordinates as they seemed a more natural choice with respect to the considered circular reactor cross-section. Additionally, as opposed to the previous model, the partitioned model considered the symmetry of the problem and only needed to learn a prediction in a quarter of the computational domain. With these modifications, the maximum errors were reduced by roughly 50 % to a maximum deviation of slightly more than 6 % which still might seem too high for a generating trustworthy predictions. However, a closer investigation revealed that the areas with the largest errors were confined to the wake of the impeller blades, especially, the error in the regions behind the baffles was significantly lower than for the parameterized model. This localization of the errors and the fact that relative deviation amounts to less than 1 % on average allows us to also answer Research Question (I b) positively. With the proposed domain decomposition approach, PINNs can predict the flow field in bioreactors accurately enough despite the formation of complex flow pattern to be employed as ROM.

Within the second part of this thesis publication, we have extensively investigated how numerous factors influence the learning behavior of different Reinforcement Learning (RL) algorithms for optimizing flow channels in profile extrusion dies.

To answer Research Question (II a), we have shown that RL offers a promising perspective for repeatedly solving similar optimization tasks. In both our investigated application cases, a fully trained RL agent using an incremental approach, choosing from 36 discrete actions, was able to optimize a geometry in less than 50 steps, whereas most likely even a gradient-based optimization algorithm would require more steps to obtain a solution. More extremely, an RL agent trained following a direct strategy obtained an optimal geometry in a single step. However, the benefits of a trained agent need to be weighted against the time required for training and we cannot make a general statement, which of the presented approaches is preferable.

Addressing the second Research Question (II b), we have shown that — given an appropriate reward function — all tested algorithms were able to find a strategy for generating optimal shapes. While we cannot make a general statement about the performance of di-

rect compared to incremental shape optimization, agents following a direct strategy overall exhibited a better convergence behavior. Among the employed algorithms, we found Proximal Policy Optimization (PPO) to be most reliable in terms of consistency among repeated runs and convergence behavior in general, i.e., PPO always managed to learn an effective strategy. With respect to convergence, PPO was only outperformed by Deep Deterministic Policy Gradient (DDPG). This algorithm exhibited a steep learning curve for both application cases and converged in a training time comparable to that of the PPO agent. Furthermore, it also showed good agreement among repeated runs. Advantage Actor Critic (A2C) was the only agent to find a way for exploiting the shaped reward function when optimizing the T-shaped geometry with a direct approach. However, for the test case of the converging channel, it converged fastest with respect to training time, highlighting the importance of properly shaping the reward function. Deep Q-Network (DQN) showed only a slow learning progression for the T-shaped geometry but did not learn anything for the converging channel geometry, rendering it less attractive for our application. As in the case of A2C, adaptions of the utilized reward function might cure this poor convergence. The learning curves of all Soft Actor Critic (SAC) agents exhibited a single sharp jump at distinctly different training steps, thus lacking confident reproducibility. This behavior was attributed to a not-optimal learning rate, as we only used the default hyper-parameters of the agents implemented in `stable-baselines3`.

Concluding the second part of this thesis, we showed with respect to Research Question (II c) that training in vectorized environments significantly reduces the wall-time. While the scaling of this approach can be expected to vary depending on the used framework, the algorithm, and the shape optimization approach, for the course of this thesis, we restricted ourselves to a PPO agent, following an incremental shape optimization approach. Under these settings, the method proposed by Rabault and Kuhnle [135] successfully worked on both geometries, achieving speed-ups of roughly 4.6 for the T-shaped geometry and even 5.6 for the converging channel.

After this summary, we can now briefly point out possible directions for future work. In this thesis, only two-dimensional examples have been considered for both methodologies. One improvement overarching both parts is consequently the extension to three dimensions. In the case of PINNs, this is challenging as we need to increase the number of sampling points significantly to cover the higher-dimensional input space. For shape optimization via RL, the number of Degrees of Freedom increases substantially as the geometry parameterization needs to allow movement along an additional dimension. Additionally, the FEM simulation performed in every step inside the environment becomes more memory- and time-consuming. In both cases, ultimately increased runtimes can be expected. While this is not critical in the case of PINNs where the NNs were trained in less than one hour, the RL-based optimizations already took multiple days for the two-dimensional cases, essentially rendering the application to three-dimensional cases infeasible with the current framework.

With respect to the application of PINNs as ROMs, the experiments conducted in Section 4.2 revealed a discrepancy between the strong Partial Differential Equation residuals that form the loss function and the actual error of the PINN prediction with respect to the true solution. Here, it would be interesting to examine if the loss function behaves similarly in different applications and if this behavior can somehow be remedied, e.g., by employing a different loss formulation or by using more elaborate sampling strategies which aim to

specifically generate samples in regions of high residual errors [31, 181]. Moreover, the proposed heuristic leaves room for further research, i.e., it would be interesting to see if it can be successfully applied to other problems, e.g., the bioreactor application case where the weights were still selected manually. With respect to this application, the next step would be to combine the results of Section 4.3.1, i.e., extend the domain decomposition model to additional parametric dimensions.

When considering RL for the shape optimization of profile extrusion dies, the motivation was to employ the same policy for optimizing not only a single geometry but multiple ones to amortize the excessively long training time. It remains subject of further research to see if a trained agent is capable of optimizing multiple geometries. Yet, to enable this further work needs to be dedicated to formulate a more general learning task when optimizing with respect to a state-of-the-art design criterion. Moreover, one could investigate ways to provide more expressive observations to the agent, i.e., information from the whole flow field or selected regions of interest thereof, making the learning process less dependent on the employed geometry parameterization. As a final idea, one could revisit the reward shaping: The reward functions used in the presented experiments were always fixed before the agent was trained, i.e., so far, no comparison of different reward functions has been performed. However, as pointed out before, shaping an appropriate reward function is crucial for the learning progression of and agent.

While the findings of this thesis undoubtedly revealed more questions that can be addressed by future research, we believe that the reported results provide a valuable contribution to the field of Scientific Machine Learning by demonstrating how classical engineering sciences can benefit from emerging Machine Learning methods.

A Additional material for Chapter 2

Derivation of the weak formulation (2.8) Starting from Equation (2.2), we multiply with test functions q and $\boldsymbol{w}$ for pressure and velocity. This yields the following weak form

Find $(p, \boldsymbol{v}) \in \mathcal{Q} \times \mathcal{S}$ *such that*

$$\int q \boldsymbol{\nabla} \cdot \boldsymbol{v} \, \mathrm{d}\Omega = 0 \,,$$
$$\int \left(\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \boldsymbol{\nabla}) \boldsymbol{v} \right) \cdot \boldsymbol{w} \, \mathrm{d}\Omega - \int (\boldsymbol{\nabla} \cdot \boldsymbol{\sigma}) \cdot \boldsymbol{w} \, \mathrm{d}\Omega = \int \rho \boldsymbol{b} \cdot \boldsymbol{w} \, \mathrm{d}\Omega \,,$$

for all $(q, \boldsymbol{w}) \in \mathcal{Q} \times \mathcal{V}$.

In the next step, we employ Gauss's theorem to transform the derivatives of the Cuachy stress tensor onto the test function:

Find $(p, \boldsymbol{v}) \in \mathcal{Q} \times \mathcal{S}$ *such that*

$$\int q \boldsymbol{\nabla} \cdot \boldsymbol{v} \, \mathrm{d}\Omega = 0 \,,$$
$$\int \left(\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \boldsymbol{\nabla}) \boldsymbol{v} \right) \cdot \boldsymbol{w} \, \mathrm{d}\Omega$$
$$- \oint (\boldsymbol{\sigma} \cdot \boldsymbol{n}) \cdot \boldsymbol{w} \, \mathrm{d}\Gamma + \int \boldsymbol{\sigma} : \boldsymbol{\nabla} \boldsymbol{w} \, \mathrm{d}\Omega = \int \rho \boldsymbol{b} \cdot \boldsymbol{w} \, \mathrm{d}\Omega \,,$$

for all $(q, \boldsymbol{w}) \in \mathcal{Q} \times \mathcal{V}$.

The boundary integral can now be simplified using the specified boundary conditions:

Find $(p, \boldsymbol{v}) \in \mathcal{Q} \times \mathcal{S}$ *such that*

$$\begin{aligned} \int q \boldsymbol{\nabla} \cdot \boldsymbol{v} \, \mathrm{d}\Omega = 0 \, , \\ \int \left(\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \boldsymbol{\nabla}) \boldsymbol{v} \right) \cdot \boldsymbol{w} \, \mathrm{d}\Omega \\ + \oint \boldsymbol{f} \cdot \boldsymbol{w} \, \mathrm{d}\Gamma_{\mathrm{N}} + \int \boldsymbol{\sigma} : \boldsymbol{\nabla} \boldsymbol{w} \, \mathrm{d}\Omega = \int \rho \boldsymbol{b} \cdot \boldsymbol{w} \, \mathrm{d}\Omega \, , \end{aligned}$$

for all $(q, \boldsymbol{w}) \in \mathcal{Q} \times \mathcal{V}$.

Adding up both equations, one obtains the weak formulation that is implemented in our FEM solver:

Find $(p, \boldsymbol{v}) \in \mathcal{Q} \times \mathcal{S}$ *such that*

$$\begin{aligned} \int q \boldsymbol{\nabla} \cdot \boldsymbol{v} \, \mathrm{d}\Omega + \int \left(\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \boldsymbol{\nabla}) \boldsymbol{v} \right) \cdot \boldsymbol{w} \, \mathrm{d}\Omega \\ + \oint \boldsymbol{f} \cdot \boldsymbol{w} \, \mathrm{d}\Gamma_{\mathrm{N}} + \int \boldsymbol{\sigma} : \boldsymbol{\nabla} \boldsymbol{w} \, \mathrm{d}\Omega = \int \rho \boldsymbol{b} \cdot \boldsymbol{w} \, \mathrm{d}\Omega \, , \end{aligned}$$

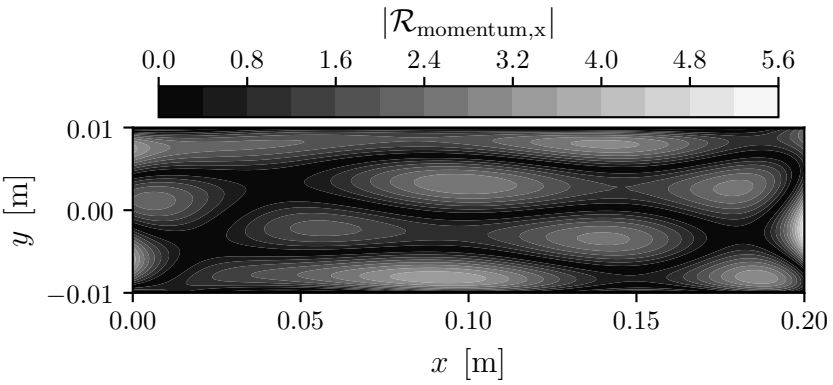
for all $(q, \boldsymbol{w}) \in \mathcal{Q} \times \mathcal{V}$.

B

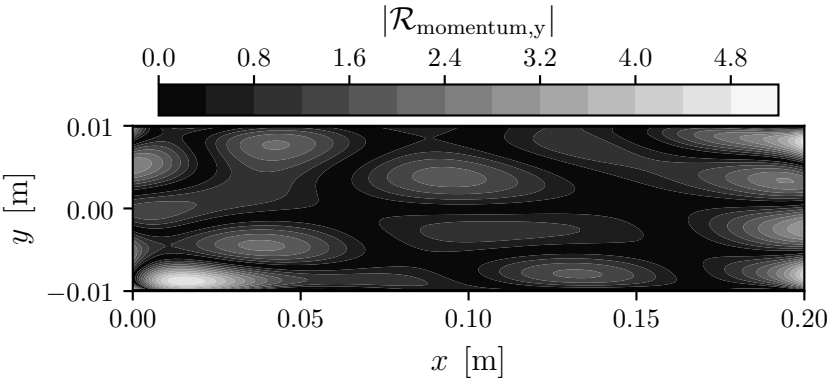
Additional material for Chapter 4

This chapter contains additional material for the chapter on PINNs as ROMs (Chapter 4).

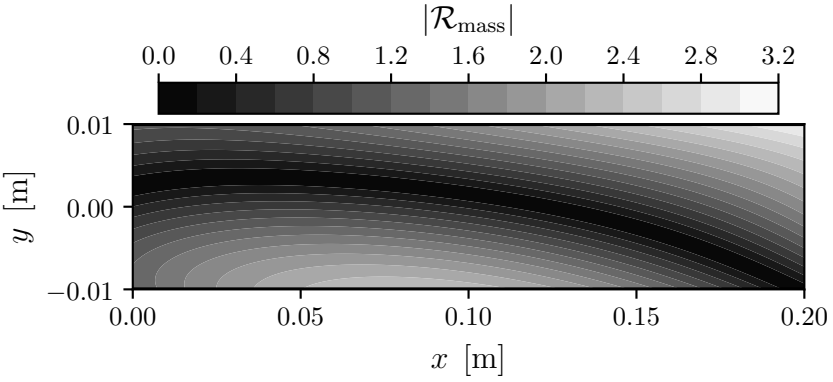
B.1. Additional material for Section 4.2.1



(a) x-momentum residual.



(b) y-momentum residual.



(c) Mass residual.

Figure B.1.: Strong PDE residuals for the trained model presented in section 4.2.1.

B.2. Additional material for Section 4.2.2

Derivation of the expression for the reference pressure (4.6) The analytical solution for the Poiseuille flow of a Newtonian fluid in a channel with height h reads

$$v_{\mathrm{x}}(y) = \frac{1}{2\mu}\frac{\partial p}{\partial x}\left(y^2 - \left(\frac{h}{2}\right)^2\right) ,$$

where the x-axis is aligned with the symmetry axis of the channel. The peak velocity is then obtained at $y = 0$, i.e.,

$$v_{\mathrm{x,max}} = v_{\mathrm{x}}(0) = -\frac{h^2}{8\mu}\frac{\partial p}{\partial x} .$$

This yields the following expression for the pressure gradient

$$\frac{\partial p}{\partial x} = -\frac{8\mu}{h^2} v_{\mathrm{x,max}} .$$

We know that for the case of a Poiseuille flow the pressure decays linearly along the channel, i.e., (assuming a channel length l) we have $\frac{\partial p}{\partial x} \approx -\frac{\Delta p}{l}$. If we moreover define the pressure difference Δp with respect to the pressure at the end of the channel, i.e., $p|_{x=l} = 0$, we obtain the expression for the reference pressure that is also stated in Equation (4.6)

$$p_{\mathrm{ref}} = \frac{8\mu}{h^2} v_{\mathrm{x,max}} l .$$

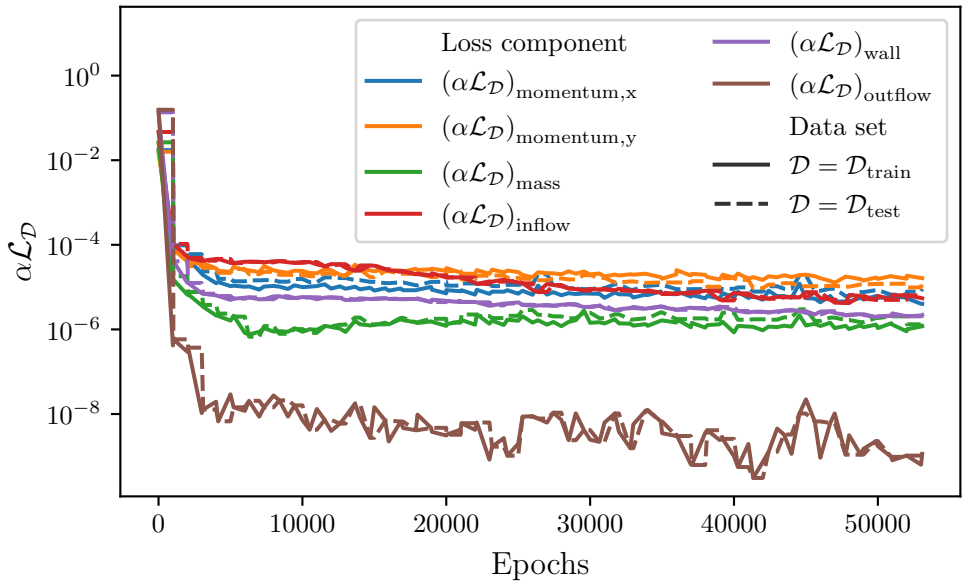
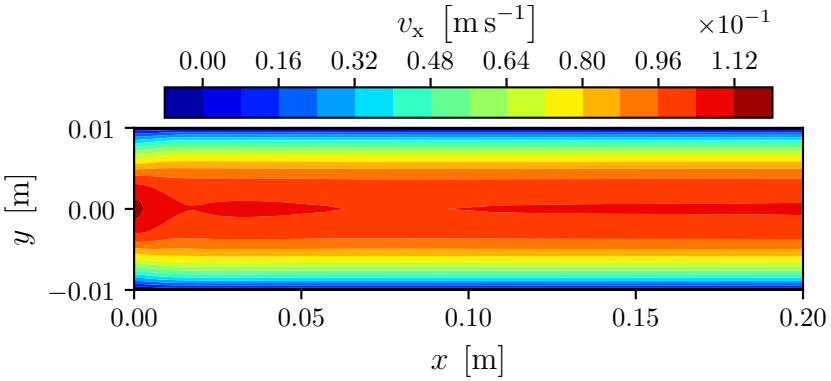
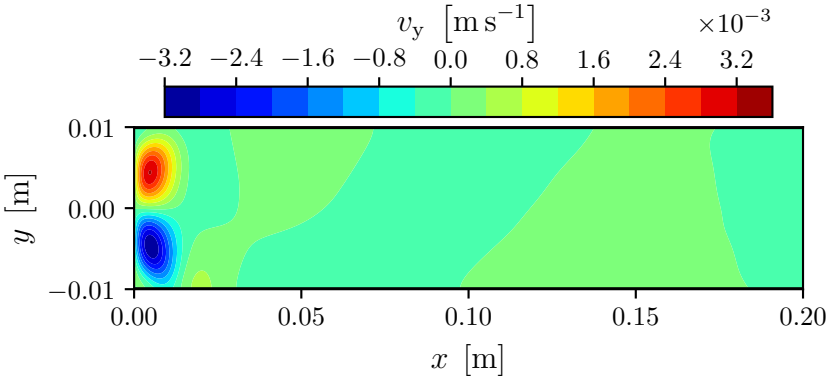


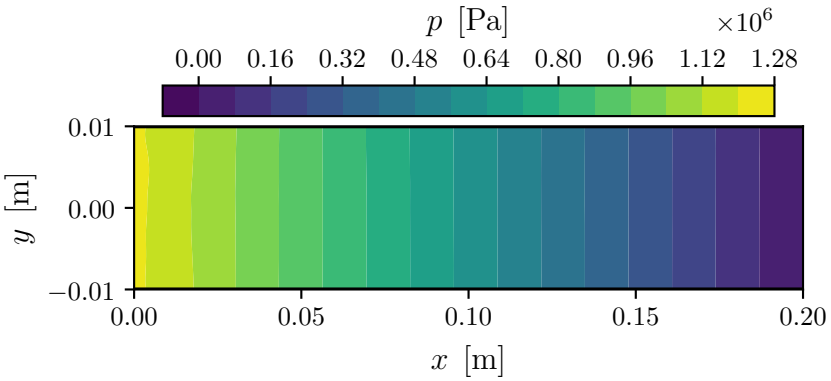
Figure B.2.: Convergence of the loss function components with respect to the number of trained epochs. At the end of training, the loss curves flatten out, indicating convergence.



(a) x-velocity field.



(b) y-velocity field.



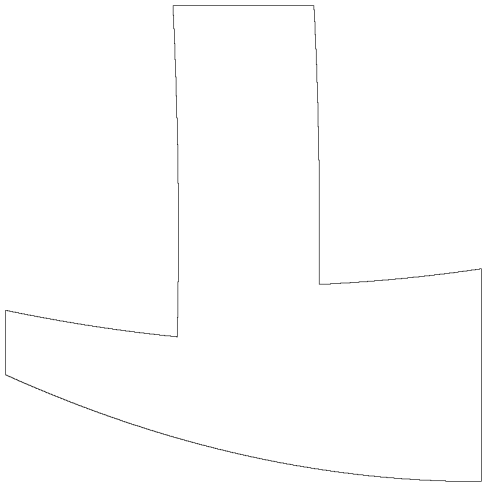
(c) Pressure field.

Figure B.3.: PINN predictions for the solution fields after training for 50000 epochs with loss function weights chosen according to the heuristic presented in section 4.2.2.

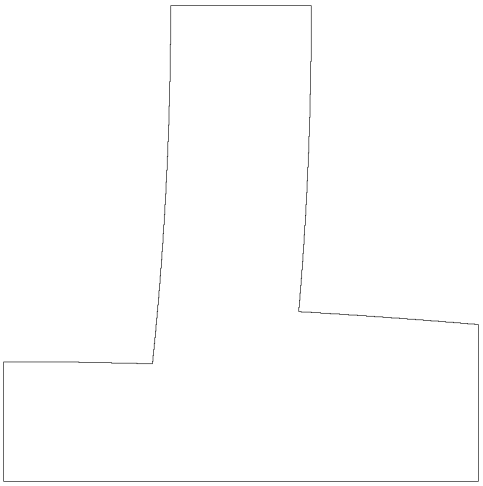
C

Additional material for Chapter 5

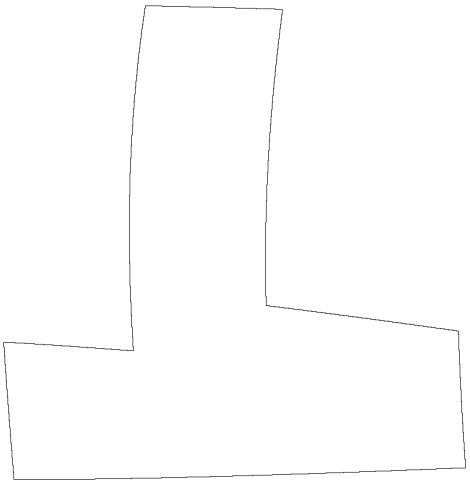
C.1. Additional material for Section 5.2.4



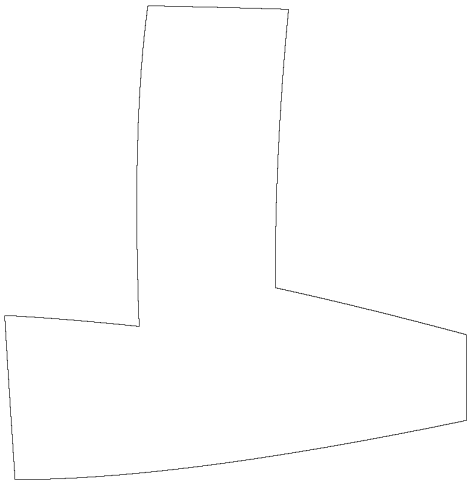
(a) Geometry obtained for desired mass-flow ratio of $\mu^\star = 0.2$.



(b) Geometry obtained for desired mass-flow ratio of $\mu^\star = 0.6$.



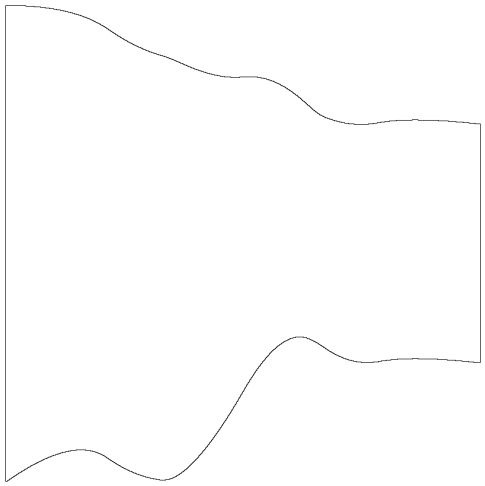
(c) Geometry obtained for desired mass-flow ratio of $\mu^\star = 0.9$.



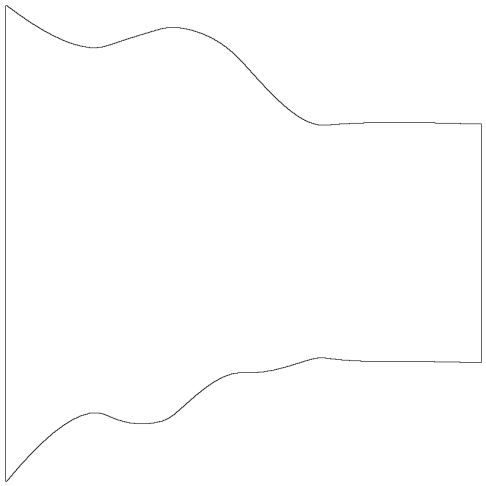
(d) Geometry obtained for desired mass-flow ratio of $\mu^\star = 1.4$.

Figure C.1.: Examples of optimal geometries obtained by a PPO agent following an incremental optimization strategy on the T-shaped geometry. One can see that the trained agent has learned a valid strategy which involves modifying the control points that strongly influence the cross-sectional area of the two outflows.

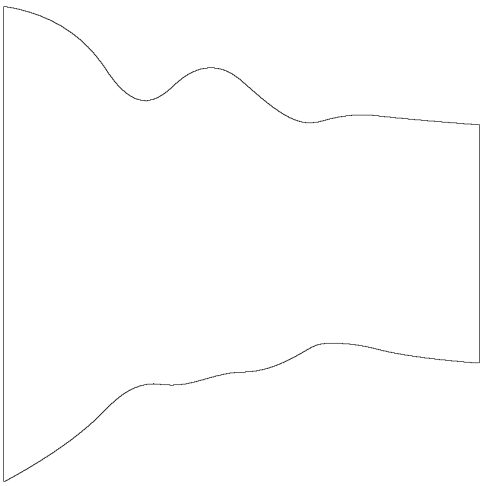
C.2. Additional material for Section 5.3.4



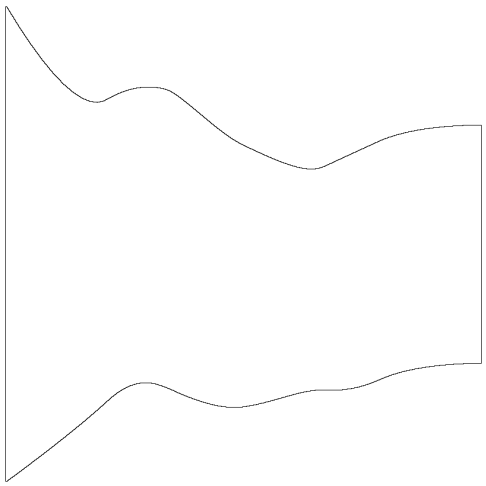
(a) Optimized geometry obtained in episode 274.



(b) Optimized geometry obtained in episode 1760.



(c) Optimized geometry obtained in episode 3127.



(d) Optimized geometry obtained in episode 3597.

Figure C.2.: Examples of optimal geometries obtained by a PPO agent following an incremental optimization strategy for the converging channel geometry. Each of the shown geometries has been generated from a random initial geometry generated by a random perturbation of the control points. Qualitatively the agent has learned to contract the channel as smoothly as possible to assert an optimal flow homogeneity.

Bibliography

[1] M. Abadi et al. "TensorFlow: A system for large-scale machine learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283 (cit. on p. 44).

[2] A. Ahamed and P. Vermette. "Enhanced enzyme production from mixed cultures of Trichoderma reesei RUT-C30 and Aspergillus niger LMA grown as fed batch in a stirred tank bioreactor". In: *Biochemical Engineering Journal* 42.1 (2008), pp. 41–46. ISSN: 1369703X. DOI: 10.1016/j.bej.2008.05.007 (cit. on p. 3).

[3] T. Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2019), pp. 2623–2631. DOI: 10.1145/3292500.3330701 (cit. on p. 46).

[4] K. Arulkumaran et al. "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38. ISSN: 10535888. DOI: 10.1109/MSP.2017.2743240 (cit. on p. 2).

[5] H. Azegami. *Shape Optimization Problems*. Vol. 164. Springer Optimization and Its Applications. Singapore: Springer Singapore, 2020. ISBN: 978-981-15-7618-8. DOI: 10.1007/978-981-15-7618-8 (cit. on p. 6).

[6] N. Baker et al. *Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence*. Tech. rep. 1. USDOE Office of Science (SC) (United States), Feb. 2019, pp. 1–109. DOI: 10.2172/1484362 (cit. on pp. 17, 18).

[7] A. G. Baydin et al. "Automatic Differentiation in Machine Learning: a Survey". In: *Journal of Machine Learning Research* 18 (2018), pp. 1–43. DOI: https://doi.org/10.48550/arXiv.1502.05767 (cit. on p. 33).

[8] Y. Bengio and Y. Lecun. "Scaling Learning Algorithms toward AI". In: *Large-Scale Kernel Machines* 1 (2019), pp. 1–41. DOI: 10.7551/mitpress/7496.003.0016 (cit. on p. 28).

[9] P. Benner et al. *Snapshot-Based Methods and Algorithms*. Ed. by P. Benner et al. Vol. 2. Berlin, Boston: De Gruyter, Jan. 2021. ISBN: 9783110671490. DOI: doi:10.1515/9783110671490 (cit. on p. 5).

[10] P. Benner et al. *System- and Data-Driven Methods and Algorithms*. Ed. by P. Benner et al. Vol. 1. Berlin, Boston: De Gruyter, Oct. 2021. ISBN: 9783110498967. DOI: doi:10.1515/9783110498967 (cit. on p. 5).

[11] J. Bergstra, D. Yamins, and D. D. Cox. "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *International Conference on Machine Learning*. Vol. 28. 1. 2013, pp. 115–123 (cit. on p. 46).

[12] A. Bērziņš et al. *Standardized Non-Intrusive Reduced Order Modeling Using Different Regression Models With Application to Complex Flow Problems*. 2021 (cit. on p. 1).

[13] M. Binder. "Shape Optimization based on Reinforcement Learning". Master Thesis. TU Wien, 2021 (cit. on p. 75).

[14] C. Bishop. *Neural Network for Pattern Recognition*. Oxford University Press, 1995 (cit. on pp. 18, 20).

[15] C. Bishop. *Pattern Recognition and Machine Learning*. Springer Berlin Heidelberg, 2006. ISBN: 9780387310732 (cit. on pp. 16, 20, 27, 30).

[16] L. Bottou and O. Bousquet. "The Tradeoffs of Large-Scale Learning". In: *Optimization for Machine Learning*. Vol. 20. The MIT Press, 2011, pp. 161–168. DOI: 10.7551/mitpress/8996.003.0015 (cit. on pp. 25, 27).

[17] G. Brockman et al. *OpenAI Gym*. 2016 (cit. on p. 77).

[18] A. N. Brooks and T. J. Hughes. "Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations". In: *Computer Methods in Applied Mechanics and Engineering* 32.1-3 (Sept. 1982), pp. 199–259. ISSN: 00457825. DOI: 10.1016/0045-7825(82)90071-8 (cit. on p. 13).

[19] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. "Machine Learning for Fluid Mechanics". In: *Annual Review of Fluid Mechanics* 52 (2020), pp. 477–508. ISSN: 00664189. DOI: 10.1146/annurev-fluid-010719-060214 (cit. on p. 5).

[20] L. Buşoniu et al. "Reinforcement learning for control: Performance, stability, and deep approximators". In: *Annual Reviews in Control* 46 (2018), pp. 8–28. ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2018.09.005 (cit. on p. 75).

[21] S. Cai et al. "DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks". In: *Journal of Computational Physics* 436 (2021), pp. 1–17. ISSN: 10902716. DOI: 10.1016/j.jcp.2021.110296 (cit. on p. 34).

[22] S. Cai et al. *Physics-informed neural networks (PINNs) for fluid mechanics: A review*. 2021 (cit. on pp. 33, 44).

[23] P. J. Carreau. "Rheological Equations from Molecular Network Theories". In: *Transactions of the Society of Rheology* 16.1 (Mar. 1972), pp. 99–127. ISSN: 0038-0032. DOI: 10.1122/1.549276 (cit. on p. 10).

[24] Y. Chen et al. "Physics-informed neural networks for inverse problems in nano-optics and metamaterials". In: *Optics Express* 28.8 (Apr. 2020), pp. 11618–11633. DOI: 10.1364/OE.384875 (cit. on p. 33).

[25] F. Chinesta, R. Keunings, and A. Leygue. *The Proper Generalized Decomposition for Advanced Numerical Simulations*. SpringerBriefs in Applied Sciences and Technology. Cham: Springer International Publishing, 2014. ISBN: 978-3-319-02864-4. DOI: 10.1007/978-3-319-02865-1 (cit. on p. 5).

[26] P.-Y. Chuang and L. Barba. "Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration". In: *Proceedings of the 21st Python in Science Conference* Scipy (2022), pp. 28–36 (cit. on pp. 1, 36, 48).

[27] C. Cortes and V. Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. ISSN: 0885-6125. DOI: 10.1007/BF00994018 (cit. on p. 18).

[28] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs. *Isogeometric Analysis*. Chichester, UK: John Wiley & Sons, Ltd, Aug. 2009. ISBN: 9780470749081. DOI: 10.1002/9780470749081 (cit. on p. 79).

[29] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274 (cit. on pp. 18, 26).

[30] Y. N. Dauphin et al. "Identifying and Attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2933–2941 (cit. on p. 29).

[31] A. Daw et al. *Mitigating Propagation Failures in PINNs using Evolutionary Sampling*. 2022 (cit. on p. 104).

[32] A. E. Deane et al. "Low-dimensional models for complex geometry flows: Application to grooved channels and circular cylinders". In: *Physics of Fluids A* 3.10 (1991), pp. 2337–2354. ISSN: 08998213. DOI: 10.1063/1.857881 (cit. on p. 5).

[33] O. Delalleau and Y. Bengio. "Shallow vs. deep sum-product networks". In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011* (2011), pp. 1–9 (cit. on p. 28).

[34] N. Demo, M. Strazzullo, and G. Rozza. *An extended physics informed neural network for preliminary analysis of parametric optimal control problems*. 2021 (cit. on p. 33).

[35] N. Dirkes. "Advanced Strategies for Improving Physics-Informed Neural Networks as Reduced Simulation Models". Master Thesis. RWTH Aachen University, 2022 (cit. on pp. 44, 69, 70, 72).

[36] J. Donea and A. Huerta. *Finite Element Methods for Flow Problems*. Wiley, Apr. 2003, pp. 28–29. ISBN: 9780471496663. DOI: 10.1002/0470013826 (cit. on pp. 9, 11, 12).

[37] F. Dworschak et al. "Reinforcement Learning for Engineering Design Automation". In: *Advanced Engineering Informatics* 52.April (2022), p. 101612. ISSN: 14740346. DOI: 10.1016/j.aei.2022.101612 (cit. on p. 75).

[38] M. Eichinger, A. Heinlein, and A. Klawonn. "Surrogate convolutional neural network models for steady computational fluid dynamics simulations". In: *ETNA - Electronic Transactions on Numerical Analysis* 56 (2022), pp. 235–255. ISSN: 1068-9613. DOI: 10.1553/etna_vol56s235 (cit. on p. 39).

[39] S. Elgeti et al. “Numerical shape optimization as an approach to extrusion die design”. In: *Finite Elements in Analysis and Design* 61 (2012), pp. 35–43. ISSN: 0168874X. DOI: 10.1016/j.finel.2012.06.008 (cit. on pp. 3, 91).

[40] L. C. Evans. *Partial Differential Equations*. Graduate studies in mathematics. American Mathematical Society, 2010. ISBN: 9780821849743 (cit. on p. 36).

[41] R. W. Fox, P. J. Pritchard, and A. T. McDonald. *Introduction to Fluid Mechanics*. 8th. John Wiley & Sons, 2010. ISBN: 9780470547557 (cit. on p. 56).

[42] L. P. Franca and S. L. Frey. “Stabilized finite element methods: II. The incompressible Navier-Stokes equations”. In: *Computer Methods in Applied Mechanics and Engineering* 99.2-3 (1992), pp. 209–233. ISSN: 00457825. DOI: 10.1016/0045-7825(92)90041-H (cit. on p. 13).

[43] C. Fricke. “Shape Optimization based on Reinforcement Learning”. Seminar Thesis. RWTH Aachen University, 2021 (cit. on pp. 75, 77).

[44] C. Fricke. “Improving Reinforcement Learning for Shape Optimization of Profile Extrusion Die Flow Channels through Multi-Environment Training and Custom Flow Field Feature Extractors”. Master Thesis. RWTH Aachen University, 2022 (cit. on p. 75).

[45] C. Fricke et al. “Investigation of reinforcement learning for shape optimization of 2D profile extrusion die geometries”. In: *Advances in Computational Science and Engineering* (2023), pp. 1–35. DOI: 10.3934/acse.2023001 (cit. on p. 75).

[46] H. Gao, L. Sun, and J. X. Wang. “PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain”. In: *Journal of Computational Physics* 428 (2021), pp. 1–57. ISSN: 10902716. DOI: 10.1016/j.jcp.2020.110079 (cit. on p. 39).

[47] P. Garnier et al. “A review on deep reinforcement learning for fluid mechanics”. In: *Computers and Fluids* 225 (2021). ISSN: 00457930. DOI: 10.1016/j.compfluid.2021.104973 (cit. on p. 76).

[48] H. Ghraieb et al. “Single-step deep reinforcement learning for two- and three-dimensional optimal shape design”. In: *AIP Advances* 12.8 (2022), p. 085108. ISSN: 21583226. DOI: 10.1063/5.0097241 (cit. on p. 76).

[49] X. Glorot and Y. Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Journal of Machine Learning Research* 9 (2010), pp. 249–256. ISSN: 15324435 (cit. on p. 29).

[50] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 15, 17, 18, 24, 27–30, 45).

[51] A. Griewank and A. Walther. *Evaluating Derivatives*. Society for Industrial and Applied Mathematics, Jan. 2008. ISBN: 978-0-89871-659-7. DOI: 10.1137/1.9780898717761 (cit. on p. 33).

[52] X. Guo, W. Li, and F. Iorio. “Convolutional Neural Networks for Steady Flow Approximation”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 481–490. ISBN: 9781450342322. DOI: 10.1145/2939672.2939738 (cit. on p. 39).

[53] T. Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *35th International Conference on Machine Learning, ICML 2018* 5 (2018), pp. 2976–2989 (cit. on p. 41).

[54] E. Haghighat and R. Juanes. “SciANN: A Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 373 (2021). ISSN: 00457825. DOI: 10.1016/j.cma.2020.113552 (cit. on p. 33).

[55] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY: Springer New York, 2009. ISBN: 978-0-387-84857-0. DOI: 10.1007/978-0-387-84858-7 (cit. on pp. 27–30, 52).

[56] K. He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE International Conference on Computer Vision* 2015 International Conference on Computer Vision, ICCV 2015 (2015), pp. 1026–1034. ISSN: 15505499. DOI: 10.1109/ICCV.2015.123 (cit. on p. 29).

[57] O. Hennigh et al. “NVIDIA SimNet™: An AI-Accelerated Multi-Physics Simulation Framework”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12746 LNCS (2021), pp. 447–461. ISSN: 16113349. DOI: 10.1007/978-3-030-77977-1_36 (cit. on p. 33).

[58] J. S. Hesthaven, G. Rozza, and B. Stamm. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. 2015, pp. 1–131. ISBN: 9783319224701. DOI: 10.1007/978-3-319-22470-1 (cit. on p. 5).

[59] C. Hopmann and W. Michaeli. *Extrusion Dies for Plastics and Rubber*. 4th. München: Carl Hanser Verlag GmbH & Co. KG, Oct. 2016, p. 469. ISBN: 978-1-56990-623-1. DOI: 10.3139/9781569906248 (cit. on pp. 2, 3, 10, 89).

[60] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8 (cit. on pp. 18, 26).

[61] Z. Hu et al. “When Do Extended Physics-Informed Neural Networks (XPINNs) Improve Generalization?” In: *SIAM Journal on Scientific Computing* 44.5 (2022), A3158–A3182. ISSN: 1064-8275. DOI: 10.1137/21m1447039 (cit. on p. 34).

[62] G. B. Huang. “Learning capability and storage capacity of two-hidden-layer feedforward networks”. In: *IEEE Transactions on Neural Networks* 14.2 (2003), pp. 274–281. ISSN: 10459227. DOI: 10.1109/TNN.2003.809401 (cit. on p. 28).

[63] A. G. Ivakhnenko, V. G. Lapa, and R. N. McDonough. *Cybernetics and forecasting techniques*. American Elsevier, NY, 1967 (cit. on p. 18).

[64] A. D. Jagtap and G. E. Karniadakis. “Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations”. In: *CEUR Workshop Proceedings* 2964.11 (2021). ISSN: 16130073 (cit. on pp. 33, 34, 69).

[65] A. D. Jagtap and G. E. Karniadakis. *How important are activation functions in regression and classification? A survey, performance comparison, and future directions*. 2022 (cit. on p. 20).

[66] A. D. Jagtap, K. Kawaguchi, and G. Em Karniadakis. "Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 476.2239 (2020). ISSN: 14712946. DOI: 10.1098/rspa.2020.0334rspa20200334 (cit. on p. 48).

[67] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis. "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems". In: *Computer Methods in Applied Mechanics and Engineering* 365 (2020), p. 113028. ISSN: 00457825. DOI: 10.1016/j.cma.2020.113028 (cit. on pp. 33, 34, 69).

[68] L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement Learning: A Survey". In: *Journal of Artificial Intelligence Research* 4.1 (Apr. 1996), pp. 237–285. ISSN: 21945365 (cit. on p. 2).

[69] G. E. Karniadakis et al. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440. ISSN: 25225820. DOI: 10.1038/s42254-021-00314-5 (cit. on pp. 32, 33, 35).

[70] N. S. Keskar et al. "On large-batch training for deep learning: Generalization gap and sharp minima". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2017), pp. 1–16 (cit. on p. 24).

[71] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. *Variational Physics-Informed Neural Networks For Solving Partial Differential Equations*. 2019 (cit. on p. 33).

[72] D. P. Kingma and J. L. Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), pp. 1–15 (cit. on pp. 24, 25).

[73] J. Kober and J. Peters. "Reinforcement Learning in Robotics: A Survey". In: *Learning Motor Skills: From Algorithms to Robot Experiments*. Cham: Springer International Publishing, 2014, pp. 9–67. ISBN: 978-3-319-03194-1. DOI: 10.1007/978-3-319-03194-1_2 (cit. on p. 75).

[74] V. R. Konda and J. N. Tsitsiklis. "Actor-critic algorithms". In: *Advances in Neural Information Processing Systems* (2000), pp. 1008–1014. ISSN: 10495258 (cit. on p. 42).

[75] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. Vol. 1. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105 (cit. on p. 18).

[76] P. L. Lagari et al. "Systematic Construction of Neural Forms for Solving Partial Differential Equations Inside Rectangular Domains, Subject to Initial, Boundary and Interface Conditions". In: *International Journal on Artificial Intelligence Tools* 29.5 (2020), pp. 1–12. ISSN: 17936349. DOI: 10.1142/S0218213020500098 (cit. on p. 37).

[77] I. E. Lagaris, A. Likas, and D. I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations". In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000. ISSN: 10459227. DOI: 10.1109/72.712178 (cit. on p. 30).

[78] U. Lämmel and J. Cleve. *Künstliche Intelligenz: Wissensverarbeitung – Neuronale Netze*. 5th. Wismar, 2020. ISBN: 9783446459144 (cit. on pp. 1, 16).

[79] A. Lampton, A. Niksch, and J. Valasek. "Morphing airfoils with four morphing parameters". In: *AIAA Guidance, Navigation and Control Conference and Exhibit* August (2008). DOI: 10.2514/6.2008-7282 (cit. on p. 76).

[80] L. D. Landau and E. M. Lifshitz. *Fluid Mechanics*. 2nd. Vol. 6. Oxford: Pergamon Press, 1987 (cit. on pp. 11, 38).

[81] J. Larson, M. Menickelly, and S. M. Wild. "Derivative-free optimization methods". In: *Acta Numerica* 28 (2019), pp. 287–404. ISSN: 14740508. DOI: 10.1017/S0962492919000060 (cit. on pp. 6, 23).

[82] T. Lassila et al. "Model Order Reduction in Fluid Dynamics: Challenges and Perspectives". In: *Reduced Order Methods for Modeling and Computational Reduction*. Cham: Springer International Publishing, 2014, pp. 235–273. DOI: 10.1007/978-3-319-02090-7_9 (cit. on p. 5).

[83] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541 (cit. on p. 18).

[84] R. Leiteritz and D. Pflüger. *How to Avoid Trivial Solutions in Physics-Informed Neural Networks*. 2021 (cit. on p. 1).

[85] M. Leshno et al. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". In: *Neural Networks* 6.6 (1993), pp. 861–867. ISSN: 08936080. DOI: 10.1016/S0893-6080(05)80131-5 (cit. on pp. 18, 26).

[86] R. Li, Y. Zhang, and H. Chen. "Learning the Aerodynamic Design of Supercritical Airfoils Through Deep Reinforcement Learning". In: *AIAA Journal* 59.10 (2021), pp. 3988–4001. ISSN: 0001-1452. DOI: 10.2514/1.j060189 (cit. on p. 76).

[87] S. Li and X. Feng. "Dynamic Weight Strategy of Physics-Informed Neural Networks for the 2D Navier–Stokes Equations". In: *Entropy* 24.9 (2022). ISSN: 10994300. DOI: 10.3390/e24091254 (cit. on p. 38).

[88] T. P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2016) (cit. on p. 41).

[89] C. Lin et al. "Operator learning for predicting multiscale bubble growth dynamics". In: *Journal of Chemical Physics* 154.10 (2021). ISSN: 10897690. DOI: 10.1063/5.0041203 (cit. on p. 34).

[90] H. W. Lin, M. Tegmark, and D. Rolnick. "Why Does Deep and Cheap Learning Work So Well?" In: *Journal of Statistical Physics* 168.6 (2017), pp. 1223–1247. ISSN: 00224715. DOI: 10.1007/s10955-017-1836-5 (cit. on p. 28).

[91] J. Ling, A. Kurzawski, and J. Templeton. "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance". In: *Journal of Fluid Mechanics* 807 (2016), pp. 155–166. ISSN: 14697645. DOI: 10.1017/jfm.2016.615 (cit. on p. 1).

[92] D. C. Liu and J. Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical Programming* 45.1-3 (Aug. 1989), pp. 503–528. ISSN: 0025-5610. DOI: 10.1007/BF01589116 (cit. on p. 25).

[93] L. Lu et al. "DeepXDE: A deep learning library for solving differential equations". In: *CEUR Workshop Proceedings* 2587 (July 2019), pp. 1–21. ISSN: 16130073. DOI: 10.1137/19M1274067 (cit. on pp. 25, 27, 33, 38, 44, 47).

[94] L. Lu et al. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". In: *Nature Machine Intelligence* 3.3 (2021), pp. 218–229. ISSN: 25225839. DOI: 10.1038/s42256-021-00302-5 (cit. on p. 34).

[95] L. Lu et al. "Physics-Informed Neural Networks with Hard Constraints for Inverse Design". In: *SIAM Journal on Scientific Computing* 43.6 (2021), B1105–B1132. DOI: 10.1137/21M1397908 (cit. on p. 37).

[96] H. Lubjuhn. "Physics-Informed Neural Networks as Reduced Simulation Models". Seminar Thesis. RWTH Aachen University, 2021 (cit. on p. 44).

[97] H. Lubjuhn. "Physics-Informed Neural Networks for Predicting the Flow Field in Bioreactors". Master Thesis. RWTH Aachen University, 2022 (cit. on pp. 44, 65).

[98] S. Luo et al. "Parameter Identification of RANS Turbulence Model Using Physics-Embedded Neural Network". In: *High Performance Computing*. Ed. by H. Jagode et al. Cham: Springer International Publishing, 2020, pp. 137–149. ISBN: 978-3-030-59851-8 (cit. on p. 33).

[99] C. Lv, L. Wang, and C. Xie. "A hybrid physics-informed neural network for nonlinear partial differential equation". In: *International Journal of Modern Physics C* (2022). ISSN: 01291831. DOI: 10.1142/S0129183123500821 (cit. on p. 65).

[100] C. W. Macosko. *Rheology: Principles, Measurements, and Applications*. Wiley-VCH, New York, 1994, p. 576 (cit. on p. 10).

[101] C.-F. Mandenius. *Bioreactors*. Ed. by C.-F. Mandenius. Weinheim, Germany: Wiley-VCH Verlag GmbH & Co. KGaA, Mar. 2016. ISBN: 9783527683369. DOI: 10.1002/9783527683369 (cit. on pp. 3–5).

[102] A. Manzoni, A. Quarteroni, and S. Salsa. *Optimal Control of Partial Differential Equations*. Vol. 207. Applied Mathematical Sciences. Cham: Springer International Publishing, 2021. ISBN: 978-3-030-77225-3. DOI: 10.1007/978-3-030-77226-0 (cit. on pp. 5, 6).

[103] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. "Physics-informed neural networks for high-speed flows". In: *Computer Methods in Applied Mechanics and Engineering* 360.March (2020), p. 112789. ISSN: 00457825. DOI: 10.1016/j.cma.2019.112789 (cit. on p. 33).

[104] Z. Mao et al. “DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators”. In: *Journal of Computational Physics* 447 (2021), pp. 1–37. ISSN: 10902716. DOI: 10.1016/j.jcp.2021.110698 (cit. on p. 34).

[105] C. C. Margossian. “A review of automatic differentiation and its efficient implementation”. In: *WIREs Data Mining and Knowledge Discovery* 9.4 (July 2019), pp. 1–32. ISSN: 1942-4787. DOI: 10.1002/widm.1305 (cit. on p. 33).

[106] J. McCarthy et al. “A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955”. In: *AI Magazine* 27.4 SE - Articles (Dec. 2006). DOI: 10.1609/aimag.v27i4.1904 (cit. on pp. 1, 16).

[107] R. van der Meer, C. W. Oosterlee, and A. Borovykh. “Optimally weighted loss functions for solving PDEs with Neural Networks”. In: *Journal of Computational and Applied Mathematics* 405 (2022), pp. 1–28. ISSN: 03770427. DOI: 10.1016/j.cam.2021.113887 (cit. on p. 38).

[108] J. M. Melenk. “On n-Widths for Elliptic Problems”. In: *Journal of Mathematical Analysis and Applications* 247.1 (2000), pp. 272–289. ISSN: 0022247X. DOI: 10.1006/jmaa.2000.6862 (cit. on p. 37).

[109] M. Minsky and S. A. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, Sept. 2017. ISBN: 9780262343930. DOI: 10.7551/mitpress/11301.001.0001 (cit. on p. 18).

[110] S. Mishra and R. Molinaro. “Estimates on the generalization error of physics-informed neural networks for approximating PDEs”. In: *IMA Journal of Numerical Analysis* (2022), pp. 1–36. ISSN: 0272-4979. DOI: 10.1093/imanum/drab093 (cit. on pp. 27, 35).

[111] T. Mitchell. *Machine Learning*. McGraw-Hill International Editions. New York: McGraw-Hill, 1997. ISBN: 9780071154673 (cit. on p. 16).

[112] V. Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013 (cit. on pp. 41, 75).

[113] V. Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *33rd International Conference on Machine Learning, ICML 2016* 4 (2016), pp. 2850–2869 (cit. on p. 41).

[114] A.-R. Mohamed, G. Dahl, and G. Hinton. “Deep belief networks for phone recognition”. In: *Nips workshop on deep learning for speech recognition and related applications*. Vol. 1. 9. 2009, pp. 1–9 (cit. on p. 18).

[115] R. Mojgani, M. Balajewicz, and P. Hassanzadeh. “Kolmogorov n–width and Lagrangian physics-informed neural networks: A causality-conforming manifold for convection-dominated PDEs”. In: *Computer Methods in Applied Mechanics and Engineering* 404 (Feb. 2023), p. 115810. ISSN: 00457825. DOI: 10.1016/j.cma.2022.115810 (cit. on pp. 37, 38).

[116] M. Möller, D. Toshniwal, and F. v. Ruiten. *Physics-informed machine learning embedded into isogeometric analysis* (cit. on p. 39).

[117] G. Montavon, G. B. Orr, and K.-R. Müller. *Neural Networks: Tricks of the Trade*. Ed. by G. Montavon, G. B. Orr, and K.-R. Müller. Lecture Notes in Computer Science 2. Berlin, Heidelberg: Springer Berlin Heidelberg, May 2012. ISBN: 978-3-642-35288-1. DOI: 10.1007/978-3-642-35289-8 (cit. on pp. 29, 52).

[118] M. A. Nabian and H. Meidani. "Physics-Driven Regularization of Deep Neural Networks for Enhanced Engineering Design and Analysis". In: *Journal of Computing and Information Science in Engineering* 20.1 (2020), pp. 1–10. ISSN: 1530-9827. DOI: 10.1115/1.4044507 (cit. on p. 35).

[119] U. Naumann. *The Art of Differentiating Computer Programs*. 2011. ISBN: 9781611972061. DOI: 10.1137/1.9781611972078 (cit. on p. 33).

[120] T. Nguyen, M. Raghu, and S. Kornblith. *Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth*. 2021 (cit. on p. 28).

[121] P. Niyogi and F. Girosi. "Generalization bounds for function approximation from scattered noisy data". In: *Advances in Computational Mathematics* 10.1 (1999), pp. 51–80. ISSN: 10197168. DOI: 10.1023/A:1018966213079 (cit. on pp. 25, 27).

[122] J. M. Nóbrega et al. "Flow balancing in extrusion dies for thermoplastic profiles, Part III: Experimental assessment". In: *International Polymer Processing* 19.3 (2004), pp. 225–235. ISSN: 0930777X. DOI: 10.3139/217.1825 (cit. on p. 3).

[123] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006, pp. 1–664. ISBN: 978-0-387-30303-1. DOI: 10.1007/978-0-387-40065-5 (cit. on pp. 6, 25).

[124] A. Offersgaard et al. "Sars-cov-2 production in a scalable high cell density bioreactor". In: *Vaccines* 9.7 (2021). ISSN: 2076393X. DOI: 10.3390/vaccines9070706 (cit. on p. 3).

[125] T. Osswald and N. Rudolph. *Polymer Rheology*. München: Carl Hanser Verlag GmbH & Co. KG, Nov. 2015, p. 237. ISBN: 978-1-56990-517-3. DOI: 10.3139/9781569905234 (cit. on pp. 3, 10).

[126] R. Owens and T. Phillips. *Computational Rheology*. Computational Rheology. Imperial College Press, 2002. ISBN: 9781860941863 (cit. on p. 10).

[127] S. Ozturk and W. S. Hu. *Cell Culture Technology for Pharmaceutical and Cell-Based Therapies*. 2005, pp. 1–769. ISBN: 9780849351068. DOI: 10.1201/9780849351068 (cit. on pp. 4, 5).

[128] L.-M. Papaspyridi et al. "Production of bioactive metabolites with pharmaceutical and nutraceutical interest by submerged fermentation of Pleurotus ostreatus in a batch stirred tank bioreactor". In: *Procedia Food Science* 1 (2011), pp. 1746–1752. ISSN: 2211601X. DOI: 10.1016/j.profoo.2011.09.257 (cit. on p. 3).

[129] L. Piegl and W. Tiller. *The NURBS Book*. Monographs in Visual Communications. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995. ISBN: 978-3-642-97387-1. DOI: 10.1007/978-3-642-97385-7 (cit. on p. 79).

[130] O. Pironneau. *Finite element methods for fluids*. eng. Chichester: Wiley, 1989. ISBN: 0471922552 (cit. on p. 12).

[131] J. F. Pittman. "Computer-aided design and optimization of profile extrusion dies for thermoplastics and rubber: A review". In: *Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering* 225.4 (2011), pp. 280–321. ISSN: 09544089. DOI: 10.1177/0954408911415324 (cit. on pp. 2, 3).

[132] D. C. Psichogios and L. H. Ungar. "A hybrid neural network-first principles approach to process modeling". In: *AIChE Journal* 38.10 (1992), pp. 1499–1511. ISSN: 15475905. DOI: 10.1002/aic.690381003 (cit. on p. 30).

[133] S. Qin et al. "Multi-objective optimization of cascade blade profile based on reinforcement learning". In: *Applied Sciences (Switzerland)* 11.1 (2021), pp. 1–27. ISSN: 20763417. DOI: 10.3390/app11010106 (cit. on p. 76).

[134] L. Quartapelle. *Numerical Solution of the Incompressible Navier-Stokes Equations*. Basel: Birkhäuser Basel, 1993. ISBN: 978-3-0348-9689-4. DOI: 10.1007/978-3-0348-8579-9 (cit. on p. 11).

[135] J. Rabault and A. Kuhnle. "Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach". In: *Physics of Fluids* 31.9 (2019). ISSN: 10897666. DOI: 10.1063/1.5116415 (cit. on pp. 42, 103).

[136] J. Rabault et al. "Deep reinforcement learning in fluid mechanics: A promising method for both active flow control and shape optimization". In: *Journal of Hydrodynamics* 32.2 (2020), pp. 234–246. ISSN: 18780342. DOI: 10.1007/s42241-020-0028-y (cit. on p. 75).

[137] A. Raffin et al. "Stable-baselines3: Reliable reinforcement learning implementations". In: *Journal of Machine Learning Research* 22 (2021), pp. 1–8. ISSN: 15337928 (cit. on p. 42).

[138] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 10902716. DOI: 10.1016/j.jcp.2018.10.045 (cit. on p. 30).

[139] M. Raissi, P. Perdikaris, and G. E. Karniadakis. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. 2017 (cit. on p. 30).

[140] M. Raissi, P. Perdikaris, and G. E. Karniadakis. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. 2017 (cit. on p. 30).

[141] A. Rajkumar et al. "Guidelines for balancing the flow in extrusion dies: The influence of the material rheology". In: *Journal of Polymer Engineering* 38.2 (2018), pp. 197–211. ISSN: 03346447. DOI: 10.1515/polyeng-2016-0449 (cit. on pp. 91, 92).

[142] C. Rao, H. Sun, and Y. Liu. "Physics-informed deep learning for incompressible laminar flows". In: *Theoretical and Applied Mechanics Letters* 10.3 (2020), pp. 207–212. ISSN: 20950349. DOI: 10.1016/j.taml.2020.01.039 (cit. on pp. 36, 38).

[143] B. Reyes et al. "Learning unknown physics of non-Newtonian fluids". In: *Physical Review Fluids* 6.7 (2021), pp. 1–12. ISSN: 2469990X. DOI: 10.1103/PhysRevFluids.6.073301 (cit. on p. 33).

[144] E. Rich. *Artificial intelligence*. McGraw-Hill, Inc., 1983 (cit. on p. 16).

[145] R. V. Roman and M. Gavrilescu. "Oxygen transfer efficiency in the biosynthesis of antibiotics in bioreactors with a modified RUSHTON turbine agitator". In: *Acta Biotechnologica* 14.2 (1994), pp. 181–192. ISSN: 15213846. DOI: 10.1002/abio.370140212 (cit. on p. 3).

[146] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 0033295X. DOI: 10.1037/h0042519 (cit. on pp. 17–19).

[147] F. Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY, 1961 (cit. on p. 20).

[148] A. Rosseburg et al. "Hydrodynamic inhomogeneities in large scale stirred tanks – Influence on mixing time". In: *Chemical Engineering Science* 188 (2018), pp. 208–220. ISSN: 00092509. DOI: 10.1016/j.ces.2018.05.008 (cit. on p. 63).

[149] G. Rozza et al. "Advances in reduced order methods for parametric industrial problems in computational fluid dynamics". In: *Proceedings of the 6th European Conference on Computational Mechanics: Solids, Structures and Coupled Problems, ECCM 2018 and 7th European Conference on Computational Fluid Dynamics, ECFD 2018* (2018), pp. 59–76 (cit. on p. 5).

[150] S. Ruder. *An overview of gradient descent optimization algorithms*. 2017 (cit. on pp. 24, 25).

[151] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 0028-0836. DOI: 10.1038/323533a0 (cit. on pp. 18, 23).

[152] F. Sahli Costabal et al. "Physics-Informed Neural Networks for Cardiac Activation Mapping". In: *Frontiers in Physics* 8.February (2020), pp. 1–12. ISSN: 2296424X. DOI: 10.3389/fphy.2020.00042 (cit. on p. 33).

[153] S. Salsa. *Partial Differential Equations in Action*. Vol. 86. UNITEXT. Cham: Springer International Publishing, 2015. ISBN: 978-3-319-15092-5. DOI: 10.1007/978-3-319-15093-2 (cit. on p. 36).

[154] J. Schulman et al. *Proximal Policy Optimization Algorithms*. 2017 (cit. on p. 41).

[155] T. W. Sederberg and S. R. Parry. "Free-form deformation of solid geometric models". In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1986* 20.4 (1986), pp. 151–160. DOI: 10.1145/15922.15903 (cit. on p. 79).

[156] P. Sermanet et al. "Pedestrian detection with unsupervised multi-stage feature learning". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2013), pp. 3626–3633. ISSN: 10636919. DOI: 10.1109/CVPR.2013.465 (cit. on p. 18).

[157] Z. Shen, H. Yang, and S. Zhang. "Optimal approximation rate of ReLU networks in terms of width and depth". In: *Journal des Mathematiques Pures et Appliquees* 157 (2022), pp. 101–135. ISSN: 00217824. DOI: 10.1016/j.matpur.2021.07.009 (cit. on p. 26).

[158] H. Sheng and C. Yang. "PFNN: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries". In: *Journal of Computational Physics* 428 (2021), p. 110085. ISSN: 00219991. DOI: 10.1016/j.jcp.2020.110085 (cit. on p. 37).

[159] Y. Shin, J. Darbon, and G. E. Karniadakis. "On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs". In: *Communications in Computational Physics* 28.5 (2020), pp. 2042–2074. ISSN: 19917120. DOI: 10.4208/CICP.OA-2020-0193 (cit. on pp. 27, 35).

[160] Y. Shin, Z. Zhang, and G. E. Karniadakis. *Error estimates of residual minimization using neural networks for linear PDEs*. 2020 (cit. on p. 35).

[161] K. Shukla, A. D. Jagtap, and G. E. Karniadakis. "Parallel physics-informed neural networks via domain decomposition". In: *Journal of Computational Physics* 447 (2021). ISSN: 10902716. DOI: 10.1016/j.jcp.2021.110683 (cit. on pp. 34, 69).

[162] K. Shukla et al. *Deep neural operators can serve as accurate surrogates for shape optimization: A case study for airfoils*. 2023 (cit. on p. 34).

[163] R. Siegbert et al. "Comparing Optimization Algorithms for Shape Optimization of Extrusion Dies". In: *PAMM* 14.1 (Dec. 2014), pp. 789–794. ISSN: 16177061. DOI: 10.1002/pamm.201410377 (cit. on p. 3).

[164] J. W. Siegel. *Optimal Approximation Rates for Deep ReLU Neural Networks on Sobolev Spaces*. 2023 (cit. on p. 26).

[165] D. Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144. DOI: 10.1126/science.aar6404 (cit. on p. 75).

[166] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018 (cit. on p. 42).

[167] R. Swischuk et al. "Projection-based model reduction: Formulations for physics-based machine learning". In: *Computers and Fluids* 179.November (2019), pp. 704–717. ISSN: 00457930. DOI: 10.1016/j.compfluid.2018.07.021 (cit. on p. 1).

[168] T. E. Tezduyar et al. "Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity-pressure elements". In: *Computer Methods in Applied Mechanics and Engineering* 95.2 (1992), pp. 221–242. ISSN: 00457825. DOI: 10.1016/0045-7825(92)90141-6 (cit. on p. 13).

[169] T. Tezduyar. "Stabilized Finite Element Formulations for Incompressible Flow Computations". In: *13th Computational Fluid Dynamics Conference*. Vol. 28. 1991, pp. 1–44. ISBN: 0120020289. DOI: 10.1016/S0065-2156(08)70153-4 (cit. on p. 13).

[170] O. Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354. ISSN: 14764687. DOI: 10.1038/s41586-019-1724-z (cit. on p. 75).

[171] J. Viquerat, P. Meliga, and E. Hachem. "A review on deep reinforcement learning for fluid mechanics: An update". In: *Physics of Fluids* 34.11 (2022), p. 111301. DOI: 10.1063/5.0128446 (cit. on pp. 76, 78).

[172] J. Viquerat et al. "Direct shape optimization through deep reinforcement learning". In: *Journal of Computational Physics* 428 (2021). ISSN: 10902716. DOI: 10.1016/j.jcp.2020.110080 (cit. on p. 76).

[173] J. Viquerat et al. "Policy-based optimization: single-step policy gradient method seen as an evolution strategy". In: *Neural Computing and Applications* 35.1 (2023), pp. 449–467. DOI: 10.1007/s00521-022-07779-0 (cit. on p. 76).

[174] S. Walczak and N. Cerpa. "Heuristic principles for the design of artificial neural networks". In: *Information and Software Technology* 41.2 (1999), pp. 107–117. ISSN: 09505849. DOI: 10.1016/S0950-5849(98)00116-5 (cit. on p. 28).

[175] S. Wang, Y. Teng, and P. Perdikaris. "Understanding and mitigating gradient flow pathologies in physics-informed neural networks". In: *SIAM Journal on Scientific Computing* 43.5 (2021), pp. 3055–3081. ISSN: 10957197. DOI: 10.1137/20M1318043 (cit. on pp. 1, 36, 38, 48).

[176] S. Wang, X. Yu, and P. Perdikaris. "When and why PINNs fail to train: A neural tangent kernel perspective". In: *Journal of Computational Physics* 449 (2022), pp. 1–29. ISSN: 10902716. DOI: 10.1016/j.jcp.2021.110768 (cit. on pp. 37, 38).

[177] Z. Wang et al. "Model identification of reduced order fluid dynamics systems using deep learning". In: *International Journal for Numerical Methods in Fluids* 86.4 (2018), pp. 255–268. ISSN: 10970363. DOI: 10.1002/fld.4416 (cit. on p. 1).

[178] E. Weinan and B. Yu. "The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems". In: *Communications in Mathematics and Statistics* 6.1 (2018), pp. 1–12. ISSN: 2194671X. DOI: 10.1007/s40304-018-0127-z (cit. on p. 39).

[179] P. H. Winston. *Artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., 1984 (cit. on p. 16).

[180] D. Wolff et al. "Towards shape optimization of flow channels in profile extrusion dies using reinforcement learning". In: *PAMM* 22.1 (2023), pp. 1–6. DOI: https://doi.org/10.1002/pamm.202200009 (cit. on pp. 81, 84).

[181] C. Wu et al. "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 403 (2023), pp. 1–23. ISSN: 00457825. DOI: 10.1016/j.cma.2022.115671 (cit. on pp. 38, 47, 104).

[182] X. Yan et al. "Aerodynamic shape optimization using a novel optimizer based on machine learning techniques". In: *Aerospace Science and Technology* 86 (2019), pp. 826–835. ISSN: 12709638. DOI: 10.1016/j.ast.2019.02.003 (cit. on p. 76).

[183] O. Yilmaz, H. Gunes, and K. Kirkkopru. "Optimization of a profile extrusion die for flow balance". In: *Fibers and Polymers* 15.4 (2014), pp. 753–761. ISSN: 12299197. DOI: 10.1007/s12221-014-0753-3 (cit. on p. 3).

[184] M. Yin et al. “Non-invasive inference of thrombus material properties with physics-informed neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 375 (2021), p. 113603. ISSN: 00457825. DOI: 10.1016/j.cma.2020.113603 (cit. on p. 33).