Approaching Partial Differential Equations with Physics-Driven Deep Learning

DISSERTATION

zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

NILS WANDEL

aus
Heidelberg

Bonn 2023

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter:	Prof. Dr. Reinhard Klein
2. Gutachter:	Prof. Dr. Jürgen Gall
Tag der Promotion:	27.10.2023
Erscheinungsjahr:	2023

Acknowledgements

First and foremost, I want to thank Prof. Dr. Reinhard Klein for giving me the opportunity to work in his lab and to pursue a Dr. rer. nat. under his supervision. Second, I want to thank Prof. Dr. Michael Weinmann for his great support regarding the achieved publications. I also want to thank Dr. Michael Neidlin for his collaboration on the Spline-PINN project. Further thanks go to Patrick for providing me with his gorgeous thesis template and all the other colleagues for the inspiring discussions during lunch breaks. Finally, I want to thank my family and Hannah for their ongoing encouragement in this endeavour.

Abstract

Partial Differential Equations (PDEs) play an important role in describing continuous physical systems such as fluids, air-flows, waves and many more. Thus, by solving these equations, one can simulate smoke and water effects in computer graphics, analyse lift and drag coefficients of vehicles in engineering applications, or compare experimental observations with predictions from fundamental theories in basic research.

However, in most scenarios, analytic solutions of PDEs are not available and traditional numerical solutions are computationally expensive. In contrast, recent deep-learning based methods promise great gains in efficiency by infering solutions in a single forward pass through a neural network and streightforwardly allow for parallelization on GPUs.

In this work, we tackle challenges of deep-learning based approaches in the context of PDEs with respect to: ground truth data generation, neural network architectures, the generalization capability of neural networks to obtain solutions for unseen domain geometries, differentiability, speed and stability.

To this end, we built on top of recent advances in *physics-driven* deep-learning that, in contrast to *data-driven* approaches, allow to train neural networks directly on the underlying PDEs and therefore alleviate the need of large ground truth datasets that are expensive to generate. We developed a novel training cycle, that, in conjunction with a physics-based loss and a training pool of randomized domains, allows end-to-end training without any precomputed ground truth data. To describe the field states of the underlying PDEs, we investigated a Marker and Cell grid representation and a continuous representation based on Hermite-splines. Our unpretentious pipeline does not rely on additional components such as a differentiable fluid solver or a particle tracer. This way, we obtained robust and accurate surrogate models that yield stable results over long time-horizons in two as well as in three dimensions. Furthermore, we developed interactive demonstrators to show the generalization capabilities of our approach in real-time. Since our models are fully differentiable, gradients can be efficiently computed throughout the simulation by using backpropagation through time. This can be exploited, for example, in inverse problems, as we have shown in a simplistic experiment on fluid control.

In the future, we hope that the insights gained in this work can form a foundation for new applications that require fast, robust and differentiable solutions of PDEs such as for example real-time physics-engines for computer games or interactive computational fluid dynamics for engineering. To this end, we share the source-code for all of our publications on github.

Contents

Part I

Introduction

Chapter 1

Introduction

Why are partial differential equations important? And why should we approach these equations with deep-learning based methods? The following chapter motivates the research field of *physics-driven deep learning* that lies at the intersection of numerical methods to solve partial differential equations and recent developments in machine learning. Furthermore, key challenges of this research field are highlighted and the contributions of this thesis are introduced.

1.1 What are PDEs and why do we need them?

Partial Differential Equations (PDEs) are special mathematical equations that put constraints on the partial derivatives of a multivariate function. They can be used for example to define the dynamics of continuous physical systems such as fluids, waves, heat distributions and many more. Continuous physical systems are usually described by multivariate functions (fields) that map domain coordinates (e.g. space and time) onto field values (e.g. velocity, pressure, height, temperature). The constraints on the partial derivatives of these fields are needed to impose physical laws such as conservation of momentum, mass or energy. More information on PDEs is provided in Section 3.1.

Solutions of such PDEs play an important role in a wide range of applications: In computer games and computer generated imagery, PDEs are required to visualize for example water or smoke effects. In engineering, PDEs are needed to complement or even replace expensive wind-tunnel experiments with computational fluid dynamics (CFD). They provide deepened insights into the velocity and pressure fields causing drag on vehicles and help out in scenarios where accurate wind-tunnel experiments are practically not feasible[1]. In basic research, fundamental theories rely on PDEs (see e.g. the Schrödinger equation for quantum mechanics or the general theory of relativity for gravity) that need to be solved in order to verify their accordance with our observations as well as to make new predictions about the universe.

[1] Imagine for example the extreme scenario of a space capsule entering the atmosphere of mars at several times the speed of sound.

1.2 Why is solving PDEs hard?

Unfortunately, for most PDEs, closed form solutions are not available. Hence, approximate solutions have to be computed numerically. However, the development of approximate PDE solvers is a complex task, and, depending on the specific application, one has to find trade-offs between various computational properties such as *speed, parallelism, memory requirements, accuracy* or *stability*. In settings such as for example optimal control or sensitivity analysis, having a *differentiable* PDE solver is an additional important property. Over the last decades many solvers have emerged for different applications with various trade-offs for the above mentioned properties and the development of new improved solvers is still an active field of ongoing research.

1.3 Why are artificial neural networks a good option for solving PDEs?

Traditional numerical solvers (such as for example finite differences, finite volumes, finite elements) are usually computationally expensive as they require solving large implicit systems of equations in order to obtain stable simulations. On the other hand, deep learning based methods can learn a heuristic or "intuition" with a neural network to provide accurate approximate solutions of PDEs. Since neural networks can infer solutions in a single forward pass and offer easy parallelization on GPUs, they are often faster and computationally significantly more efficient compared to traditional methods.

Several works (see Section 2) based on neural networks have already shown promising results for various PDEs such as Burgers equation, wave equations, Darcy flows, the Navier-Stokes equation etc. but are still limited by their generalization capability to novel simulation domain geometries, computational speed or high demands of training data.

In the future, such deep-learning based methods could allow for accelerated fluid and smoke simulations in computer games and graphics, more rapid prototyping in engineering[2] or quick checks whether physical theories are in accordance with our observations. Over the next decade, we believe that deep learning will have a similarly profound impact on the field of solving partial differential equations as deep learning already had over the past decade on numerous other fields in computer science such as for example computer vision or reinforcement learning.

1.4 What are the challenges of neural network based approaches?

Before explaining the specific design-decisions and contributions of this work in more detail in Section 1.5, we want to highlight some of the general challenges in deep learning, as well

[2] For example extrality.ai is a start-up that aims to accelerate computational fluid dynamics with deep-learning to decrease time-to-market.

as challenges that arise in particular in the context of solving PDEs.

Data representation When facing a deep-learning problem, first, a suitable data representation must be chosen. A fluid can be described for example by a set of smoothed particles with parameters such as velocity, density or shape in methods such as smoothed particle hydrodynamics that rely on a Lagrangian frame of reference (see Section 2.1). Furthermore, grid or mesh based representations are common in methods such as lattice-Boltzmann methods or finite element methods that rely on an Eulerian frame of reference (see Section 2.3). And implicit functions can be used to obtain continuous field descriptions. The wide variety of different data representations makes the field of solving PDEs and fluid dynamics in particular a fruitful ground for deep learning based approaches as these representations require various different neural network designs to exploit the underlying properties and symmetries of the data.

Neural network design Neural networks are often considered to be a kind of blackbox universal function approximator. However, choosing a network architecture that reflects the underlying data structure is crucial to achieve good performance [Bronstein et al., 2017]. For example, on grids, convolutional neural networks exploit the translational symmetry of the data. On mesh data, graph neural networks or message passing neural networks allow for efficient processing of data on nodes and edges. And on point cloud data, PointNets can handle permutation equivariance. The mentioned network architectures directly incorporate the symmetries of the underlying data representation and thus are able to make efficient use of network parameters by sharing weights across input signals. Apart from reduced memory and computational requirements, this also enables training on significantly less training data and leads to better generalization performance.

On top of these commonly used design decisions for neural network architectures in deep-learning, in fluid dynamics, several works have extended the neural network pipeline by harnessing additional components from traditional solvers such as e.g. a particle tracer to deal with the advection term [Tompson et al., 2017] or a "traditional", differentiable low resolution fluid solver [Um et al., 2020]. However, these components increase the complexity of the overall method and impede end-to-end training if differentiability of a component is not given.

In contrast to these explicit neural architectures that work directly on the field representations of the underlying PDEs, implicit neural architectures map domain coordinates to field values and thus implicitly encode field representations by their network parameters. These implicit neural architectures yield continuous results and play an important role in physics-informed neural networks, however, they are not capable to significantly generalize beyond given training domains and thus cannot be applied in interactive scenarios.

Training data and loss function When training a neural network, typically, the first step consists of gathering ground truth training data. In the context of learning partial differential equations, this ground truth training data could be real data (for example satellite images captured for weather forecasting), or, when real data is not available, synthetic data that needs to be generated by traditional PDE solvers. Since the performance of a neural network usually can not significantly surpass the ground truth data it is trained on, it is important to generate a dataset of very high quality. Furthermore, the more data is available, the better a neural network can generalize. However, generating a large and highly accurate dataset using traditional PDE solvers is computationally demanding and large real-world datasets are often not available.

Once the training data is set up, a neural network can be fit (trained) to that data by minimizing a certain error (loss) function using gradient descent. There exist various common loss terms and choosing proper ones for the loss function is a crucial part of training as it strongly impacts the final performance of the neural network. Important loss terms include: L_1 error, L_2 error, cross-entropy, adversarial loss functions in conjunction with various regularization terms and often multiple of these loss terms are combined. Recently, *physics-informed loss terms* that directly incorporate the residuals of the underlying PDEs became more and more popular in the context of learning PDEs as they allow to train with less ground truth data compared to other data-driven methods and sometimes even allow to train with no ground truth data at all.

Testing, Stability and Generalization Capability In order to test the performance of a deep learning model, typically, a certain subset of the dataset (e.g. 20 percent) is kept secret which is referred to as test data. After training the model only on the remaining training data (i.e. here: 80 percent), we can evaluate the performance of the model on the test data with respect to a specified metric and check, if our model is able to *generalize* to this unseen data or if it overfits the training data.

However, this strategy fails if a model is evaluated on chaotic PDEs over long time horizons. Solutions of chaotic PDEs diverge even after tiny initial perturbations and, thus, it is not possible to make exact predictions over long time horizons that can be compared to predictions of a respective test dataset. An important example of chaotic PDEs are turbulent fluid dynamics at high Reynolds numbers [3]. Thus, since comparing against test data on reasonably long time-horizons is not possible, we can check instead, how well the PDEs themselves are fulfilled. This can be done by investigating the residuals of the PDEs.

Another important property of PDE solvers is *stability*. This means that simulations should not diverge towards unrealistic states after many iterations. For example in fluid simulations, the solver must not introduce unrealistic high energies or pressure gradients. This can be investiged again by checking the residuals of the PDEs over time.

Finally, in order to test the generalization capability of a PDE model, it should be confronted with boundary conditions that were not captured in the training dataset.

[3] In popular literature this sensitivity to initial perturbations is often illustrated by the *butterfly effect*, which states that the flap of a butterfly could cause or prevent even a tornado on the other side of the earth when enough time has elapsed.

1.5 What are the contributions of this work?

In this work, we addressed the aforementioned challenges of deep learning in the context of solving PDEs in order to learn fast, interactive, differentiable simulations without ground truth data using neural networks. To this end, we made the following contributions regarding data representation, neural network design, training and testing:

Data representation Over the course of this thesis, we developed code for two different data representations based on uniform grids. First, we chose a Marker and Cell (MAC) grid in 2d and 3d, which allows for convenient computations of finite differences that can be used to formulate a physics-*constrained* loss. Second, we implemented a representation based on Hermite spline interpolations on a uniform grid. This yields continuous results that can be derived analytically and thus allows to compute a physics-*informed* loss.

Neural network design Since our data representations are based on uniform grids, convolutional neural network architectures are a natural design choice for our neural surrogate models. In particular, we investigated the use of a U-Net architecture [Ronneberger et al., 2015] that can be considered a multi-resolution hierarchy of convolutional neural networks. However, our approach is model agnostic and we also tested smaller architectures for increased simulation speed. Since our method does not rely on additional components from traditional solvers such as particle tracers, computing gradients with backpropagation through time (BPTT) is straight forward and allows to apply our method for example in inverse problems or fluid control scenarios. We demonstrated this in an example application where BPTT was used to control the frequency of a vortex street behind an obstacle by adjusting the flow velocity.

Training data and loss function We developed a novel training cycle that does not require any precomputed training data. To this end, we implemented a physics-constrained loss for MAC grid representations and a physics-informed loss for Hermite spline representations respectively. Furthermore, we make use of a training pool that gets filled automatically with more and more realistic data produced by the neural networks themselves as a byproduct during training.

Testing, Stability and Generalization Capability We tested our approach for the incompressible Navier-Stokes as well as the wave equation and performed quantitative comparisons to established solvers. By investigating errors over hundreds of iterations we demonstrate the stability of our method and interactive simulations show the generalization capabilities of the trained models to new domain geometries.

The contributions of this thesis have led to three peer-reviewed publications accompanied by open-source implementations:

- *Learning Incompressible Fluid Dynamics from Scratch - Towards Fast, Differentiable Fluid Models that Generalize*
  Nils Wandel, Michael Weinmann, Reinhard Klein,
  International Conference on Learning Representations (ICLR) 2021
  Code: `https://github.com/wandeln/Unsupervised_Deep_Learning_of_Incompressible_Fluid_Dynamics`
  Chapter 4 and Appendix 1 of this thesis
- *Teaching the Incompressible Navier Stokes Equations to Fast Neural Surrogate Models in 3D*
  Nils Wandel, Michael Weinmann, Reinhard Klein,
  Physics of Fluids, Volume 33
  Code: `https://github.com/wandeln/Teaching_Incompressible_Fluid_Dynamics_to_3D_CNNs`
  Chapter 5 and Appendix 2 of this thesis
- *Spline-PINN: Approaching PDEs without Data using Fast, Physics-Informed Hermite-Spline CNNs*
  Nils Wandel, Michael Weinmann, Michael Neidlin, Reinhard Klein
  Proceedings of the AAAI Conference on Artificial Intelligence 2022
  Code: `https://github.com/wandeln/Spline_PINN`
  Chapter 6 and Appendix 3 of this thesis

1.6 Overview

In the next section, we will discuss related research and in Section 3, we will provide foundations for a more in depth insight into our methodology. Sections 4, 5 and 6 contain short summaries of our papers. For more details, we recommend reading the full papers that are attached in the Appendix. Since each of these papers comes with its own specific related work, method, results and conclusion section, the corresponding sections of the overall thesis are kept broader to provide a more general overview. In Section 7, we sum up the results of our work, discuss its current limitations as well as possible future research directions and talk about its potential impact on the field of physics-driven deep-learning. Finally, we want to conclude with a broader vision for physics-driven deep-learning and an outlook regarding its positive implications for prospective real-world applications.

Chapter 2
Related work

In general, partial differential equations do not possess analytic solutions. Thus, computing numerical approximations is often the only viable option. Over the past decades, a wide variety of numerical solvers for PDEs evolved including common traditional methods such as finite differences, finite volumes, finite elements or particle based methods. These solvers do not require learning of parameters but rely on handcrafted algorithms with high computational complexity. On top of these traditional methods, recently, deep-learning based strategies emerged that aim to drastically reduce computational costs and speed up simulations by means of neural surrogate models.

In this section, a short review of related work is given. We will look at how different "traditional" numerical solvers can be complemented by corresponding deep-learning based methods. Furthermore, we will look at data-based learning strategies that rely on ground truth training data as well as physics-based learning strategies that rely on the underlying partial differential equations. Since our contributions focus in particular on the incompressible Navier-Stokes equation, the main emphasis is put on fluid simulations.

The behavior of incompressible fluids can be described by the incompressible Navier-Stokes equation. As discussed in Section 3.2 in more detail, there are two alternative views on the velocity field of a fluid: First, the *Lagrangian* view considers the velocity field from the perspective of individual particles moving within the fluid. Second, the *Eulerian* view considers the velocity field from a fixed outside perspective. Depending on the particular application, one or the other frame of reference can facilitate computations and in certain scenarios, "hybrid"-approaches that incorporate both views can be beneficial.

In the following, for each view (Lagrangian / Hybrid / Eulerian), we will first cover traditional numerical approaches followed by recent deep learning based advances. Finally, we will also briefly discuss related work in the field of deep-learning based fluid control.

	Lagrangian	Hybrid	Eulerian
"traditional"	• Distinct Element Method [Cundall and Strack, 1979] • Smoothed Particle Hydrodynamics [Gingold and Monaghan, 1977; Lucy, 1977] • Position-Based Dynamics [Müller et al., 2007] • Position-Based Fluids [Macklin and Müller, 2013]	• Particle In Cell [Harlow, 1962] • FLIP [Brackbill et al., 1988] • Material-Point Method [Sulsky et al., 1995; Stomakhin et al., 2013] • Animation of Liquids [Foster and Metaxas, 1996] • Stable Fluids [Stam, 1999]	• Finite Differences [Harlow and Welch, 1965] • Lattice Boltzmann [Chen and Doolen, 1998] • Finite Volumes [Eymard et al., 2000] • Finite Elements [Quarteroni and Quarteroni, 2009]
deep-learning	• Regression Forests [Ladický et al., 2015] • SP-Nets [Schenck and Fox, 2018] • Continuous Convolutions [Ummenhofer et al., 2020] • Graph Neural Networks [Sanchez-Gonzalez et al., 2020]	• Pressure Projection [Tompson et al., 2017]	Data-Driven: • Solver in the Loop [Um et al., 2020] • RANS [Thuerey et al., 2019] • GNN [Pfaff et al., 2021] Physics-Based: • Physics-Informed NN [Raissi et al., 2018] • Physics-Constrained NN [Zhu et al., 2019; Geneva and Zabaras, 2020]

Table 2.1: A selection of influential Lagrangian / Hybrid / Eulerian Fluid-Solvers relying on "traditional" and recent deep-learning based solvers

2.1 Lagrangian methods

Lagrangian methods consider a dynamical system in the frame of reference of individual particles. For example, distinct element methods [Cundall and Strack, 1979] simulate distinct particles of a granular medium that interact only at direct contact. However, if the number of particles is large (e.g. because the domain is big or because the particles are small such as molecules or atoms), this method becomes computationally infeasible. Thus, new artificial particles need to be introduced that describe quantities such as density or velocity of a *collection* of underlying physical particles. For example, in Smoothed Particle Hydrodynamics (SPH) [Gingold and Monaghan, 1977; Lucy, 1977], (isotropic Gaussian) smoothing kernels are used to simulate collections of particles in order to test astrophysical models of star formation. To allow for smoothing kernels that adjust their shape depending on the spacing to neighboring particles, adaptive SPH have been introduced in the context of high strain hydrodynamics [Liu et al., 2006]. Position Based Dynamics (PBD) [Müller et al., 2007] directly work on particle positions yielding unconditionally stable simulations over time and thus is a popular choice for game engines or in CGI. On top of PBD, Position Based Fluids (PBF) [Macklin and Müller, 2013] have been developed to enforce the incompressibility constraint of fluid simulations using an iterative Newton solver.

However, these traditional methods are computationally expensive even for medium sized scenes and require small time-steps for numerical stability. Thus, recently, deep-learning based approaches are employed more and more often that aim to drastically speed-up simulations. To this end, methods based on regression forests [Ladický et al., 2015], Graph Neural Networks [Mrowca et al., 2018; Li et al., 2019; Sanchez-Gonzalez et al., 2020] or continuous convolutions [Ummenhofer et al., 2020] have been introduced. Furthermore, Smooth Particle Nets were developed [Schenck and Fox, 2018] that implement the Position Based Fluid algorithm [Macklin and Müller, 2013] in a differentiable manner such that it can be used as a neural network layer in a deep-learning pipeline to solve control tasks.

While Lagrangian methods can provide robust simulations that automatically ensure conservation of mass for free-surface flows or multi-phase flows, they suffer from high dissipation and low accuracy within a fluid domain when in lack of sufficiently many particles. Furthermore, error estimation tends to be difficult in particle based methods.

2.2 Hybrid methods

Several works combine the Lagrangian and Eulerian frames of reference such that individual terms of the underlying PDEs can be handled in their most suitable frames of reference. For this reason, we call these methods "hybrids". Typically, advection terms are treated in the Lagrangian frame and the remaining terms are treated in the Eulerian frame of reference.

For example in "Stable fluids" [Stam, 1999], the advection term of the Navier-Stokes equation is computed by a particle tracer in the Lagrangian frame of reference and the viscosity as well as the pressure term are solved on a grid in the Eulerian frame of reference:

$$\partial_t \boldsymbol{v} = -\underbrace{\boldsymbol{v} \cdot \nabla \boldsymbol{v}}_{\text{Lagrangian}} + \frac{1}{\rho} \underbrace{(\mu \Delta \boldsymbol{v} - \nabla p + \boldsymbol{f}_{ext})}_{\text{Eulerian}}$$

Further "traditional" hybrid methods include Particle-in-Cell methods [Harlow, 1962], FLIP [Brackbill et al., 1988] or material-point methods [Sulsky et al., 1995; Stomakhin et al., 2013].

To accelerate the pipeline of [Stam, 1999], [Tompson et al., 2017] propose to use a CNN to solve the pressure term more efficiently (see also Section 3.2, "Helmholtz Decomposition Theorem").

While hybrid methods combine the benefits of Eulerian and Lagrangian methods in that they allow for fairly accurate simulations within a domain as well as dynamic free-surface boundaries, their computational pipeline is fairly complex. Furthermore, the mapping between grid and particle representations can introduce additional numerical diffusion.

2.3 Eulerian methods

Eulerian methods consider a dynamical system from a fixed point of view. This could be for a example a grid, a mesh or an implicit function. Common traditional Eulerian solvers include finite difference methods [Harlow and Welch, 1965; Foster and Metaxas, 1996] that rely on a staggered Marker And Cell (MAC) grid, Lattice-Boltzman methods [Chen and Doolen, 1998; Guo, 2021], finite volume methods and finite element methods. These traditional Eulerian methods can produce highly accurate results and are wildly used in industrial applications of computational fluid dynamics, but they are computationally expensive as they typically require solving large systems of linear equations to obtain stable simulations.

Therefore, in recent years, several papers investigated deep-learning strategies to speed-up Eulerian methods. These works can be categorized into data-driven and physics-based works:

Data-driven methods rely on existing ground truth data that is usually generated by a high quality but computationally expensive traditional CFD solver. By training a neural network to mimic the traditional solver, these methods aim to achieve a similar performance at significantly reduced computational costs. For example, [Wiewel et al., 2019] train an auto-encoder to project a fluid state onto a smaller latent description and a recurrent prediction network to efficiently evolve the fluid state in this latent space over time. [Geneva and Zabaras, 2022] also rely on an autoencoder to obtain latent representations but replace the recurrent prediction network by a transformer network. [Um et al., 2020] aim to correct numerical errors of a low resolution differentiable fluid solver by a convolutional neural network (CNN) that is trained on data obtained from a higher resolution fluid solver. Similarly, [Kochkov et al., 2021] use a CNN to achieve significant speed-ups while maintaining high accuracy by learning corrections steps for a finite volume method. [Thuerey et al., 2019] use a CNN to learn solutions of the Reynolds-averaged Navier-Stokes equation around airfoil profiles. Generative Adversarial Network (GAN) based approaches were developed by [Xie et al., 2018], who propose a temporally coherent, volumetric GAN (Tempo-GAN) to increase the resolution of smoke simulations and by [Kim et al., 2019], who propose a recurrent GAN for simulations of turbulent flow fields in pipe-domains. And Graph Neural Network (GNN) based methods are used in works by [Gao et al., 2021; Harsch and Riedelbauch, 2021; Pfaff et al., 2021] to learn the dynamics of various physical systems such as cloth, metal plates and fluids on a mesh. This allows to focus on parts of the fluid domain that are of special interest by choosing a mesh with higher resolution for example at boundary layers.

While these data-driven methods are relatively straight forward to train using standard losses such as L_2 for steady-state or short-term predictions of laminar flows, long-term predictions at high Reynolds-numbers can become problematic due to the chaotic nature of fluids. In this case, minimal changes of the input fields lead to vastly different outcomes and optimizing for an L_2 loss would result in a blurry mean value, which is usually not a desirable outcome. Thus, more sophisticated training strategies are needed that are based for example on adversarial losses, which on the other hand are challenging due to issues such as mode collapse, temporal coherence or lack of physical plausibility. Furthermore, significant generalization beyond domain geometries of the training data and stable simulations over long time horizons tend to be difficult. To this end, traditional, differentiable low-resolution solvers can be incorporated into the simulation pipeline, however, they come at the cost of significantly increasing the overall complexity of the method. Finally, the accuracy of data-driven methods is always limited by the quality and amount of the underlying ground truth data generated by traditional methods. However, generating large datasets of high quality is computationally expensive and might become a bottleneck. Thus, in the long run, if deep-learning based approaches are to surpass the accuracy of traditional solvers, they probably have to get emancipated from traditional methods and should learn directly on the underlying PDEs.

To this end, *physics-based* loss terms were introduced to drastically reduce or even completely avoid the need for ground truth data by directly penalizing residuals of the underlying PDEs. Here, physics-informed methods can be discerned from physics-constrained methods.

Physics-Informed neural networks (PINNs) [Grohs et al., 2018; Sirignano and Spiliopoulos, 2018; Khoo et al., 2019; Raissi et al., 2019] are based on implicit neural representations that continuously map domain coordinates in space and time (e.g. x, y, t) to corresponding field values (e.g. v, p). This allows to deal with high dimensional PDEs that could not be dealt with in discretized settings due to the curse of dimensionality [Grohs et al., 2018]. To train such models, a physics-informed loss that incorporates the residuals of a PDE is computed by automatic differentiation. PINNs have been popularized by [Raissi et al., 2019] and have been applied to fluids [Yang et al., 2016; Raissi et al., 2018], the Hamilton-Jacobi-Bellman PDE and Burgers equation [Sirignano and Spiliopoulos, 2018]. Further applications include flow simulations through porous media [Tripathy and Bilionis, 2018; Zhu and Zabaras, 2018; Zhu et al., 2019], turbulence modeling [Ling et al., 2016; Geneva and Zabaras, 2019] and simulations of propagating waves [Sitzmann et al., 2020; Rasht-Behesht et al., 2021]. Unfortunately, by the nature of implicit neural representation networks, these methods are trained to overfit on one single domain. Thus, they cannot generalize to new domains and require retraining of the entire neural network which is computationally expensive.

Physics-Constrained methods on the other hand are usually applied on discrete domain representations. These discrete representations (e.g. a grid) are then explicitly mapped from one time-step t to the next time-step $t + dt$ by a neural network (e.g. a CNN). To compute a physics-constrained loss, the residuals of the underlying PDE are evaluated on the grid based on finite differences. This method has been used in [Tompson et al., 2017] to solve the Poisson equation for a pressure projection step in the context of inviscid fluids. However, in this work, a particle tracer was needed to deal with advection. Furthermore, although no labeled data is needed, this method still requires some fluid data generated by a traditional solver in order to achieve better generalization performance by training the model on fluid states that resemble real-world data. In contrast, [Zhu et al., 2019] do not require any ground truth data to solve Darcy flow problems by using on a physics-constrained loss. However, Darcy flows describe stationary "creeping" flows through porous media and thus do not result in dynamic simulations. Shortly after, dynamic fluid simulations without ground truth data were obtained by [Geneva and Zabaras, 2020]. They used a physics-constrained loss to solve Burger's equation which produces interesting shock patterns. But in contrast to the full incompressible Navier-Stokes equation, the Burger's equation does not contain a pressure term. Furthermore, this work does not consider variable boundary conditions that would allow for interactions with the neural surrogate model in real-time.

Techniques that combine the advantages of physics-informed and physics-constrained neural networks, namely continuous field representations and fast, interactive simulations, can be achieved with continuous spline interpolations as we show in our work on Spline-PINNs [Wandel et al., 2022].

2.4 Splines in Neural Networks

The usage of splines in neural networks has been explored by numerous previous works. In [Igelnik and Parikh, 2003] and [Fakhoury et al., 2022], splines were proposed to obtain flexible activation functions. [Fey et al., 2017] propose to train continuous convolutional kernels based on B-splines to deal with irregular structured data. On uniform grids, B-Splines have been used in conjunction with CNNs by [Barrowclough et al., 2021] to do binary segmentation of medical images. Furthermore, [Cho et al., 2021] developed differentiable spline approximations to solve Poisson equations on top of performing image segmentation and point cloud reconstruction. However, to the best of our knowledge, splines in conjunction with CNNs have not yet been exploited to approach complex dynamical systems of partial differential equations such as the wave equation or the incompressible Navier-Stokes equation.

2.5 Differentiable models for fluid control

Several works have investigated the use of differentiable fluid models and neural networks in fluid control settings. For example, [Schenck and Fox, 2018] implemented a differentiable version of the lagrangian position based fluid (PBF) solver with a deep neural network in order to learn control tasks such as pouring water into a bowl or steering a water puddle on a plate. And an eulerian differentiable fluid solver (Phiflow) was used by [Holl et al., 2020] in combination with neural networks to plan and control flow trajectories. However, both differentiable fluid solvers rely on complex handcrafted algorithms with only few learned paramters.

In this thesis, we build on top of these groundbreaking works and develop novel approaches that combine some of the mentioned approaches and their individual advantages: By using a physics-based loss we *forgo any ground truth data* for training. Our surrogate models *generalize* to variable boundary conditions, thereby allowing for *real-time interactive simulations*. Since our light-weight inference pipeline is *fully differentiable*, it can be used for example for gradient-based optimization of control algorithms. Finally, in our last work on Spline-PINNs, we aimed to combine the benefits of physics-constrained with physics-informed neural networks in order to achieve *continuous* simulations with fewer discretization artifacts on top of our fast, interactive and differentiable simulation pipeline.

Chapter 3

Foundations

In this chapter, we want to develop some basic theoretical foundations in order to explain our work in more detail.

3.1 Partial Differential Equations

In general, partial differential equations are equations that impose relationships between the partial derivatives of a multivariable function. Here, we focus on physical systems, where multivariable functions are often used to describe fields such as pressure, velocity or displacement and their partial derivatives describe quantities such as the force of a pressure gradient, change of velocity over time, viscous friction, acceleration etc.

Based on these field descriptions and their derivatives we can formulate partial differential equations that impose the laws of physics in order to make predictions about their future dynamics.

Examples of PDEs in physics are: Wave equation, Maxwell equations, Navier-Stokes equation, and many more. In this work, we focus mainly on the incompressible Navier-Stokes equation as well as the damped wave equation.

3.1.1 Differential operators

To abbreviate the notations of partial derivatives, certain differential operators are commonly used. In this thesis, the following operators are of importance:

Nabla-Operator The Nabla [1] Operator is a vector that comprises partial derivatives:

$$\nabla = \begin{pmatrix} \partial_x \\ \partial_y \\ \partial_z \end{pmatrix}$$

Note: Here, the Nabla operator only refers to spatial derivatives. Temporal derivatives are still denoted by ∂_t.

Gradient and Jacobian Using the Nabla operator, the gradient of a scalar field s and the jacobian [2] of a vector field $\boldsymbol{v}$ can be conveniently denoted by:

$$\nabla s = \begin{pmatrix} \partial_x s \\ \partial_y s \\ \partial_z s \end{pmatrix}; \nabla \boldsymbol{v} = \begin{pmatrix} \partial_x v_x & \partial_y v_x & \partial_z v_x \\ \partial_x v_y & \partial_y v_y & \partial_z v_y \\ \partial_x v_z & \partial_y v_z & \partial_z v_z \end{pmatrix}$$

Divergence The divergence of a vector field $\boldsymbol{v}$ can be considered as the scalar product of the Nabla operator with that vector field:

$$\nabla \cdot \boldsymbol{v} = \partial_x v_x + \partial_y v_y + \partial_z v_z$$

Curl The curl of a vector field $\boldsymbol{v}$ can be computed by taking the cross product of the Nabla operator with that vector field:

$$\nabla \times \boldsymbol{v} = \begin{pmatrix} \partial_y v_z - \partial_z v_y \\ \partial_z v_x - \partial_x v_z \\ \partial_x v_y - \partial_y v_x \end{pmatrix}$$

Laplace Operator The Laplace[3] operator Δ is defined as the divergence of the gradient. Applying it onto a scalar field s or a vector field $\boldsymbol{v}$ respectively yields:

$$\Delta s = \nabla \cdot \nabla s = \partial_x^2 s + \partial_y^2 s + \partial_z^2 s \, ; \, \Delta \boldsymbol{v} = \nabla \cdot \nabla \boldsymbol{v} = \begin{pmatrix} \partial_x^2 v_x + \partial_y^2 v_x + \partial_z^2 v_x \\ \partial_x^2 v_y + \partial_y^2 v_y + \partial_z^2 v_y \\ \partial_x^2 v_z + \partial_y^2 v_z + \partial_z^2 v_z \end{pmatrix}$$

[1] The name "Nabla" originates from a harp-like Phoenician stringed instrument that resembled this sign.
[2] Carl Gustav Jacobi, German mathematician (1804 - 1851)
[3] Pierre-Simon de Laplace, French mathematician (1749–1827)

3.1.2 The initial-boundary-value-problem (IBVP)

Usually, we want to restrict the solution of a PDE to a certain domain Ω and its boundary $\partial\Omega$. In computer simulations, this is done by limiting Ω to a given spatial geometry as well as to a given time-range.

To specify the solution of the PDE at the domain boundaries $\partial\Omega$, we introduce boundary and initial conditions that need to be fulfilled at the spatial boundaries of the geometry and at the beginning of the simulation respectively.

Boundary Conditions In order to model interactions with the surroundings at the domain boundaries, additional constraints need to be imposed. In this thesis, we focus on Dirichlet boundary conditions that specify directly the field values at the domain boundaries $\partial\Omega$. Further types of boundary conditions such as for example Neumann boundary conditions might also impose constraints on the derivatives of a field.

Initial Conditions Initial conditions are commonly used to specify the state at the beginning of a simulation (i.e. usually at $t = 0$). For example, in case of the Navier-Stokes equation (see Section 3.2), this would be the initial velocity and pressure fields v_0 and p_0 respectively.

3.1.3 Properties of PDEs

We can classify PDEs by certain characteristics in order to choose appropriate techniques to solve or approximate their solutions. In this section, some basic important properties of PDEs are highlighted.

Order of a PDE The order of a PDE is given by its highest partial derivatives in a certain dimension. For example, the Laplace equation is a second order PDE:

$$\Delta u = \partial_x^2 u + \partial_y^2 u + \partial_z^2 u = 0 \tag{3.1}$$

The order of a PDE is important for example in finite difference or spline methods since it directly affects the size of a finite difference kernel or the order of spline-kernels that needs to be chosen.

Homogeneous and Nonhomogeneos PDEs Let's assume, we have a PDE for an unknown function $u(t, x, y)$ in 3 variables t, x, y. To distinguish homogeneous from nonhomogeneous PDEs, we first group all terms that contain u and its derivatives into $\mathcal{D}(u)$ and all terms that do not contain u into $f(t, x, y)$ in order to obtain:

$$\mathcal{D}(u) = f(t, x, y) \tag{3.2}$$

If the right hand side, $f(t, x, y)$, is zero, the PDE is called homogeneous, otherwise, it is called nonhomogeneous.

Linear and Nonlinear PDEs If $\mathcal{D}(u)$ in Equation 3.2 is linear, that means $\mathcal{D}(\alpha u + v) = \alpha\mathcal{D}(u) + \mathcal{D}(v)$, then the PDE is also called linear. Otherwise, the PDE is called nonlinear. For example, the wave equation (see section 3.3) is a linear PDE while the Navier-Stokes equation (see section 3.2) is nonlinear. If an Operator $\mathcal{D}$ is linear, we can compute eigenfunctions u_i and eigenvalues λ_i of the form:

$$\mathcal{D}(u_i) = \lambda_i u_i$$

If we can find coefficients c_i, such that:

$$f = \sum_i c_i u_i$$

we can construct a solution for $\mathcal{D}(u) = f$ by setting:

$$u = \sum_i \frac{c_i}{\lambda_i} u_i \Rightarrow \mathcal{D}(u) = \sum_i \lambda_i \frac{c_i}{\lambda_i} u_i = \sum_i c_i u_i = f$$

If $\mathcal{D}$ is symmetric, (that means $\langle \mathcal{D}u, v\rangle = \langle u, \mathcal{D}v\rangle$), we can find orthonormal eigenfunctions (that means $\langle u_i, u_j\rangle = \delta_{i,j}$) and thus the coefficients can be determined by taking the scalar-product between u_i and f:

$$c_i = \langle u_i, f\rangle$$

Furthermore, if we can separate the variables of $\mathcal{D}$, we can transform the PDE into separate ODEs which drastically reduces the complexity and in some cases enables analytical solutions.

Following the notation of Farlow, 1993, a linear second order PDE of a function $u(x, y)$ in two variables x, y can be written for example as:

$$A\partial_x^2 u + B\partial_x\partial_y u + C\partial_y^2 u + D\partial_x u + E\partial_y u + Fu = G \tag{3.3}$$

where A, B, C, D, E, F, G can be considered as arbitrary functions of x, y.

If we consider the special case of a PDE that on top of being linear is also homogeneous (in Equation 3.3 that would mean: $G = 0$), we can linearly combine solutions of the PDE to obtain new solutions. This allows for methods such as separation of variables that can potentially even provide analytical solutions.

Elliptic / Parabolic / Hyperbolic PDEs By investigating the coefficients of equation 3.3 we can classify linear second order PDEs into 3 categories: elliptic, parabolic and hyperbolic PDEs .

1. If $B^2 - 4AC < 0$, the PDE is said to be *elliptic* and describes steady-state phenomena such as the Laplace-equation (see Equation 3.1).
2. If $B^2 - 4AC = 0$, the PDE is said to be *parabolic* and describes for example diffusion processes such as heat flow.
3. If $B^2 - 4AC > 0$, the PDE is said to be *hyperbolic* and describes vibrating systems such as the wave-equation. (See Section 3.3)

This naming convention stems from an analogy to conic sections. If we plot solutions of the general equation for conic sections:

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \tag{3.4}$$

we obtain ellipses for $B^2 - 4AC < 0$, parabolas for $B^2 - 4AC = 0$ and hyperbolas for $B^2 - 4AC > 0$. Examples for these 3 cases are shown in Figure 3.1.

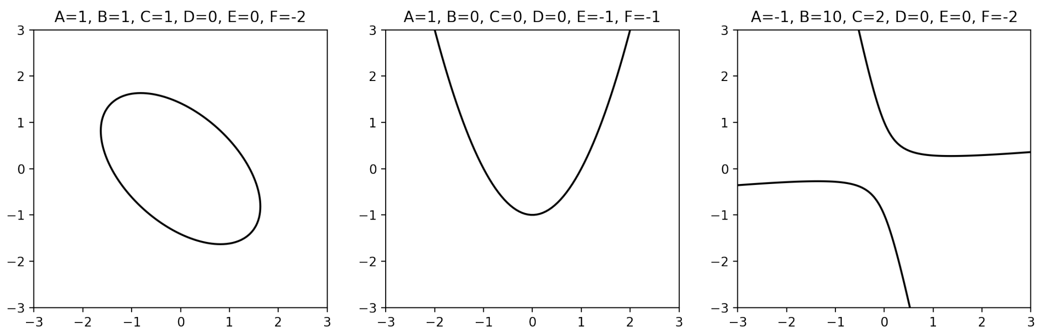


Figure 3.1: Conic sections for different values of A, B, C, D, E, F. Left: Ellipse, Center: Parabola, Right: Hyperbola

Knowledge about the type of the underlying PDE might be important to choose a proper solver.

In the following, we will look at 2 particular PDEs in more detail, namely the incompressible Navier-Stokes equation and the damped wave equation.

3.2 Incompressible Navier-Stokes Equation

The incompressible Navier[4]-Stokes[5] equation[6] is a non-linear PDE that describes the dynamics of an incompressible fluid by means of a vector field v that contains the fluids velocity

[4] Claude-Louis Navier, French engineer and physicist (1785 - 1836)

[5] Sir George Stokes, Anglo-Irish physicist and mathematician, (1819 - 1903)

[6] Interesting side note: The existence of smooth solutions of the Navier-Stokes equation has not been proven yet and proving or disproving its existence remains a 1.000.000 $ millennium prize problem. However, in this work we do not attempt to make a contribution in this regard.

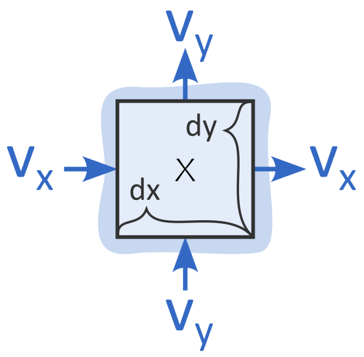


Figure 3.2: The divergence of a vector field $\nabla \boldsymbol{v}$ measures the total amount of inflow into and outflow from an infinitesimal volume around a point x of the domain. If $\boldsymbol{v}$ is divergence-free, the inflow and outflow have to cancel each other at any given point.

and a scalar field p that contains the fluids pressure. It comprises 2 equations that need to be fulfilled at any given point within the domain Ω. First, the incompressibility equation (Equation 3.5) and second, the momentum equation (Equation 3.6):

$$\nabla \cdot \boldsymbol{v} = 0 \qquad \text{in } \Omega \tag{3.5}$$

$$\rho \frac{D}{Dt} \boldsymbol{v} = \rho(\partial_t \boldsymbol{v} + \boldsymbol{v} \cdot \nabla \boldsymbol{v}) = \mu \Delta \boldsymbol{v} - \nabla p + \boldsymbol{f}_{\text{ext}} \qquad \text{in } \Omega \tag{3.6}$$

At the domain boundary $\partial\Omega$, we specify Dirichlet boundary conditions as follows:

$$\boldsymbol{v} = \boldsymbol{v}_d \qquad \text{on } \partial\Omega \tag{3.7}$$

If $\boldsymbol{v}_d$ corresponds to the velocity of an obstacle within the domain Ω, this boundary condition is often referred to as *no-slip* condition which holds under the common assumption that fluid particles exactly follow the velocity of an obstacle at its surface.

In the following, we will investigate the incompressibility and momentum equation in more detail.

3.2.1 Incompressibility equation

The incompressibility equation ensures that the fluid remains incompressible. It states that the amount of liquid entering any infinitesimal small patch in Ω must match the amount exiting that infiniteximal patch (see also Figure 3.2). This is equivalent to enforcing the divergence of the velocity field $\boldsymbol{v}$ to be 0:

$$\nabla \cdot \boldsymbol{v} = \partial_x v_x + \partial_y v_y = 0$$

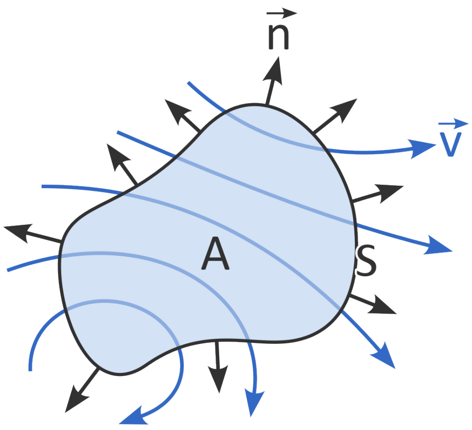


Figure 3.3: If a vector field $\boldsymbol{v}$ is divergence-free ($\nabla \cdot \boldsymbol{v} = 0$), the total amount of liquid entering a subdomain A through its surface S always matches the amount of fluid exiting that subdomain again.

By using Gauss's[7] theorem, we can show that the total amount of inflow and outflow into an arbitrary subdomain A of Ω is always 0 if the velocity field is divergence-free (see also Figure 3.3):

$$\oint_S \boldsymbol{v} \cdot \boldsymbol{n}\, dS \overset{\text{Gauss}}{=} \int_A \nabla \cdot \boldsymbol{v}\, dA \overset{\nabla \cdot \boldsymbol{v}=0}{=} 0$$

Helmholtz Decomposition Theorem The Helmholtz[8] decomposition theorem states that every vector field $\boldsymbol{v}$ can be written as the sum of the gradient of a scalar potential q and the curl of a vector potential $\boldsymbol{a}$:

$$\boldsymbol{v} = \nabla q + \nabla \times \boldsymbol{a}$$

By plugging in the definitions of gradient, divergence and curl, it can be shown that the curl of ∇q is always zero ($\nabla \times (\nabla q) = 0$) and that the divergence of $\nabla \times \boldsymbol{a}$ is always zero as well ($\nabla \cdot (\nabla \times \boldsymbol{a}) = 0$). This result can be used to ensure that the velocity field fulfills incompressibility ($\nabla \cdot \boldsymbol{v} = 0$).

A common way to ensure incompressibility based on the Helmholtz decomposition theorem is to compute a pressure projection step (Stam, 1999; Tompson et al., 2017). Here, we decompose a given vector field $\tilde{\boldsymbol{v}}$ that might not be divergence-free into the gradient of a "pressure" field q and a divergence-free velocity field $\boldsymbol{v} = \nabla \times \boldsymbol{a}$:

$$\tilde{\boldsymbol{v}} = \nabla q + \underbrace{\nabla \times \boldsymbol{a}}_{\boldsymbol{v}}$$

Since q and $\boldsymbol{a}$ are unknown, we solve a Poisson equation for the "pressure" field q given $\tilde{\boldsymbol{v}}$ as

[7] Carl Friedrich Gauß, German mathematician and physicist (1777 - 1855)
[8] Hermann von Helmholtz, German physicist and physician (1821 - 1894)

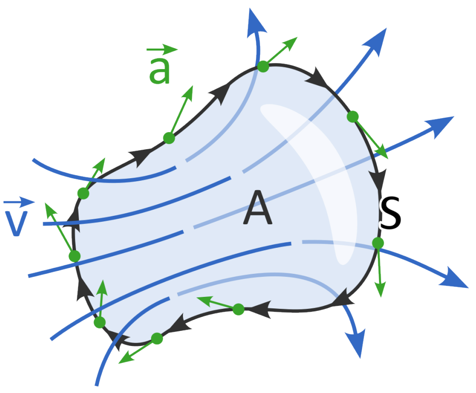


Figure 3.4: The amount of liquid flowing through an area A can be computed by integrating the vector potential $\boldsymbol{a}$ along the border S of that area.

follows:

$$\nabla \cdot \tilde{\boldsymbol{v}} = \nabla \cdot (\nabla q) + \underbrace{\nabla \cdot (\nabla \times \boldsymbol{a})}_{=0} = \Delta q$$

By subtracting the gradient of the obtained "pressure" field, ∇q, we can project $\tilde{\boldsymbol{v}}$ onto the divergence-free vector field $\boldsymbol{v}$:

$$\tilde{\boldsymbol{v}} - \nabla q = \underbrace{\boldsymbol{v}}_{\sim \nabla \times \boldsymbol{a}} \Rightarrow \nabla \cdot \boldsymbol{v} = \nabla \cdot \tilde{\boldsymbol{v}} - \Delta q = 0$$

Solving the Poisson[9] equation can be done efficiently in frequency domain using the fast fourier transform in case of periodic boundary conditions [Stam, 1999]. In more evolved settings, this step is computationally more expensive, thus Tompson et al., 2017 approached this step with a deep learning based method. However, depending on the accuracy of the Poisson solver, this method does not guarantee exact solutions for the incompressibility equation.

On the other hand, it is also possible to directly work on the vector potential $\boldsymbol{a}$ and set $\boldsymbol{v} = \nabla \times \boldsymbol{a}$ [Kim et al., 2019; Raissi et al., 2019; Mohan et al., 2020]. This automatically ensures strict incompressibility and thus is the method of choice in this work.

Stokes theorem (see Figure 3.4) provides some intuition for the vector potential $\boldsymbol{a}$:

$$\oint_S \boldsymbol{a} \cdot dS \overset{\text{Stokes}}{=} \int_A (\nabla \times \boldsymbol{a}) \cdot dA \overset{\nabla \times \boldsymbol{a} = \boldsymbol{v}}{=} \int_A \boldsymbol{v} \cdot dA$$

This implies that we can compute the amount of liquid flowing through a surface-area A by integrating the vector potential $\boldsymbol{a}$ along the border S of that area. This way, the complexity of a 2D surface integral can be reduced to a simpler 1D line integral.

[9] Baron Siméon Denis Poisson, French mathematician and physicist (1781 - 1840)

3.2.2 Momentum equation

The momentum equation describes how different forces such as viscous drag ($\mu\Delta\boldsymbol{v}$), a pressure gradient (∇p) or external forces ($\boldsymbol{f}_{\text{ext}}$) act on a fluid. There are two different view points to approach the momentum equation: the Lagrangian and the Eulerian frame of reference.

Lagrangian frame of reference In the Lagrangian[10] frame of reference, the fluid is viewed from the perspective of individual particles. Thus, the accelerations of the fluid particles are denoted by $\frac{D}{Dt}\boldsymbol{v}$ and the change of momentum per volume is given by $\rho\frac{D}{Dt}\boldsymbol{v}$ (see left hand side of Equation 3.6). By Newtons second law, this change of momentum must equal the sum of forces acting on the fluid such as viscous, pressure and external forces (see right hand side of Equation 3.6).

Eulerian frame of reference In the Eulerian[11] frame of reference, the fluid is viewed from a fixed perspective. Thus, the accelerations of the fluid particles not only depend on the partial derivative of the velocity field with respect to time ($\partial_t\boldsymbol{v}$) but also on an advection term ($\boldsymbol{v}\cdot\nabla\boldsymbol{v}$). The intuition of this advection term is that particles get accelerated if they move into regions of different velocities. For example, in Figure 3.9, particles get accelerated when they move into the narrow pipe region and decelerated when they exit again into the wide pipe region. Note that these accelerations take place although the fluids velocity field is in a steady state ($\partial_t\boldsymbol{v} = 0$), which might be a bit counter-intuitive at first. Furthermore, note that this advection term is also the reason why the Navier-Stokes equation is non-linear and thus particularly hard to solve.

In the following, we want to develop more intuition on the different parts of the momentum equation and derive the units for ρ, μ and p.

3.2.3 What is density ρ?

The density ρ of a liquid is defined as mass per volume of a liquid. Thus, in SI-units, it is specified by kg/m^3. Since we consider the *incompressible* Navier-Stokes equation, we can assume that the density is constant everywhere within the liquid.

In the Lagrangian view, the density can be considered as the inertia of fluid particles that get affected by the surrounding forces such as the pressure gradient, viscous friction or external forces. If we use the SI-units of ρ and plug them into the left hand side of the momentum equation (see Equation 3.6), we obtain the following SI units:

[10] Joseph-Louis Lagrange, Italian mathematician and astronomer (1736 - 1813)
[11] Leonhard Euler, Swiss mathematician, physicist and astronomer (1707 - 1783)

$$\underbrace{\frac{kg}{m^3}}_{\rho} \cdot \underbrace{\frac{1}{s}}_{\frac{D}{Dt}} \cdot \underbrace{\frac{m}{s}}_{v} = \frac{kg}{m^2 s^2} = \frac{N}{m^3} \tag{3.8}$$

This corresponds to a force (here in Newton $N = \frac{kg \cdot m}{s^2}$) per unit volume. These SI-units have to be matched as well by every individual term on the right hand side of Equation 3.6.

3.2.4 What is viscosity μ?

The viscosity μ of a fluid describes the siziness of a fluid. For example, honey has high viscosity while oil is less viscous and water has low viscosity. In the momentum equation, the viscosity-term ($\mu\Delta\boldsymbol{v}$) can be considered as a drag force between fluid particles. If the surrounding of a particle moves in average in a different direction (denoted by the Laplacian of the velocity field $\Delta\boldsymbol{v}$), it applies a drag force proportional to the viscosity μ (see Figure 3.5 a). If we consider the SI-units of the viscosity term ($\mu\Delta\boldsymbol{v}$) in the momentum equation (Equation 3.6) and set them equal to Equation 3.8, we obtain the following units for μ:

$$\underbrace{\frac{Ns}{m^2}}_{\mu} \cdot \underbrace{\frac{1}{m^2}}_{\Delta} \cdot \underbrace{\frac{m}{s}}_{v} = \frac{N}{m^3}$$

Thus, the SI-units of viscosity μ have to match $\frac{Ns}{m^2}$, which is also called one Poiseuille [12] (PI).

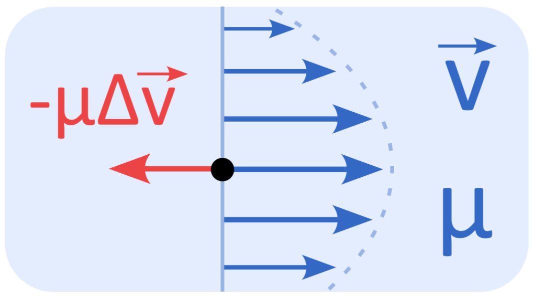
On top of applying a drag force on particles within a fluid, viscosity also causes drag on the domain boundaries $\partial\Omega$ given by $\mu(\nabla\boldsymbol{v})\boldsymbol{n}$. This boundary drag force per surface area is proportional to the viscosity μ and the velocity-gradient $\nabla\boldsymbol{v}$ along the surface normal $\boldsymbol{n}$ (see Figure 3.5 b). To verify that this drag force term ($\mu(\nabla\boldsymbol{v})\boldsymbol{n}$) indeed yields a force per unit area, we can check the SI-units again:

$$\underbrace{\frac{Ns}{m^2}}_{\mu} \cdot \underbrace{\frac{1}{m}}_{\nabla} \cdot \underbrace{\frac{m}{s}}_{v} \cdot \underbrace{\text{(unit-free)}}_{\boldsymbol{n}} = \frac{N}{m^2}$$

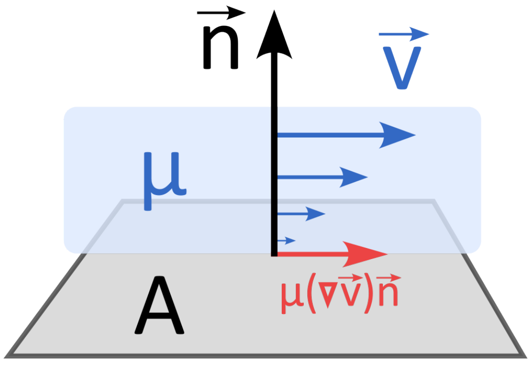
Thus, in order to compute the drag force $\boldsymbol{F}_\mu$ caused by viscous friction on an object within a fluid, we can integrate over its surface S:

$$\boldsymbol{F}_\mu = \oint_S \mu(\nabla\boldsymbol{v})\boldsymbol{n}\,ds$$

[12] Jean Léonard Marie Poiseuille, French physicist and physiologist (1797 - 1869)



(a) Velocity profile around a fluid particle. Viscous friction causes a drag force on a particle to assimilate the velocity of the local surrounding.



(b) Slice of a velocity field (blue) perpendicular to a No-Slip boundary surface (grey). The drag force caused by the fluid on the boundary is proportional to the viscosity μ and the velocity gradient $\nabla \boldsymbol{v}$ along the surface normal $\boldsymbol{n}$.

Figure 3.5: Effects of viscosity on particles within a fluid and on domain boundaries.

3.2.5 What is pressure p?

The pressure field p acts like a scalar potential and applies a force proportional to its gradient on the fluid particles. By investigating the SI-units of the pressure term in the momentum equation (∇p in Equation 3.6) and setting them equal to Equation 3.8, we obtain the following units for p:

$$\underbrace{\frac{1}{m}}_{\nabla} \cdot \underbrace{\frac{N}{m^2}}_{p} = \frac{N}{m^3}$$

Thus, the SI-units of the pressure field p must match $\frac{N}{m^2}$, which is in accordance with the definition of one Pascal[13] (Pa) and our intuition of pressure being a force per unit (surface) area.

If we want to compute the pressure force $\boldsymbol{F}_p$ on an object within the fluid, we have to integrate the pressure forces p along the surface normals $\boldsymbol{n}$ over the objects surface S:

$$\boldsymbol{F}_p = \oint_S p\boldsymbol{n}ds$$

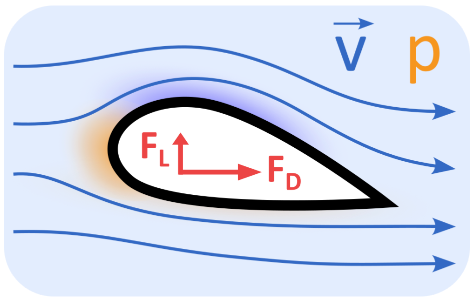


Figure 3.6: Drag (F_D) and lift (F_L) forces acting on an object within a flow field

3.2.6 Drag and Lift coefficients

By combining the viscous force $\boldsymbol{F}_\mu$ and the pressure force $\boldsymbol{F}_p$, we can obtain the total force $\boldsymbol{F}_{\text{tot}}$ that a fluid exerts on an object.

$$\boldsymbol{F}_{\text{tot}} = \boldsymbol{F}_\mu + \boldsymbol{F}_p$$

Now, we take the parallel drag-force component F_D and the perpendicular lift-force component F_L of $\boldsymbol{F}_{\text{tot}}$ with respect to the average velocity of the fluid stream (see Figure 3.6) and define the drag and lift coefficient C_D and C_L respectively as follows:

$$C_D = \frac{2F_D}{\rho U_{\text{mean}}^2 A}, \; C_L = \frac{2F_L}{\rho U_{\text{mean}}^2 A} \tag{3.9}$$

Here, U_{mean} is the average velocity of the surrounding fluid stream and A denotes the surface area of the object. The drag and lift coefficients are dimension-less quantities as can be easily shown by investigating the SI-units of a force term ($F \sim N$) divided by a density term ($\rho \sim \frac{kg}{m^3}$), a squared velocity term ($U^2 \sim \left(\frac{m}{s}\right)^2$) and a surface area term ($A \sim m^2$):

$$\frac{N}{\frac{kg}{m^3} \cdot \left(\frac{m}{s}\right)^2 \cdot m^2} = \frac{\frac{kg\,m}{s^2}}{\frac{kg}{m^3} \cdot \frac{m^2}{s^2} \cdot m^2} = \frac{\frac{kg\,m}{s^2}}{\frac{kg\,m}{s^2}} = 1$$

3.2.7 Reynolds-Number

The Reynolds[14]-number is an important unit-free quantity that describes the qualitative behavior of a fluid. It is defined as:

$$Re = \frac{\rho U_{\text{mean}} L}{\mu} \tag{3.10}$$

[13] Blaise Pascal, French mathematician, physicist, and theologian (1623 - 1662)
[14] Osborne Reynolds, Irish mathematician and engineer (1842 - 1912)

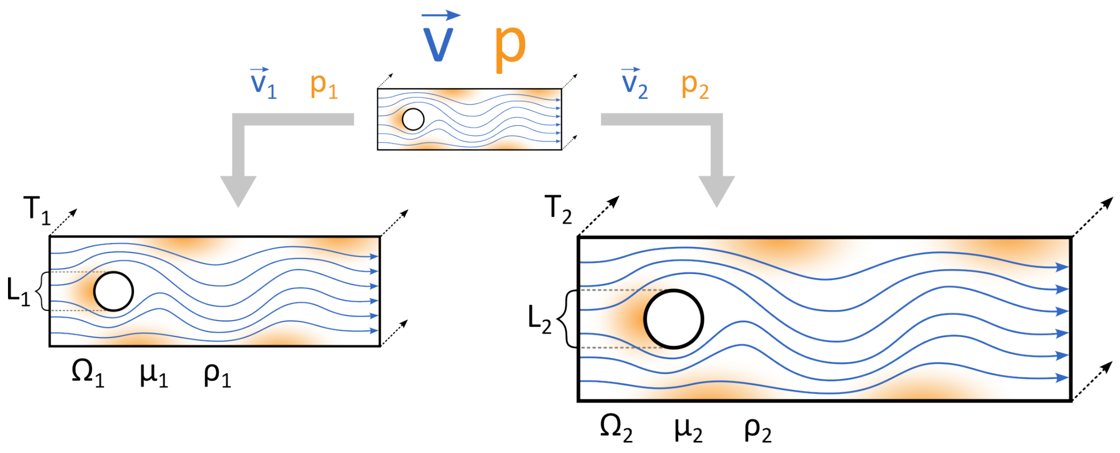


Figure 3.7: If 2 Systems have the same Reynolds-Number, solutions of the Navier-Stokes equations can be deduced from the same underlying velocity (v) and pressure (p) fields up to constant scaling factors.

Here, U_{mean} is the average fluid velocity and L is the characteristic length scale of the domain (e.g. the diameter of an obstacle). By plugging SI-units into the Reynolds-number, we can easily verify that the Reynolds-number is indeed unit-free:

$$\frac{\frac{kg}{m^3}\cdot\frac{m}{s}\cdot m}{N\frac{s}{m^2}} = \frac{\frac{kg}{m\,s}}{\frac{kg\,m}{s^2}\cdot\frac{s}{m^2}} = \frac{\frac{kg}{m\,s}}{\frac{kg}{m\,s}} = 1$$

Now, one might wonder: Why do systems with equal Reynolds-numbers show qualitatively similar behavior - even if the fluids exhibit very different viscosities, densities and spatial / temporal scales? The answer is given by a scaling argument: Let us suppose we have two systems (System 1 and System 2) with identical (but scaled) fluid domains Ω_1 and Ω_2 (see Figure 3.7). Now, we say that System 1 and System 2 behave qualitatively equivalent, if their velocity- and pressure-fields (v_1, p_1) and (v_2, p_2) can be ascribed to the same underlying velocity-/pressure-fields (v, p) up to scaling factors (U_1, P_1, T_1, L_1) and (U_2, P_2, T_2, L_2):

$$v_1(t,x) = U_1\, v(\frac{t}{T_1}, \frac{x}{L_1}) \qquad p_1(t,x) = P_1\, p(\frac{t}{T_1}, \frac{x}{L_1})$$
$$v_2(t,x) = U_2\, v(\frac{t}{T_2}, \frac{x}{L_2}) \qquad p_2(t,x) = P_2\, p(\frac{t}{T_2}, \frac{x}{L_2})$$

To this end, the Navier-Stokes equation have to be fulfilled in both systems.

The incompressibility equation is automatically fulfilled in both systems, since, if $\nabla \cdot v = 0$ is true in one system, this is also true for any scaled version $\boldsymbol{v}_i$:

$$\nabla \cdot \boldsymbol{v}_i = \nabla \cdot U_i \boldsymbol{v}\left(\frac{t}{T_i}, \frac{\boldsymbol{x}}{L_i}\right) = \frac{U_i}{L_i} \nabla \cdot \boldsymbol{v} \overset{\nabla\cdot \boldsymbol{v}=0}{=} 0$$

However, for the momentum equation (Equation 3.6), one has to take care that the prefactors of the indiviudual terms are in correspondence with each other (here, we ignore external forces):

$$\begin{aligned} &\rho_i \partial_t \boldsymbol{v}_i + \rho_i \boldsymbol{v}_i \cdot \nabla \boldsymbol{v}_i = \mu_i \Delta \boldsymbol{v}_i - \nabla p_i \\ = &\underbrace{\frac{\rho_i U_i}{T_i}}_{\textcircled{1}} \partial_t \boldsymbol{v} + \underbrace{\frac{\rho_i U_i^2}{L_i}}_{\textcircled{2}} \boldsymbol{v} \cdot \nabla \boldsymbol{v} = \underbrace{\frac{\mu_i U_i}{L_i^2}}_{\textcircled{3}} \Delta \boldsymbol{v} - \underbrace{\frac{P_i}{L_i}}_{\textcircled{4}} \nabla p \end{aligned}$$

To ensure that this equation can be fulfilled by the same underlying fields $\boldsymbol{v}$ and p for both systems ($i = 1$ and $i = 2$), $\textcircled{1}$, $\textcircled{2}$, $\textcircled{3}$, $\textcircled{4}$ must have the same proportions, in the following denoted as $\textcircled{A}$, $\textcircled{B}$, $\textcircled{C}$, $\textcircled{D}$:

$$\underbrace{\frac{\rho_1 U_1}{T_1} = \alpha \frac{\rho_2 U_2}{T_2}}_{\textcircled{A}};\ \underbrace{\frac{\rho_1 U_1^2}{L_1} = \alpha \frac{\rho_2 U_2^2}{L_2}}_{\textcircled{B}};\ \underbrace{\frac{\mu_1 U_1}{L_1^2} = \alpha \frac{\mu_2 U_2}{L_2^2}}_{\textcircled{C}};\ \underbrace{\frac{P_1}{L_1} = \alpha \frac{P_2}{L_2}}_{\textcircled{D}}$$

where α is a proportionality constant.

If we divide $\textcircled{B}$ by $\textcircled{C}$, we obtain:

$$\frac{\frac{\rho_1 U_1^2}{L_1}}{\frac{\mu_1 U_1}{L_1^2}} = \frac{\alpha \frac{\rho_2 U_2^2}{L_2}}{\alpha \frac{\mu_2 U_2}{L_2^2}} \Leftrightarrow \frac{\rho_1 U_1 L_1}{\mu_1} = \frac{\rho_2 U_2 L_2}{\mu_2}$$

As you can see immediately, these are the Reynolds-numbers of System 1 and System 2 that have to match.

We proceed analogously to obtain a relationship between the time-scalings T_1 and T_2. To this end, we divide $\textcircled{B}$ by $\textcircled{A}$:

$$\frac{\frac{\rho_1 U_1^2}{L_1}}{\frac{\rho_1 U_1}{T_1}} = \frac{\alpha \frac{\rho_2 U_2^2}{L_2}}{\alpha \frac{\rho_2 U_2}{T_2}} \Leftrightarrow T_1 \frac{U_1}{L_1} = T_2 \frac{U_2}{L_2}$$

This relationship states that the smaller the velocity U_i or the bigger the characteristic length L_i, the bigger is the time-scaling T_i.

To obtain a ratio between the pressure scalings P_1 and P_2 we divide $\textcircled{D}$ by $\textcircled{B}$:

$$\frac{\frac{P_1}{L_1}}{\frac{\rho_1 U_1^2}{L_1}} = \frac{\alpha \frac{P_2}{L_2}}{\alpha \frac{\rho_2 U_2^2}{L_2}} \Leftrightarrow \frac{P_1}{\rho_1 U_1^2} = \frac{P_2}{\rho_2 U_2^2}$$

Thus, the bigger the density ρ_i or the squared velocity U_i^2, the bigger is the pressure scaling P_i.

Knowing about these scaling arguments is crucial when making e.g. wind-tunnel experiments with down-scaled prototypes.

Furthermore, it comes in handy when working with grid-units: In grid-units, we define μ, ρ and t in terms of grid-cell lengths (in contrast to SI-units where we use m or s). By rescaling μ, ρ and the time-step dt according to the ratios discussed above, we can easily change for example the resolution of the grid without having to write new code.

Figure 3.8 shows examples of qualitative behavior of the Navier-Stokes equation at different Reynolds-Numbers: At very low Reynolds-Numbers, the flow becomes time-reversible, which can be seen by the symmetry of the stream-lines around obstacles. At slightly higher Reynolds-Numbers ($Re \gtrapprox 10$), the flow forms a laminar wake behind obstacles but remains steady ($\partial_t \boldsymbol{v} = 0$). When further increasing the Reynolds-Number ($Re \gtrapprox 90$), the wake begins to form a regularly oscillating pattern of vortices. This pattern is also called a von Kármán vortex street[15]. At even higher Reynolds-numbers ($Re \gtrapprox 1000$), the wake dynamics become turbulent resulting in a chaotic and unpredictable behavior.

An intuition for this behavior could be, that at low Reynolds-numbers the viscous forces dominate in the momentum equation and, thus, the PDE resembles an elliptic PDE. In contrast, at high Reynolds-numbers, the non-linear advection term dominates over the friction term in the momentum equation and produces chaotic, turbulent behavior.

[15] Theodore von Kármán, Austria-hungarian physicist (1881-1963)

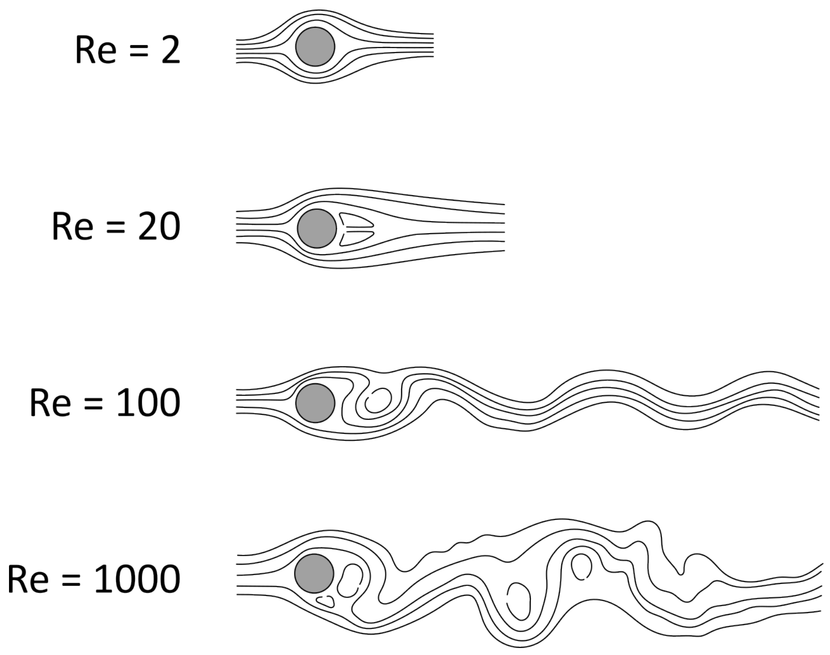


Figure 3.8: At different Reynolds-numbers, the dynamics of fluids exhibit vastly different qualitative behaviors.

3.2.8 Important effects

Fluid dynamics comprise a wide variety of different behaviors that sometimes seem to contradict our intuitions. This makes fluid dynamics a very challenging but also interesting deep-learning problem for neural networks. In this section, we highlight some important effects that need to be reproduced by a neural surrogate model in order to obtain realistic simulations.

Venturi effect If the fluids viscosity μ and external forces $\boldsymbol{f}_{\text{ext}}$ can be assumed negligible, Bernoulli's[16] principle states for steady flows ($\partial_t \boldsymbol{v} = 0$) that the pressure p decreases as the fluid's velocity $\boldsymbol{v}$ increases:

$$p + \frac{\rho}{2}\|\boldsymbol{v}\|^2 = \text{const}$$

Bernoulli's principle can be counterintuitive if the velocity field $\boldsymbol{v}$ gets confused with the root-mean-square velocity of individual fluid particles $\left(v_{\text{rms}} = \sqrt{\langle v^2 \rangle}\right)$. In this case, it might be tempting to apply the kinetic theory of ideal gases, which states that v_{rms}^2 is proportional to temperature, which in turn is proportional to pressure. Or in simpler terms: if particles inside a fluid become faster, they slam faster into the domain boundaries and thus increase pressure. However, $\boldsymbol{v}$ does not describe the velocities of individual particles but the average speed of particles at a certain point inside the domain and the temperature can be considered

[16] Daniel Bernoulli, Swiss mathematician and physicist (1700 - 1782)

to be fixed within the fluid. Figure 3.9 of the Venturi[17] effect provides a more intuitive explanation: A pipe contraction results in a higher flow velocity inside the narrow section in order to satisfy the incompressibility equation (Equation 3.5). Thus, the fluid entering the narrow section must be accelerated while the fluid exiting the narrow section must be decelerated. However, if we follow our initial assumption and neglect $\mu\Delta \boldsymbol{v}$ and $\boldsymbol{f}_{\text{ext}}$ of the momentum equation (Equation 3.6), there is only the negative gradient of the pressure field left to accelerate the fluid at the entry of the narrow section, or, respectively, a positive pressure gradient to decelerate the fluid at the exit of the narrow section. Thus, the pressure within the narrow section must be smaller compared to the wider sections. A more formal proof of Bernoulli's principle can be derived from the law of conservation of energy where p is considered a potential energy term and $\frac{\rho}{2}\|\boldsymbol{v}\|^2$ a kinetic energy term so the sum of both terms must stay constant.

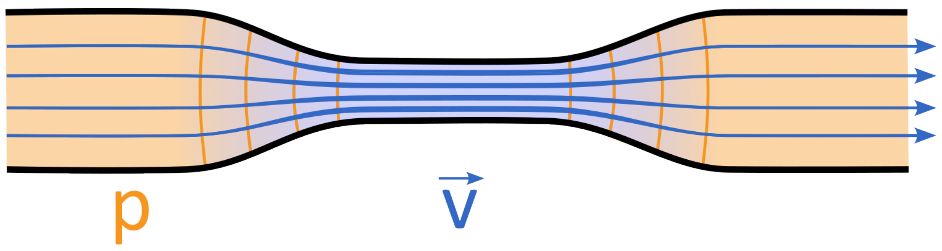


Figure 3.9: Qualitative visualization of the Venturi effect: The contraction of a pipe (e.g. a stenosis of an artery) results in a higher flow velocity and a lower pressure field inside the narrow section.

Magnus effect The Magnus[18] effect appears at spinning objects within a flow field and produces a charactistic Magnus force ($\boldsymbol{f}_m \propto \boldsymbol{v} \times \boldsymbol{\omega}$) proportional to the cross-product of the fluid velocity $\boldsymbol{v}$ and the objects angular velocity $\boldsymbol{\omega}$. A qualitative visualization of this effect is provided in Figure 3.10. Here, the pressure fields can be explained by Bernoullis principle (see paragraph above), which states that higher fluid velocities result in lower pressure fields and vice versa. At the top of the rotating cylinder in Figure 3.10, the surface of the cylinder moves in the same direction as the fluid resulting in a higher velocity compared to the bottom of the cylinder, where the surface moves in the opposite direction of the fluid and thus slows the fluid down due to viscous friction. This results in a lower pressure field at the top of the cylinder compared to the bottom and consequently in a Magnus force pointing upwards.

The Magnus effect is harnessed in many common ballsports such as soccer, golf or tennis by spinning the ball to deflect its path. Furthermore, prototypes of planes and sailboats have been developed that replaced wings and sails by rotating cylinders.

Karman vortex street The Karman vortex street is a phenomenon that produces periodically oscillating flow patterns behind an obstacle (see e.g. Figure 3.8 at Re=100). It starts to appear at Reynolds-Numbers bigger than 90 and plays an important role in engineering as the

[17] Giovanni Battista Venturi, Italian physicist (1746 – 1822)
[18] Heinrich Gustav Magnus, German physicist (1802 - 1870)

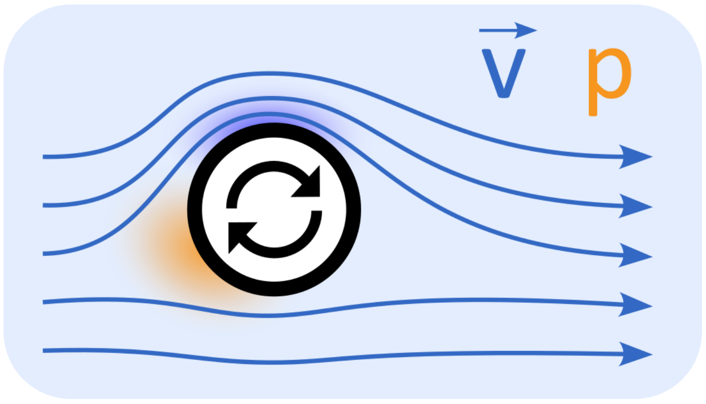


Figure 3.10: Qualitative visualization of the Magnus effect: A spinning object causes a characteristic reduced pressure field on the side that moves along the fluids velocity.

oscillating flow patterns can cause resonance when hitting the natural frequency of an object such as e.g. a chimney or an antenna and, thus, result in material fatigue.

3.2.9 Simplifications of the Navier-Stokes equation

Solving the full incompressivle Navier-Stokes equation is hard. Thus, several simplifications were developed that hold under certain assumptions:

Incompressible Stokes equation Stokes flows describe incompressible fluids at very low Reynolds numbers ($Re \ll 1$). In this case, accelerations within the fluid become nearly zero ($\rho \frac{D}{Dt} \boldsymbol{v} \longrightarrow 0$) and the momentum equation becomes:

$$\mu \Delta \boldsymbol{v} - \nabla p + \boldsymbol{f}_{\text{ext}} = 0$$

Since solutions of this equation are steady state and usually describe highly viscous fluids, they are sometimes also referred to as "creeping flows".

Incompressible Euler equation In contrast to Stokes flows, the Euler equation describes incompressible fluids at very high Reynolds numbers ($Re \longrightarrow \infty$). Under the assumption that the fluids viscosity is nearly zero ($\mu \longrightarrow 0$), the momentum equation becomes:

$$\rho \frac{D}{Dt} \boldsymbol{v} = \rho(\partial_t \boldsymbol{v} + \boldsymbol{v} \cdot \nabla \boldsymbol{v}) = -\nabla p + \boldsymbol{f}_{\text{ext}}$$

Burgers equation Burgers[19] equation describes fluids without a pressure term and external forces:

$$\rho \frac{D}{Dt} \boldsymbol{v} = \rho(\partial_t \boldsymbol{v} + \boldsymbol{v} \cdot \nabla \boldsymbol{v}) = \mu \Delta \boldsymbol{v}$$

In contrast to the incompressible Stokes / Euler / Navier-Stokes equations, the Burgers equation does not enforce incompressibility and solutions of the Burgers equation can form interesting shock patterns.

3.3 Damped Wave Equation

The damped wave equation is a linear partial differential equation that can be used to describe for example the dynamics of a thin membrane at small excitations (see Figure 3.11). It imposes the following equations on a displacement field z and a velocity field v_z:

$$\partial_t z = v_z \tag{3.11}$$

$$\partial_t v_z = k\Delta z - \delta v_z \tag{3.12}$$

Here, k is a factor that can be interpreted as a tension-force on the membrane divided by its density. A common choice is to use c^2 instead of k where c is the propagation speed of waves within the medium. δ can be considered a damping factor proportional to the velocity of the membrane.

These 2 equations are often combined into a single second order PDE in time by plugging Equation 3.11 into v_z of Equation 3.12:

$$\partial_t^2 z = k\Delta z - \delta \partial_t z \tag{3.13}$$

However in the following, for the sake of intuitive simplicity and consistency with our Spline-PINN paper, we stick with the first notation.

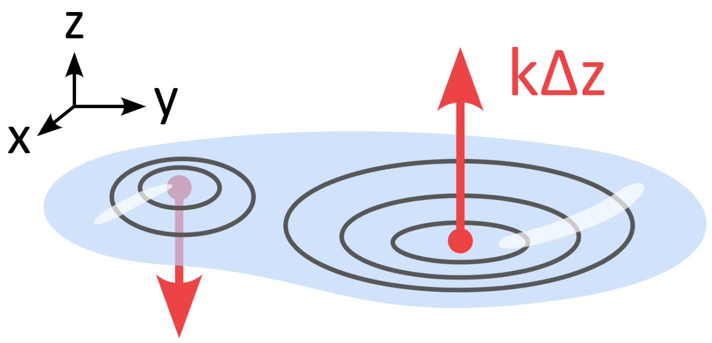


Figure 3.11: On a thin membrane described by $z(t, x, y)$, tension forces act proportional to the curvature Δz.

[19] Johannes Martinus Burgers, Dutch physicist (1895 - 1981)

3.3.1 Important effects

The wave equation exhibits several interesting effects. In particular, the Doppler effect and interference patterns, which will be covered in the following.

Doppler effect The Doppler[20] effect appears if an oscillator moves at a certain speed through the domain. This results in a shortened wavelength in front of the oscillator and an elongated wavelength behind the oscillator (see Figure 3.12, left). The Doppler effect can be observed, for example, in the changing pitch of a siren when an ambulance passes by and has important applications in radar technology, for example, where the reflected wavelength of an object can be used to determine its speed.

Interference Interference patterns appear when waves of the same frequency get superimposed from different directions. This leads to regions where waves accumulate or cancel out each other, which is also referred to as constructive and destructive interference (see Figure 3.12, right). These interference patterns play an important role in many fields such as acoustics (e.g. in active noise-cancelling (ANC) earbuds) or quantum mechanics (see e.g. the double-slit experiment).

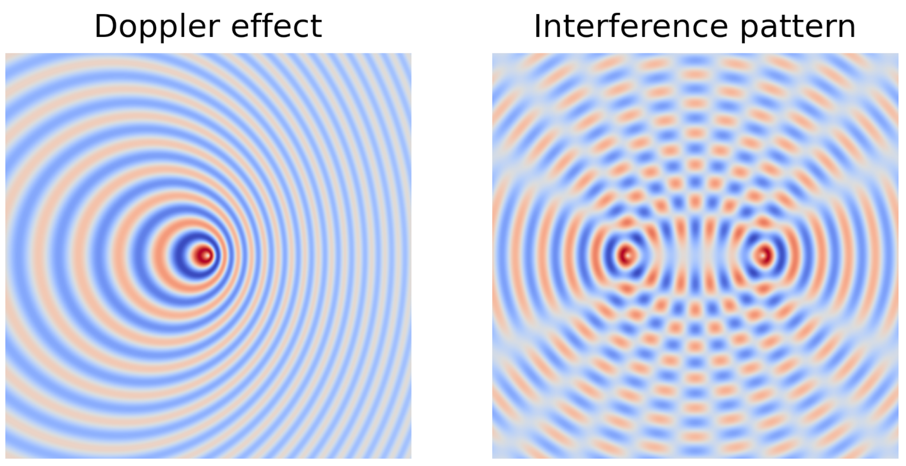
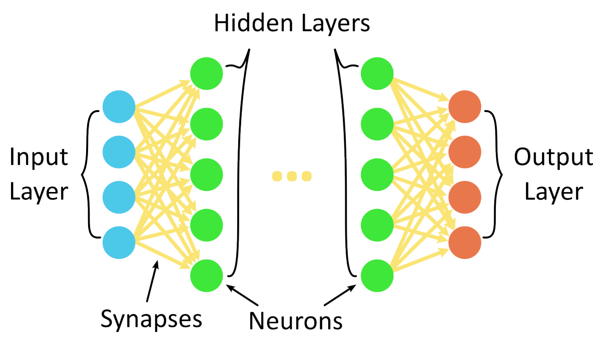


Figure 3.12: Left: Doppler effect of an oscillator moving to the right. Right: Interference patterns of two oscillators

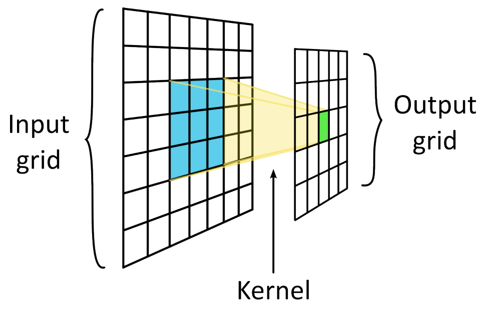
3.4 Artificial Neural Networks

Artificial neural networks (ANNs) can be considered as a special class of functions inspired by the functioning of biological neural networks (see Figure 3.13 a).

[20] Christian Andreas Doppler, Austrian mathematician and physicist (1803 - 1853)



(a) Basic schematic of a feed-forward neural network: A neural network consists of several layers that comprise individual neurons ("units") and are interconnected by synapses.



(b) Convolutional neural networks usually work on grid data. The synaptic weights are comprised in a kernel that is swept over the input grid to compute the output.

Figure 3.13: Fundamental concepts of artificial neural networks.

They consist of individual units ("neurons") that take a weighted sum of input signals ("dendrites with synaptic weights") and apply a non-linear activation function to compute an output value ("firing rate"). Usually, multiple units form a layer and in fully connected NN, every unit of a certain layer is connected with all units of the preceding layer. The first layer is usually called the input layer and the last layer is called the output layer. However, in contrast to biological neural networks, ANNs are trained by gradient descent instead of Hebbian learning rules and often require a large amount of (labelled) training data that is not temporally ordered. Furthermore, while biological neurons produce individual spikes, units in ANNs return firing rates and rely on activation functions that often lack biological plausibility (e.g. activation functions can be negative, unbounded or even periodic while firing rates of biological neurons must be bigger than zero and saturate at high frequencies). On top of that, ANNs are often feed-forward networks while biological NN inherently contain a high degree of recurrent connectivity.

Despite these differences, over the past decade, ANNs have proven to be extremely successful in a variety of applications ranging from computer vision to natural language processing to computer graphics to reinforcement learning and even outperform humans in several specific tasks such as image classification [He et al., 2015] or board games [Silver et al., 2016]. Lately, ANNs have also found their way into numerics and solving partial differential equations.

In the following, we introduce different neural architectures that can be used to approach PDEs and that are of importance in this work.

3.4.1 Implicit Neural Representation Networks

Neural networks can be used to learn scene representations by mapping domain coordinates onto field values. For example, a neural network can represent a 3D geometry by learning a signed distance function that maps coordinates x, y, z onto signed distance values s. Further

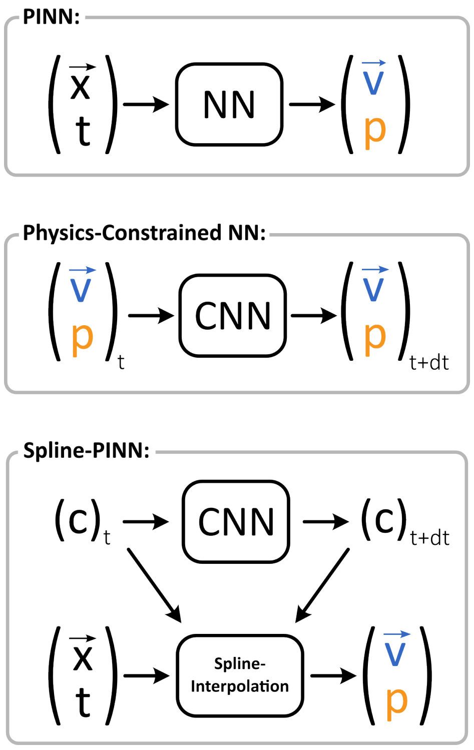


Figure 3.14: Conceptual differences between Physics-Informed NNs, Physics-Constrained NNs, and our Spline-PINN architecture. While the original PINN architecture directly maps domain coordinates to field values, Physics-Constrained NN map discrete field representations from a time-point t to a time-point $t + dt$. Spline-PINNs combine both approaches and build a latent representation based on Hermite-Spline coefficients that can be continuously interpolated within the domain between t and $t + dt$.

applications[21] include image representations (map coordinates x, y onto color values r, g, b), radiance fields (map coordinates and view directions x, y, z, ϕ, θ onto color and density values r, g, b, σ) or partial differential equations (map domain coordinates $x, y, z, t \in \Omega$ onto field values, such as for example flow and pressure fields v, p).

Physics-Informed Neural Networks (PINNs) In this work, the application of implicit neural representations to solve PDEs is of particular interest as it allows training using gradient descent on a *physics-informed* loss.

To this end, derivatives of the neural representation can be obtained by automatic differentiation and a loss function can be formulated such that residuals of the underlying PDE get penalized. This way, the physics-informed neural network is automatically incentivised to represent a solution of the PDE. Note that, in order to compute higher order derivatives of the neural network by automatic differentiation, proper activation functions must be chosen to ensure differentiability up to the degree of the underlying PDE. Another benefit of

[21] A more elaborate, curated list of implicit neural representations can be found at Sitzmann, 2021.

neural implicit representations is that they deliver continuous results and can overcome the curse of dimensionality. However, implicit neural representations need to be retrained if the domain-geometry is changed which is computationally expensive and prohibits their use in interactive scenarios.

3.4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) usually work on grid representations and are widely applied for example in audio-processing (1D grids), image classification / segmentation / generation (2D pixel grids) or in analysing 3D CT/MRI scans (3D voxel grids). Instead of connecting every grid point to every unit in a layer (see fully connected NN), in a CNN, each unit considers only a local receptive field. Furthermore, the synaptic weights are shared between the individual units of a layer. This allows for efficient implementations based on convolutions (see Figure 3.13 b) by sweeping a kernel over the input grid and requires fewer parameters compared to a fully connected NN while taking local correlations and translational equivariance of grid values into account.

U-Net A popular CNN architecture for segmentation tasks is the U-Net architecture by Ronneberger et al., 2015. U-Nets make use of pooling layers and strided transposed convolutions to incorporate global features at low resolution layers while preserving fine details at high resolution layers through shortcut connections. This architecture is of particular importance for solving the the Navier-Stokes equation as forces from viscous friction act locally, while the pressure field needs to be solved globally.

Physics-Constrained Neural Networks In contrast to implicit PINNs, CNNs do not represent field values implicitly but explicitly map a grid-based fluid state from one time-step t to the next time-step $t + dt$ (see Figure 3.14, Physics-Constrained NN). Thus, the residuals of the underlying PDE cannot be computed with automatic differentiation but we must use finite differences on the grid to compute a *physics-constrained* loss. While the discrete nature of physics-constrained approaches can lead to discretization-artifacts, these methods are often able to generalize to new domain-geometries without requiring retraining and can be used in interactive scenarios.

Marker And Cell grid A convenient grid representations for fluid simulations is the Marker And Cell (MAC) grid (see Figure 3.15). Its staggered arrangement of field values allows to compute centered finite differences more efficiently compared to on a regular grid. For example, when computing the divergence of the velocity field, we can simply compare the incoming and outgoing velocity components of a cell that are orthogonally placed at the centers of the cell faces. In contrast, on a regular grid, computing the divergence would require taking both adjacent cells into account thus increasing dx by a factor of two. Similarly, the momentum equation can be handled more efficiently as well, since finite-difference gradients of the pressure field align automatically with the corresponding velocity components.

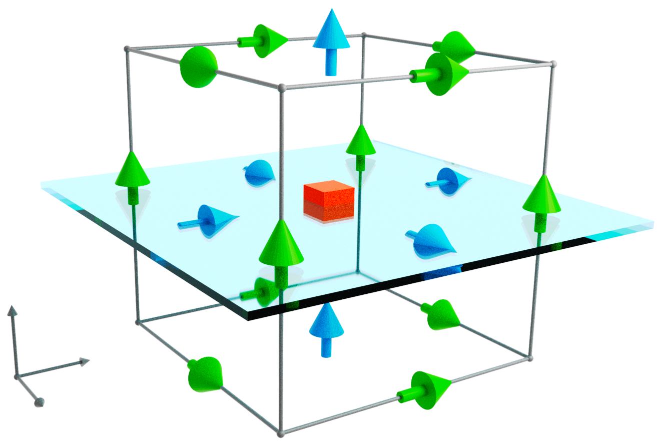


Figure 3.15: Single unit cell of a 3D Marker And Cell (MAC) grid. On a MAC grid, the different field values are placed in a staggered manner to facilitate the computation of finite differences. The red box denotes pressure, blue arrows denote the x, y, z components of the velocity field and the green arrows denote the x, y, z components of the vector potential. The glass plane marks the components contained in a 2D MAC grid.

3.5 Spline-Interpolation

In our work on Spline-PINNs we make use of spline-interpolation to obtain continuous field representations from a discrete set of coefficients on a uniform grid (see Figure 3.14). This allows to combine the advantages of physics-informed and physics-constrained approaches, namely continuous results with fewer discretization artifacts and interactive applications due to the generalization capabilities of CNNs on uniform grids.

Splines [22] are piecewise polynomials that are widely used in computer graphics and computer aided design. They can be defined as follows:

Let $[a, b] \subset \mathbb{R}$ be an interval in $\mathbb{R}$ and $t_i \in [a, b], i = 0, 1, ..., k$ be so called *knots* that fulfill $a = t_0 \leq t_1 \leq ... \leq t_k = b$ and divide the interval $[a, b]$ into k pieces $[t_i, t_{i+1}]$. Furthermore, let P_i be polynomials of degree n that map $[t_i, t_{i+1}] \to \mathbb{R}$. Then, the piecewise polynomial:

$$S : [a, b] \longrightarrow \mathbb{R}, t \mapsto S(t) = \{P_i(t) \text{ if } t \in [t_i, t_i + 1]$$

is called a *spline* of smoothness r, if $S \in C^r$ is r times continuously differentiable. The resulting space of splines is often denoted as $\mathcal{S}_n^r$.

[22] The term "spline" originates from flexible wooden strips used by draftsmen to draw smooth lines (an interesting blog post about the history of splines can be found at Noe, 2016)

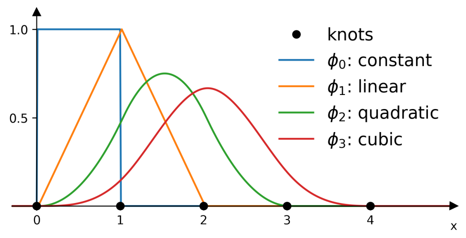


Figure 3.16: Cardinal B-spline kernels of degree 0 (constant), 1 (linear), 2 (quadratic) and 3 (cubic).

To facilitate the handling of such a spline space $\mathcal{S}_n^r$, usually, a set of basis splines $\phi_j(x) \in \mathcal{S}_n^r$ is introduced that already fulfills the given smoothness properties. This allows to form arbitrary splines of that spline space by linear combination of $\phi_j(x)$ with spline coefficients c_j:

$$s(x) = \sum_j c_j \phi_j(x)$$

A popular basis for splines in $\mathcal{S}_n^{n-1}$ are B-splines. They are denoted as $B_{i,n}(x)$ and defined as functions in $\mathcal{S}_n^{n-1}$ that only have support in $[t_i, t_{i+n+1}]$ and sum up to 1 ($\sum_i B_{i,n}(x) = 1$). These constraints yield unique solutions for $B_{i,n}(x)$ that can be elegantly computed by the De Boor's [23] recursion formula:

$$B_{i,0}(x) = \begin{cases} 1, \text{if } t_i \leq x < t_{i+1} \\ 0 \text{ else} \end{cases} \tag{3.14}$$

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x) \tag{3.15}$$

Since we are using CNNs that work on uniform grids, we are particularly interested in equidistant distributions of knots ($t_{i+1} - t_i = h$). This makes handling of splines a lot easier as the basis kernels always have the same shape and thus can be efficiently combined at different grid offsets using convolutions. B-splines with equidistant knots are also called *cardinal* B-splines and are shown in Figure 3.16 for degrees $0, 1, 2, 3$. Apart from De Boor's recursion formula, another equivalent way to compute cardinal B-splines is to set $\phi_0(t)$ equal to a box function that returns 1, if $t \in [0, h]$ and 0 everywhere else. Then, $\phi_n = \phi_0 * \phi_{n-1}$ is recursively defined as a convolution of ϕ_0 with the preceding B-spline ϕ_{n-1}. As follows from the central limit theorem, for large n, ϕ_n converges towards a normal distribution and the support of ϕ_n grows linearly with n. Thus, cardinal B-splines of order n can be considered as smoothing filters that corresponds to a convolution with a Gaussian kernel approximated by piecewise polynomials of order n or, if we go into frequency domain, they can be considered as a multiplication by $\text{sinc}(x)^{n+1}$.

However, in our work on Spline-PINNs, we rely on Hermite splines. In contrast to B-splines, Hermite spline representations only span two neighboring spline segments and specify the

[23] Carl-Wilhelm Reinhold de Boor, German-American mathematician (*1937)

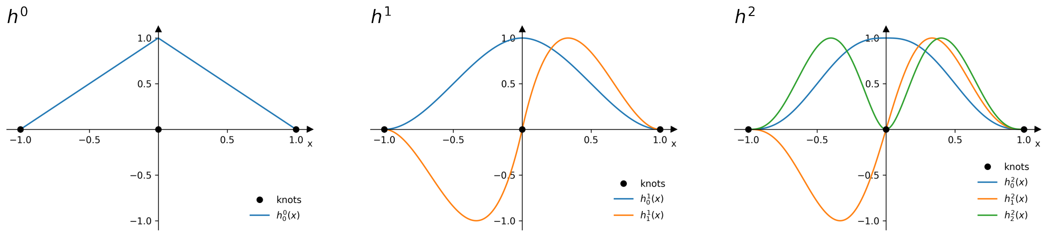


Figure 3.17: Hermite-spline basis kernels.

value and first n derivatives at each knot. Thus, Hermite basis functions (see $h_i^j(x)$ in Figure 3.17) have a smaller support compared to B-splines for $n > 2$ and thus can be computed more efficiently due to smaller convolutional kernel sizes and can restore high frequency signals more easily. For Hermite splines, each interval must have $2n+2$ degrees of freedom since both adjacent knots impose $n+1$ constraints (for the first n derivatives as well as the actual function value). Thus, the piecewise polynomials must be at least of degree $2n+1$ and the basis splines h_i^n are in S_{2n+1}^n. The polynome coefficients of the hermite spline kernels can be obtained by solving the corresponding systems of linear equations.

For B-splines as well as Hermite splines, higher dimensional grid representations can be obtained by taking the outer product of the individual basis functions. Finally, these basis functions can be combined on a multidimensional grid with coordinates $\hat{x}, \hat{y}, \hat{t} \in \hat{X} \times \hat{Y} \times \hat{T}$ as follows in order to obtain a continuous function $g(x,y,t)$:

$$g(x,y,t) = \sum_{\substack{i,j,k \in [0:l]\times[0:m]\times[0:n] \\ \hat{x},\hat{y},\hat{t} \in \hat{X}\times\hat{Y}\times\hat{T}}} c_{\hat{x},\hat{y},\hat{t}}^{i,j,k} h_i^l(x-\hat{x}) h_j^m(y-\hat{y}) h_k^n(t-\hat{t}) \tag{3.16}$$

here, $c_{\hat{x},\hat{y},\hat{t}}^{i,j,k}$ are the discrete spline coefficients that can be efficiently processed by a CNN and $h_i^l(x-\hat{x})h_j^m(y-\hat{y})h_k^n(t-\hat{t})$ is the outer product of Hermite spline kernel functions. The orders l, m, n of the Hermite spline kernel in x, y, t directions must be chosen such that $g(x,y,t)$ is sufficiently smooth to be processed by the differential operators of the underlying PDE (e.g.: if the underlying PDE is of third order in x and second order in y, l must be at least 2 and m must be at least 1).

Part II

Publications

Chapter 4

Learning Incompressible Fluid Dynamics from Scratch - Towards Fast, Differentiable Fluid Models that Generalize

Nils Wandel, Michael Weinmann, and Reinhard Klein.
"Learning Incompressible Fluid Dynamics from Scratch - Towards Fast, Differentiable Fluid Models that Generalize."
9th International Conference on Learning Representations, ICLR, 2021.

4.1 Summary

In our work on "Learning Incompressible Fluid Dynamics from Scratch - Towards Fast, Differentiable Fluid Models that Generalize", we developed a novel approach that allows to learn the dynamics of the incompressible Navier-Stokes equation in two dimensions by a convolutional neural network without any precomputed ground truth data. To this end, we use a physics constrained loss function on a Marker-and-Cell grid in conjunction with a novel training data recycling strategy. Furthermore, we argue that, using back-propagation through time, the learned fluid model is fully differentiable and consequently show, how this can be exploited in a flow control application.

Solving the incompressible Navier-Stokes equation is a hard problem as in general there exist no analytic solutions and classical numerical methods are computationally expensive. Following the advances of deep-learning based methods over the past decade, recent works also investigate the application of such strategies in the realm of computational fluid dynamics in order to lower the computational burden of traditional methods.

Prior work that aims to learn solutions of partial differential equations on grids using convolutional neural networks includes Tompson et al., 2017; Thuerey et al., 2019; Zhu et al., 2019; Geneva and Zabaras, 2020. While [Thuerey et al., 2019] relies on a large set of training data, further works [Tompson et al., 2017; Zhu et al., 2019; Geneva and Zabaras, 2020] use a physics constrained loss to reduce the amount of required training data. Zhu et al., 2019; Geneva and Zabaras, 2020 even completely avoid any training data, however, they consider Darcy flows and Burgers equation instead of the incompressible Navier-Stokes equation and do not allow for dynamic domain boundaries. Computational pipelines that aim to fully simulate incompressible fluids with dynamic domain boundaries often require additional components to yield stable simulations such as a particle tracer [Tompson et al., 2017] or a differentiable fluid solver [Um et al., 2020]. In contrast, our approach allows to train a neural surrogate model on the full incompressible Navier-Stokes equation that interacts with changing domain boundaries without the need for any ground truth data or additional components in the pipeline such as a particle tracer or a differentiable fluid-solver.

To this end, we use a convolutional neural network $nn : (\boldsymbol{a}^t, p^t, \Omega^t, \boldsymbol{v}_d^t) \mapsto (\boldsymbol{a}^{t+dt}, p^{t+dt})$ based on the U-Net architecture [Ronneberger et al., 2015] to map a fluid state and boundary conditions at timestep t to a new fluid state at timestep $t + dt$. The fluid state is defined by a vector-potential $\boldsymbol{a}^t$ and a pressure field p^t on a 2D Marker-and-Cell (MAC) grid (see Figure 3.15) which allows to compute finite-differences very conveniently. For the velocity-field $\boldsymbol{v}^t$, we exploit the Helmholtz decomposition theorem by setting $\boldsymbol{v}^t = \nabla \times \boldsymbol{a}^t$ as the curl of the vector potential $\boldsymbol{a}^t$. This automatically ensures that the incompressibility equation ($\nabla \cdot \boldsymbol{v} = 0$) is fulfilled and reduces degrees of freedom of the fluid state since for symmetric reasons in 2D only the z-component of $\boldsymbol{a}$, a_z, is of interest. The domain geometry Ω^t is given by a binary mask and the Dirichlet boundary conditions are given by $\boldsymbol{v}_d^t$.

To train this model without precomputed ground truth data we introduce a data recycling strategy that makes use of a training pool and a physics constrained loss (see Figure 4.1). First, we 0-initialize a training pool with initial vector-potentials $\boldsymbol{a}_i^0 = 0$ and pressure fields $p_i^0 = 0$. Furthermore, we assign for every sample i of the training pool randomized domain geometries Ω_i^0 and Dirichlet conditions $\boldsymbol{d}_{d,i}^0$. Now, in every training step, we draw a random mini-batch $(\boldsymbol{a}_i^t, p_i^t, \Omega_i^t, \boldsymbol{v}_{i,d}^t)$ from the training pool and make a prediction for $(\boldsymbol{a}_i^{t+dt}, p_i^{t+dt})$ using the PDE model. Based on these predictions, we can compute a physics-constrained loss that penalizes residuals of the momentum equation and deviations from the given boundary conditions. By using gradient descent, we can optimize the fluid model to make better predictions over time. Finally, to close the cycle, we feed the predictions of the fluid model back into the training pool. This way, we not only optimize the model but also fill the training pool over time with more and more realistic training data. One could consider this strategy also as a way to generate our own training data on the fly during training.

We tested our trained models at a wide range of Reynolds-numbers and obtained time-reversible, laminar and turbulent flows. Our models were able to successfully reproduce important qualitative effects such as the Magnus effect, Karman Vortex streets or the Bernoulli effect and run at over 300 iterations per second on a 100×100 grid. Furthermore, we compared our trained models to a recent state of the art differentiable fluid solver (Phiflow) that relies on a MAC grid as well and found our approach to be superior with respect to computational

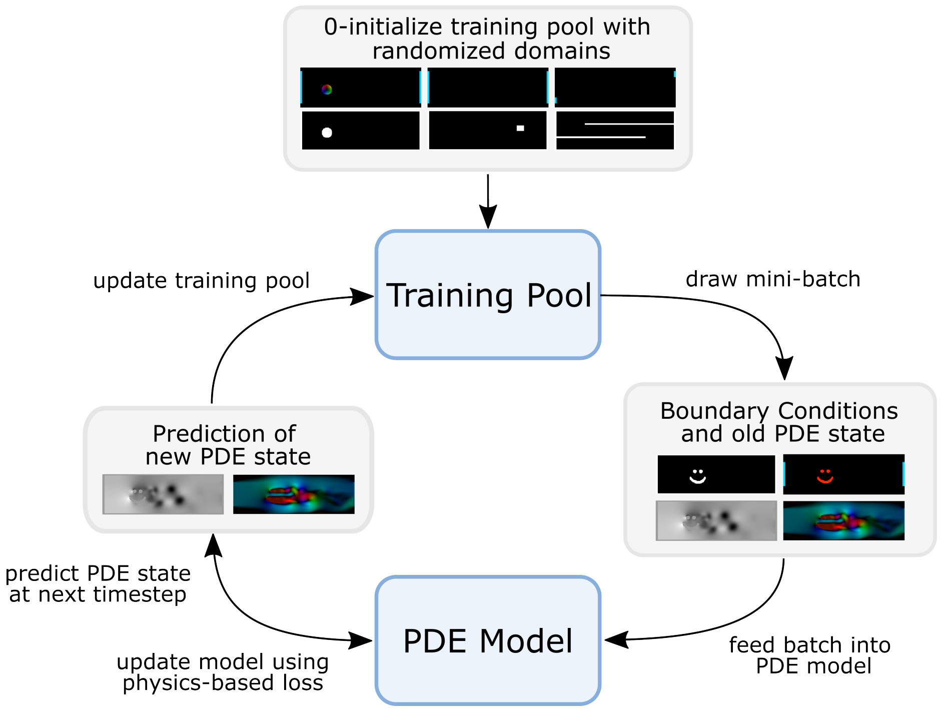

0-initialize training pool with randomized domains
Training Pool
update training pool
draw mini-batch
Prediction of new PDE state
Boundary Conditions and old PDE state
predict PDE state at next timestep
PDE Model
update model using physics-based loss
feed batch into PDE model

Figure 4.1: Using a training cycle we not only improve the PDE model but also the pool of training domains in every turn. Since the training pool can be 0-initialized, no precomputed training data is needed.

complexity as well as accuracy for viscous fluids. Finally, we demonstrated that our trained models can be used as an efficient differentiable fluid solver by controlling the fluid velocity using gradient descent in order to produce a vortex street behind an obstacle with a specified frequency.

Code and pretrained models to reproduce our results are made publicly available on Github:

`https://github.com/wandeln/Unsupervised_Deep_Learning_of_Incompressible_Fluid_Dynamics`

4.2 Author Contributions

In this publication, I developed and implemented the code including the novel training cycle, the physics-constrained loss based on finite differences on a 2D MAC grid and the control experiment. Furthermore, I conducted the presented evaluations.

Chapter 5

Teaching the Incompressible Navier-Stokes Equations to Fast Neural Surrogate Models in 3D

Nils Wandel, Michael Weinmann, and Reinhard Klein.
"Teaching the Incompressible Navier-Stokes Equations to Fast Neural Surrogate Models in 3D."
Physics of Fluids, 2021.
DOI: 10.1063/5.0047428

5.1 Summary

In our work "Teaching the Incompressible Navier-Stokes Equations to Fast Neural Surrogate Models in 3D" we introduced significant extensions to our previous work on fluid simulations that worked only in two dimensions (see Section 4). In particular we focused on two major improvements: First, we went from two dimensions to three dimensions. This not only imposes much higher computational and memory requirements but also requires learning all 3 degrees of freedom for the vector-potential $\boldsymbol{a}$ as opposed to only one for a_z in the two-dimensional case and thus significantly increases the complexity of the learning task [1]. Second, we aimed to use the same model in various fluid settings with different viscosities and densities by conditioning the fluid model on the fluids viscosity and density. In our previous work, handling different fluids required different, separately trained models.

Accelerating or refining 3D fluid simulations on a grid using neural networks motivated several prior works such as Tompson et al., 2017; Xie et al., 2018; Um et al., 2020. However, these works require precomputed ground truth data and often incorporate additional parts in the pipeline such as a particle tracer [Tompson et al., 2017] or a differentiable fluid solver

[1] Indeed, learning a 3D vector potential was considered infeasible by reviewers of our previous publication in 2D, which highlights the difficulty of this learning task.

[Um et al., 2020] in order to obtain stable simulations over many timesteps. In this work, we wanted to simplify the pipeline and training process by building up on our previous experiences in 2D to obtain stable and versatile fluid models in 3D.

While the methodology of training a 3 dimensional fluid model without ground truth data shares commonalities with the 2 dimensional case, there are several significant differences that required reimplementing major parts of our code base from our previous work: First, a 3D marker and cell grid was implemented and the residuals of the Navier-Stokes equation were extended to 3D finite-difference kernels. In order to train the 3D fluid model without ground truth data, we rely again on the concept of a training cycle that was developed in our preceding publication. To this end, we 0-initialize a training pool of 3D fluid states consisting of vector-potentials $\boldsymbol{a}$ and pressure fields p. From this pool, we draw random mini-batches to make predictions using the fluid model. Then, we compute the physics-constrained loss function by taking the mean square residuals of the Navier-Stokes equation and boundary conditions and use gradient descent to optimize the fluid model. However, in contrast to our previous work, since we train the model on multiple viscosities and densities at the same time, we extended the training pool as well as the inputs of the fluid model by two additional channels that contain the viscosity and density parameters. Finally, we feed the model predictions back into the training pool in order to improve the training set with more realistic data as the fluid model gets better over time.

Using this approach, we were able to produce dynamic, interactive simulations of various fluids ranging from time-reversible to laminar to turbulent flows at Reynolds-Numbers ranging from 0.64 to 800 using only a single model. Since 3D grids are computationally expensive, we tested two neural network architectures: First, a 3D U-Net as proposed by Ronneberger et al., 2015 which is able to simulate a $64 \times 64 \times 128$ grid at 16 steps per seconds, and second, a pruned version with slightly lower accuracy that increased the inference speed to 36 steps per second. Our models are able to capture many intriguing effects in 3D such as Karman Vortex Streets or the Magnus effect and generalize well to new domain geometries that were not contained in the set of randomized domains of the training pool. By investigating the evolution of errors over time, we showed that our models deliver stable simulations over tousands of timesteps. Visualizations of the resulting flow and pressure fields were plotted in Paraview (an Example is shown in Figure 5.1). An animated video can be found on youtube: www.youtube.com/watch?v=tKcYJaJtHJE

To play with our interactive demo or to reproduce our results we made code and pretrained models publicly available on Github:

https://github.com/wandeln/Teaching_Incompressible_Fluid_Dynamics_to_3D_CNNs

5.2 Author Contributions

In this publication, I extended our previous work from 2D to 3D. This includes implementing new 3D training domains, a modified physics-constrained loss based on a 3D MAC grid and 3D convolutional neural network models that take additional input channels for fluid

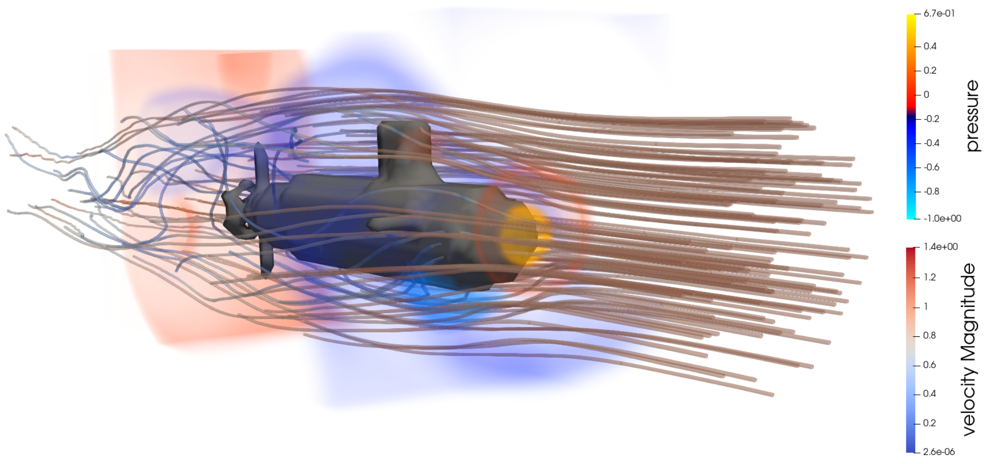


Figure 5.1: Exemplary visualization of our 3D network predictions in Paraview. The presented submarine geometry was not part of the set of randomized training domains, which demonstrates the generalization capabilities of our method.

parameters such as density and pressure. Finally, I evaluated our method and visualized the 3D results in Paraview.

Chapter 6

Spline-PINN: Approaching PDEs without Data using Fast, Physics-Informed Hermite-Spline CNNs

Nils Wandel, Michael Weinmann, Michael Neidlin, and Reinhard Klein.
"Spline-PINN: Approaching PDEs without Data using Fast, Physics-Informed Hermite-Spline CNNs."
Proceedings of the 36th AAAI Conference on Artificial Intelligence, 2022.
DOI: 10.1609/aaai.v36i8.20830

6.1 Summary

In our work "Spline-PINN: Approaching PDEs without Data using Fast, Physics-Informed Hermite-Spline CNNs", we combine the benefits of continuous physics-informed neural networks with the generalization capabilities of discrete physics-constrained approaches. To this end, we introduce Spline-PINNs - an extension to convolutional fluid models based on Hermite-Splines to obtain continuous field representations. By interpolating spline coefficients on a grid, we can evaluate field values as well as a physics-informed loss analytically at arbitrary points within the continuous domain Ω. This way, we aim to obtain higher fidelity results with fewer discretization artifacts compared to our previous finite-difference methods based on a marker and cell grid. To assess our method quantitatively we not only investigated the progression of the physics-based loss over the course of a simulation but also computed drag and lift coefficients on a benchmark setup [*The CFD Benchmarking Project* 2021] for fluid simulations and compared the results with an industrial CFD solver. To show that this versatile method is suitable for a wide range of PDEs, we not only investigated the incompressible Navier-Stokes equation, but also the damped wave equation.

Physics-informed Neural Networks (PINNs) as popularized by Raissi et al., 2019 offer continuous solutions of PDEs by learning an implicit neural representation that maps domain coordinates (e.g. t, x, y) to field values (e.g. v, p). By incorporating the residuals of the underlying PDEs directly into the training loss, they can be trained with little to no ground truth data. However, since these implicit neural network representations overfit to only one single domain, they do not generalize to new domain geometries and, thus, are not suitable for interactive applications. On the other hand, physics-constrained approaches rely on neural networks that map an explicit field representation (e.g. a discrete grid) from one timepoint t to another timepoint $t + dt$ and allow to compute a loss based on the residuals of a PDE using finite differences. These physics-constrained approaches have already shown to successfully generalize to new domains [Tompson et al., 2017; Wandel et al., 2021a] but are prone to discretization artifacts at high Reynolds numbers. Thus, in this work, we combine both approaches by using a convolutional neural network that evolves Hermite spline coefficients in time. This way, we obtain continuous solutions and get rid of discretization artifacts as in physics-informed neural networks while maintaining the good generalization capabilities of physics-constrained approaches. As we show in Figure 3.14, the Hermite spline coefficients can be interpreted as a latent grid-based representation of an implicit continuous field description.

To train the Spline-PINN model, we make use of a training cycle as introduced in our work on "Learning Incompressible Fluid Dynamics from Scratch - Towards Fast, Differentiable Fluid Models that Generalize" (see Figure 4.1 in Chapter 4): First, we initialize a training pool with randomized boundary conditions and set the initial conditions (i.e. the initial spline coefficients) to zero. Next, we draw a random mini-batch from that training pool and feed it into the PDE model in order to obtain the spline coefficients of the next timestep. This allows to evaluate the field values as well as their derivatives analytically at any point in space and time and thus to compute a physics-*informed* loss based on the mean squared residuals of the underlying PDEs. Since Hermite splines of order l are only of bounded variation up to derivatives of order $l + 1$, one has to choose a high enough order l depending on the order of the underlying PDE to be able to compute the loss. After updating the neural surrogate model based on that physics-based loss using gradient descent, we also update the training pool with the predicted spline coefficients. This way, in every training cycle, we not only improve the PDE model but also the training pool with more and more realistic spline coefficients.

Once the Spline-PINN model is trained, it can be used to infer fast solutions of PDEs. For the incompressible Navier-Stokes equation, we used a Spline-PINN to obtain fluid simulations and were able to reproduce various important effects such as the Magnus effect, Bernoulli effect or Kármán Vortex streets. To increase the interactivity of our demo and show the generalization capabilities of our approach, we also allow the user to paint arbitrary new boundary conditions inside the fluid domain (see Figure 6.1). The resulting continuous flow fields effectively reduced discretization artifacts that become a problem in MAC-grid based approaches at high Reynolds-numbers. Furthermore, we computed drag an lift forces of a liquid on a cylindrical obstacle by integrating pressure and viscous forces along the boundary of that obstacle. This way, we could determine the corresponding drag and lift coefficients on a standardized benchmark domain [*The CFD Benchmarking Project* 2021] and compare them

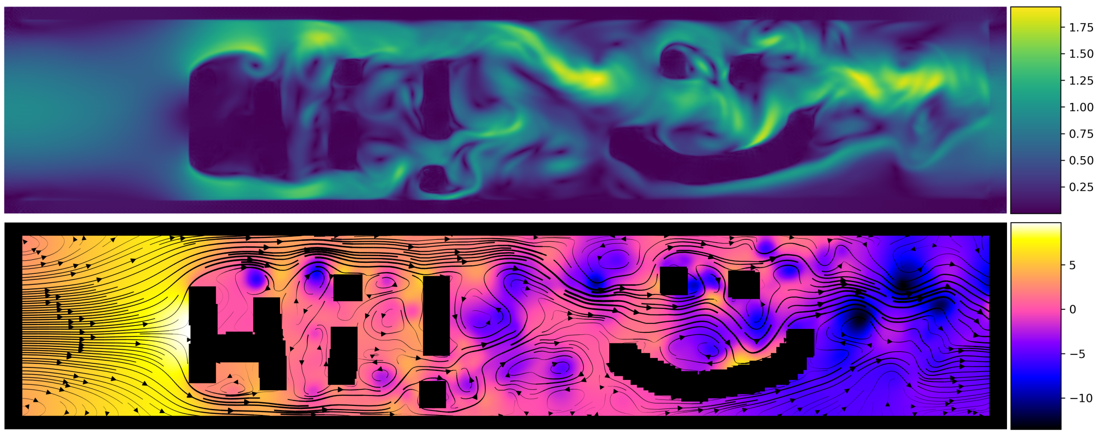


Figure 6.1: Flow velocity (top) and pressure / streamline visualizations (bottom) of the incompressible Navier-Stokes equation produced by our Spline-PINN approach. The boundaries were painted in an interactive setting to show the generalization capabilities of our method.

with official results and a professional industrial CFD solver (AnSys). Compared to previous data-free deep-learning approaches, the accuracy of the computed drag and lift coefficients got significantly closer to the accuracy of an industrial CFD solver while being considerably faster. To show that this approach is not limited to fluid simulations, we also trained a model on the damped wave equation. These experiments revealed further important effects such as interference patterns, the Doppler effect and wave reflections at the domain boundaries.

We made the code as well as pretrained models publicly available on Github so it is possible to try out our interactive demo and reproduce our results:

`https://github.com/wandeln/Spline_PINN`

6.2 Author Contributions

In this publication, I replaced the MAC grid that was used in our previous publication by Hermite-Splines. This way, a continuous field representation is obtained. To train the PDE models, I implemented a physics-informed loss function and to show the interactive speed and generalization capabilities of the resulting fluid simulations, I implemented a proof of concept paint application that allows to dynamically modify the domain boundaries. Furthermore, I compared the results to our previous MAC grid based solver and an implicit PINN. Special thanks go to Michael Neidlin who provided additional results with AnSys for a more in depth comparison with an industrial CFD solver. Finally, an additional Spline-PINN was developed to solve the damped wave equation in order to demonstrate that this concept is not limited to the Navier-Stokes equation but also works for further PDEs.

Part III

Conclusion

Chapter 7

Conclusion

In this section, we first summarize the contributions and results of the publications covered by this thesis. Then, current limitations of our method as well as potential further research directions to mitigate mentioned limitations are pointed out. Finally, we venture an outlook on future applications of physics-based deep learning and discuss the potential impact of our works on the field of physics-based deep-learning.

7.1 Summary

In this work, we explored how solutions of partial differential equations such as the incompressible Navier-Stokes equation or the damped wave equation can be efficiently approximated by neural surrogate models. To this end, we developed a novel training approach that allows to solve PDEs using convolutional neural networks without any precomputed ground truth data. For this purpose, we developed a training cycle that works as follows: At the beginning, a training pool of PDE states is 0-initialized and randomized domains and boundary conditions are generated that have to be solved by the PDE model. Then, we iterate over the following steps: First, we draw a random mini-batch from the training pool and feed it into the PDE model. Then, the PDE model makes a prediction for the fluid state at the next timestep. This prediction is used to compute a physics-based loss that penalizes errors of the underlying PDE. Next, we improve the parameters of the PDE model doing a gradient descent step. Finally, we update the PDE states of the mini-batch inside the training pool with the predictions made by the PDE model. This way, in every cycle, not only the neural PDE model gets improved by gradient descent but also the training pool gets improved with more and more realistic PDE states. Since the training pool is 0-initialized, no precomputed ground truth data is required in the entire training process.

In our papers, we applied this training strategy to different state representations based on a MAC grid and Hermite-splines, different neural surrogate models as well as different PDEs such as the incompressible Navier-Stokes equation and the wave equation.

In our first work, we represent the state of a fluid domain by a pressure and velocity field on a 2 dimensional marker and cell grid and trained a convolutional neural network (U-Net) to solve the incompressible Navier-Stokes equation. The marker and cell grid allows to efficiently compute a physics-constrained loss based on finite differences. After training, we obtained fast simulations that allowed for over 300 iterations per second on a 100×100 grid. This speed-up compared to traditional methods was made possible since only one single forward pass of the fluid state through a neural surrogate model is needed in order to obtain the next state instead of having to solve large systems of equations that are of high computational complexity. Furthermore, we observed that our fluid models generalize well to new domain geometries that were not considered during training. We tested this approach for a wide range of Reynolds-Numbers and were able to obtain many characteristic qualitative results such as von Karman vortex streets, the Magnus effect or the Bernoulli effect. Furthermore, since the entire recurrent inference pipeline is differentiable, gradients can be efficiently computed throughout the simulation using backpropagation through time. We demonstrated the effectiveness of this approach in a rudimentary flow control experiment that aimed to control the frequency of a von Karman vortex street behind an obstacle by adjusting the fluids velocity with gradient descent. However, this work also had limitations: first, only two dimensional domains were considered and second, fluids with different viscosities and densities required separate models that had to be retrained for every fluid modality. Furthermore, discretization artifacts start to appear at high Reynolds numbers.

Thus, in our second work, we addressed limitations of our previous work by representing the fluid state on a 3D marker and cell grid and by extending the neural PDE model from a 2D U-Net to a 3D U-Net. Simulating 3D fluid dynamics is considered to be a significantly more complex problem compared to 2D as it not only increases computational and memory demands but also increases the degrees of freedom of the vector potential that describes the velocity field by a factor of three. Furthermore, we conditioned the fluid model not only on the previous fluid state but also hand over the fluids viscosity and density as additional inputs. To train the model, we reapplied the training cycle as introduced in our first work but extended the 2D physics-constrained loss to 3 dimensions and enabled variable viscosity and density values. This way, after training, we were able to simulate fluid dynamics in 3D efficiently. We tested two models with different speed / accuracy trade-offs and were able to achieve 16 iterations per second on a $128 \times 64 \times 64$ grid with our large model and 36 iterations per second on the same domain with a smaller but slightly less accurate model. Furthermore, we were able to continuously interpolate between different fluid viscosities and densities and simulate various Reynolds-Numbers ranging from laminar to turbulent flows by the same neural surrogate model. Finally, we showed that, as in 2D, our models were able to generalize to new domain geometries that were not contained in the set of randomized training domains.

In our third work, we moved from a MAC grid based fluid representation to a continuous representation based on Hermite-spline coefficients on a grid. This allows to evaluate field values at arbitrary points within the domain by continuous interpolation and enables the use of a physics-informed loss instead of a physics-constrained loss inside the training cycle. Using this representation, we observed fewer discretization artifacts and more detailed flow fields at lower grid resolutions compared to our previous MAC grid based approaches. Furthermore, we computed drag and lift coefficients of a cylinder at different Reynolds numbers and compared our results to official benchmark results, a professional industrial CFD solver (AnSys) and results from other physics-based deep-learning approaches namely an implicit PINN approach and our marker and cell grid based approach. While the deep-learning based approaches generally did not achieve as high accuracy as the professional solver, our Spline-PINN approach made a big leap forward and achieved results that came significantly closer to the performance of AnSys. Regarding computational speed for unsteady flow simulations, our Spline-PINN method outperformed AnSys by two orders of magnitude and returned from simulation after 10 seconds while AnSys took about 37 minutes. Finally, we also tested this approach on the damped wave equation to demonstrate the general applicability of this method to a wide range of PDEs.

In conclusion, this work demonstrates the feasibility and effectiveness of unsupervised physics-based deep-learning to obtain simulations that are significantly faster than common traditional PDE solvers. Our framework allows to train neural surrogate models without any precomputed ground truth data and does not require additional components such as differentiable fluid-solvers or a particle tracer. After training, our models generalize to new domains that were not considered during training and yield stable simulations for thousands of timesteps. Furthermore, since the entire pipeline is differentiable, gradients can be efficiently computed by backpropagation through time. Finally, we demonstrated that this approach is not limited to fluid simulations but can be employed to further PDEs such as the damped wave equation as well.

7.2 Limitations and Future Work

The presented approach still exhibits a few limitations that might be subject of future research. So far, we mainly relied on U-Net architectures. However, since our training cycle is model agnostic, additional neural network architectures could be tested to further increase accuracy. For example, transformer architectures could tested as in Geneva and Zabaras, 2022 or steerable CNNs [Weiler and Cesa, 2019] could be used to exploit rotational symmetries that are prevalent in many PDEs on top of translational symmeties. Furthermore, the accuracy of the model predictions could be iteratively refined, similar to the stacked hourglass architecture that was proposed in context of human pose estimation [Newell et al., 2016].

To put more focus on important areas of the domain (for example boundary layers), a multigrid approach based on sparse convolutions [Graham and Maaten, 2017] could be implemented on top of our current implementations, which so far rely solely on uniform grids.

In this work, we focused on Dirichlet boundary conditions, but in the future, more boundary conditions such as Neumann boundaries, free surfaces or multi-phase flows could be considered as well.

On top of that, our approach could be tested for a wider variety of PDEs such as Darcy flows, the Burgers equation or the compressible Navier-Stokes equation to investigate further interesting effects such as for example shock wave formation.

7.3 Outlook

We believe that physics-based deep-learning approaches will have a profound impact on a wide range of applications that rely on solving PDEs. In computer generated imagery and computer games, these methods could enable fast, photo-realistic fluid and smoke visualizations and allow physics engines to simulate more plausible interactions with the environment in real-time. In engineering, these approaches could build the foundation for significantly faster CFD simulations in order to speed-up rapid prototyping and, thus, decrease time to market. Furthermore, since neural network based models are inherently differentiable, they could be applied in inverse problems, gradient-based shape optimization (e.g. to minimize drag while maximizing volume), in control algorithms or in reinforcement learning scenarios.

In the long run, in order to surpass the accuracy of traditional fluid solvers on complex domains, we believe that deep learning-based approaches must emancipate from ground-truth data because the accuracy as well as the computational complexity of generating a diverse dataset will become a limiting factor. Instead, physics-based loss functions can offer the required learning incentives, and a training cycle in conjunction with a constantly updated training pool will render any ground truth data unnecessary. This development could resemble the latest developments in reinforcement learning where algorithms that rely solely on self-play outperform all previous reinforcement learning techniques that (partially) relied on prior human knowledge by a significant margin [Silver et al., 2018].

At the current speed of progress in physics-driven deep-learning, we believe that in the foreseeable future similar accuracies to traditional methods can be achieved while maintaining advantages with respect to computational speed and efficiency. We hope that our work can make a strong contribution towards this goal by presenting a way to obtain fast, accurate, differentiable surrogate models based on physics-driven deep-learning that can be applied for a wide variety of PDEs and domain geometries without precomputed ground truth data.

Bibliography

Ainsworth, Mark and Justin Dong (2021). "Galerkin Neural Networks: A Framework for Approximating Variational Equations with Error Control." *arXiv preprint arXiv:2105.14094*.

Avila Belbute-Peres, Filipe de, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J. Zico Kolter (2018). "End-to-End Differentiable Physics for Learning and Control." *Advances in Neural Information Processing Systems 31*.

Bar-Sinai, Yohai, Stephan Hoyer, Jason Hickey, and Michael P Brenner (2019). "Learning data-driven discretizations for partial differential equations." *Proceedings of the National Academy of Sciences*.

Barrowclough, Oliver JD, Georg Muntingh, Varatharajan Nainamalai, and Ivar Stangeby (2021). "Binary segmentation of medical images using implicit spline representations and deep learning." *Computer Aided Geometric Design*.

Berg, Jens and Kaj Nyström (2018). "A unified deep artificial neural network approach to partial differential equations in complex geometries." *Neurocomputing*. ISSN: 0925-2312. DOI: `https://doi.org/10.1016/j.neucom.2018.06.056`. URL: `http://www.sciencedirect.com/science/article/pii/S092523121830794X`.

Brackbill, Jeremiah U, Douglas B Kothe, and Hans M Ruppel (1988). "FLIP: a low-dissipation, particle-in-cell method for fluid flow." *Computer Physics Communications*.

Bronstein, Michael M, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst (2017). "Geometric deep learning: going beyond euclidean data." *IEEE Signal Processing Magazine*.

Chen, Shiyi and Gary D Doolen (1998). "Lattice Boltzmann method for fluid flows." *Annual review of fluid mechanics*.

Cho, Minsu, Aditya Balu, Ameya Joshi, Anjana Deva Prasad, Biswajit Khara, Soumik Sarkar, Baskar Ganapathysubramanian, Adarsh Krishnamurthy, and Chinmay Hegde (2021). "Differentiable Spline Approximations." *Advances in Neural Information Processing Systems*. Edited by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. URL: `https://proceedings.neurips.cc/paper/2021/file/a952ddeda0b7e2c20744e52e728e5594-Paper.pdf`.

Çiçek, Özgün, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger (2016). "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation." *CoRR*. arXiv: `1606.06650`. URL: `http://arxiv.org/abs/1606.06650`.

Cundall, Peter A and Otto DL Strack (1979). "A discrete numerical model for granular assemblies." *geotechnique*.

Degrave, Jonas, Michiel Hermans, Joni Dambre, and Francis Wyffels (2019). "A differentiable physics engine for deep learning in robotics." *Frontiers in neurorobotics*.

Durkan, Conor, Artur Bekasov, Iain Murray, and George Papamakarios (2019). "Neural spline flows." *Advances in neural information processing systems*.

Eymard, Robert, Thierry Gallouët, and Raphaèle Herbin (2000). "Finite volume methods." *Handbook of numerical analysis*.

Fabregat, Alexandre, Ferran Gisbert, Anton Vernet, Josep Anton Ferré, Ketan Mittal, Som Dutta, and Jordi Pallarès (2021). "Direct numerical simulation of turbulent dispersion of evaporative aerosol clouds produced by an intense expiratory event." *Physics of Fluids*. DOI: `10.1063/5.0045416`. eprint: `https://doi.org/10.1063/5.0045416`. URL: `https://doi.org/10.1063/5.0045416`.

Fakhoury, Daniele, Emanuele Fakhoury, and Hendrik Speleers (2022). "ExSpliNet: An interpretable and expressive spline-based neural network." *Neural Networks*.

Farlow, Stanley J (1993). *Partial differential equations for scientists and engineers*.

Fey, Matthias, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller (2017). "SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels." *CoRR*. arXiv: `1711.08920`. URL: `http://arxiv.org/abs/1711.08920`.

Foster, Nick and Dimitri Metaxas (1996). "Realistic Animation of Liquids." *Graphical Models and Image Processing*. ISSN: 1077-3169. DOI: `https://doi.org/10.1006/gmip.1996.0039`. URL: `http://www.sciencedirect.com/science/article/pii/S1077316996900398`.

Gao, Han, Matthew J. Zahr, and Jian-Xun Wang (2021). "Physics-informed graph neural Galerkin networks: A unified framework for solving PDE-governed forward and inverse problems." *CoRR*. arXiv: `2107.12146`. URL: `https://arxiv.org/abs/2107.12146`.

Geneva, Nicholas and Nicholas Zabaras (2019). “Quantifying model form uncertainty in Reynolds-averaged turbulence models with Bayesian deep neural networks.” *Journal of Computational Physics*. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2019.01.021`. URL: `http://www.sciencedirect.com/science/article/pii/S0021999119300464`.

Geneva, Nicholas and Nicholas Zabaras (2020). “Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks.” *Journal of Computational Physics*.

Geneva, Nicholas and Nicholas Zabaras (2022). “Transformers for modeling physical systems.” *Neural Networks*.

Gingold, Robert A. and Joseph J. Monaghan (1977). “Smoothed particle hydrodynamics: theory and application to non-spherical stars.” *Monthly notices of the royal astronomical society*.

Graham, Benjamin and Laurens van der Maaten (2017). “Submanifold Sparse Convolutional Networks.” *arXiv preprint arXiv:1706.01307*.

Grohs, Philipp, Fabian Hornung, Arnulf Jentzen, and Philippe Von Wurstemberger (2018). “A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations.” *arXiv preprint arXiv:1809.02362*.

Guo, Zhaoli (2021). “Well-balanced lattice Boltzmann model for two-phase systems.” *Physics of Fluids*. DOI: `10.1063/5.0041446`. eprint: `https://doi.org/10.1063/5.0041446`. URL: `https://doi.org/10.1063/5.0041446`.

Harlow, Francis H (1962). *The particle-in-cell method for numerical solution of problems in fluid dynamics*. Technical report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Harlow, Francis H. and J. Eddie Welch (1965). “Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface.” *The physics of fluids*.

Harsch, Lukas and Stefan Riedelbauch (2021). *Direct Prediction of Steady-State Flow Fields in Meshed Domain with Graph Networks*. arXiv: `2105.02575 [physics.flu-dyn]`.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.” *Proceedings of the IEEE international conference on computer vision*.

Holl, Philipp, Vladlen Koltun, and Nils Thuerey (2020). “Learning to Control PDEs with Differentiable Physics.” *ICLR*.

Hsieh, Jun-Ting, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon (2019). "Learning neural PDE solvers with convergence guarantees." *arXiv preprint arXiv:1906.01200*.

Hu, Yuanming, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik (2019). "ChainQueen: A real-time differentiable physical simulator for soft robotics." *2019 International Conference on Robotics and Automation (ICRA)*.

Igelnik, Boris and Neel Parikh (2003). "Kolmogorov's spline network." *IEEE transactions on neural networks*.

Ingraham, John, Adam Riesselman, Chris Sander, and Debora Marks (2019). "Learning protein structure with a differentiable simulator." *International Conference on Learning Representations*.

Jin, Xiaowei, Shengze Cai, Hui Li, and George Em Karniadakis (2021). "NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations." *Journal of Computational Physics*. ISSN: 0021-9991. DOI: `10.1016/j.jcp.2020.109951`. URL: `http://dx.doi.org/10.1016/j.jcp.2020.109951`.

Karumuri, Sharmila, Rohit Tripathy, Ilias Bilionis, and Jitesh Panchal (2020). "Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks." *Journal of Computational Physics*.

Khoo, Yuehaw, Jianfeng Lu, and Lexing Ying (2019). "Solving for high-dimensional committor functions using artificial neural networks." *Research in the Mathematical Sciences*.

Kim, Byungsoo, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler (2019). "Deep fluids: A generative network for parameterized fluid simulations." *Computer Graphics Forum*.

Kim, Junhyuk and Changhoon Lee (2020). "Deep unsupervised learning of turbulence for inflow generation at various Reynolds numbers." *Journal of Computational Physics*. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2019.109216`. URL: `http://www.sciencedirect.com/science/article/pii/S0021999119309210`.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization." *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kochkov, Dmitrii, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer (2021). "Machine learning–accelerated computational fluid dynamics." *Proceedings of the National Academy of Sciences*.

Ladický, L'ubor, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross (2015). "Data-Driven Fluid Simulations Using Regression Forests." *ACM Trans. Graph.* ISSN: 0730-0301. DOI: `10.1145/2816795.2818129`. URL: `https://doi.org/10.1145/2816795.2818129`.

Lagaris, I. E., A. Likas, and D. I. Fotiadis (1998). "Artificial neural networks for solving ordinary and partial differential equations." *IEEE Transactions on Neural Networks*.

Lagaris, I. E., A. C. Likas, and D. G. Papageorgiou (2000). "Neural-network methods for boundary value problems with irregular boundaries." *IEEE Transactions on Neural Networks*.

Li, Yunzhu, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba (2019). "Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids." *ICLR*.

Li, Zhaobin, Xinlei Zhang, Ting Wu, Lixing Zhu, Jianhua Qin, and Xiaolei Yang (2021). "Effects of slope and speed of escalator on the dispersion of cough-generated droplets from a passenger." *Physics of Fluids*. DOI: `10.1063/5.0046870`. eprint: `https://doi.org/10.1063/5.0046870`. URL: `https://doi.org/10.1063/5.0046870`.

Liang, Junbang, Ming Lin, and Vladlen Koltun (2019). "Differentiable Cloth Simulation for Inverse Problems." *Advances in Neural Information Processing Systems*.

Ling, Julia, Andrew Kurzawski, and Jeremy Templeton (2016). "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance." *Journal of Fluid Mechanics*.

Liu, MB, GR Liu, and KY Lam (2006). "Adaptive smoothed particle hydrodynamics for high strain hydrodynamics with material strength." *Shock Waves*.

Long, Zichao, Yiping Lu, Xianzhong Ma, and Bin Dong (2018). "PDE-Net: Learning PDEs from data." *35th International Conference on Machine Learning, ICML 2018*.

Lu, Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis (2021). "DeepXDE: A deep learning library for solving differential equations." *SIAM Review*.

Lucy, Leon B (1977). "A numerical approach to the testing of the fission hypothesis." *The astronomical journal*.

Ma, Pingchuan, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha (2018). "Fluid directed rigid body control using deep reinforcement learning." *ACM Transactions on Graphics (TOG)*.

Macklin, Miles and Matthias Müller (2013). "Position based fluids." *ACM Transactions on Graphics (TOG)*.

Mao, Zhiping, Ameya D Jagtap, and George Em Karniadakis (2020). "Physics-informed neural networks for high-speed flows." *Computer Methods in Applied Mechanics and Engineering*.

Meade, A.J. and A.A. Fernandez (1994a). "Solution of nonlinear ordinary differential equations by feedforward neural networks." *Mathematical and Computer Modelling*. ISSN: 0895-7177. DOI: `https://doi.org/10.1016/0895-7177(94)00160-X`. URL: `http://www.sciencedirect.com/science/article/pii/089571779400160X`.

Meade, A.J. and A.A. Fernandez (1994b). "The numerical solution of linear ordinary differential equations by feedforward neural networks." *Mathematical and Computer Modelling*. ISSN: 0895-7177. DOI: `https://doi.org/10.1016/0895-7177(94)90095-7`. URL: `http://www.sciencedirect.com/science/article/pii/0895717794900957`.

Meng, Xuhui and George Em Karniadakis (2020). "A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems." *Journal of Computational Physics*.

Meng, Xuhui, Zhen Li, Dongkun Zhang, and George Em Karniadakis (2020). "PPINN: Parareal physics-informed neural network for time-dependent PDEs." *Computer Methods in Applied Mechanics and Engineering*.

Mohan, Arvind T., Nicholas Lubbers, Daniel Livescu, and Michael Chertkov (2020). *Embedding Hard Physical Constraints in Neural Network Coarse-Graining of 3D Turbulence*. arXiv: `2002.00021 [physics.comp-ph]`.

Morton, Jeremy, Antony Jameson, Mykel J Kochenderfer, and Freddie Witherden (2018). "Deep Dynamical Modeling and Control of Unsteady Fluid Flows." *Advances in Neural Information Processing Systems 31*. Edited by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. URL: `http://papers.nips.cc/paper/8138-deep-dynamical-modeling-and-control-of-unsteady-fluid-flows.pdf`.

Mrowca, Damian, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B. Tenenbaum, and Daniel L. K. Yamins (2018). "Flexible Neural Representation for Physics Prediction." *Proceedings of the 32nd International Conference on Neural Information Processing Systems*.

Müller, Matthias, Bruno Heidelberger, Marcus Hennix, and John Ratcliff (2007). "Position based dynamics." *Journal of Visual Communication and Image Representation*.

Newell, Alejandro, Kaiyu Yang, and Jia Deng (2016). "Stacked hourglass networks for human pose estimation." *European conference on computer vision*.

Noe, Rain (2016). *When Splines Were Physical Objects*. `https://www.core77.com/posts/55368/When-Splines-Were-Physical-Objects`. (accessed at 08-Jan-2023).

O'Reilly, H. and Jeffrey M. Beck (2006). "A Family of Large-Stencil Discrete Laplacian Approximations in Three Dimensions." *International Journal For Numerical Methods in Engineering*.

OpenCFD (2007). *OpenFOAM - The Open Source CFD Toolbox - User's Guide*. 1.4.

Pfaff, Tobias, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia (2021). *Learning Mesh-Based Simulation with Graph Networks*. ICLR 2021: `2010.03409` (cs.LG).

Psichogios, Dimitris C. and Lyle H. Ungar (1992). "A hybrid neural network-first principles approach to process modeling." *AIChE Journal*. DOI: `10.1002/aic.690381003`. eprint: `https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690381003`. URL: `https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690381003`.

Quarteroni, Alfio and Silvia Quarteroni (2009). *Numerical models for differential problems*.

Raissi, Maziar, P. Perdikaris, and George Em Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." *Journal of Computational Physics*. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2018.10.045`. URL: `http://www.sciencedirect.com/science/article/pii/S0021999118307125`.

Raissi, Maziar, Alireza Yazdani, and George Em Karniadakis (2018). "Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data." *arXiv preprint arXiv:1808.04327*.

Rasht-Behesht, Majid, Christian Huber, Khemraj Shukla, and George Em Karniadakis (2021). *Physics-informed Neural Networks (PINNs) for Wave Propagation and Full Waveform Inversions*. arXiv: `2108.12035 [physics.geo-ph]`.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation." *International Conference on Medical image computing and computer-assisted intervention*.

Sanchez-Gonzalez, Alvaro, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia (2020). "Learning to simulate complex physics with graph networks." *International Conference on Machine Learning*.

Schenck, Connor and Dieter Fox (2018). "SPNets: Differentiable Fluid Dynamics for Deep Neural Networks." *Conference on Robot Learning*.

Schöberl, Markus, Nicholas Zabaras, and Phaedon-Stelios Koutsourelakis (2019). "Predictive collective variable discovery with deep Bayesian models." *The Journal of Chemical Physics*. DOI: `10.1063/1.5058063`. eprint: `https://doi.org/10.1063/1.5058063`. URL: `https://doi.org/10.1063/1.5058063`.

Schultz, M. G., C. Betancourt, B. Gong, F. Kleinert, M. Langguth, L. H. Leufen, A. Mozaffari, and S. Stadtler (2021). "Can deep learning beat numerical weather prediction?" *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*. DOI: 10.1098/rsta.2020.0097. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2020.0097. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2020.0097.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search." *nature*.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science*.

Sirignano, Justin and Konstantinos Spiliopoulos (2018). "DGM: A deep learning algorithm for solving partial differential equations." *Journal of computational physics*.

Sitzmann, Vincent (2021). *List of Implicit Representations*. https://github.com/vsitzmann/awesome-implicit-representations. (accessed at 08-Jan-2023).

Sitzmann, Vincent, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein (2020). "Implicit neural representations with periodic activation functions." *Advances in Neural Information Processing Systems*.

Stam, Jos (1999). "Stable fluids." *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*.

Stomakhin, Alexey, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle (2013). "A material point method for snow simulation." *ACM Transactions on Graphics (TOG)*.

Sulsky, Deborah, Shi-Jian Zhou, and Howard L Schreyer (1995). "Application of a particle-in-cell method to solid mechanics." *Computer physics communications*.

Takahashi, Tetsuya, Junbang Liang, Yi-Ling Qiao, and Ming C Lin (2021). "Differentiable Fluids with Solid Coupling for Learning and Control."

The CFD Benchmarking Project (2021). http://www.mathematik.tu-dortmund.de/featflow/en/benchmarks/cfdbenchmarking.html. (accessed at 08-Sep-2021).

Thuerey, Nils, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu (2019). "Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows." *AIAA Journal*.

Tompson, Jonathan, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin (2017). "Accelerating eulerian fluid simulation with convolutional networks." *Proceedings of the 34th International Conference on Machine Learning-Volume 70*.

Toussaint, Marc, Kelsey R Allen, Kevin A Smith, and Joshua B Tenenbaum (2019). "Differentiable physics and stable modes for tool-use and manipulation planning." *Proceedings of the 28th International Joint Conference on Artificial Intelligence*.

Tripathy, Rohit K. and Ilias Bilionis (2018). "Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification." *Journal of Computational Physics*. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.08.036. URL: http://www.sciencedirect.com/science/article/pii/S0021999118305655.

Um, Kiwon, Raymond Fei, Philipp Holl, Robert Brand, and Nils Thuerey (2020). *Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers*. arXiv: 2007.00016 [physics.comp-ph].

Ummenhofer, Benjamin, Lukas Prantl, Nils Thuerey, and Vladlen Koltun (2020). "Lagrangian Fluid Simulation with Continuous Convolutions." *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. URL: https://openreview.net/forum?id=B1lDoJSYDH.

Vecci, Lorenzo, Francesco Piazza, and Aurelio Uncini (1998). "Learning and approximation capabilities of adaptive spline activation function neural networks." *Neural Networks*.

Wandel, Nils, Michael Weinmann, and Reinhard Klein (2021a). "Learning Incompressible Fluid Dynamics from Scratch - Towards Fast, Differentiable Fluid Models that Generalize." *9th International Conference on Learning Representations, ICLR*. URL: https://openreview.net/forum?id=KUDUoRsEphu.

Wandel, Nils, Michael Weinmann, and Reinhard Klein (2021b). "Teaching the Incompressible Navier-Stokes Equations to Fast Neural Surrogate Models in 3D." *Physics of Fluids (AIP), Volume 33, Issue 4*. DOI: 10.1063/5.0047428.

Wandel, Nils, Michael Weinmann, Michael Neidlin, and Reinhard Klein (2022). "Spline-PINN: Approaching PDEs without Data using Fast, Physics-Informed Hermite-Spline CNNs." *Proceedings of the 36th AAAI Conference on Artificial Intelligence*. DOI: 10.1609/aaai.v36i8.20830.

Wang, Hengjie, Robert Planas, Aparna Chandramowlishwaran, and Ramin Bostanabad (2021). "Train Once and Use Forever: Solving Boundary Value Problems in Unseen Domains with Pre-trained Deep Learning Models." *CoRR*. arXiv: 2104.10873. URL: https://arxiv.org/abs/2104.10873.

Wang, Rui, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu (2019). "Towards Physics-informed Deep Learning for Turbulent Flow Prediction." *arXiv preprint arXiv:1911.08655*.

Weiler, Maurice and Gabriele Cesa (2019). "General E(2)-Equivariant Steerable CNNs." *Conference on Neural Information Processing Systems (NeurIPS)*.

Wiewel, Steffen, Moritz Becher, and Nils Thuerey (2019). "Latent space physics: Towards learning the temporal evolution of fluid flow." *Computer graphics forum*.

Xie, You, Erik Franz, Mengyu Chu, and Nils Thuerey (2018). "TempoGAN: A Temporally Coherent, Volumetric GAN for Super-Resolution Fluid Flow." *ACM Trans. Graph.* ISSN: 0730-0301. DOI: 10.1145/3197517.3201304. URL: https://doi.org/10.1145/3197517.3201304.

Yang, Cheng, Xubo Yang, and Xiangyun Xiao (2016). "Data-driven projection method in fluid simulation." *Computer Animation and Virtual Worlds*. DOI: 10.1002/cav.1695. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cav.1695. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1695.

Yang, Liu, Xuhui Meng, and George Em Karniadakis (2021). "B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data." *Journal of Computational Physics*.

Yang, Liu, Dongkun Zhang, and George Em Karniadakis (2020). "Physics-informed generative adversarial networks for stochastic differential equations." *SIAM Journal on Scientific Computing*.

Zhang, Dongkun, Lu Lu, Ling Guo, and George Em Karniadakis (2019). "Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems." *Journal of Computational Physics*.

Zhu, Yinhao and Nicholas Zabaras (2018). "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification." *Journal of Computational Physics*. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.04.018. URL: http://www.sciencedirect.com/science/article/pii/S0021999118302341.

Zhu, Yinhao, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris (2019). "Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data." *Journal of Computational Physics*. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2019.05.024. URL: http://www.sciencedirect.com/science/article/pii/S0021999119303559.

List of Figures

List of Tables

Part IV

Appendix

Publication:
“Learning Incompressible Fluid Dynamics from Scratch - Towards Fast, Differentiable Fluid Models that Generalize”

Nils Wandel, Michael Weinmann, and Reinhard Klein

9th International Conference on Learning Representations, ICLR

2021

LEARNING INCOMPRESSIBLE FLUID DYNAMICS FROM SCRATCH - TOWARDS FAST, DIFFERENTIABLE FLUID MODELS THAT GENERALIZE

Nils Wandel
Department of Computer Science
University of Bonn
wandeln@cs.uni-bonn.de

Michael Weinmann
Department of Computer Science
University of Bonn
mw@cs.uni-bonn.de

Reinhard Klein
Department of Computer Science
University of Bonn
rk@cs.uni-bonn.de

ABSTRACT

Fast and stable fluid simulations are an essential prerequisite for applications ranging from computer-generated imagery to computer-aided design in research and development. However, solving the partial differential equations of incompressible fluids is a challenging task and traditional numerical approximation schemes come at high computational costs. Recent deep learning based approaches promise vast speed-ups but do not generalize to new fluid domains, require fluid simulation data for training, or rely on complex pipelines that outsource major parts of the fluid simulation to traditional methods.

In this work, we propose a novel physics-constrained training approach that generalizes to new fluid domains, requires no fluid simulation data, and allows convolutional neural networks to map a fluid state from time-point t to a subsequent state at time $t + dt$ in a single forward pass. This simplifies the pipeline to train and evaluate neural fluid models. After training, the framework yields models that are capable of fast fluid simulations and can handle various fluid phenomena including the Magnus effect and Kármán vortex streets. We present an interactive real-time demo to show the speed and generalization capabilities of our trained models. Moreover, the trained neural networks are efficient differentiable fluid solvers as they offer a differentiable update step to advance the fluid simulation in time. We exploit this fact in a proof-of-concept optimal control experiment. Our models significantly outperform a recent differentiable fluid solver in terms of computational speed and accuracy.

1 INTRODUCTION

Simulating the behavior of fluids by solving the incompressible Navier-Stokes equations is of great importance for a wide range of applications and accurate as well as fast fluid simulations are a long-standing research goal. On top of simulating the behavior of fluids, several applications such as sensitivity analysis of fluids or gradient-based control algorithms rely on differentiable fluid simulators that allow to propagate gradients throughout the simulation (Holl et al. (2020)).

Recent advances in deep learning aim for fast and accurate fluid simulations but rely on vast datasets and / or do not generalize to new fluid domains. Kim et al. (2019) present a framework to learn parameterized fluid simulations and allow to interpolate efficiently in between such simulations. However, their work does not generalize to new domain geometries that lay outside the training data. Kim & Lee (2020) train a RNN-GAN that produces turbulent flow fields within a pipe domain, but do not show generalization results beyond pipe domains. Xie et al. (2018) introduce a tempoGAN to perform temporally consistent superresolution of smoke simulations. This allows to produce plausible high-resolution smoke-density fields for arbitrary low-resolution inputs, but our fluid model should output a complete fluid state description consisting of a velocity and a pressure field. Tompson et al. (2017) present how a Helmholtz projection step can be learned to accelerate Eulerian fluid simulations. This method generalizes to new domain geometries, but a particle tracer is needed to deal with the advection term of the Navier-Stokes equations. Furthermore, as Eulerian fluids do not model viscosity, effects like e.g. the Magnus effect or Kármán vortex streets cannot be simulated. Geneva & Zabaras (2020) propose a physics-informed framework to learn the entire update step for the Burgers equations in 1D and 2D, but no generalization results for new domain geometries are demonstrated. All of the aforementioned methods rely on the availability of vast amounts of data from fluid-solvers such as FEniCS, OpenFOAM or Mantaflow. Most of these methods do not generalize well or outsource a major part of the fluid simulation to traditional methods such as low-resolution fluid solvers or a particle tracer.

In this work, we propose a novel unsupervised training framework to learn incompressible fluid dynamics from scratch. It does not require any simulated fluid-data (neither as ground truth data, nor to train an adversarial network, nor to initialize frames for a physics-constrained loss) and generalizes to fluid domains unseen during training. It allows CNNs to learn the entire update-step of mapping a fluid domain from time-point t to $t + dt$ without having to rely on low resolution fluid-solvers or a particle-tracer. In fact, we will demonstrate that a physics-constrained loss function combined with a simple strategy to recycle fluid-data generated by the neural network at training time suffices to teach CNNs fluid dynamics on increasingly realistic statistics of fluid states. This drastically simplifies the training pipeline. Fluid simulations get efficiently unrolled in time by recurrently applying the trained model on a fluid state. Furthermore, the fluid models include viscous friction and handle effects such as the Magnus effect and Kármán vortex streets. On top of that, we show by a gradient-based optimal control example how backpropagation through time can be used to differentiate the fluid simulation. Code and pretrained models are publicly available at `https://github.com/aschethor/Unsupervised_Deep_Learning_of_Incompressible_Fluid_Dynamics/`.

2 Related Work

In literature, several different approaches can be found that aim to approximate the dynamics of PDEs in general and fluids in particular with efficient, learning-based surrogate models.

Lagrangian methods such as smoothed particle hydrodynamcs (SPH) Gingold & Monaghan (1977) handle fluids from the perspective of many individual particles that move with the velocity field. Following this approach, learning-based methods using regression forests by Ladický et al. (2015), graph neural networks by Mrowca et al. (2018); Li et al. (2019) and continuous convolutions by Ummenhofer et al. (2020) have been developed. In addition, Smooth Particle Networks (SP-Nets) by Schenck & Fox (2018) allow for differentiable fluid simulations within the Lagrangian frame of reference. These Lagrangian methods are particularly suitable when a fluid domain exhibits large, dynamic surfaces (e.g. waves or droplets). However, to simulate the dynamics within a fluid domain accurately, *Eulerian methods*, that treat the Navier-Stokes equations in a fixed frame of reference, are usually better suited.

Continuous Eulerian methods allow for mesh-free solutions by mapping domain coordinates (e.g. x,y,t) directly onto field values (e.g. velocity $\vec{v}$ / pressure p) (Sirignano & Spiliopoulos (2018); Grohs et al. (2018); Khoo et al. (2019)). Recent applications focused on flow through porous media (Zhu & Zabaras (2018); Zhu et al. (2019); Tripathy & Bilionis (2018)), fluid modeling (Yang et al. (2016); Raissi et al. (2018)), turbulence modeling (Geneva & Zabaras (2019); Ling et al. (2016)) and modeling of molecular dynamics (Schöberl et al. (2019)). Training is usually based on physics-constrained loss functions that penalize residuals of the underlying PDEs. Similar to our approach, Raissi et al. (2019) uses vector potentials to obtain continuous divergence-free velocity fields to approximate the incompressible Navier-Stokes equations. Continuous methods return smooth, accurate results and can overcome the curse of dimensionality of discrete techniques in high-dimensional PDEs (Grohs et al. (2018)). However, these networks are trained on a specific domain and cannot generalize to new environments or be used in interactive scenarios.

Discrete Eulerian methods, on the other hand, aim to solve the underlying PDEs on a grid and early work dates back to Harlow & Welch (1965) and Stam (1999). Accelerating such traditional works with deep learning techniques is a major field of research and all of the methods mentioned in the introduction fall into this category. Further methods include the approach by Thuerey et al. (2019) to learn solutions of the Reynolds-averaged Navier-Stokes equations for airfoil flows, but requires large amounts of training data and does not generalize beyond airfoil flows. In the work by Um et al. (2020), a correction step is learned that brings solutions of a low-resolution differentiable fluid solver closer to solutions of a high-resolution fluid simulation. However, generalization results for new domain geometries were not presented. The works of Mohan et al. (2020) and Kim et al. (2019) show that vector potentials are suitable to enforce the incompressibility constraint in fluids but do not generalize to new fluid domains beyond their training data.

3 METHOD

In this section, we briefly review the incompressible Navier-Stokes equations, which are to be solved by the neural network. Then, we explain how the Helmholtz decomposition can be exploited to ensure incompressibility within the fluid domain. Furthermore, we provide details of our discrete spatio-temporal fluid representation and introduce the fluid model. Afterwards, we formulate a physics-constrained loss function based on residuals of the Navier-Stokes equations and introduce a pressure regularization term for very high Reynolds numbers. Finally, we explain the unsupervised training strategy.

3.1 INCOMPRESSIBLE NAVIER-STOKES EQUATIONS

Most fluids can be modeled with the incompressible Navier-Stokes equations - a set of non-linear equations that describe the interplay of a velocity field $\vec{v}$ and a pressure field p within a fluid domain Ω:

$$\nabla \cdot \vec{v} = 0 \qquad \text{incompressibility on } \Omega \tag{1}$$

$$\rho \dot{\vec{v}} = \rho \left(\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\, \vec{v} \right) = -\nabla p + \mu \Delta \vec{v} + \vec{f} \qquad \text{conservation of momentum on } \Omega \tag{2}$$

Here, ρ describes the fluid density and μ the viscosity. Equation 1 states that the fluid is incompressible and thus $\vec{v}$ is divergence-free. Equation 2 states that the change in momentum of fluid particles must correspond to the sum of forces that arise from the pressure gradient, viscous friction and external forces. Here, external forces on the fluid (such as e.g. gravity) can be neglected, so we set $\vec{f} = 0$.

These incompressible Navier-Stokes equations shall be solved by a CNN given initial conditions $\vec{v}^0$ and p^0 at the beginning of the simulation and Dirichlet boundary conditions which constrain the velocity field at the domain boundary $\partial\Omega$:

$$\vec{v} = \vec{v}_d \qquad \text{Dirichlet boundary condition on } \partial\Omega \tag{3}$$

3.2 HELMHOLTZ DECOMPOSITION

A common method to ensure incompressibility of a fluid (see Equation 1) is to project the flow field onto the divergence-free part of its Helmholtz decomposition. The Helmholtz theorem states that every vector field $\vec{v}$ can be decomposed into a curl-free part (∇q) and a divergence-free part ($\nabla \times \vec{a}$):

$$\vec{v} = \nabla q + \nabla \times \vec{a} \tag{4}$$

Note, that $\nabla \times (\nabla q) = \vec{0}$ and $\nabla \cdot (\nabla \times \vec{a}) = 0$. The Helmholtz projection consists of solving the Poisson problem $\nabla \cdot \vec{v} = \Delta q$ for q, followed by substracting ∇q from the original flow field. However, solving the Poisson equation on arbitrary domains comes at high computational costs for classical methods and one has to rely e.g. on conjugate gradient methods to approximate its solution.

Here, we propose a different approach and directly try to learn a vector potential $\vec{a}$ with $\vec{v} = \nabla \times \vec{a}$. This ensures that the network outputs a divergence-free velocity field within the domain Ω and automatically solves Equation 1. In this work, we consider 2D fluid simulations, so only the z-component of $\vec{a}$, a_z, is of interest since v_z and all derivatives with respect to the z-axis are zero:

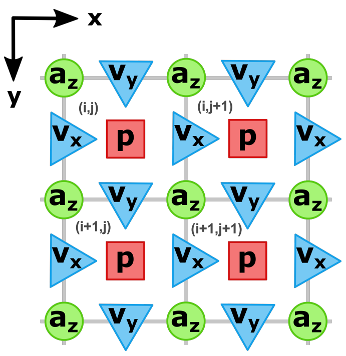
$$\nabla \times \vec{a} = \begin{pmatrix} \partial_y a_z - \partial_z a_y \\ \partial_z a_x - \partial_x a_z \\ \partial_x a_y - \partial_y a_x \end{pmatrix} = \begin{pmatrix} \partial_y a_z \\ -\partial_x a_z \\ 0 \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ 0 \end{pmatrix} = \vec{v} \tag{5}$$

3.3 Discrete Spatio-temporal Fluid Representation

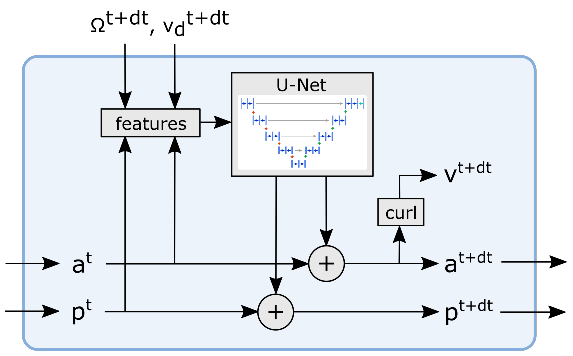
Marker-And-Cell (MAC) grid To solve the Navier-Stokes equations, we represent the relation between a_z, v_x, v_y, p on a 2D staggered marker-and-cell (MAC) grid (see Figure 1a). Therefore, we discretise time and space as follows:

$$\vec{a}(x,y,t) = \begin{pmatrix} 0 \\ 0 \\ (a_z)_{i,j}^t \end{pmatrix} ;\ \vec{v}(x,y,t) = \begin{pmatrix} (v_x)_{i,j}^t \\ (v_y)_{i,j}^t \end{pmatrix} ;\ p(x,y,t) = p_{i,j}^t \tag{6}$$

Obtaining gradient, divergence, Laplace and curl operations on this grid with finite differences is straight forward and can be efficiently implemented with convolutions (see appendix A).



(a) Layout of Staggered Marker-And-Cell (MAC) grid in 2D.



(b) Diagram of the fluid model. By recurrently applying the model on the fluid state (p^t and a^t), we can unroll the fluid simulation in time.

Figure 1: MAC grid and diagram of the fluid model.

Explicit, Implicit, Implicit-Explicit (IMEX) time integration methods The discretization of the time domain is needed to deal with the time-derivative of the velocity field $\frac{\partial \vec{v}}{\partial t}$ in Equation 2, which becomes:

$$\rho \left(\frac{\vec{v}^{t+dt} - \vec{v}^t}{dt} + \left(\vec{v}^{t'} \cdot \nabla \right) \vec{v}^{t'} \right) = -\nabla p^{t+dt} + \mu \Delta \vec{v}^{t'} + \vec{f} \tag{7}$$

The goal is to take as large as possible timesteps dt while maintaining stable and accurate solutions. Stability and accuracy largely depend on the definition of $v^{t'}$. In literature, choosing $v^{t'} = v^t$ is often referred to as explicit integration methods and frequently leads to unstable behavior. Choosing $v^{t'} = v^{t+dt}$ is usually associated with implicit integration methods and gives stable solutions at the cost of numerical dissipation. Implicit-Explicit (IMEX) methods, which set $v^{t'} = (v^t + v^{t+dt})/2$ are a compromise between both methods and considered to be more accurate but less stable than implicit methods.

3.4 Fluid Model

We represent the fluid dynamics by a recurrent model that maps the fluid state $p^t, \vec{a}^t$ for timestep t and the domain description $\Omega^{t+dt}, \vec{v}_d^{t+dt}$ to the fluid state $p^{t+dt}, \vec{a}^{t+dt}$ of the next timestep. Here, p^t describes the pressure field and $\vec{a}^t$ describes the vector potential of $\vec{v}^t$. For $t = 0$, we consider initial states $p^0 = 0$ and $\vec{a}^0 = \vec{0}$, however, other initial conditions could be considered as well. Ω^{t+dt} is a binary mask that contains the domain geometry and is 1 for the fluid domain and 0 everywhere else. For the boundary of the domain, we simply take the inverse of Ω: $\partial\Omega = 1 - \Omega$. $\vec{v}_d^{t+dt}$ represents the Dirichlet boundary conditions and contains a velocity field that must be matched by $\vec{v}^{t+dt}$ at the domain boundaries. Figure 1b shows a diagram of the fluid model. First, $\left(p^t, \vec{a}^t, \Omega^{t+dt}, \vec{v}_d^{t+dt}\right)$ are taken to derive a slightly more meaningful feature representation that comprises $\left(p^t, a^t, \nabla \times a^t, \Omega^{t+dt}, \partial\Omega^{t+dt}, \Omega^{t+dt} \cdot \nabla \times a^t, \Omega^{t+dt} \cdot p^t, \partial\Omega^{t+dt} \cdot \vec{v}_d^{t+dt}\right)$. These features can be very efficiently computed with convolutions and are then fed into a U-Net (Ronneberger et al. (2015)) with a reduced number of channels (the exact network configuration can be found in appendix B). The mean of the U-Net output is set to 0 in order to keep p and $\vec{a}$ well defined and prevent drifting offset values. Finally, the output is added to p^t and $\vec{a}^t$ to obtain the updated fluid state p^{t+dt} and $\vec{a}^{t+dt}$.

3.5 Physics-constrained Loss function

Using the residuals of the Navier-Stokes equations (Equations 1 and 2), we can formulate the following loss terms on Ω and $\partial\Omega$:

$$L_d = \|\nabla \cdot \vec{v}\|^2 \qquad \text{divergence loss on } \Omega \tag{8}$$

$$L_p = \left\|\rho\left(\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\, \vec{v}\right) + \nabla p - \mu \Delta \vec{v} - \vec{f}\right\|^2 \qquad \text{momentum loss on } \Omega \tag{9}$$

$$L_b = \|\vec{v} - \vec{v}_d\|^2 \qquad \text{boundary loss on } \partial\Omega \tag{10}$$

Combining the described loss terms, we obtain the following loss function:

$$L = \alpha L_d + \beta L_p + \gamma L_b \tag{11}$$

where α, β, γ are hyperparameters that weight the contributions of the different loss terms. Note that if we use a vector potential $\vec{v} = \nabla \times \vec{a}$, $L_d = 0$ is automatically fulfilled and we can set $\alpha = 0$. This loss function can be computed very efficiently with convolutions in $O(N)$ (where N = number of grid cells), whereas solving the Navier-Stokes equations explicitly would be computationally a lot more expensive. For detailed descriptions regarding the fully discretized loss-function, we refer to appendix A.

3.6 Pressure Regularization

For very high Reynolds numbers (see Equation 13) and inviscid flows, training becomes unstable as viscous friction cannot dissipate enough energy out of the system. This leads to unrealistic gradients in $\vec{v}$ and p. For such cases, we introduce an additional regularization term for the loss function (11) that can be traded off with L_p to stabilize training:

$$L_r = \|\nabla p\|^2 \tag{12}$$

The intuition behind this regularization term is, that we want to penalize unrealistically high energies in the pressure field.

3.7 TRAINING STRATEGY

Training starts with initializing a pool $\{\Omega_k^0, (v_d)_k^0, (a_z)_k^0, p_k^0\}$ of randomized domains Ω_k^0 and boundary conditions $(v_d)_k^0$ as well as initial conditions for the vector potential and pressure fields that we both set to zero ($(a_z)_k^0 = 0$ and $p_k^0 = 0$). The resolution of our training domains is 100x300 grid cells and example-domains of the training pool are shown in appendix C. Note that our training pool does not rely on any previously simulated fluid-data.

At each training step, a random mini-batch $\{\Omega_k^t, (v_d)_k^t, (a_z)_k^t, p_k^t\}_{\{k \in \text{minibatch}\}}$ is drawn from the pool and fed into the neural network which is designed to predict the velocity ($\vec{v}_k^{t+dt} = \nabla \times \vec{a}_k^{t+dt}$) and pressure ($p_k^{t+dt}$) fields of the next time step. Based on a physics-constrained loss-function (Equation 11), we update the weights of the network using the Adam optimizer (Kingma & Ba (2015)). At the end of each training step, the pool is updated by replacing the old vector potential and pressure fields $(a_z)_k^t, p_k^t$ by the newly predicted ones $(a_z)_k^{t+dt}, p_k^{t+dt}$. This recycling strategy fills the training pool with more and more realistic fluid states as the model becomes better at simulating fluid dynamics.

From time to time, old environments of the training pool are replaced by new randomized environments and the vector potential as well as the pressure fields are reset to 0. This increases the variance of the training pool and helps the neural network to learn "cold starts" from $\vec{0}$-velocity and 0-pressure fields.

Besides the fluid model described above, which we denote as $\vec{a}$-Net in the following, we also trained an ablation model, $\vec{v}$-Net, that directly learns to predict the velocity field without a vector potential. For the implementation of both models, we used the popular machine learning framework Pytorch and trained the models on a NVidia GeForce RTX 2080 Ti. Training converged after about 1 day. The hyperparameters in the loss-function for the $\vec{a}$-Net were $\beta = 1$ and $\gamma = 20$. The reason for choosing a higher weight for the loss term L_b than for L_p was the observation, that errors in L_b can lead to unrealistic flows leaking through boundaries. For the ablation study ($\vec{v}$-Net), we used $\alpha = 100, \beta = 1, \gamma = 0.001$. Here, we had to choose a very high weight for L_d to ensure incompressibility of the fluid, otherwise unrealistic source and sink effects start to appear. For L_b, on the other hand, we used a very low weight as the boundary conditions can be trivially learned by the $\vec{v}$-Net. We used these parameter settings for all experiments.

4 RESULTS

To evaluate the potential of our method, we assess its ability to reproduce physical effects such as Kármán vortex streets and the Magnus effect. In addition, we demonstrate its generalization capability and real-time performance. Finally, we test the fluid models quantitatively.

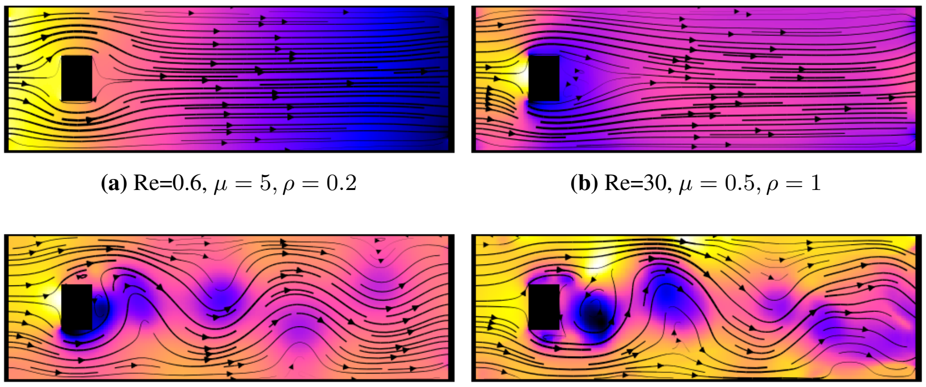
4.1 QUALITATIVE EVALUATION

Qualitative analysis of wake dynamics Qualitative effects in fluid dynamics such as the wake dynamics behind an obstacle are closely related to the Reynolds number. It is a dimensionless quantity defined by:

$$Re = \frac{\rho \, \|\vec{v}\| \, D}{\mu} \tag{13}$$

Here, ρ is the fluid density, $\|\vec{v}\|$ is the fluid speed, D is the diameter of the obstacle, and μ is the viscosity. (We use the units of the grid).

We retrained models for different values of μ and ρ to compare the fluid behavior for a wide range of Reynolds numbers. Figure 2 shows, that the trained models are able to predict the wake dynamics behind an obstacle in good accordance with qualitative expectations from fluid dynamics. As a rule of thumb, for $Re \ll 1$, the flow becomes time-reversible. This can be noticed in Figure 2a by the symmetry of the flow before and after the obstacle and the nearly constant pressure gradient within the pipe. Starting from $Re \approx 10$, the flow is still laminar but a static wake is forming behind the obstacle (see Figure 2b). For Reynolds numbers $Re >\approx 90$, Kármán vortex streets start to appear (see Figure 2c). A Kármán vortex street consists of clock and counterclockwise spinning vortices that are generated at the obstacle and then start moving in a regularly oscillating pattern with the flow. For very large Reynolds numbers or inviscid flows, the flow field becomes turbulent, which can be recognized by the irregular patterns behind the obstacle in Fig 2d.


(a) Re=0.6, $\mu = 5, \rho = 0.2$

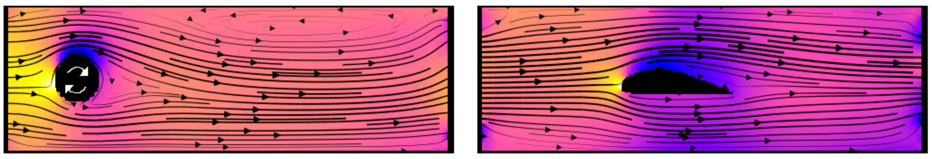
(b) Re=30, $\mu = 0.5, \rho = 1$

(c) Re=600, $\mu = 0.1, \rho = 4$

(d) Re$\rightarrow \infty$, $\mu = 0, \rho = 1$ (obtained with regularization on ∇p)

Figure 2: After training, our models are able to show correct wake flow dynamics for a wide range of different Reynolds numbers. ($D = 30, \|\vec{v}\| = 0.5$). Streamlines indicate flow direction, linewidth indicates speed and colors represent the pressure field (blue: low pressure / yellow: high pressure).

Magnus effect The Magnus effect appears when a flow interacts with a rotating body. It is widely known e.g. in sports such as soccer or tennis where spin is used to deflect the path of a ball. The reason for the deflection stems from a low pressure field where the surface of the object moves along flow direction and a high pressure field where the object surface moves against the flow. Figure 3a shows, that our models are able to reproduce the Magnus effect around a rotating cylinder.


(a) Magnus effect on a clock-wise turning cylinder.

(b) Generalization example: Note that the fluid model has never been confronted with wing-profiles during training.

Figure 3: Our models feature the Magnus effect and generalize to new fluid domains. Further examples are presented in appendix D and the video.

Analysis of generalization capability We tested the networks capability to generalize to objects not seen during training. Figure 3b shows the networks capability to meet boundary conditions of an airfoil and return a plausible pressure field that produces lift (see low pressure on top of wing). Note that in contrast to the approach by Thuerey et al. (2019), which learns simplified, time-averaged solutions of the Navier-Stokes equations, our method is able to simulate the full incompressible Navier-Stokes equations for an airfoil without relying on any ground truth data or having seen airfoil-geometries during training. In fact, the network was only trained on simple randomized domains as highlighted in appendix C and Figure 7. Possible reasons for the networks generalization capabilities are:

- During training, the network gets confronted with an infinite number of different flow-fields and randomized domain configurations because the training pool gets updated at every training step. This prevents the network from over-fitting.

- The dynamics of a fluid-particle are mostly determined by its local neighborhood / surrounding particles. This means, the update step for a certain cell on the MAC grid is mostly determined by close / neighboring MAC-grid cells. Since more complicated shapes can be seen locally as a composition of basic shapes (e.g. the front of the wing can be locally regarded as a cylinder), it suffices to train on basic shapes that provide the network with enough examples to generalize to more complicated shapes.

Further generalization examples are provided in appendix D.

Real-time capability The fluid simulation can be easily parallelized and takes low computational costs as one time-integration step consists just of a single forward pass through a convolutional neural network. This enables for example interactive real-time simulations. We implemented a demo that allows to interact with a fluid by moving obstacles, rotating spheres and changing the flow speed within a pipe (see video in supplementary material and source code). Our method runs at 250 timesteps per second on a 100x300 grid. In the respective experiments, we used a NVidia GeForce RTX 2080 Ti consuming about 860 MB of GPU memory.

4.2 Quantitative Evaluation

We compare our method ($\vec{a}$-Net) quantitatively with PhiFlow by Holl et al. (2020). Phiflow is a recent, open source, differentiable fluid simulator based on a MAC grid data structure. Furthermore, we provide an ablation study ($\vec{v}$-Net) that does not make use of the Helmholtz decomposition but directly works on the velocity field $\vec{v}$.

Quantitative comparison of different fluid solvers is challenging, as their performance is highly dependent on factors like the geometry of the domain, fluid parameters such as viscosity or density, flow speed or the timestep of the integrator. As benchmarks for fluid simulations on MAC grids are not yet available, we built a simple toy domain on a 100 x 100 grid which simulates a flow around an obstacle within a pipe (more details are provided in appendix E).

First, we compared the computational speed on a CPU and GPU by comparing the integration time-steps per second (see Table 1). The $\vec{v}$-Net as well as the $\vec{a}$-Net are significantly faster than PhiFlow (11x on CPU and 40x on GPU) as they do not rely on an iterative conjugate gradient solver but instead use a single forward pass through a convolutional neural network that can be easily parallelized on a GPU. To provide a fair comparison on L_d, we set the velocity field at the boundaries equal to $\vec{v}_d$. This enables us to compute L_d for the $\vec{a}$-Net architecture on the domain boundaries which would otherwise have zero divergence everywhere. This way, L_d can be interpreted as a metric on how well the orthogonal components of the Dirichlet boundary conditions are met (i.e. no flow leaks through the boundaries). For $dt = 4$, we outperformed Phiflow by several orders of magnitude. For both, L_d and L_p, the $\vec{a}$-Net architecture significantly outperformed the more naive $\vec{v}$-Net approach.

Furthermore, we investigated stability by evaluating the evolution of L_p and L_d for the $\vec{a}$-Net over time (see Figure 4). As the fluid state is initialized with $a_z = 0$ and $p = 0$, the $\vec{a}$-Net has to perform a cold-start which is the reason for high L_p and L_d during the first circa 70 steps. Afterwards, the $\vec{a}$-Net continues an accurate and stable fluid simulation.

Method	CPU [TPS]	GPU [TPS]	L_d	L_p
PhiFlow	7	-	6.2e-4	-
$\vec{v}$-Net (ours)	**82**	**311**	8.66e-7	4.87e-5
$\vec{a}$-Net (ours)	**82**	**311**	**5.44e-7**	**1.56e-5**

Table 1: Quantitative comparison of timesteps per second (TPS) on CPU / GPU as well as divergence loss and momentum loss for differentiable fluid solvers on a 100x100 grid for viscosity $\mu = 0.1$, density $\rho = 4$ and timesteps of size $dt = 4$.

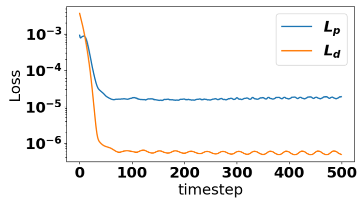
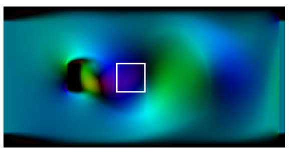
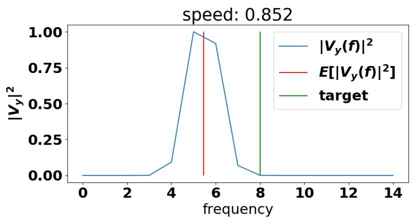


Figure 4: Long term stability of fluid simulations performed by the $\vec{a}$-Net

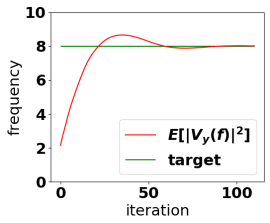
4.3 Optimal Control of Vortex Shedding Frequency

In this section, we present a proof-of-concept experiment that aims at controlling the shedding frequency of a Kármán vortex street behind an obstacle by changing the flow speed (see Figure 5a). To this end, we exploit our previously trained differentiable fluid models.


(a) control setup (domain size: 200x100 grid cells)



(b) frequency distribution before reaching convergence



(c) optimization curve

Figure 5: The frequency of vortex streets can be controlled using our differentiable fluid models.

First, we measure the y-component of the velocity field $v_y(t)$ behind an obstacle (see white box in Figure 5a) over 200 time steps. Then, we compute the frequency spectrum $V_y(f)$ of $v_y(t)$ using the fast Fourier transform (see Figure 5b). Now, we want to adjust the inflow / outflow boundary conditions in $\vec{v}_d$ such that $E[|V_y(f)|^2] = \hat{f}$. Here, $\hat{f}$ is the target frequency. To optimize $\vec{v}_d$, we define a loss function $L = (E[|V_y(f)|^2] - \hat{f})^2$ and compute the gradients $\frac{\partial L}{\partial \vec{v}_d}$ with backpropagation through time. This is possible since all parts of the loss function including the fluid simulation that is performed by our trained neural fluid model as well as the fast Fourier transform are differentiable. Computing the gradients with a standard automatic differentiation library (Pytorch) took 3.5 seconds for all 200 time steps on our 200x100 domain setup. This is considerably faster than the current state-of-the-art differentiable fluid solver by Takahashi et al. (2021) which takes 5.42 seconds for only 30 time steps on a smaller 128x128 grid. The update steps of $\vec{v}_d$ are done using the ADAM-optimizer and converge after approximately 70 iterations (see Figure 5c). We want to emphasize that differentiable fluid simulations are limited to scenarios with low Reynolds numbers as in the presence of turbulences, chaotic behavior will lead to exploding gradients.

5 Discussion and Outlook

In this work, we present an unsupervised learning scheme for the incompressible Navier-Stokes equations and introduce a fluid model that uses a vector potential to output divergence-free velocity fields. Qualitative results of our trained fluid models are in good accordance with expectations from fluid dynamics for a wide range of Reynolds numbers and generalize to unknown fluid domains. Quantitative assessment showed superior performance in terms of accuracy and speed compared to Phiflow and an ablation study that directly predicts the velocity field. We present a real-time demo and demonstrate how differentiability can be used in a proof-of-concept fluid control scenario. We believe that our fluid models can significantly speed up more sophisticated fluid control pipelines such as described by Holl et al. (2020).

First experiments of extending this approach to 3D deliver encouraging results and are topic of future research. Furthermore, on top of Dirichlet boundary conditions, Neumann boundary conditions and multi-phase domains could be incorporated in future fluid models as well.

REFERENCES

Nicholas Geneva and Nicholas Zabaras. Quantifying model form uncertainty in reynolds-averaged turbulence models with bayesian deep neural networks. *Journal of Computational Physics*, 383: 125 – 147, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.01.021. URL `http://www.sciencedirect.com/science/article/pii/S0021999119300464`.

Nicholas Geneva and Nicholas Zabaras. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*, 403:109056, 2020.

Robert A. Gingold and Joseph J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3): 375–389, 1977.

Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philippe Von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *arXiv preprint arXiv:1809.02362*, 2018.

Francis H. Harlow and J. Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12):2182–2189, 1965.

Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *ICLR*, 2020.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving for high-dimensional committor functions using artificial neural networks. *Research in the Mathematical Sciences*, 6(1):1, 2019.

Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, pp. 59–70. Wiley Online Library, 2019.

Junhyuk Kim and Changhoon Lee. Deep unsupervised learning of turbulence for inflow generation at various reynolds numbers. *Journal of Computational Physics*, 406:109216, 2020. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.109216. URL `http://www.sciencedirect.com/science/article/pii/S0021999119309210`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven fluid simulations using regression forests. *ACM Trans. Graph.*, 34(6), October 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818129. URL `https://doi.org/10.1145/2816795.2818129`.

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.

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

Arvind T. Mohan, Nicholas Lubbers, Daniel Livescu, and Michael Chertkov. Embedding hard physical constraints in neural network coarse-graining of 3d turbulence, 2020.

Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B. Tenenbaum, and Daniel L. K. Yamins. Flexible neural representation for physics prediction. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 8813–8824, Red Hook, NY, USA, 2018. Curran Associates Inc.

Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data. *arXiv preprint arXiv:1808.04327*, 2018.

Maziar Raissi, P. Perdikaris, and George Em Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686 – 707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL `http://www.sciencedirect.com/science/article/pii/S0021999118307125`.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Conference on Robot Learning*, pp. 317–335, 2018.

Markus Schöberl, Nicholas Zabaras, and Phaedon-Stelios Koutsourelakis. Predictive collective variable discovery with deep bayesian models. *The Journal of Chemical Physics*, 150(2):024109, 2019. doi: 10.1063/1.5058063. URL `https://doi.org/10.1063/1.5058063`.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339 – 1364, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.08.029. URL `http://www.sciencedirect.com/science/article/pii/S0021999118305527`.

Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 121–128, 1999.

Tetsuya Takahashi, Junbang Liang, Yi-Ling Qiao, and Ming C Lin. Differentiable fluids with solid coupling for learning and control. 2021.

Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, pp. 1–12, 2019.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3424–3433. JMLR. org, 2017.

Rohit K. Tripathy and Ilias Bilionis. Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics*, 375:565 – 588, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.08.036. URL `http://www.sciencedirect.com/science/article/pii/S0021999118305655`.

Kiwon Um, Raymond Fei, Philipp Holl, Robert Brand, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers, 2020.

Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=B1lDoJSYDH`.

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. Tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Trans. Graph.*, 37(4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201304. URL `https://doi.org/10.1145/3197517.3201304`.

Cheng Yang, Xubo Yang, and Xiangyun Xiao. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424, 2016. doi: 10.1002/cav.1695. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1695`.

Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415 – 447, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.04.018. URL `http://www.sciencedirect.com/science/article/pii/S0021999118302341`.

Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56 – 81, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.05.024. URL `http://www.sciencedirect.com/science/article/pii/S0021999119303559`.

A Physics-Constrained Loss on a MAC Grid

As mentioned in Section 3.3 of the paper, our method relies on a staggered marker-and-cell grid representation for the vector potential as well as the velocity and pressure fields. In the following, we provide further details on how to apply this representation to learn incompressible fluid dynamics.

To calculate the velocity field $\vec{v} = \nabla \times \vec{a}$ of a vector potential $\vec{a}$ on a MAC grid in 2D, we have to compute the curl as follows:

$$\begin{aligned} (v_x)_{i,j} &= (a_z)_{i+1,j} - (a_z)_{i,j} \\ (v_y)_{i,j} &= (a_z)_{i,j} - (a_z)_{i,j+1} \end{aligned} \tag{14}$$

If this vector potential is inserted into the divergence operator on a MAC grid, we can show that $\nabla \cdot \vec{v}_{i,j} = 0$ is indeed fulfilled:

$$\nabla \cdot \vec{v}_{i,j} = (v_x)_{i,j+1} - (v_x)_{i,j} + (v_y)_{i+1,j} - (v_y)_{i,j} \tag{15}$$

$$= \begin{matrix} ((a_z)_{i+1,j+1} - (a_z)_{i,j+1}) - ((a_z)_{i+1,j} - (a_z)_{i,j}) \\ + ((a_z)_{i+1,j} - (a_z)_{i+1,j+1}) - ((a_z)_{i,j} - (a_z)_{i,j+1}) \end{matrix} \tag{16}$$

$$= 0 \tag{17}$$

Thus, for the $\vec{a}$-Net, the incompressibility equation is automatically fulfilled and no further training on the divergence loss L_d is required. However, for the $\vec{v}$-Net, the residuals of the divergence are still of importance:

$$(R_d)^{t+dt}_{i,j} = \nabla \cdot \vec{v}^{t+dt}_{i,j} (= 0 \text{ for } \vec{a}\text{-Net}) \tag{18}$$

The residuals of the momentum equation in x-direction can be computed as follows:

$$\begin{aligned} (R_{p_x})^{t+dt}_{i,j} = &\rho \Bigg(\frac{(v_x)^{t+dt}_{i,j} - (v_x)^{t}_{i,j}}{dt} + (v_x)^{t'}_{i,j} \cdot \frac{(v_x)^{t'}_{i,j+1} - (v_x)^{t'}_{i,j-1}}{2} \\ &+ \frac{\frac{(v_y)^{t'}_{i,j-1} + (v_y)^{t'}_{i,j}}{2} \cdot \left((v_x)^{t'}_{i,j} - (v_x)^{t'}_{i-1,j} \right) + \frac{(v_y)^{t'}_{i+1,j-1} + (v_y)^{t'}_{i+1,j}}{2} \cdot \left((v_x)^{t'}_{i+1,j} - (v_x)^{t'}_{i,j} \right)}{2} \Bigg) \\ &+ \left(p^{t+dt}_{i,j} - p^{t+dt}_{i,j-1} \right) - \mu \cdot \Delta (v_x)^{t'}_{i,j} \end{aligned} \tag{19}$$

Here, we use the following isotropic Laplace operator:

$$\begin{aligned} \Delta s_{i,j} = \frac{1}{4}(&1 * s_{i-1,j-1} + 2 * s_{i-1,j} + 1 * s_{i-1,j+1} \\ &+2 * s_{i,j-1} - 12 * s_{i,j} + 2 * s_{i,j+1} \\ &+1 * s_{i+1,j-1} + 2 * s_{i+1,j} + 1 * s_{i+1,j+1}) \end{aligned} \tag{20}$$

The derivation of the advection term for R_{p_x} is a bit more complex since on a MAC grid, v_x and v_y are displaced by half a pixel in x-direction and y-direction. To obtain the residuals of the momentum equation in y-direction, $(R_{p_y})_{i,j}$, one has to take $(R_{p_x})_{i,j}$ and swap x and y and the indices respectively.

Now, the discretized loss terms can be written as follows:

$$L^{t+dt}_d = \sum_{i,j} \Omega^{t+dt}_{i,j} ((R_d)^{t+dt}_{i,j})^2 \tag{21}$$

$$L^{t+dt}_p = \sum_{i,j} \Omega^{t+dt}_{i,j} \left(((R_{p_x})^{t+dt}_{i,j})^2 + ((R_{p_y})^{t+dt}_{i,j})^2 \right) \tag{22}$$

$$L^{t+dt}_b = \sum_{i,j} \partial\Omega^{t+dt}_{i,j} \left\| \vec{v}^{t+dt}_d - \vec{v}^{t+dt} \right\|^2 \tag{23}$$

Note, that all mentioned operations can be efficiently implemented with convolutions. To obtain the final velocities on a square grid, we project the velocity fields of the MAC grid back onto the $\vec{a}$-grid using linear interpolation:

$$\vec{v} = \frac{1}{2} \begin{pmatrix} (v_x)_{i-1,j} + (v_x)_{i,j} \\ (v_y)_{i,j-1} + (v_y)_{i,j} \end{pmatrix} \tag{24}$$

B Network Architecture

Our fluid model is based on the U-Net architecture (Ronneberger et al. (2015)) with fewer channels (see Figure 6). As the pressure field and vector potential can have an arbitrary offset, we always normalize the mean of the pressure (Δp) and vector potential (Δa_z) to 0 to keep these fields well-defined and prevent drifting offset values.

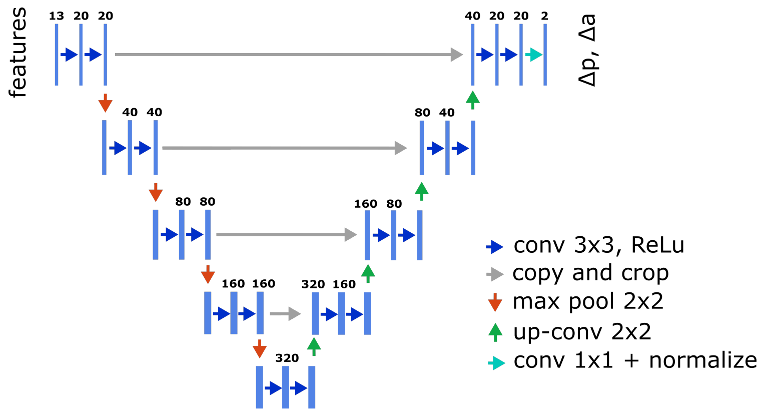


Figure 6: U-Net architecture with fewer channels.

C Examples of Training Domains

The domains we used for training consist of 100×300 grids. We used 3 different randomized domains as exemplary depicted in Figure 7. First, we have boxes with randomized height and width that float on randomized paths inspired by Brownian motion in a pipe with randomized flow speed. Second, we have the same setup but replaced the boxes by cylinders with randomized radii and angular velocities in order to learn the Magnus effect. Finally, we have a folded pipe system with randomized flow speed, that is randomly flipped along the x-axis.

D Further Examples of Generalization

Note that the network was only trained on simple domain geometries as presented in appendix C. Still, as can be seen in Figure 8, the network is capable of generalizing to far more complicated domain geometries (e.g. shark, car). Figure 8c shows that it can generalize to multiple objects in the scene, although the training set contained at most one object per scene. And Figure 8d shows that we can alter the outer boundary conditions as well. For real-time simulations, please have a look at our source code and the supplementary video.

E Quantitative Analysis: The Benchmark Problem

Figure 9 shows the domain Ω and v_d on a 100×100 grid which was used as the benchmark problem for quantitative analysis. The flow speed for the inlet and outlet was set to 0.5. The timestep of the integrator was set to $dt = 4$ and the viscosity and fluid density were set to $\mu = 0.1$ and $\rho = 4$ respectively.

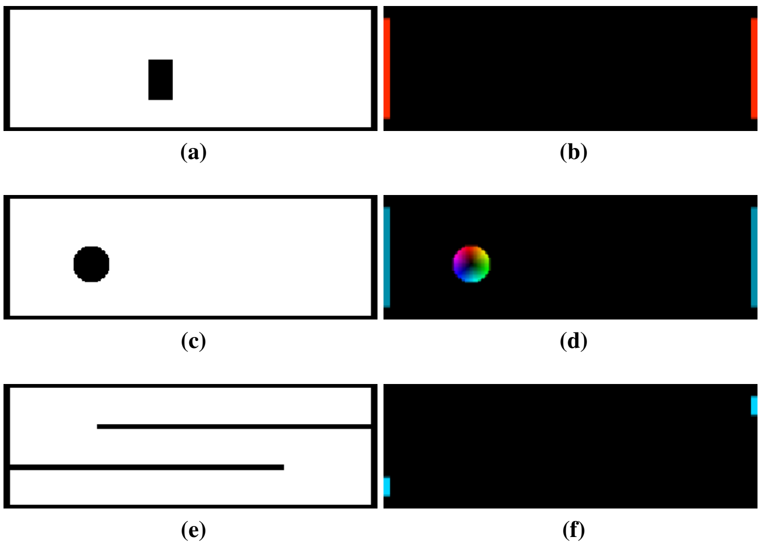


Figure 7: The left column shows Ω (in white) / $\partial\Omega$ (in black) and the right column shows $\vec{v}_d$ for three examples of training domains. (Colors indicate the direction and magnitude of $\vec{v}_d$ as depicted in Figure 9a)

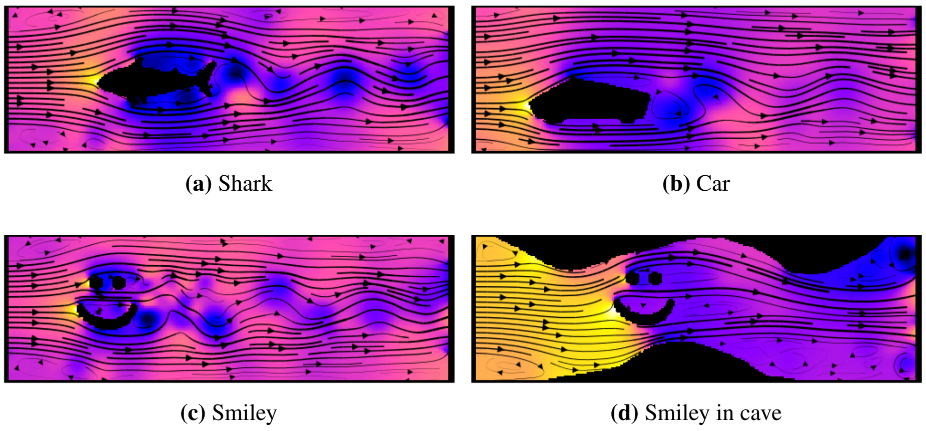


Figure 8: Our models generalize to various domain geometries, although being trained only on simple shapes (see Figure 7)

F Qualitative Comparison of $\vec{a}$-Net and $\vec{v}$-Net

We give a qualitative example to show the benefits of using a vector potential. Figure 10 demonstrates that the $\vec{a}$-Net finds plausible solutions for a folded pipe domain while the $\vec{v}$-Net looses most of the flow in the center of the domain. This is in good accordance with quantitative results shown in section 1. The folded pipe domain is particularly difficult to learn as the flow field contains long range dependencies to the inlet and outlet (as shown in the bottom row in Figure 7).

G Training without resetting environments

We performed an ablation study to investigate what happens if we do not reset old environments from time to time and, thus, do not continuously present the fluid model with cold starts during training. Figure 11 shows that in this case, large error spikes appear in the validation curve. These error spikes appear since the model has troubles to perform a cold start as can be seen in Figure 11b: compared

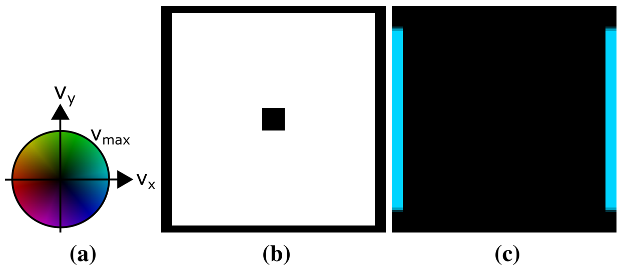
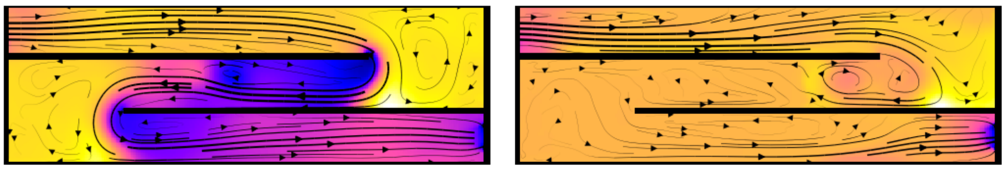


Figure 9: a) shows legend for $\vec{v}_d$; b) shows Ω (in white) / $\partial\Omega$ (in black) for the benchmark problem; c) shows $\vec{v}_d$ for the benchmark problem. (Colors indicate the direction of $\vec{v}_d$ as depicted in a)



(a) $\vec{a}$-Net **(b)** $\vec{v}$-Net

Figure 10: Qualitative comparison of $\vec{a}$-Net and $\vec{v}$-Net in a folded pipe domain

to a properly trained model (see Figure 4) the model takes longer to perform a cold start (ca 100 steps) and converges to a solution with high L_p- and L_d- losses. By resetting the environments from time to time during training, we can prevent these error spikes as shown in Figure 11c.

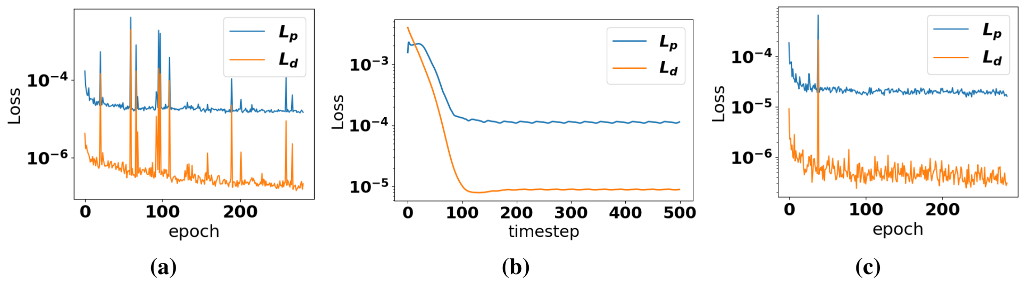


Figure 11: a) ablation study without resetting environments: validation curve shows large error spikes during training; b) error spike: the fluid model takes longer to perform a cold start and converges to a solution with high losses; c) original training with resetting environments: validation curve is stable

Publication: “Teaching the Incompressible Navier-Stokes Equations to Fast Neural Surrogate Models in 3D”

Nils Wandel, Michael Weinmann, and Reinhard Klein

Physics of Fluids

2021

Teaching the incompressible Navier–Stokes equations to fast neural surrogate models in three dimensions

Cite as: Phys. Fluids **33**, 047117 (2021); https://doi.org/10.1063/5.0047428
Submitted: 13 February 2021 • Accepted: 06 April 2021 • Published Online: 30 April 2021

Nils Wandel, Michael Weinmann and Reinhard Klein

Teaching the incompressible Navier–Stokes equations to fast neural surrogate models in three dimensions

Cite as: Phys. Fluids **33**, 047117 (2021); doi: 10.1063/5.0047428
Submitted: 13 February 2021 · Accepted: 6 April 2021 ·
Published Online: 30 April 2021

Nils Wandel,[a] Michael Weinmann,[b] and Reinhard Klein[c]

AFFILIATIONS

Institute of Computer Science, University of Bonn, Friedrich-Hirzebruch-Allee 8, 53115 Bonn, Germany

[a]Author to whom correspondence should be addressed: wandeln@cs.uni-bonn.de
[b]Electronic mail: mw@cs.uni-bonn.de
[c]Electronic mail: rk@cs.uni-bonn.de

ABSTRACT

Physically plausible fluid simulations play an important role in modern computer graphics and engineering. However, in order to achieve real-time performance, computational speed needs to be traded-off with physical accuracy. Surrogate fluid models based on neural networks (NN) have the potential to achieve both fast fluid simulations and high physical accuracy. However, these approaches rely on massive amounts of training data, require complex pipelines for training and inference, or do not generalize to new fluid domains. In this work, we present significant extensions to a recently proposed deep learning framework, which addresses the aforementioned challenges in two dimensions (2D). We go from 2D to three dimensions (3D) and propose an efficient architecture to cope with the high demands of 3D grids in terms of memory and computational complexity. Furthermore, we condition the neural fluid model on additional information about the fluid's viscosity and density, which allows for simulating laminar as well as turbulent flows based on the same surrogate model. Our method allows us to train fluid models without requiring fluid simulation data beforehand. Inference is fast and simple, as the fluid model directly maps a fluid state and boundary conditions at a moment t to a subsequent fluid state at $t + dt$. We obtain real-time fluid simulations on a $128 \times 64 \times 64$ grid that include various fluid phenomena such as the Magnus effect or Kármán vortex streets and generalize to domain geometries not considered during training. Our method indicates strong improvements in terms of accuracy, speed, and generalization capabilities over current 3D NN-based fluid models.

I. INTRODUCTION

Computational fluid dynamics play an important role in applications ranging from computer games to computer generated imagery to applications in industry to basic research in fields such as weather forecasting[1] or simulations of aerosol clouds produced by expiratory events.[2,3] For all of these applications, certain tradeoffs with respect to speed, accuracy, and stability have to be made.

While physical correctness is of utmost importance for engineering applications like aerodynamic design, applications in computer graphics such as simulations for games or movies particularly focus on computational efficiency under the constraint of producing visually plausible results. Often, physical accuracy is sacrificed by means of introducing pseudo-forces in order to get more appealing curls or by a physically inaccurate notion of viscosity to mitigate numerical dissipation.

As solving the equations of fluid dynamics based on numerical approximation schemes comes at high computational costs that cannot be handled in real time, recent developments particularly focused on exploiting the potential of deep learning in the context of surrogate fluid solvers to significantly reduce the computational burden while maintaining high physical accuracy.[4–8] Grid-based, physics-informed learning strategies have been shown to allow convolutional neural networks (CNNs) to advance a fluid state in time efficiently and to generalize well to new domain geometries that are not contained in the training set.[4,5] In contrast to the approach by Tompson *et al.*,[4] the approach by Wandel *et al.*[5] can handle viscous fluids and dynamic boundary conditions while not relying on the availability of training data from fluid simulations. Nevertheless, this method is limited to simulations in two dimensions (2D) domains and, hence, not suitable for the description of general three dimensions (3D) fluid behavior since symmetry along the third dimension cannot be expected in general. Using 3D grids, in turn, comes at the cost of significantly increasing computational complexity and memory requirements. Additional challenges are given by the increasing number of degrees of freedom for 3D fluid motion as well as the more complex boundary conditions in 3D domains.

In this paper, we address these challenges by a novel unsupervised approach of learning incompressible fluid dynamics, i.e., how a fluid state at time step t changes to a subsequent state at time step $t + dt$, in 3D. For this purpose, we represent the fluid on a 3D staggered marker-and-cell grid and formulate a physics-informed loss function by penalizing residuals of the Navier–Stokes equations on this representation. We test our method on a 3D U-Net[9] architecture and, to meet the demands for real-time performance, we also propose an efficient pruned 3D U-Net-based architecture. This allows for fluid simulations on a $128 \times 64 \times 64$ grid at 36 time steps per second while taking into account various fluid phenomena such as the Magnus effect and Kármán vortex streets. Furthermore, our framework allows us to generalize to 3D domain geometries not considered during training and does not rely on previously generated fluid data, hence significantly increasing its practical relevance as there is no need to consider large amounts of data from fluid solvers such as FEniCS, OpenFOAM[10] or Mantaflow for the training of the fluid model. As demonstrated by our experiments, our method indicates strong improvements in terms of accuracy, speed, and generalization capabilities compared to existing deep learning-based approaches. The code and data to reproduce our results are made available at: https://github.com/aschethor/Teaching_Incompressible_Fluid_Dynamics_to_3D_CNNs.

II. RELATED WORK

In recent years, the rapid progress in deep learning inspired several approaches to approximate the dynamics of partial differential equations (PDEs) with efficient, learning-based surrogate models.

Lagrangian methods such as smoothed particle hydrodynamics (SPH)[11] model fluids based on a large number of individual particles that move with the fluid's velocity field, i.e., each particle has different properties like mass or velocity. As a result, the conservation of mass can be easily preserved. Respective Lagrangian learning-based approaches for fluid simulation have been proposed based on regression forests,[12] graph neural networks[13,14] and continuous convolutions.[15] Furthermore, differentiable fluid simulations have been achieved based on Smooth Particle Networks (SP-Nets).[16] Whereas Lagrangian methods are particularly suitable for fluid domains with large, dynamic surfaces such as waves or droplets, the accurate simulation of fluid dynamics within a fluid domain usually can be better achieved with *Eulerian methods*.

Eulerian methods model fluid properties such as the fluid's velocity or pressure field on a fixed frame of reference. This includes methods that describe the fluid state using implicit neural representations, finite elements, or grid structures.

Continuous Eulerian methods map domain coordinates (e.g., x, y, t) directly onto field values (e.g., velocity $\vec{v}$/pressure p) using, e.g., implicit neural representations, thereby allowing for mesh-free solutions.[17–19] Respective applications include the modeling of flow through porous media,[20–22] fluid modeling,[23,24] turbulence modeling[25,26] and modeling of molecular dynamics.[27] Such learning-based approaches typically involve a training process that includes a physics-informed loss function to penalize residuals of the underlying PDEs. Furthermore, similar to our approach, Raissi *et al.*[28] focused on the approximation of the incompressible Navier–Stokes equations based on leveraging vector potentials to obtain continuous divergence-free velocity fields. While such continuous methods allow smooth and accurate simulations as well as overcoming the curse of dimensionality of discrete techniques in high-dimensional PDEs,[18] the training of the respective networks relies on a specific domain. Hence, these networks are not capable of generalizing to new domains or being used in interactive scenarios.

In contrast, *discrete Eulerian methods* such as finite difference approaches, lattice Boltzmann[29,30] or finite element methods solve the underlying PDEs on a grid or a mesh. While seminal work has been published decades ago,[31,32] recent techniques particularly focus on leveraging the potential of deep learning techniques to achieve speedups while maintaining accuracy. Frameworks to learn parameterized fluid simulations[7] allow an efficient interpolation between such simulations. Furthermore, a recurrent generative adversarial network (RNN-GAN) has been used to produce turbulent flow fields within a pipe domain.[33] However, in both cases, a generalization to new domain geometries not considered during training has not been achieved. The tempoGAN introduced by Xi *et al.*[34] allows temporally consistent super-resolution in the context of smoke simulations, thereby producing plausible high-resolution smoke-density fields for low-resolution inputs. This, however, is not in accordance with our goal to obtain a fluid model that provides complete fluid state representations including velocity and pressure fields. With a focus on accelerating the simulation of Eulerian fluids, Tompson *et al.*[4] have shown how a Helmholtz projection step can be learned. While this method is capable to generalize to domain geometries not considered during training, the technique relies on a particle tracer to deal with the advection term of the Navier–Stokes equations. In addition, characteristic effects such as the Magnus effect or Kármán vortex streets cannot be simulated since Eulerian fluids do not model viscosity and dynamics boundary conditions were not considered. Several works[7,35] make use of discretized vector potentials to ensure incompressibility within the fluid domain but do not generalize to new fluid domains beyond their training data. Discarding the pressure term in the Navier–Stokes equation, Geneva *et al.*[8] introduced a physics-informed framework to learn the update step for the Burgers' equation. Thuerey *et al.*[36] proposed to learn solutions of the Reynolds-averaged Navier–Stokes equations for airfoil flows. However, their approach does not generalize beyond airfoil flows and Reynolds-averaged Navier–Stokes equations do not model the temporal evolution of a fluid state. Furthermore, Um *et al.*[6] focused on learning a correction step so that solutions of a high-resolution fluid simulation can be approximated by a low-resolution differentiable fluid solver. However, generalization to new domain geometries has not been demonstrated.

The approach of Wandel *et al.*[5] also falls into this category of discrete Eulerian approaches. However, unlike the aforementioned approaches, this approach does not rely on the availability of vast amounts of data from fluid solvers such as FEniCS, OpenFOAM[10] or Mantaflow and handles dynamic boundary conditions allowing for interactions with the fluid in 2D. In this paper, we extend this approach to 3D fluid dynamics and extend the networks capability toward also handling changes of fluid parameters such as viscosity and density during simulation.

III. METHOD

In this section, we first provide a brief introduction of the incompressible Navier–Stokes equations, which describe the dynamics of most incompressible fluids. This is followed by a review of the Helmholtz decomposition that can be used to ensure incompressibility within a fluid domain. Afterwards, we present the details of our discrete neural fluid model and show how a physics-informed loss function can be used to learn fluid dynamics in 3D without training data.

A. Incompressible Navier–Stokes Equations

The Navier–Stokes equations are a well-established model for the dynamics of most incompressible fluids. If we consider the state of an incompressible fluid consisting of a velocity field $\vec{v}$ and a pressure field p on a fluid domain Ω, then the incompressible Navier–Stokes equations describe its evolution over time by a set of two partial differential equations, which are often referred to as incompressibility equation and momentum equation.

The *incompressibility equation* ensures incompressibility of the fluid by enforcing that $\vec{v}$ is divergence-free

$$\nabla \cdot \vec{v} = 0 \text{ in } \Omega. \tag{1}$$

The *momentum equation* ensures conservation of momentum within the fluid

$$\rho\dot{\vec{v}} = \rho\left(\frac{\partial \vec{v}}{\partial t} + (\vec{v}\cdot\nabla)\vec{v}\right) = -\nabla p + \mu\Delta\vec{v} + \vec{f} \text{ in } \Omega. \tag{2}$$

Here, ρ denotes the fluid density and μ the viscosity. The left-hand side of this equation can be interpreted as the change in momentum of fluid particles and the right-hand side represents the sum of the forces acting on them. These forces include the pressure gradient ∇p, viscous friction $\mu\Delta\vec{v}$ and external forces $\vec{f}$. In this work, we set $\vec{f} = 0$ since external forces, such as gravity, can be neglected.

On top of ensuring incompressibility and conservation of momentum within Ω, we also have to match initial conditions $\vec{v}^0$ and p^0 at the beginning of the simulation and fulfill Dirichlet boundary (no-slip) conditions at the boundary of the domain $\partial\Omega$. The Dirichlet boundary conditions state that the velocity field $\vec{v}$ has to match the velocity $\vec{v}_d$ at the domain boundaries, i.e.,

$$\vec{v} = \vec{v}_d \text{ on } \partial\Omega. \tag{3}$$

B. Ensuring incompressibility using a vector potential

The Helmholtz theorem states that every vector field $\vec{v}$ can be decomposed into a curl-free part ∇q and a divergence-free part $\nabla \times \vec{a}$, that is

$$\vec{v} = \nabla q + \nabla \times \vec{a}. \tag{4}$$

Note that ∇q is curl-free $(\nabla \times (\nabla q) = \vec{0})$ and $\nabla \times \vec{a}$ is divergence-free $(\nabla \cdot (\nabla \times \vec{a}) = 0)$.

A common method to ensure incompressibility is to project $\vec{v}$ onto its divergence free part by solving the Poisson problem $\nabla \cdot \vec{v} = \Delta q$ followed by subtracting ∇q from $\vec{v}$.[4,32] Solving the Poisson problem, however, comes at high computational costs. Approximate, learned solutions[4] cannot guarantee proper projections onto the divergence-free part and thus might not fulfill incompressibility within the domain exactly.

For this reason, we directly predict a vector potential $\vec{a}$ similar to previous work.[5,7,28] This guarantees incompressibility of the velocity field $\vec{v} = \nabla \times \vec{a}$ within the domain and automatically solves the incompressibility equation [Eq. (1)].

C. Discrete spatio-temporal 3D fluid representation

In order to process the fluid state with a 3D convolutional neural network, we consider the following spatial and temporal discretizations:

$$\vec{a} = \begin{pmatrix} (a_x)^t_{i,j,k} \\ (a_y)^t_{i,j,k} \\ (a_z)^t_{i,j,k} \end{pmatrix}; \quad \vec{v} = \begin{pmatrix} (v_x)^t_{i,j,k} \\ (v_y)^t_{i,j,k} \\ (v_z)^t_{i,j,k} \end{pmatrix}; \quad p = p^t_{i,j,k}. \tag{5}$$

The relationship between $\vec{a}$, $\vec{v}$ and p can be efficiently represented by arranging the discretized quantities on a Marker-And-Cell (MAC) grid as depicted in Fig. 1(a). This grid representation allows us to compute gradients, divergence, curl, and Laplace operations for the Navier–Stokes equations in a straightforward manner.

To calculate the velocity field $\vec{v} = \nabla \times \vec{a}$ of a vector potential $\vec{a}$ on a MAC grid in 3D, we have to compute the curl as follows:

$$\begin{pmatrix} (v_x)_{i,j,k} \\ (v_y)_{i,j,k} \\ (v_z)_{i,j,k} \end{pmatrix} = \begin{pmatrix} (a_z)_{i,j+1,k} - (a_z)_{i,j,k} - (a_y)_{i,j,k+1} + (a_y)_{i,j,k} \\ (a_x)_{i,j,k+1} - (a_x)_{i,j,k} - (a_z)_{i+1,j,k} + (a_z)_{i,j,k} \\ (a_y)_{i+1,j,k} - (a_y)_{i,j,k} - (a_x)_{i,j+1,k} + (a_x)_{i,j,k} \end{pmatrix}. \tag{6}$$

The divergence of the velocity field $\vec{v}$ can be computed as follows:

$$\begin{aligned} \nabla \cdot \vec{v}_{i,j,k} &= (v_x)_{i+1,j,k} - (v_x)_{i,j,k} + (v_y)_{i,j+1,k} - (v_y)_{i,j,k} \\ &\quad + (v_z)_{i,j,k+1} - (v_z)_{i,j,k} \qquad (7) \\ &= 0. \qquad (8) \end{aligned}$$

By inserting the results from Eq. (6) into Eq. (7), we immediately arrive at Eq. (8) ($\vec{v}$ is divergence-free and the fluid thus incompressible).

For the Laplace operator, we use a 3D convolution with the 27-point stencil by[37] as it provides a more isotropic estimate of the Laplacian at similar computational costs compared to 7-point or 19-point stencils.

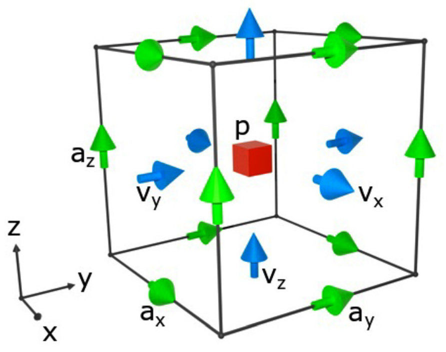
The temporal derivative in Eq. (2) is handled as follows:

$$\rho\left(\frac{\vec{v}^{t+dt} - \vec{v}^t}{dt} + \left(\vec{v}^{t'}\cdot\nabla\right)\vec{v}^{t'}\right) = -\nabla p^{t+dt} + \mu\Delta\vec{v}^{t'} + \vec{f}. \tag{9}$$

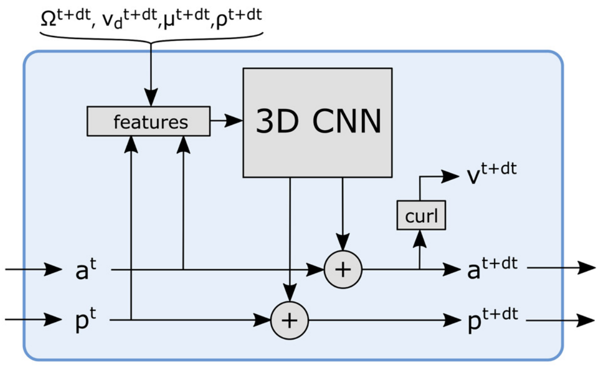
In literature, there are several different methods to assign $\vec{v}^{t'}$. The explicit method sets $\vec{v}^{t'} = \vec{v}^t$ whereas the implicit method sets $\vec{v}^{t'} = \vec{v}^{t+dt}$. Here, we focus on an implicit–explicit (IMEX) scheme that sets $\vec{v}^{t'} = \frac{\vec{v}^t + \vec{v}^{t+dt}}{2}$.

D. Fluid model

Building upon this discrete representation, we now introduce a recurrent model for fluid dynamics, F, that maps the fluid state specified by the vector potential $\vec{a}^t$ and the pressure field p^0 at time point t



(a)



(b)

FIG. 1. (a) 3D Marker-And-Cell (MAC) grid and (b) diagram of the fluid model. (a) Positions and directions of pressure coordinates (red box in the center), velocity coordinates (blue arrows perpendicular to lattice faces) and vector potential coordinates (green arrows along lattice edges) in a 3D staggered Marker-And-Cell (MAC) grid. (b) Pipeline of the 3D CNN-based fluid model. We unroll the fluid simulation in time by recurrently applying this fluid model on the fluid state $(\vec{a}, p)^t$. We tested 2 different 3D CNNs (see Fig. 2).

to its subsequent state $\vec{a}^{t+dt}$ and p^{t+dt} at time point $t + dt$, given the domain Ω^{t+dt} with boundary conditions $\vec{v}^{t+dt} = \vec{v}_d^{t+dt}$ and fluid parameters μ^{t+dt}, ρ^{t+dt}:

$$(\vec{a}, p)^{t+dt} = F((\vec{a}, p)^t, \Omega^{t+dt}, \vec{v}_d^{t+dt}, \mu^{t+dt}, \rho^{t+dt}). \tag{10}$$

By recurrently applying F on the initial fluid state $(\vec{a}^0, p^0)$, the fluid simulation can be unrolled in time for given boundary conditions and fluid parameters $(\Omega^t, \vec{v}_d^t, \mu^t, \rho^t)$:

$$(\vec{a},p)^{n\cdot dt} = F(\ldots F((\vec{a},p)^0, \Omega^{dt}, \vec{v}_d^{dt}, \mu^{dt}, \rho^{dt})\ldots, \Omega^{n\cdot dt}, \vec{v}_d^{n\cdot dt}, \mu^{n\cdot dt}, \rho^{n\cdot dt}). \tag{11}$$

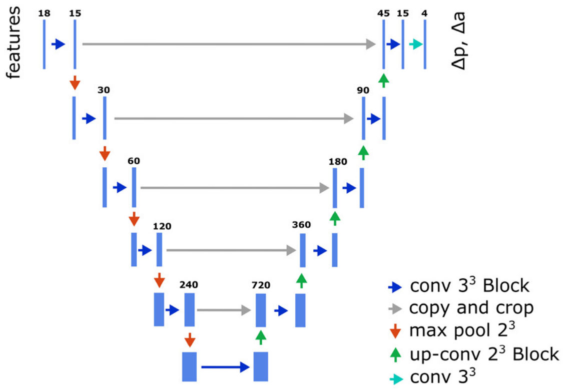
Figure 1(b) gives an overview over the fluid model F. First, a feature representation is build based on the inputs:

$$\text{Features} = (p^t, \vec{a}^t, \nabla \times \vec{a}^t, \Omega^{t+dt}, \partial\Omega^{t+dt}, \Omega^{t+dt} \cdot \nabla \times \vec{a}^t, \Omega^{t+dt} \cdot p^t, \partial\Omega^{t+dt} \cdot \vec{v}_d^{t+dt}, \ln(\mu^{t+dt}), \ln(\rho^{t+dt})). \tag{12}$$

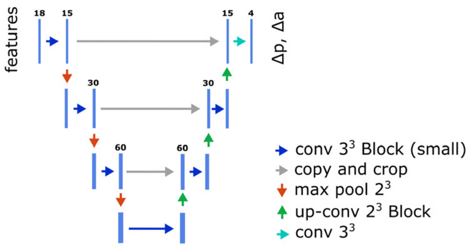
Here, the boundary $(\partial\Omega)$ is simply set to $1 - \Omega$. These features can be efficiently computed with convolutions and are then fed into a 3D CNN. We can make arbitrary choices for the 3D CNN and tested 2 different variants (see Fig. 2): a 3D U-Net[9] version for accurate simulations and a pruned 3D U-Net version that is less accurate but considerably faster. For this smaller model, we replaced concatenations with sums and removed 2 pooling stages as well as hidden layers at every stage. Then, the output of the 3D CNN is mean-normalized to prevent drifting offsets of $\vec{a}/p$ and added to the previous state of $\vec{a}^t$ and p^t to obtain the fluid state of the next time step $\vec{a}^{t+dt}$ and p^{t+dt}.

E. Physics-informed loss function

In the following, we introduce a loss function based on the residuals of the Navier–Stokes equations [Eqs. (1) and (2)] as well as the boundary conditions [see Eq. (3)]. Incompressibility [Eq. (1)] is already ensured by the vector potential. To enforce the momentum



(a)



(b)

FIG. 2. Two variants of 3D convolutional neural networks that were used inside the fluid model [see Fig. 1(b)]. (a) 3D U-Net architecture and (b) pruned 3D U-Net architecture. A U-Net consists of a downsampling part (performed by max pool operations marked in red), an upsampling part (performed by up-convolutions marked in green) and shortcut connections to preserve high-resolution features (marked in grey). (a) 3D U-Net architecture as introduced by Cicek *et al.*[9] (b) Pruned 3D U-Net architecture.

equation [Eq. (2)] to be fulfilled, we formulate the following momentum loss term:

$$L_p = \left\| \rho\left(\frac{\vec{v}^{t+dt} - \vec{v}^t}{dt} + \left(\vec{v}^{t'} \cdot \nabla\right)\vec{v}^{t'}\right) + \nabla p^{t+dt} - \mu\Delta\vec{v}^{t'} - \vec{f}\right\|^2 \text{ in } \Omega. \tag{13}$$

Furthermore, the compliance with the Dirichlet boundary conditions [Eq. (3)] is enforced by a boundary loss term:

$$L_b = \|\vec{v}^{t+dt} - \vec{v}_d^{t+dt}\|^2 \text{ on } \partial\Omega. \tag{14}$$

Combining the described loss terms, we obtain the following loss function:

$$L = \alpha L_p + \beta L_b. \tag{15}$$

α and β are hyperparameters to weight the different loss terms. We chose $\alpha = 1$ and $\beta = 20$, because errors in L_b lead to very unrealistic fluxes penetrating the boundaries. Note that in contrast to solving the Navier–Stokes equations explicitly, computing these loss terms can be done very efficiently by convolutions in $O(N)$ where N corresponds to the number of grid cells.

F. Training strategy

To start training, we initialize a pool $\{\vec{a}_k^0, p_k^0, \Omega_k^0, (\vec{v}_d)_k^0, \mu_k, \rho_k\}_{\{k\in\text{pool}\}}$ of initial states for the vector potential $\vec{a}_k^0$ and pressure field p_k^0 as well as randomized domains Ω_k^0, boundary conditions $(\vec{v}_d)_k^0$, and fluid parameters μ_k, ρ_k. For simplicity, the initial fluid states are set to 0 ($\vec{a}_k^0 = 0$ and $p_k^0 = 0$). The randomized domains contain primitive shapes such as boxes, spinning balls, or cylinders, and the resolution of these domains is $128 \times 64 \times 64$ voxels. Figure 3 shows examples of such training domains. In the future, more complex training domains could be added to further refine the fluid models. Note that in contrast to other 3D grid-based training methods (including Refs. 4 and 8) we do not need any simulated fluid-data.

For every training step, we draw a random mini-batch $\{\vec{a}_k^t, p_k^t, \Omega_k^t, (\vec{v}_d)_k^t, \mu_k, \rho_k\}_{\{k\in\text{minibatch}\}}$ (batch size = 14) from the pool and feed it into the neural network. Then, the neural network is asked to predict the velocity ($\vec{v}_k^{t+dt} = \nabla \times \vec{a}_k^{t+dt}$) and pressure ($p_k^{t+dt}$) fields of the next time step. Based on a physics-informed loss-function [Eq. (15)], we update the weights of the network using the Adam optimizer[38] (learning rate = 0.000 5). At the end of each training step, the pool is updated by replacing the old vector potential and pressure fields $\vec{a}_k^t, p_k^t$ by the newly predicted ones $\vec{a}_k^{t+dt}, p_k^{t+dt}$. This recycling strategy fills the training pool with more and more realistic fluid states as the model becomes better at simulating fluid dynamics.

From time to time, old environments of the training pool are replaced by new randomized environments, and the vector potential as well as the pressure fields is reset to 0. This increases the variance of the training pool and helps the neural network to learn "cold starts" from $\vec{0}$-velocity and 0-pressure fields.

For the implementation of the fluid models, we used the machine learning framework Pytorch and trained the models on an NVidia GeForce RTX 2080 Ti. Training converged after about 5 days.

IV. RESULTS

In the following, we present qualitative results for various different Reynolds numbers as well as quantitative results to compare the performance of the U-Net with the small model version.

A. Qualitative evaluation

Here, we provide a qualitative analysis of the wakeflow dynamics for different Reynolds numbers and show that our technique is capable of handling the Magnus effect as well as generalizing to new domains not seen during training.

1. Wakeflow dynamics

Snapshots of the velocity and pressure fields around an elongated obstacle that were generated by our fluid model are visualized in Fig. 4 with Paraview. In the following, we will discuss the produced wake dynamics and pressure fields qualitatively.

The wake dynamics behind an obstacle depend largely on the Reynolds number of a flow field. The Reynolds number is defined as follows:

$$Re = \frac{\rho\|\vec{v}\|D}{\mu}, \tag{16}$$

where ρ and μ are the fluid density and viscosity, respectively, $\|\vec{v}\|$ is the flow speed, and D the obstacle's diameter. For very small Reynolds numbers [see Fig. 4(a)], the flow becomes time-reversible. This means, if we would reverse the simulation, the streamlines would still look the same. This can be recognized by the symmetry of the streamlines before and after the obstacle and the pressure gradient. For Reynolds numbers around 10, the fluid starts to form a laminar wake behind the obstacle. This can be seen in Fig. 4(b), where two vortices are forming

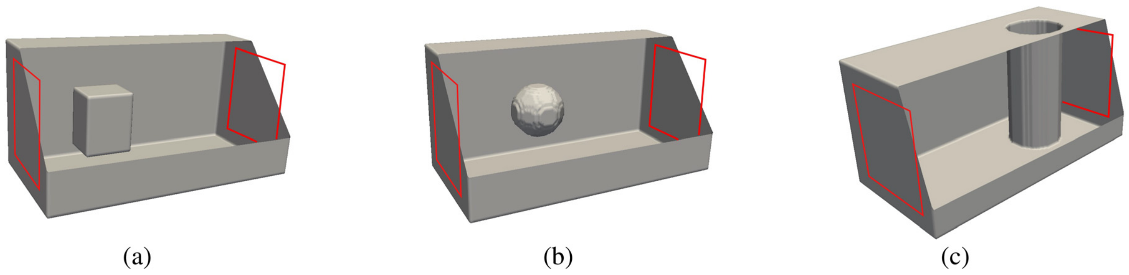


FIG. 3. (a)/(b)/(c) show examples of randomized training domains (resolution: $128 \times 64 \times 64$ voxels). The inflow/outflow boundaries are on the left/right sides of the domains (see red boxes). (a) Box environment. (b) Ball environment. (c) Cylinder environment.

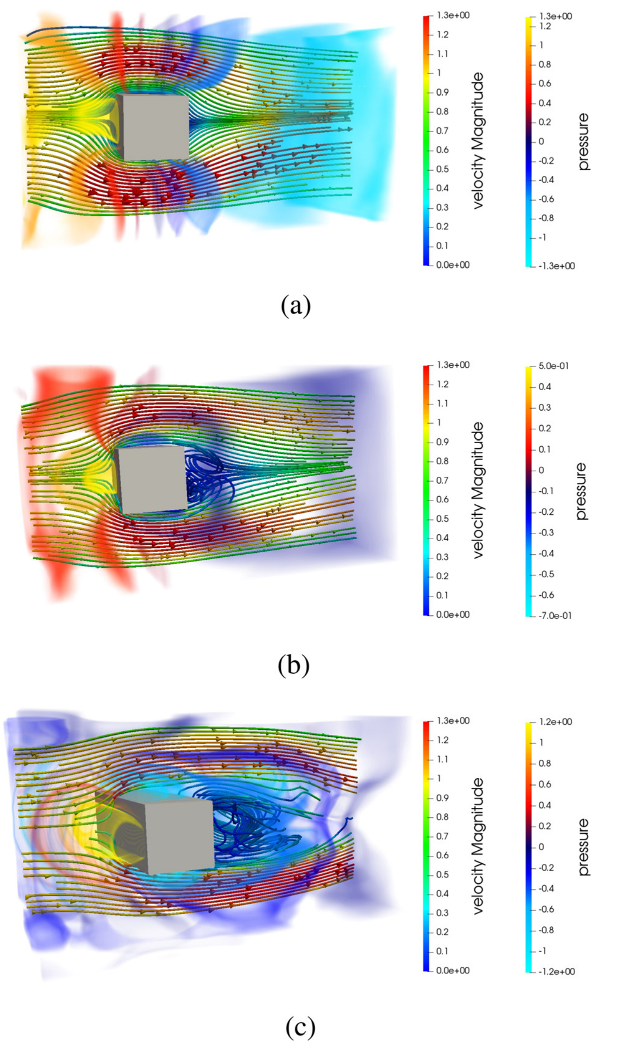


FIG. 4. (a)/(b)/(c) show streamlines of the velocity field and transparent isosurfaces of the pressure field for flows around square rods at different Reynolds numbers. The diameter of the rod is $D = 16$ (we use the units of the grid). All of these results were obtained by the same model (U-Net) and visualized with Paraview. Seamless interpolation between these states is possible as demonstrated in the supplementary video. (a) Time-reversible flow ($Re = 0.64, \mu = 5, \rho = 0.2$). (b) Laminar flow ($Re = 80, \mu = 0.2, \rho = 1$). (c) Turbulent flow ($Re = 800, \mu = 0.1, \rho = 5$).

behind the obstacle. For Reynolds numbers beyond 100, the wake becomes unstable and vortices generated at the obstacle start to detach and travel downstream. Figure 4(c) ($Re = 800$) clearly shows this turbulent behavior.

2. Magnus effect

The Magnus effect appears if a fluid streams around a rotating body. In this case, a high pressure field arises where the surface of the rotating body moves against flow direction and a low pressure field arises where the surface moves along flow direction. The Magnus effect plays a crucial role in sports such as, e.g., soccer or tennis where it is used to deflect the path of a spinning ball or in Flettner rotors to create a force perpendicular to a stream of air. In Fig. 5, this effect can be clearly recognized.

3. Generalization

We also tested the model's capability to generalize to new domain geometries that were not contained in the training dataset. In particular, we considered the shapes of a fish and 3 boxes (see Fig. 6). Generalizing to multiple objects was considered to be notably hard, since none of our randomized training domains contains more than one obstacle (see Fig. 3). Still, in both cases, our fluid model is able to match the boundary conditions and produce plausible flow and pressure fields (see streamlines evading the obstacles and high pressure fields in front of the obstacles).

To further improve the performance on domains not seen during training, pretrained fluid models can be fine-tuned on new domains.

4. Video

Impressions of the time-dependent fluid dynamics produced by our model are provided in the supplementary video [see Fig. 7 (Multimedia view)]. The frame rate of the renderings is synchronized with the speed of the fluid model to demonstrate its real-time capability. We show examples for the magnus effect, interpolation of different fluid viscosities and densities as well as generalization results for domains not considered during training.

B. Quantitative evaluation

In the following, we provide quantitative results of our method and investigate the stability of the fluid simulations over time.

1. U-Net/Pruned U-Net/Phiflow

We compare the performance of the U-Net, the pruned U-Net, and the recently released, open-source fluid simulation package Phiflow[39] quantitatively on a $128 \times 64 \times 64$ benchmark problem [see Fig. 8(a)]. Table I summarizes our measurements of the speed in time steps per second on an Intel Core i9-9940X (CPU) and a Nvidia GeForce RTX 2080 Ti (GPU) as well as the accuracy with respect to L_p, L_d and $E[||\nabla \cdot \vec{v}||]$. $E[||\nabla \cdot \vec{v}||]$ is defined as the mean L_2 norm of the velocity divergence. To compute L_d and $E[||\nabla \cdot \vec{v}||]$ for the (pruned) U-Net, we set the velocity field at the boundaries equal to the boundary conditions. Otherwise, $\nabla \cdot \vec{v}$ would be 0 everywhere due to the underlying vector potential. While the U-Net provides highly accurate results for L_p, L_d, and $E[||\nabla \cdot \vec{v}||]$, the small model yields considerably faster solutions with slightly less accuracy and is suitable for real-time simulations. Furthermore, the small model gets along with a drastically reduced number of parameters.

We also tested Phiflow on the benchmark problem and modeled viscosity with a diffusion step on the velocity field.[32] For $\mu = 0.1$, $\rho = 4$, we observed significantly higher losses compared to our approach. Furthermore, the U-Net as well as the pruned U-Net are considerably faster than Phiflow since they only require one forward pass through a convolutional neural network which can be easily parallelized and Phiflow relies on an iterative conjugate gradient solver. For $\mu = 1, \rho = 1$, the simulation with Phiflow became unstable. This

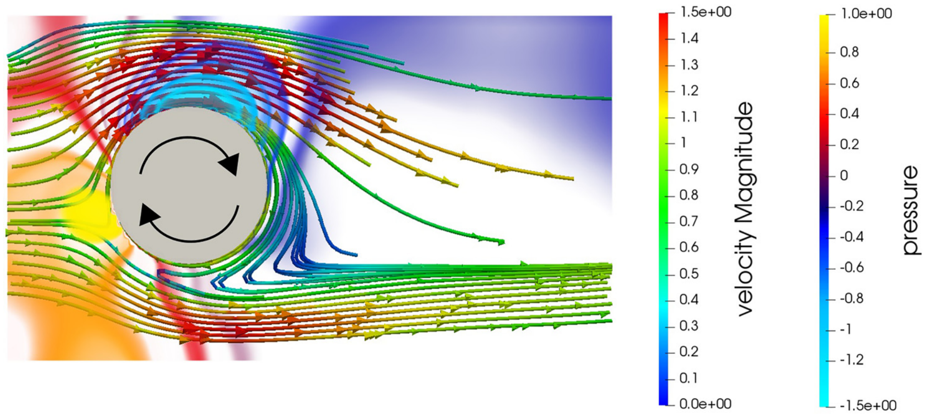


FIG. 5. Streamlines of the velocity field and transparent isosurfaces of the pressure field for Magnus effect on a clockwise spinning cylinder ($\mu = 0.5, \rho = 1$).

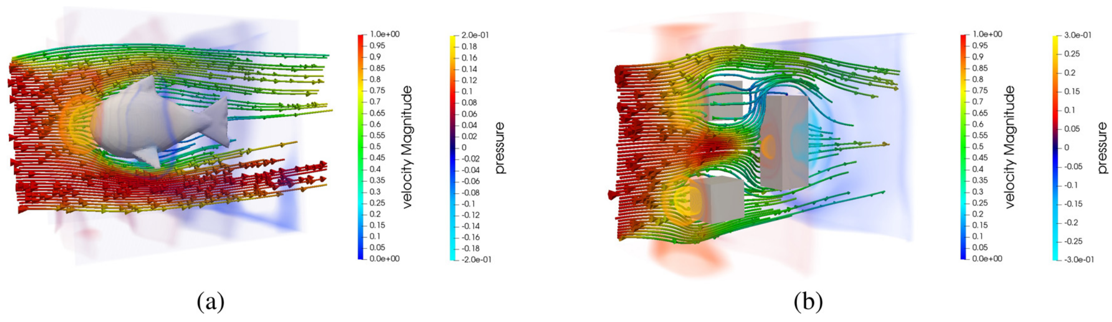


FIG. 6. Streamlines of the velocity field and transparent isosurfaces of the pressure field of generalization examples for objects not seen during training. (a) shows flow around a fish shape and (b) shows flow around multiple objects. For a better visualization of the dynamic fluid behavior, we refer to the supplementary video. (a) Exemplary simulation result for fish shapes that were not considered in the training set. ($\mu = 0.5, \rho = 1$). (b) Exemplary simulation result for multiple objects. During training, the domain contained only one obstacle. ($\mu = 0.5, \rho = 1$.)

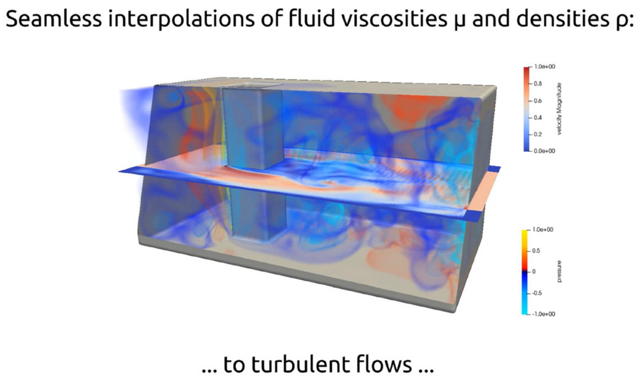

Seamless interpolations of fluid viscosities μ and densities ρ:
velocity Magnitude
pressure
... to turbulent flows ...

FIG. 7. Video of results obtained by the fluid model. Multimedia view: https://doi.org/10.1063/5.0047428.1

could be avoided by choosing smaller time steps, however, smaller time steps would further slowdown the simulation with Phiflow.

Note that a direct comparison to the approach by Tompson *et al.*[4] is not possible since their approach only considers Eulerian fluids and therefore does not model viscosity. However, when considering $E[||\nabla \cdot \vec{v}||]$ our method indicates significantly lower divergence of the velocity field (by 3 orders of magnitude)—presumably because our method learns a vector field instead of a Helmholtz projection step. Furthermore, our approach is significantly faster than the approach by Um *et al.*,[6] which reports 7.6 time steps per second on a smaller 64 × 32 × 32 fluid-domain. In contrast, our simulation runs at 36 time steps per second for a domain of size 128 × 64 × 64. This may result from the fact that our method does not rely on a differentiable fluid solver.

2. Stability

Figure 8(b) shows the evolution of $E[||\nabla \cdot \vec{v}||]$ and L_p over time of a simulation performed by the U-Net on the benchmark setup [see Fig. 8(a)]. Since the simulation starts with the vector potential and pressure field set to 0 ($\vec{a}^0 = \vec{0}$ and $p^0 = 0$), several time steps are needed for warm-up. After about 500 time steps, good stability characteristics are shown over thousands of time steps with only marginal increases in $E[||\nabla \cdot \vec{v}||]$ and L_p.

V. DISCUSSION AND OUTLOOK

In this work, we proposed a novel unsupervised approach of learning incompressible fluid dynamics in 3D using an efficient surrogate fluid model based on a convolutional neural network. For this purpose, we used the combination of a physics-informed loss function

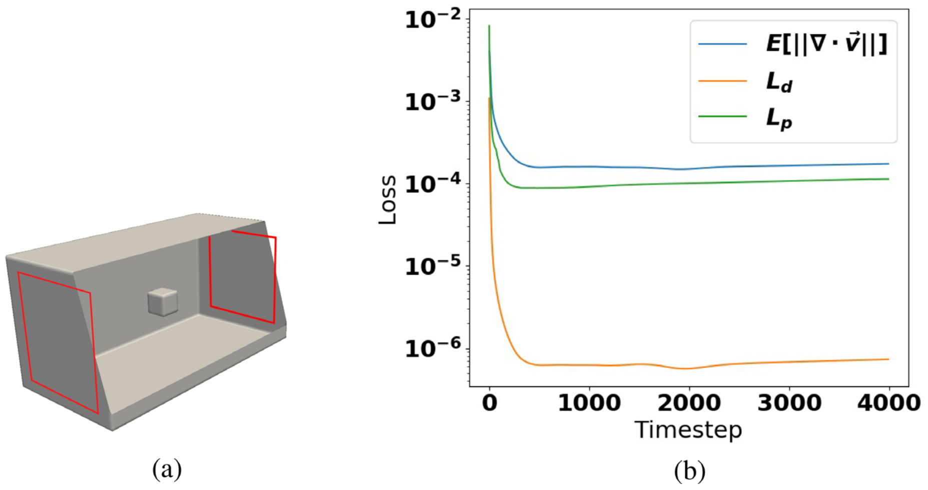


FIG. 8. (a) Benchmark setup; (b) stability analysis of the U-Net over time. (a) Benchmark setup. The red rectangles on the left and right side mark the inflow/outflow boundaries of the domain. (b) Stability of the U-Net for $\mu = 0.1, \rho = 4, dt = 4$. Section IV B 2 discusses the stability behavior over time.

TABLE I. Quantitative comparison of accuracy with respect to L_p, L_d and $E[||\nabla \cdot \vec{v}||]$ for different μ/ρ and computational speed in time steps per second (TPS) on a CPU and GPU as well as number of trainable parameters (n_{params}). The grid size was 128 × 64 × 64 and $dt = 4$. $*$ unstable loss due to diffusion step.

	Speed (TPS)		$\mu = 0.1, \rho = 4$			$\mu = 1, \rho = 1$			
Method	CPU	GPU	L_p	L_d	$E[\|\|\nabla \cdot \vec{v}\|\|]$	L_p	L_d	$E[\|\|\nabla \cdot \vec{v}\|\|]$	n_{params}
PhiFlow	0.22	···	···	$2.668\,48 \times 10^{-4}$	$1.631\,7 \times 10^{-3}$	···	$1.261\,4 \times 10^{5\,*}$	4.8894^{*}	0
U-Net	0.5	16	$1.056\,18 \times 10^{-4}$	$6.589\,4 \times 10^{-7}$	$1.619\,95 \times 10^{-4}$	$1.732\,59 \times 10^{-4}$	$7.531\,65 \times 10^{-7}$	$1.519\,11 \times 10^{-4}$	29M
Pruned U-Net	1.19	36	$1.182\,33 \times 10^{-4}$	$1.057\,7 \times 10^{-6}$	$1.825\,98 \times 10^{-4}$	$5.329\,99 \times 10^{-4}$	$1.548\,98 \times 10^{-6}$	$2.056\,91 \times 10^{-4}$	649k

on a 3D staggered grid and a data pool that automatically gets filled with more and more realistic fluid states over the course of training. In contrast to other approaches, our approach does not rely on the availability of any data from fluid solvers such as FEniCS, OpenFOAM[10] or Mantaflow. Our fluid models allow for fast fluid simulations while taking into account various fluid phenomena such as the Magnus effect and Kármán vortex streets. Furthermore, they can handle dynamically changing boundary conditions as required for interactive scenarios and generalize to new domains.

The speed of our method allows for real-time graphics applications such as games. In addition, the fluid models are fully differentiable and thus enable efficient gradient propagation throughout the fluid simulation as shown by Wandel *et al.*[5] This could be exploited for sensitivity analysis, to estimate the viscosity and density of a fluid by investigating its velocity and pressure fields or in machine learning scenarios that aim at controlling fluid fields using gradient-based methods. In the future, more sophisticated network architectures could be explored to further increase speed and accuracy of the simulation. Furthermore, Neumann boundary and external force fields could be incorporated into the surrogate model.

REFERENCES

[1]M. G. Schultz, C. Betancourt, B. Gong, F. Kleinert, M. Langguth, L. H. Leufen, A. Mozaffari, and S. Stadtler, "Can deep learning beat numerical weather prediction?," Philos. Trans. R. Soc. A **379**(2194), 20200097 (2021).
[2]A. Fabregat, F. Gisbert, A. Vernet, J. A. Ferré, K. Mittal, S. Dutta, and J. Pallarès, "Direct numerical simulation of turbulent dispersion of evaporative aerosol clouds produced by an intense expiratory event," Phys. Fluids **33**(3), 033329 (2021).
[3]Z. Li, X. Zhang, T. Wu, L. Zhu, J. Qin, and X. Yang, "Effects of slope and speed of escalator on the dispersion of cough-generated droplets from a passenger," Phys. Fluids **33**(4), 041701 (2021).
[4]J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, "Accelerating Eulerian fluid simulation with convolutional networks," in Proceedings of the 34th International Conference on Machine Learning (2017), Vol. 70, pp. 3424–3433.
[5]W. Nils, W. Michael, and K. Reinhard, "Learning incompressible fluid dynamics from scratch—towards fast, differentiable fluid models that generalize," 9th International Conference on Learning Representations (ICLR) (2021).
[6]U. Kiwon, R. Fei, P. Holl, R. Brand, and N. Thuerey, "Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers," Adv. Neural Info. Process. Syst. (published online 2020).
[7]B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and, and B. Solenthaler, "Deep fluids: A generative network for parameterized fluid simulations," Comput. Graphics Forum **38**, 59–70 (2019).
[8]N. Geneva and N. Zabaras, "Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks," J. Comput. Phys. **403**, 109056 (2020).
[9]Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3d U-Net: Learning dense volumetric segmentation from sparse annotation," arXiv:1606.06650 (2016).
[10]OpenCFD, *OpenFOAM—The Open Source CFD Toolbox—User's Guide*, 1.4 ed. (OpenCFD Ltd., United Kingdom, 2007).
[11]R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: Theory and application to non-spherical stars," Mon. Not. R. Astron. Soc. **181**(3), 375–389 (1977).

[12]L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross, "Data-driven fluid simulations using regression forests," ACM Trans. Graphics **34**(6), 1 (2015).
[13]D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum, and D. L. K. Yamins, "Flexible neural representation for physics prediction," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (NIPS'18) (Curran Associates Inc., Red Hook, NY, USA, 2018), pp. 8813–8824.
[14]Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids," ICLR (2019).
[15]U. Benjamin, P. Lukas, T. Nils, and K., Vladlen, "Lagrangian fluid simulation with continuous convolutions," in 8th International Conference on Learning Representations (ICLR), Addis Ababa, Ethiopia, 26–30 April, 2020.
[16]S. Connor and F. Dieter, "SPNets: Differentiable fluid dynamics for deep neural networks," in Conference on Robot Learning (2018), pp. 317–335.
[17]S. Justin and S. Konstantinos, "DGM: A deep learning algorithm for solving partial differential equations," J. Comput. Phys. **375**, 1339–1364 (2018).
[18]G. Philipp, H. Fabian, J. Arnulf, and P. V. Wurstemberger, "A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations," arXiv:1809.02362 (2018).
[19]Y. Khoo, J. Lu, and L. Ying, "Solving for high-dimensional committor functions using artificial neural networks," Res. Math. Sci. **6**(1), 1 (2019).
[20]Y. Zhu and N. Zabaras, "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification," J. Comput. Phys. **366**, 415–447 (2018).
[21]Y. Zhu and N. Zabaras, "Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data," J. Comput. Phys. **394**, 56–81 (2019).
[22]R. Tripathy and I. Bilionis, "Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification," J. Comput. Phys. **375**, 565–588 (2018).
[23]C. Yang, X. Yang, and X. Xiao, "Data-driven projection method in fluid simulation," Comput. Animation Virtual Worlds **27**(3–4), 415–424 (2016).
[24]R. Maziar, Y. Alireza, and G. E. Karniadakis, "Hidden fluid mechanics: A Navier-Stokes informed deep learning framework for assimilating flow visualization data," arXiv:1808.04327 (2018).
[25]N. Geneva and N. Zabaras, "Quantifying model form uncertainty in Reynolds-averaged turbulence models with Bayesian deep neural networks," J. Comput. Phys. **383**, 125–147 (2019).
[26]J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," J. Fluid Mech. **807**, 155–166 (2016).
[27]M. Schőberl, N. Zabaras, and P.-S. Koutsourelakis, "Predictive collective variable discovery with deep Bayesian models," J. Chem. Phys. **150**(2), 024109 (2019).
[28]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," J. Comput. Phys. **378**, 686–707 (2019).
[29]S. Chen and G. D. Doolen, "Lattice Boltzmann method for fluid flows," Annu. Rev. Fluid Mech. **30**(1), 329–364 (1998).
[30]Z. Guo, "Well-balanced lattice Boltzmann model for two-phase systems," Phys. Fluids **33**(3), 031709 (2021).
[31]F. H. Harlow and J. E. Welch, "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface," Phys. Fluids **8**(12), 2182–2189 (1965).
[32]J. Stam, "Stable fluids," in Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (1999), pp. 121–128.
[33]J. Kim and C. Lee, "Deep unsupervised learning of turbulence for inflow generation at various Reynolds numbers," J. Comput. Phys. **406**, 109216 (2020).
[34]Y. Xie, E. Franz, M. Chu, and N. Thuerey, "tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow," ACM Trans. Graphics **37**(4), 1 (2018).
[35]A. T. Mohan, N. Lubbers, D. Livescu, and M. Chertkov, "Embedding hard physical constraints in neural network coarse-graining of 3d turbulence," in *International Conference on Learning Representations* (ICLR, 2020).
[36]N. Thuerey, K. Weißenow, L. Prantl, and X. Hu, "Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows," AIAA J. **58**, 25–12 (2020).
[37]H. O'Reilly and J. M. Beck, "A family of large-stencil discrete Laplacian approximations in three dimensions," Int. J. Numer. Methods Eng. **2006**, 1–16.
[38]D. P. Kingma and B. Jimmy, "Adam: A method for stochastic optimization," in 3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May, 2015.
[39]P. Holl, V. Koltun, and and N. Thuerey, "Learning to control PDEs with differentiable physics," ICLR (2020).

Publication: “Spline-PINN: Approaching PDEs without Data using Fast, Physics-Informed Hermite-Spline CNNs”

Nils Wandel, Michael Weinmann, Michael Neidlin, and Reinhard Klein

Proceedings of the 36th AAAI Conference on Artificial Intelligence

2022

Spline-PINN: Approaching PDEs without Data Using Fast, Physics-Informed Hermite-Spline CNNs

Nils Wandel,[1] Michael Weinmann,[2] Michael Neidlin,[3] Reinhard Klein [1]

[1] University of Bonn
[2] Delft University of Technology
[3] RWTH Aachen University
wandeln@cs.uni-bonn.de, m.weinmann@tudelft.nl, neidlin@ame.rwth-aachen.de, rk@cs.uni-bonn.de

Abstract

Partial Differential Equations (PDEs) are notoriously difficult to solve. In general, closed-form solutions are not available and numerical approximation schemes are computationally expensive. In this paper, we propose to approach the solution of PDEs based on a novel technique that combines the advantages of two recently emerging machine learning based approaches. First, physics-informed neural networks (PINNs) learn continuous solutions of PDEs and can be trained with little to no ground truth data. However, PINNs do not generalize well to unseen domains. Second, convolutional neural networks provide fast inference and generalize but either require large amounts of training data or a physics-constrained loss based on finite differences that can lead to inaccuracies and discretization artifacts.

We leverage the advantages of both of these approaches by using Hermite spline kernels in order to continuously interpolate a grid-based state representation that can be handled by a CNN. This allows for training without any precomputed training data using a physics-informed loss function only and provides fast, continuous solutions that generalize to unseen domains. We demonstrate the potential of our method at the examples of the incompressible Navier-Stokes equation and the damped wave equation. Our models are able to learn several intriguing phenomena such as Karman vortex streets, the Magnus effect, Doppler effect, interference patterns and wave reflections. Our quantitative assessment and an interactive real-time demo show that we are narrowing the gap in accuracy of unsupervised ML based methods to industrial solvers for computational fluid dynamics (CFD) while being orders of magnitude faster.

Introduction

Partial differential equations (PDEs) are an important mathematical concept to describe for example the motion of fluids, the propagation of waves, the evolution of stock markets, gravity and more. However, solving partial differential equations is a hard problem since closed-form solutions are rarely available. Thus, developing fast and accurate numerical schemes in order to find approximate solutions is of great interest for applications such as e.g. physics engines in computer games, computer generated imagery (CGI) for movies, or computational fluid dynamics (CFD) to help engineers with simulated wind tunnel experiments.

Recently, advances of machine learning (ML) based approaches have led to promising results coping with the high computational costs associated with classical numerical methods. Furthermore, some ML based approaches are by design differentiable. This means that they offer gradients that can be used for stability analysis, optimal control or reinforcement learning. However, ML based approaches often do not generalize to domains not seen during training [Kim and Lee 2020, Mohan et al. 2020, Thuerey et al. 2019, Um et al. 2020, Raissi, Perdikaris, and Karniadakis 2019] or rely on large amounts of training data which capture the variations to be expected in foreseen scenarios [Pfaff et al. 2021, Kim et al. 2019, Ladický et al. 2015].Recent physics-constrained approaches based on finite differences [Tompson et al. 2017, Zhu et al. 2019, Wandel, Weinmann, and Klein 2021a,b] exhibit the potential to mitigate these problems but might lead to inaccuracies and discretization artifacts, especially at high Reynolds-Numbers.

Here, we propose to use a Hermite spline CNN to obtain continuous solutions that can be trained with a physics-informed loss only. This approach combines the advantages of i) leveraging physics-informed neural networks (PINNs) to overcome the need for large amounts of training data [Jin et al. 2021, Raissi, Yazdani, and Karniadakis 2018, Raissi, Perdikaris, and Karniadakis 2019] and ii) leveraging the faster inference and better generalization capabilities offered by convolutional neural networks [Tompson et al. 2017, Wandel, Weinmann, and Klein 2021a,b]. We demonstrate the effectiveness of our approach for the incompressible Navier-Stokes equations as well as the damped wave equation. The incompressible Navier-Stokes equations, which are particularly hard to solve due to the non-linear advection term and thus the main focus of this paper, are investigated for Reynolds numbers ranging from 2-10000. To assess the accuracy of our method, we compute drag and lift coefficients on a CFD benchmark domain [Schäfer and Turek 1996] and compare the results with official benchmark values. For both equations, we perform generalization experiments, evaluate the stability over long time horizons and present an interactive real-time demonstration. To ensure full reproducibility, our code is publicly available on github: *https://github.com/aschethor/Spline_PINN*.

Related Work

Recent developments indicate the potential of efficient surrogate models based on machine learning, and in particular deep learning, to approximate the dynamics of partial differential equations (PDEs).

Lagrangian methods like smoothed particle hydrodynamics (SPH) [Gingold and Monaghan 1977] represent fluids in the reference frame of individual particles that follow the fluid's velocity field. Relying on this principle, learning-based Lagrangian approaches include the simulation of fluids based on regression forests [Ladický et al. 2015], graph neural networks [Mrowca et al. 2018, Li et al. 2019], continuous convolutions [Ummenhofer et al. 2020] and Smooth Particle Networks (SP-Nets) [Schenck and Fox 2018]. While Lagrangian methods are particularly suitable for fluid domains that exhibit large, dynamic surfaces such as waves and droplets, however, Eulerian methods typically allow a more accurate simulation of fluid dynamics within a certain fluid domain (see [Foster and Metaxas 1996]).

Eulerian methods model domain characteristics such as a velocity or pressure field on a fixed reference frame. Respective techniques leverage implicit neural representations, grid- and mesh-structures to describe the domain of a PDE.

Continuous Eulerian methods exploit the direct mapping of domain coordinates (i.e. positional coordinates and time) to field values (i.e. velocity, pressure, etc.) relying e.g. on implicit neural representations to obtain smooth and accurate simulations [Sirignano and Spiliopoulos 2018a, Grohs et al. 2018, Khoo, Lu, and Ying 2019, Raissi, Perdikaris, and Karniadakis 2019, Lu et al. 2021] and handle the curse of dimensionality faced by discrete techniques in the case of high-dimensional PDEs [Grohs et al. 2018]. Respective applications include the modeling of flow through porous media [Zhu and Zabaras 2018, Zhu et al. 2019, Tripathy and Bilionis 2018], fluid modeling [Yang, Yang, and Xiao 2016, Raissi, Yazdani, and Karniadakis 2018], turbulence modeling [Geneva and Zabaras 2019, Ling, Kurzawski, and Templeton 2016], wave propagation [Rasht-Behesht et al. 2021, Sitzmann et al. 2020] and the modeling of molecular dynamics [Schöberl, Zabaras, and Koutsourelakis 2019]. Training typically relies on penalizing residuals of the underlying PDEs based on physics-informed loss functions for a specific domain, which prevents the networks from generalizing to new domain geometries and being used in the scope of interactive scenarios. Recently, Wang et al. [Wang et al. 2021] proposed to stitch pretrained Deep Learning models together to speed up the learning process for boundary value problems. However, temporal dependencies are not embedded in their method yet preventing interactive applications.

Discrete Eulerian methods such as finite difference, lattice Boltzmann [Chen and Doolen 1998, Guo 2021] or finite element methods instead tackle the underlying PDEs on an explicit representation such as a discrete grid or a graph structure. Beyond early seminal work [Harlow and Welch 1965, Stam 1999], recent developments focus on exploiting the potential of deep learning techniques to achieve significant speed-ups while preserving accuracy. Learning fluid simulations has been approached with deep generative models for efficient interpolations between various simulation settings [Kim et al. 2019] and turbulent flow fields within pipe domains [Kim and Lee 2020]. However, both of these approaches do not generalize to new domain geometries that have not been used during training. To speed up Eulerian fluid simulations, Tompson et al. [Tompson et al. 2017] proposed to learn a Helmholtz projection step with a CNN that generalizes to new domain geometries that have not been used during the training. However, this approach requires a path tracer to deal with the advection term of the Navier-Stokes equations. Furthermore, as viscosity is not considered in Eulerian fluid models, phenomena such as the Magnus effect and Karman vortex streets cannot be simulated. Ensuring incompressibility of fluids has been achieved with discretized vector potentials [Kim et al. 2019, Mohan et al. 2020], however, these techniques are not capable of generalizing to domain geometries not seen during the training. The approach by Geneva et al. [Geneva and Zabaras 2020] discards the pressure term in the Navier-Stokes equations which leads to the simpler Burgers' equations, for which they learned the update step based on a physics-constrained framework. Thuerey et al. [Thuerey et al. 2019] learn to solve the Reynolds-averaged Navier-Stokes equations, but the specific approach prevents a generalization beyond airfoil flows and discards the temporal evolution of the fluid state. Um et al. [Um et al. 2020] proposed to learn a correction step which allows approximating solutions of a high-resolution fluid simulations in terms of a low-resolution differentiable fluid solver, but the generalization capability to domain geometries that have not been used during training has not been shown. In contrast to the aforementioned techniques, the approach by Wandel et al. [Wandel, Weinmann, and Klein 2021a] and its 3D extension [Wandel, Weinmann, and Klein 2021b] overcome the dependence on huge amounts of training data by using a physics-constrained loss and introducing a training cycle that recycles data generated by the network during training. This allows handling dynamic boundary conditions as required for interactions with the fluid [Wandel, Weinmann, and Klein 2021a,b] and, with an improved network [Wandel, Weinmann, and Klein 2021b], also dealing with temporally varying fluid parameters such as viscosity and density during simulation. However, their physics-constrained loss can lead to discretization artifacts and requires a pressure gradient regularizer for high Reynolds numbers. Further learning-based approaches also exploit graph representations in terms of graph neural networks [Harsch and Riedelbauch 2021, Sanchez-Gonzalez et al. 2020], graph convolutional networks [Gao, Zahr, and Wang 2021] as well as mesh representations [Pfaff et al. 2021] or subspace representations [Sirignano and Spiliopoulos 2018b, Ainsworth and Dong 2021]. Unfortunately, graph neural networks cannot make use of the highly efficient implementations for convolutional operations on grids and thus usually come with higher computational complexity per node compared to grid based CNNs.

To the best of our knowledge, the potential of Spline CNNs [Fey et al. 2017] has not yet been investigated for learning the dynamics of PDEs and, hence, the presented physics-informed Hermite Spline CNN (Spline-PINN) approach is the first method that leverages both implicit and explicit characteristics to obtain fast, interactive, continuous surrogate models that generalize and can be trained without training data using a physics-informed loss based on the underlying PDEs.

Method

In the main part of our paper, we first briefly discuss the background regarding partial differential equations with a focus on the Navier-Stokes equations and the damped wave equation, before we introduce our physics-informed Hermite spline CNN approach (Spline-PINN), that combines the advantages of PINNs regarding physics-informed training without training data and Spline-CNNs regarding their generalization capability.

Partial Differential Equations

Partial Differential Equations describe the dependencies between partial derivatives of a multivariate function inside a domain Ω and usually need to be solved for given initial conditions at the beginning of the simulation and boundary conditions at the domain boundaries $\partial\Omega$.

The Incompressible Navier-Stokes Equations describe the dynamics of a fluid with a pressure-field p and a velocity-field $\vec{v}$ inside a domain Ω by means of the incompressibility and momentum equation:

$$\nabla \cdot \vec{v} = 0 \quad \text{in } \Omega \quad (1)$$

$$\rho\dot{\vec{v}} = \rho(\partial_t \vec{v} + \vec{v} \cdot \nabla\vec{v}) = \mu\Delta\vec{v} - \nabla p + \vec{f} \quad \text{in } \Omega \quad (2)$$

Here, ρ represents the fluid's density and μ its viscosity. The external forces $\vec{f}$ are neglected in our experiments. These two equations have to be solved for given initial conditions $\vec{v}_0, p_0$ and boundary conditions (BCs). Here, we consider the Dirichlet BC to set the velocity field $\vec{v} = \vec{v}_d$ at the domain boundaries $\partial\Omega$. In this work, we exploit the Helmholtz decomposition theorem and use a vector potential $\vec{a}$ with $\vec{v} = \nabla \times \vec{a}$ to automatically ensure incompressibility. Furthermore, we restrict our considerations to 2D flows and, hence, only the z-component of $\vec{a}$, a_z, is needed. The Navier-Stokes equations are considered particularly hard to solve due to the non-linear advection term $(\vec{v} \cdot \nabla\vec{v})$ and, therefore, are the main focus of our investigations.

The Damped Wave Equation can be used to describe for example the dynamics of a thin membrane with height-field z and velocity-field v_z.

$$\partial_t z = v_z \quad \text{in } \Omega \quad (3)$$

$$\partial_t v_z = k\Delta z - \delta v_z \quad \text{in } \Omega \quad (4)$$

Here, k is the stiffness constant of the membrane and δ is a damping constant. As for the Navier-Stokes equations, we solve these equations for given initial conditions z_0, v_{z0} and Dirichlet boundary conditions: $z = z_d$ on $\partial\Omega$.

Although the damped wave equation contains only linear components, we present this additional example to demonstrate that our method might also work for different classes of PDEs. The wave equation lays the foundation for many more complex equations such as the electromagnetic wave equation or the Schrödinger equation. Furthermore, for high damping constants δ, z converges towards the solution of the Laplace equation.

Spline-PINN

In this section, we introduce our Physics-Informed Hermite Spline CNN. First, we provide an overview on Hermite splines and then show how to incorporate them into a CNN for solving PDEs.

Hermite Splines In general, Hermite splines are piecewise polynomials that are defined by their values and their first n derivatives at given support points. To facilitate CNN-based processing, we arrange the support points on a regular grid. Figure 1 shows an example of Hermite spline kernel functions in 1D for $n = 0, 1, 2$. We define the kernel functions $h_i^n(x)$, such that the values and first n derivatives at the support points ($x = -1, 0, 1$) are set to 0 except the ith derivative at $x = 0$ which takes a value such that $h_i^n(x) \in [-1, 1]$. In contrast to B-Spline kernels, the support of Hermite spline kernels ranges only over the directly neighboring grid cells. This facilitates learning of high frequencies and allows for computationally more efficient interpolations with comparatively small transposed convolution kernels. By linearly combining Hermite spline kernel functions of order n at every grid-cell we obtain continuous piecewise polynomials with $(n + 1)$-th derivatives of bounded variation. Choosing the right spline order n is important to be able to compute the physics informed loss for a given PDE as will be discussed below.

To obtain kernel functions in multiple dimensions, we use the tensor product of multiple 1D kernel functions:

$$h_{i,j,k}^{l,m,n}(x, y, t) = h_i^l(x)h_j^m(y)h_k^n(t) \quad (5)$$

In the supplementary, we show how basis flow fields can be obtained by taking the curl of $h_{i,j}^{l,m}(x, y)$. On a grid $\hat{x}, \hat{y}, \hat{t} \in \hat{X} \times \hat{Y} \times \hat{T}$ with discrete spline coefficients $c_{\hat{x},\hat{y},\hat{t}}^{i,j,k}$, we obtain a continuous Hermite spline $g(x, y, t)$ as follows:

$$g(x, y, t) = \sum_{\substack{i,j,k \in [0:l] \times [0:m] \times [0:n] \\ \hat{x},\hat{y},\hat{t} \in \hat{X} \times \hat{Y} \times \hat{T}}} c_{\hat{x},\hat{y},\hat{t}}^{i,j,k} h_{i,j,k}^{l,m,n}(x - \hat{x}, y - \hat{y}, t - \hat{t}) \quad (6)$$

Our goal is, to find spline coefficients $c_{\hat{x},\hat{y},\hat{t}}^{i,j,k}$ such that $g(x, y, t)$ matches the solution of PDEs as closely as possible. The partial derivatives of g with respect to x, y, t can be directly obtained by taking the corresponding derivatives of the spline kernel functions.

Pipeline Figure 2 depicts the pipeline of our Spline-PINN. We use a CNN (PDE Model) to map discrete Hermite-spline coefficients and boundary conditions from a timepoint $\hat{t}$ to spline coefficients at a succeeding timepoint $\hat{t} + dt$. By recurrently applying the PDE Model on the spline coefficients, the simulation can be unrolled in time. From these discrete spline-coefficients, continuous Hermite splines can be efficiently obtained using transposed convolutions (see Equation 6) with interpolation kernels as depicted in Figure 1.

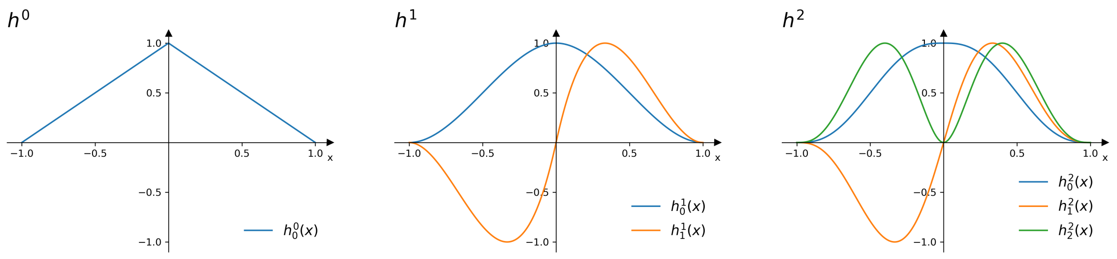


Figure 1: 1D Hermite spline kernels for $n = 0, 1, 2$ (scaled between -1 and 1). Note, that these kernel functions are in C^n and thus, the $(n + 1)$-th derivatives are of bounded variation. The case $n = 0$ could be considered as a linear interpolation between spline coefficients.

This way, training samples as well as evaluation samples can be taken at any point in space and time.

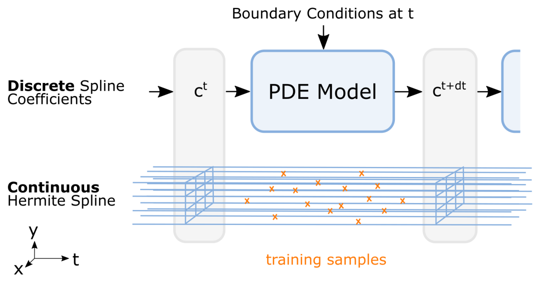


Figure 2: Pipeline of PDE Model with Hermite spline interpolation. Since the solution is continuous, training samples and evaluation samples can be obtained at any point in space and time. More detailed views of the PDE Models used for the Navier-Stokes equations and damped wave equation are provided in the supplementary material.

Physics Informed Loss When training neural networks on PDEs, the information provided by the PDEs themselves can be used to save or even completely spare out training-data. In literature, two major approaches have been established:

Physics-*constrained* approaches [Zhu et al. 2019] compute losses *explicitly* - usually on a grid topology using finite differences in order to approximate the derivatives of a PDE. This approach is suitable to train CNNs and allows neural networks to learn e.g. the incompressible Navier-Stokes equations without any ground truth training-data and generalize to new domains [Wandel, Weinmann, and Klein 2021a]. However, relying on finite differences may lead to inaccuracies and strong discretization artifacts - especially at high Reynolds-numbers (see Figure 3).

In contrast, physics-*informed* approaches [Raissi, Perdikaris, and Karniadakis 2019] compute losses *implicitly* - usually by taking derivatives of an implicit field description. This approach enables efficient training of implicit neural networks and yields continuous solutions that can be evaluated at any point in space and time. However, implicit neural networks do not generalize well to novel domains but usually require retraining of the network.

Here, we can combine the advantages of both approaches: By using a convolutional neural network that processes spline coefficients which can be considered as a discrete hidden latent description for a continuous implicit field description based on Hermite splines, our Spline-PINN approach is capable to generalize to new domain geometries and yields continuous solutions that avoid the detrimental effects of a discretized loss function based on finite differences.

We aim to optimize the spline coefficients such that the integrals of the squared residuals of the PDEs over the domain / domain-boundaries and time steps are minimized. To compute these integrals, we uniform randomly sample points within the given integration domains.

For the Navier-Stokes equation, we consider a momentum loss term L_p defined as:

$$L_p = \int_\Omega \int_{\hat{t}}^{\hat{t}+dt} ||\rho(\partial_t \vec{v} + \vec{v} \cdot \nabla \vec{v}) - \mu \Delta \vec{v} + \nabla p||^2 \quad (7)$$

It is important that the residuals are of bounded variation, otherwise we can not compute the integral. L_p contains third order derivatives for a_z in space (see viscosity term: $\mu\Delta(\nabla \times a_z)$) and thus, following the argument in Figure 1, we have to choose at least $l, m = 2$ for the Hermite spline kernels of a_z. The boundary loss term L_b is given by:

$$L_b = \int_{\partial\Omega} \int_{\hat{t}}^{\hat{t}+dt} ||\vec{v} - \vec{v}_d||^2 \quad (8)$$

L_p and L_b are then combined in the final loss term $L_{\text{tot}}^{\text{flow}}$ according to:

$$L_{\text{tot}}^{\text{flow}} = \alpha L_p + \beta L_b \quad (9)$$

Here, we set hyperparameters $\alpha = 10$ and $\beta = 20$. β was chosen higher compared to α to prevent the flow field from leaking through boundaries. A more thorough discussion regarding the choice of hyperparameters is provided in the supplementary material.

For the damped wave equation, we need two loss terms inside the domain Ω:

$$L_z = \int_\Omega \int_{\hat{t}}^{\hat{t}+dt} ||\partial_t z - v_z||^2 \quad (10)$$

$$L_v = \int_\Omega \int_{\hat{t}}^{\hat{t}+dt} ||\partial_t v_z - k\Delta z + \delta v_z||^2 \quad (11)$$

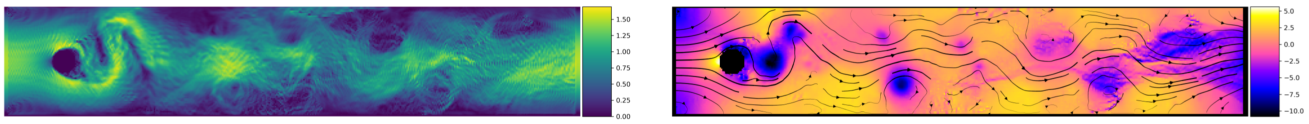


Figure 3: Severe discretization artifacts appear using a physics-constrained loss based on a finite differences Marker and Cell grid at $Re = 10000$ - even at twice the resolution compared to our Hermite spline approach.

Similar to the incompressible Navier-Stokes equation, we have second order derivatives of z in space (see $k\Delta z$-Term) and thus have to choose at least $l, m = 1$ for the Hermite spline kernels of z. The boundary loss term is:

$$L_b = \int_{\partial\Omega} \int_{\hat{t}}^{\hat{t}+dt} ||z - z_d||^2 \tag{12}$$

L_z, L_v and L_b are then combined in the final loss term $L_{\text{tot}}^{\text{wave}}$:

$$L_{\text{tot}}^{\text{wave}} = \alpha L_z + \beta L_v + \gamma L_b \tag{13}$$

For the wave equation, we set the hyperparameters $\alpha = 1, \beta = 0.1, \gamma = 10$.

Training Procedure As presented in Figure 4, we use a training procedure similar to [Wandel, Weinmann, and Klein 2021a] but replace the physics-constrained loss based on finite differences by the just introduced physics-informed loss on the spline coefficients. First, we initialize a training pool of randomized training domains and spline coefficients. At the beginning, all spline coefficients can be set to 0, thus no ground truth data is required. Then, we draw a random minibatch (batchsize = 50) from the training pool and feed it into the PDE Model, which predicts the spline coefficients for the next time step. Then, we compute a physics-informed loss inside the volume spanned by the spline coefficients of the minibatch and the predicted spline coefficients in order to optimize the weights of the PDE Model with gradient descent. To this end, we use the Adam optimizer (learning rate = 0.0001). Finally, we update the training pool with the just predicted spline coefficients in order to fill the pool with more and more realistic training data over the course of training. From time to time, we reset spline coefficients to 0 in the training pool in order to also learn the warm-up phase from 0 spline coefficients. Training took 1-2 days on a NVidia GeForce RTX 2080 Ti.

Results

In the following, we provide qualitative and quantitative evaluations of our approach for fluid and wave simulations. Additional results as well as demonstrations where the user can interact with the PDE by dynamically changing boundary conditions, are contained in our supplemental.

Fluid Simulation

Qualitative Evaluation The dynamics of a fluid depends strongly on the Reynolds number, which is a dimensionless quantity that relates the fluid density ρ, mean velocity $||\vec{v}||$, obstacle diameter L and viscosity μ in the following way:

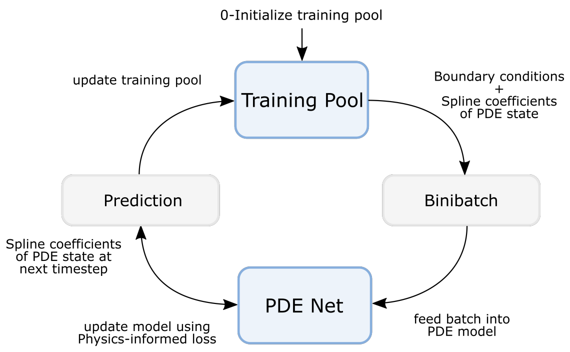

0-Initialize training pool
update training pool
Training Pool
Boundary conditions + Spline coefficients of PDE state
Prediction
Binibatch
Spline coefficients of PDE state at next timestep
PDE Net
update model using Physics-informed loss
feed batch into PDE model

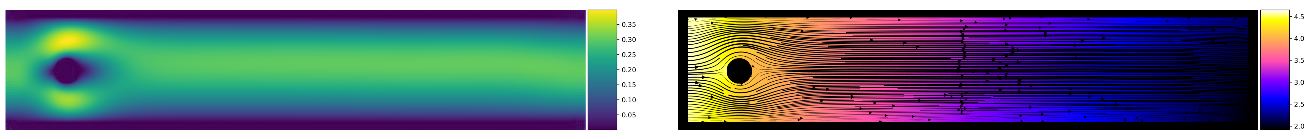
Figure 4: Training cycle similar to [Wandel, Weinmann, and Klein 2021a]

$$Re = \frac{\rho ||\vec{v}|| L}{\mu} \tag{14}$$

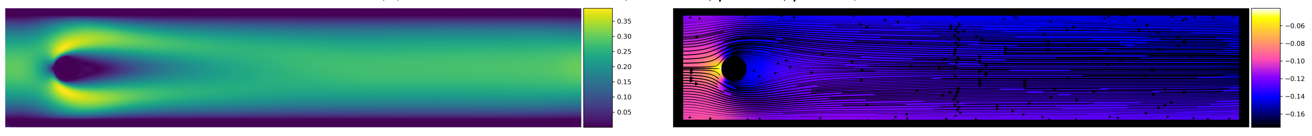
To validate our method, we reconstructed the DFG benchmark setup [Schäfer and Turek 1996] and scaled its size and viscosity by a factor of 100. This way, we obtained a grid size of 41 × 220 cells for the fluid domain and the obstacle diameter is 10 cells.

For very small Reynolds numbers, the flow field becomes basically time-reversible which can be recognized by the symmetry of the stream lines in Figure 5 a). At $Re = 20$ a steady wake is forming behind the obstacle (see Figure 5 b). For higher Reynolds numbers such as $Re = 100$, this wake becomes unstable and a von Karman vortex street is forming (see Figure 5 c). Figure 5 d) and e) show turbulent flow fields at very high Reynolds numbers ($Re = 1000$ and $Re = 10000$). Notice the high level of details in the velocity field learned by our method without any ground truth data.

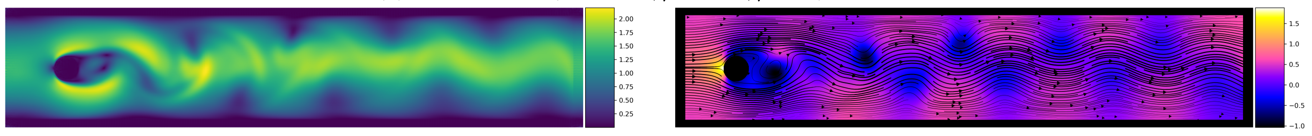
Quantitative Evaluation To evaluate our method quantitatively, we computed the forces exerted by the fluid on a cylinder (see Figure 6) and compared the resulting drag and lift coefficients to an implicit physics-informed neural network similar to [Raissi, Yazdani, and Karniadakis 2018], a MAC grid based physics-constrained neural network [Wandel, Weinmann, and Klein 2021a], an industrial CFD solver (Ansys) and official results reported on a CFD benchmark [Schäfer and Turek 1996]. To the best of our knowledge, this is the first time such forces were obtained by a fully unsupervised machine learning based approach. The forces can be separated into two terms stemming from viscous friction



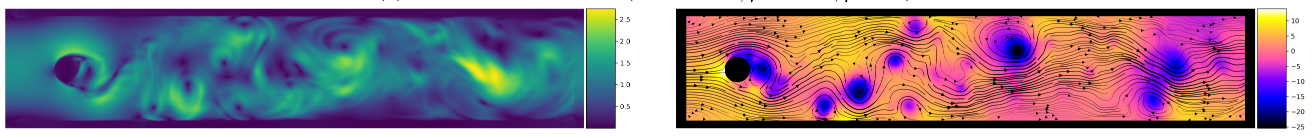
(a) time-reversible flow($Re = 2, \mu = 1, \rho = 1$)



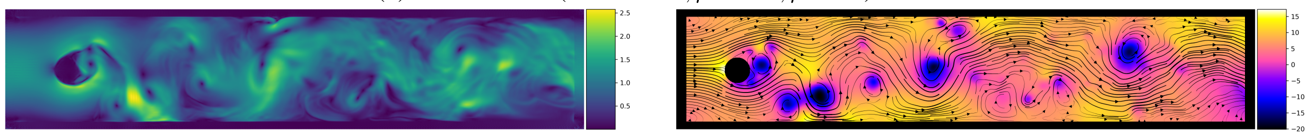
(b) laminar flow ($Re = 20, \mu = 0.1, \rho = 1$)



(c) laminar vortex street($Re = 100, \mu = 0.1, \rho = 1$)



(d) turbulent flow($Re = 1000, \mu = 0.1, \rho = 10$)



(e) turbulent flow ($Re = 10000, \mu = 0.01, \rho = 10$)

Figure 5: Flow and pressure fields around a cylinder obtained by our method at different Reynolds numbers. Left side: velocity magnitude; Right side: pressure field and stream lines of velocity field. An animated real-time visualization of these experiments is provided in the supplementary video (*www.youtube.com/watch?v=QC98LCtCZn0*).

($\vec{F}_\mu$) and the pressure field ($\vec{F}_p$):

$$\vec{F}_\mu = \int_S \mu(\nabla \vec{v})\vec{n}ds\,;\; \vec{F}_p = \int_S -p\vec{n}ds \tag{15}$$

$$\vec{F}_{\text{tot}} = \vec{F}_\mu + \vec{F}_p \tag{16}$$

Figure 6 shows the distribution of such viscous drag and pressure forces along the cylinder surface S with surface normals $\vec{n}$.

The drag-force F_D is the parallel force-component of $\vec{F}_{\text{tot}}$ to the flow direction ($F_D = \vec{F}_{\text{tot},x}$) while the lift force F_L is its orthogonal component ($F_L = \vec{F}_{\text{tot},y}$). From these forces, it is possible to compute drag and lift coefficients as follows:

$$C_D = \frac{2F_D}{U^2_{\text{mean}}L},\; C_L = \frac{2F_L}{U^2_{\text{mean}}L} \tag{17}$$

Tables 1 and 2 compare the drag and lift coefficients obtained by our method to an implicit PINN, a finite-difference based MAC grid approach [Wandel, Weinmann, and Klein 2021a], an industrial CFD solver (Ansys) and the official DFG benchmark [Schäfer and Turek 1996] for Reynolds numbers 2, 20 and 100. For all Reynolds numbers, the Spline-PINN approach returned significantly improved drag coefficients compared to the implicit PINN and

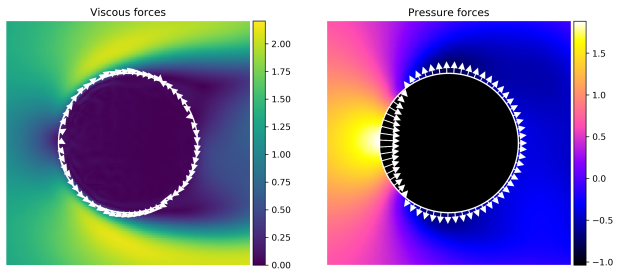


Figure 6: White arrows indicate viscous and pressure forces (see Equation 15) acting on a cylinder at $Re = 100$ (see Figure 5 c). While forces from viscous friction are parallel to the obstacle's surface, pressure forces are always perpendicular to the surface.

MAC-approach. For $Re = 100$, the implicit PINN fails to capture the dynamics of the von Karman vortex street as it is only trained on the domain boundaries without any additional information stemming for example from a moving die [Raissi, Yazdani, and Karniadakis 2018] or dynamic boundary conditions [Jin et al. 2021]. In contrast, our method is able to reproduce such oscillations as can be seen in Figure 7. The implicit PINN required retraining for every setting which took about 1 day while performing the computations with the MAC grid based approach or our Spline-PINN was achieved basically in real-time at 60 and 30 iterations per second respectively. The Ansys solver took 9 sec for $Re = 2$, 13 sec for $Re = 20$ and 37 min for $Re = 100$. A comparison of errors in $E[|\nabla \cdot \vec{v}|]$ and L_p as well as a stability analysis can be found in the supplemental.

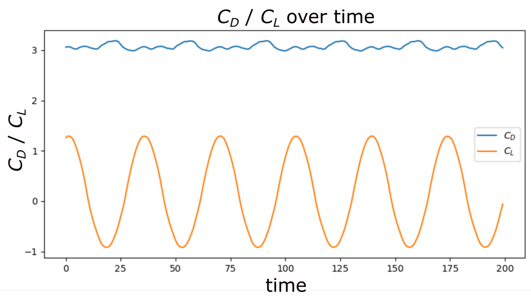


Figure 7: Oscillating drag and lift coefficients over time obtained by our Spline-PINN at $Re = 100$.

Wave Simulation

Qualitative Evaluation Figure 8 shows several experimental results that were obtained by our method for the damped wave equation on a 200 × 200 domain. First, we investigated interference patterns that arise when waves from different directions are superimposed. In Figure 8 a), the waves of 4 oscillators interfere with each other. Then, in Figure 8 b), we investigate whether our method is able to learn the Doppler effect for moving oscillators. This effect is well-known e.g. for changing the pitch of an ambulance-car that drives by. Finally, in Figure 8 c), we show the reflection and interference behavior of waves hitting Dirichlet boundaries.

As for the fluid simulations, all of these results were obtained without relying on any ground truth data. Furthermore, the domain in Figure 8 c) was not contained in the randomized set of training domains indicating good generalization performance.

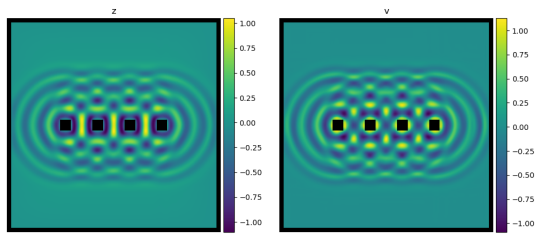
Quantitative Evaluation Table 3 compares the losses of our method on the oscillator domain (see Figure 8 a). We trained two versions with two different spline orders in the spatial dimensions ($l, m = 1$ and $l, m = 2$) and observed significantly better performance for $l, m = 2$. The stability of our approach for the wave equation is examined in the supplementary material.

Conclusion

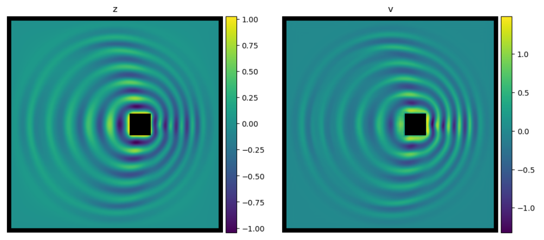
In this work, we approach the incompressible Navier-Stokes equation and the damped wave equation by training a continuous Hermite Spline CNN using physics-informed loss functions only. While finite-difference based methods break down at high Reynolds-numbers due to discretization artifacts, our method still returns visually appealing results at $Re = 10000$. Furthermore, we investigated drag and lift coefficients on a CFD benchmark setup and observed reasonable accordance with officially reported values which is remarkable given the fact that our method does not rely on any ground truth data.

In the future, further boundary conditions (e.g. von Neumann BC) could be incorporated into the PDE model as well. To further refine the solutions at the boundary layers, a multigrid Hermite spline CNN could be considered. The fully differentiable nature of our method may also help in reinforcement learning scenarios, optimal control, sensitivity analysis or gradient based shape optimization.

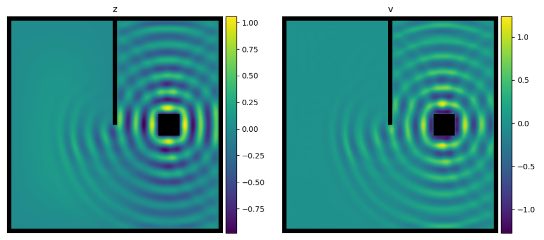
We believe that our method could have applications in physics engines of computer games or in computer-generated imagery as it provides fast and visually plausible solutions. The obtained drag and lift coefficients indicate that in the future, unsupervised ML based methods could reach levels of accuracy that are sufficiently close to traditional industrial CFD solvers making them suitable for fast prototyping in engineering applications. We firmly believe



(a) Interference patterns forming around 4 oscillators.



(b) Doppler effect of an oscillator moving to the right.



(c) Wave reflections on domain boundaries.

Figure 8: Results of our method for the wave equation ($k = 10, \delta = 0.1$). Left side: height field z; Right side: velocity field v_z. The domain boundaries are marked in black. For better performance, second-order splines were used in x and y. An animated real-time visualization of these experiments can be found at *www.youtube.com/watch?v=QC98LCtCZn0.*

Method	Re=2		Re=20	
	C_D	C_L	C_D	C_L
implicit PINN	25.3	0.478	3.299	0.0744
MAC grid [Wandel, Weinmann, and Klein 2021a]	25.76	-0.824	4.414	-0.597
Spline-PINN (ours)	29.7	-0.456	4.7	5.64e-04
Ansys	32.035	0.774	5.57020	0.00786
DFG-Benchmark [Schäfer and Turek 1996]	-	-	5.58	0.0106

Table 1: Drag and lift coefficients obtained by an implicit PINN, our Hermite spline approach, an industrial CFD solver (Ansys) and official results from the DFG-benchmark [Schäfer and Turek 1996] for $Re = 2, 20$.

Method	Re=100		
	C_D	C_L	time
implicit PINN	1.853	-0.02445	$\sim$ 1 day
MAC grid [Wandel, Weinmann, and Klein 2021a]	(2.655 / 2.693 / 2.725)*	(-0.757 / 0.0184 / 0.86)*	$\sim$ 15 sec
Spline-PINN (ours)	(2.985 / 3.068 / 3.188)*	(-0.926 / 0.179 / 1.295)*	$\sim$ 10 sec
Ansys	(3.234 / 3.273 / 3.31)*	(-1.14 / -0.059 / 1.07)*	37 min
DFG-Benchmark [Schäfer and Turek 1996]	(3.1569 / 3.1884 / 3.220)*	(-1.0206 / -0.0173 / 0.9859)*	-

Table 2: Drag and lift coefficients obtained by an implicit PINN, our Hermite spline approach, an industrial CFD solver (Ansys) and official results from the DFG-benchmark [Schäfer and Turek 1996] for $Re = 100$. *:(minimum/average/maximum)-values for oscillating coefficients

Spline order	L_z	L_v	L_b
$l, m = 1$	8.511e-02	1.127e-02	1.425e-03
$l, m = 2$	5.294e-02	6.756e-03	1.356e-03

Table 3: Quantitative results of wave equation.

that moving from explicit physics-constrained losses to implicit physics-informed losses on continuous fields based on discrete latent descriptions such as spline coefficients will positively influence the performance of future ML based PDE solvers that generalize.

Acknowledgements

This work has been supported by the German Research Foundation (DFG), FOR 2535 Anticipating Human Behavior.

References

Ainsworth, M.; and Dong, J. 2021. Galerkin Neural Networks: A Framework for Approximating Variational Equations with Error Control. *arXiv preprint arXiv:2105.14094*.

Chen, S.; and Doolen, G. D. 1998. Lattice Boltzmann method for fluid flows. *Annual review of fluid mechanics*, 30(1): 329–364.

Fey, M.; Lenssen, J. E.; Weichert, F.; and Müller, H. 2017. SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels. *CoRR*, abs/1711.08920.

Foster, N.; and Metaxas, D. 1996. Realistic Animation of Liquids. *Graphical Models and Image Processing*, 58(5): 471 – 483.

Gao, H.; Zahr, M. J.; and Wang, J. 2021. Physics-informed graph neural Galerkin networks: A unified framework for solving PDE-governed forward and inverse problems. *CoRR*, abs/2107.12146.

Geneva, N.; and Zabaras, N. 2019. Quantifying model form uncertainty in Reynolds-averaged turbulence models with Bayesian deep neural networks. *Journal of Computational Physics*, 383: 125 – 147.

Geneva, N.; and Zabaras, N. 2020. Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*, 403: 109056.

Gingold, R. A.; and Monaghan, J. J. 1977. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3): 375–389.

Grohs, P.; Hornung, F.; Jentzen, A.; and Von Wurstemberger, P. 2018. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *arXiv preprint arXiv:1809.02362*.

Guo, Z. 2021. Well-balanced lattice Boltzmann model for two-phase systems. *Physics of Fluids*, 33(3): 031709.

Harlow, F. H.; and Welch, J. E. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12): 2182–2189.

Harsch, L.; and Riedelbauch, S. 2021. Direct Prediction of Steady-State Flow Fields in Meshed Domain with Graph Networks. arXiv:2105.02575.

Jin, X.; Cai, S.; Li, H.; and Karniadakis, G. E. 2021. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 426: 109951.

Khoo, Y.; Lu, J.; and Ying, L. 2019. Solving for high-dimensional committor functions using artificial neural networks. *Research in the Mathematical Sciences*, 6(1): 1.

Kim, B.; Azevedo, V. C.; Thuerey, N.; Kim, T.; Gross, M.; and Solenthaler, B. 2019. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, 59–70. Wiley Online Library.

Kim, J.; and Lee, C. 2020. Deep unsupervised learning of turbulence for inflow generation at various Reynolds numbers. *Journal of Computational Physics*, 406: 109216.

Ladický, L.; Jeong, S.; Solenthaler, B.; Pollefeys, M.; and Gross, M. 2015. Data-Driven Fluid Simulations Using Regression Forests. *ACM Trans. Graph.*, 34(6).

Li, Y.; Wu, J.; Tedrake, R.; Tenenbaum, J. B.; and Torralba, A. 2019. Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids. In *ICLR*.

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

Lu, L.; Meng, X.; Mao, Z.; and Karniadakis, G. E. 2021. DeepXDE: A Deep Learning Library for Solving Differential Equations. *SIAM Review*, 63(1): 208–228.

Mohan, A. T.; Lubbers, N.; Livescu, D.; and Chertkov, M. 2020. Embedding Hard Physical Constraints in Neural Network Coarse-Graining of 3D Turbulence. arXiv:2002.00021.

Mrowca, D.; Zhuang, C.; Wang, E.; Haber, N.; Fei-Fei, L.; Tenenbaum, J. B.; and Yamins, D. L. K. 2018. Flexible Neural Representation for Physics Prediction. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 8813–8824. Red Hook, NY, USA: Curran Associates Inc.

Pfaff, T.; Fortunato, M.; Sanchez-Gonzalez, A.; and Battaglia, P. W. 2021. Learning Mesh-Based Simulation with Graph Networks. ICLR 2021:2010.03409.

Raissi, M.; Perdikaris, P.; and Karniadakis, G. E. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378: 686 – 707.

Raissi, M.; Yazdani, A.; and Karniadakis, G. E. 2018. Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data. *arXiv preprint arXiv:1808.04327*.

Rasht-Behesht, M.; Huber, C.; Shukla, K.; and Karniadakis, G. E. 2021. Physics-informed Neural Networks (PINNs) for Wave Propagation and Full Waveform Inversions. arXiv:2108.12035.

Sanchez-Gonzalez, A.; Godwin, J.; Pfaff, T.; Ying, R.; Leskovec, J.; and Battaglia, P. W. 2020. Learning to Simulate Complex Physics with Graph Networks. arXiv:2002.09405.

Schenck, C.; and Fox, D. 2018. SPNets: Differentiable Fluid Dynamics for Deep Neural Networks. In *Conference on Robot Learning*, 317–335.

Schäfer, M.; and Turek, S. 1996. Benchmark Computations of Laminar Flow Around a Cylinder (The CFD Benchmarking Project). http://www.mathematik.tu-dortmund.de/ featflow/en/benchmarks/cfdbenchmarking.html. (accessed at 08-Sep-2021).

Schöberl, M.; Zabaras, N.; and Koutsourelakis, P.-S. 2019. Predictive collective variable discovery with deep Bayesian models. *The Journal of Chemical Physics*, 150(2): 024109.

Sirignano, J.; and Spiliopoulos, K. 2018a. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375: 1339 – 1364.

Sirignano, J.; and Spiliopoulos, K. 2018b. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375: 1339–1364.

Sitzmann, V.; Martel, J.; Bergman, A.; Lindell, D.; and Wetzstein, G. 2020. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33.

Stam, J. 1999. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 121–128.

Thuerey, N.; Weißenow, K.; Prantl, L.; and Hu, X. 2019. Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows. *AIAA Journal*, 1–12.

Tompson, J.; Schlachter, K.; Sprechmann, P.; and Perlin, K. 2017. Accelerating eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 3424–3433. JMLR. org.

Tripathy, R. K.; and Bilionis, I. 2018. Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics*, 375: 565 – 588.

Um, K.; Fei, R.; Holl, P.; Brand, R.; and Thuerey, N. 2020. Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers. arXiv:2007.00016.

Ummenhofer, B.; Prantl, L.; Thuerey, N.; and Koltun, V. 2020. Lagrangian Fluid Simulation with Continuous Convolutions. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Wandel, N.; Weinmann, M.; and Klein, R. 2021a. Learning Incompressible Fluid Dynamics from Scratch - Towards Fast, Differentiable Fluid Models that Generalize. *9th International Conference on Learning Representations, ICLR 2021*.

Wandel, N.; Weinmann, M.; and Klein, R. 2021b. Teaching the Incompressible Navier-Stokes Equations to Fast Neural Surrogate Models in 3D. *Physics of Fluids (AIP)*.

Wang, H.; Planas, R.; Chandramowlishwaran, A.; and Bostanabad, R. 2021. Train Once and Use Forever: Solving Boundary Value Problems in Unseen Domains with Pre-trained Deep Learning Models. *CoRR*, abs/2104.10873.

Yang, C.; Yang, X.; and Xiao, X. 2016. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4): 415–424.

Zhu, Y.; and Zabaras, N. 2018. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366: 415 – 447.

Zhu, Y.; Zabaras, N.; Koutsourelakis, P.-S.; and Perdikaris, P. 2019. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394: 56 – 81.

Supplementary

Magnus effect and generalization examples

Figure 3 a) demonstrates that our Spline-PINN approach is capable of handling the Magnus effect. The Magnus effect appears e.g. on rotating bodies within a stream and leads to a characteristic low pressure field on the side that moves in the same direction as the flow field. This effect is often exploited in ballsports such as soccer (banana cick) or tennis (topspin) to deflect the path of a ball. Figure 3 b) shows a generalization example of the Spline-PINN. During training, we confronted the Spline-PINN with randomized environments that contained at most one obstacle. Nevertheless, our approach is able to generalize to multiple obstacles as shown here by the smiley face.

Video - interactive demo

In our supplementary video, we present dynamic simulations obtained by our Spline-PINN method. All simulations were done in real time on a NVidia GeForce RTX 2080 Ti. To allow for dynamic interactions, we allow the user to dynamically paint new boundaries into the fluid domain during simulation. This proves the high generalization capabilities of our method to scenarios not considered during training. Furthermore, we show dynamic solutions for the wave equation.

Stability of Spline-PINNs

Here, we evaluate the stability of our approach at the example of the incompressible Navier-Stokes equations and the damped wave equation.

Incompressible Navier-Stokes equation We investigated the stability of Spline-PINNs on the incompressible Navier-Stokes equations over hundreds of iterations on the DFG benchmark [1] problem at $Re = 100$. As can be seen in Figure 1, after a short warm-up phase of around 50 iterations, in which the network has to start up the simulation from 0 pressure and velocity fields, our method is able to provide stable results over long time horizons.

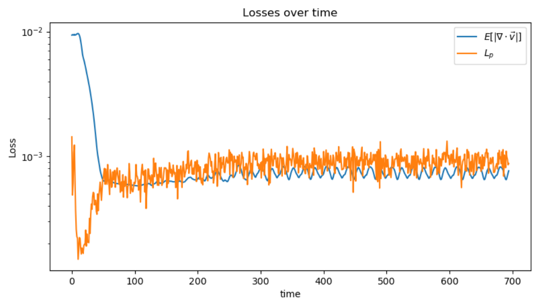


Figure 1: Stability of Spline Net while solving the Navier Stokes equation on the DFG benchmark problem at $Re = 100$.

Damped wave equation As can be seen in Figure 2, our method delivers stable results for the damped wave equation on the interference problem setup (see Figure 8 a) in paper). In contrast to the loss curves for the Navier Stokes equation (see Figure 1), there is no significant warmup phase as 0 spline coefficients already fulfill the boundary conditions and wave equation at the beginning of the simulation very well.

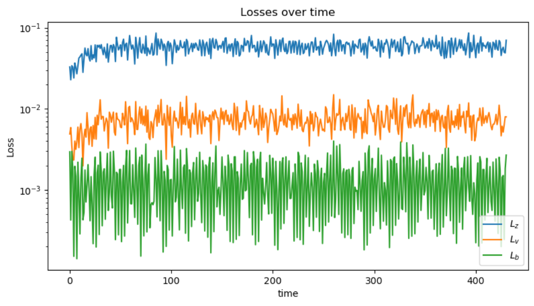


Figure 2: Stability of solution for wave equation ($l, m = 2$) on the interference problem setup.

Neural network architecture

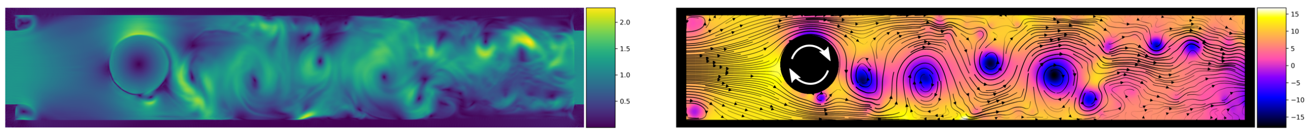
In Figure 4, we present the network architectures that we used for the incompressible Navier-Stokes equation and the damped wave equation. Ω^t contains the occupancy grid of the domain at a timepoint t and $\vec{v}_d^t / z_d^t$ contain respective Dirichlet boundary conditions. For the Navier-Stokes equation, we internally rely on a U-Net architecture [2] while for the damped wave equation, a simple 3-layer CNN was already sufficient. These networks were used to compute a residual ($\Delta c_{i,j}$) to be added to $c_{i,j}^t$ in order to obtain the spline coefficients of the next time-step $c_{i,j}^{t+dt}$. In the case of the Navier-Stokes equations, the vector potential a_z as well as the pressure field p are only defined up to a constant. Thus, we mean-normalize the coefficients of the first Spline-mode for these fields to zero ($\sum_{\hat{x},\hat{y}} c_{\hat{x},\hat{y},\hat{t}}^{0,0,0} = 0$).

Basis flow fields

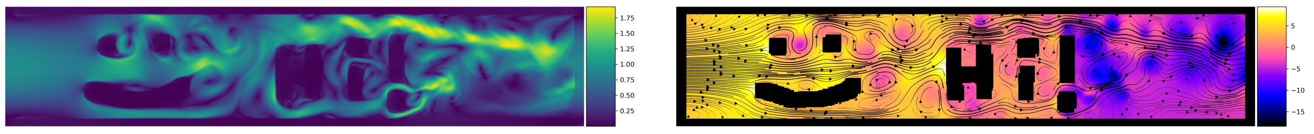
Figure 5 shows how flow fields can be obtained by taking the curl of individual Hermite spline kernel functions for the vector potential a_z. These vector fields could be considered as basis flow fields for the velocity field $\vec{v}$.

Choice of hyperparameters α, β, γ

We investigated the sensitivity of our method with respect to different loss weights in Equation 9. Figure 6 shows how different ratios between α and β affect the losses for the momentum equation L_p and boundary conditions L_b after training. If we put too little weight on the fluid domain (α), the accuracy of the momentum equation drops, while if we put too little weight on the boundaries (β), we get unrealistic flows leaking through the boundary. Tuning these parameters is fairly intuitive and depending on the application, one



(a) Magnus effect on clockwise turning cylinder ($\mu = 0.01, \rho = 10$)



(b) Generalization example (drawing of smiley) ($\mu = 0.1, \rho = 10$)

Figure 3: Flow and pressure fields for Magnus effect and a generalization example. Left side: velocity magnitude; Right side: pressure field and stream lines of velocity field.

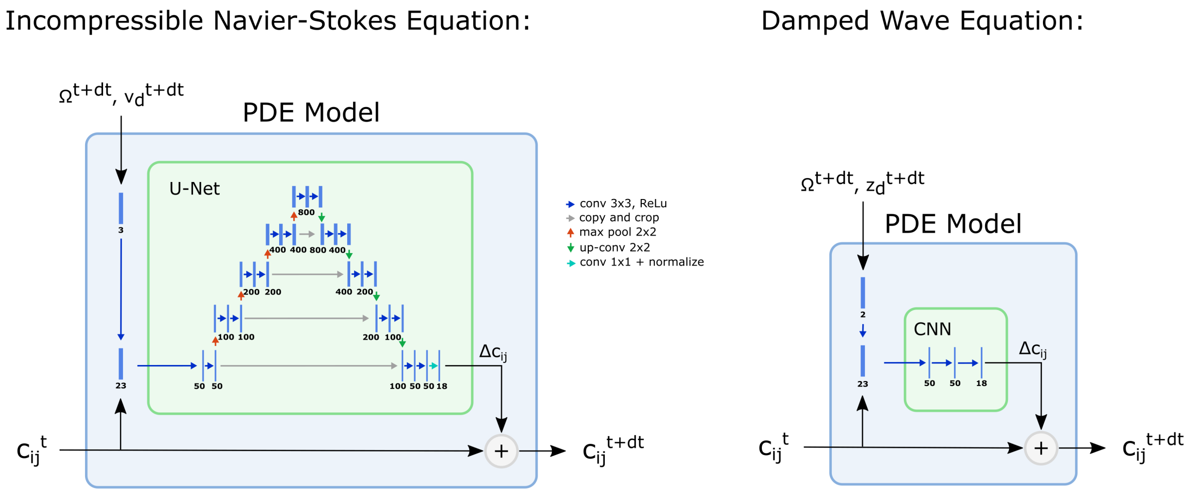


Figure 4: Network architectures used for the incompressible Navier-Stokes equation (left) and damped wave equation (right). The numbers located below the blue bars indicate the number of channels of the individual layers.

can choose a corresponding sweet-spot. For the wave equation, one can proceed similarly. Too little weight on α and β decreases the performance of the Spline CNN within Ω, while too little weight on the boundaries (γ) leads to inferior adherence to the boundary conditions. In the future, more elaborate methods such as proposed e.g. by Wang et al. [3] could be used to automate the process of finding proper loss weights.

Loss values for Navier-Stokes equation

We compared $E[|\nabla \cdot \vec{v}|]$ and L_p of an implicit physics-informed neural network, a MAC grid based physics-constrained neural network [4], our Hermite Spline approach and an industrial CFD solver (Ansys) on the DFG Fluid-Benchmark at $Re = 2, Re = 20$ and $Re = 100$. All machine learning based approaches were trained without any ground truth data but only using a physics-based loss and the boundary conditions of the benchmark setup [1]. Since a vector potential is used for the ML-based methods, we computed $E[|\nabla \cdot \vec{v}|]$ by measuring the flow leaking through the boundaries as an effect of not perfectly matching the boundary conditions. This corresponds to a thin layer that does not preserve the incompressibility equation. The losses on the MAC grid were computed using finite differences and the loss for Ansys was computed based on the field-values and derivatives provided by Ansys.

While the implicit PINN provides a very high accuracy for the momentum equation (see low L_p values), it fails to achieve similar performance on matching the boundary conditions (see high $E[|\nabla \cdot \vec{v}|]$ values). The Marker-and-Cell grid based approach delivers the lowest loss values. However, these losses do not consider the inaccuracies delivered by the finite-difference approximations within the MAC grid itself. The high losses for the Ansys solver could stem from the fact that solutions are internally not optimized in strong form but based on Galerkin / Finite-Elements methods and

	Re=2		Re=20		Re=100	
Method	$E[\lvert\nabla \cdot \vec{v}\rvert]$	L_p	$E[\lvert\nabla \cdot \vec{v}\rvert]$	L_p	$E[\lvert\nabla \cdot \vec{v}\rvert]$	L_p
implicit PINN	3.692e-04	5.543e-08	2.371e-04	1.142e-07	1.855e-03	1.465e-06
MAC grid	6.039e-05	1.986e-05	8.99e-06	3.436e-07	5.796e-05	4.358e-06
Spline Net	8.32e-05	3.847e-03	1.07e-04	1.92e-05	7.492e-04	9.04e-04
Ansys	1.46e-04	1.511e-02	1.072e-05	2.348e-05	8.912e-04	2.038e-02

Table 1: Quantitative results for implicit PINN, MAC grid, Spline Net and Ansys. Note: A direct comparison of these loss values is difficult as the losses had to be computed in different ways: The values for the implicit PINN and Spline Net were computed based on physics-informed losses while the losses for the MAC grid method were based on physics-constrained losses and the values of the Ansys-solver were computed based on the outputs on the underlying mesh representation. Thus, we argue that our results on drag and lift coefficients (see Table 1 and 2) are more expressive in terms of overall performance.

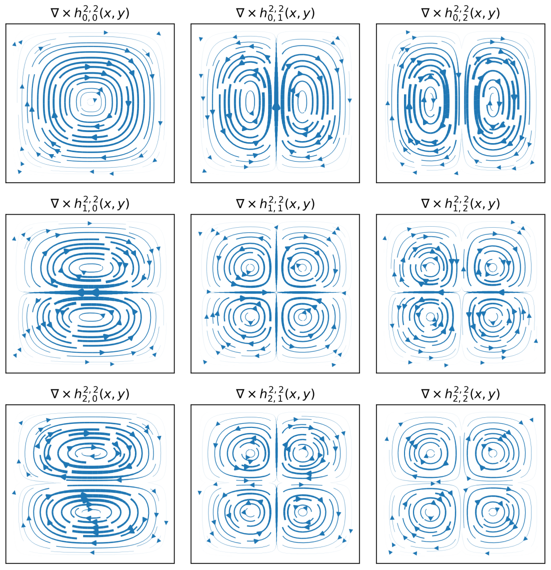


Figure 5: Curl of 2D Hermite spline kernels with $l, m = 2$. Note that these "basis flow fields" all conserve $\nabla \cdot \vec{v} = 0$, since $\nabla \cdot \nabla \times a_z = 0$. Thus the total flow field is incompressible as well.

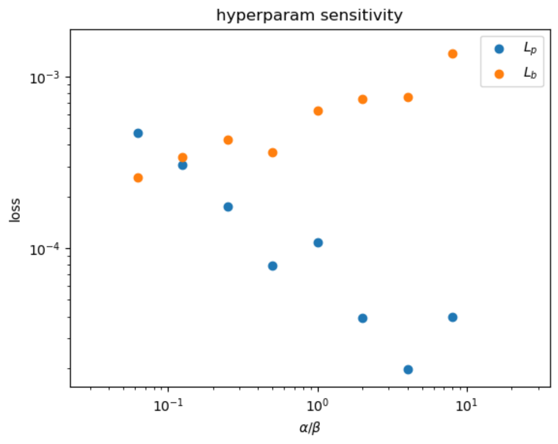


Figure 6: Impact of different ratios between α and β in $L^{\text{flow}}_{\text{tot}}$ on L_p and L_b after training (see Equation 9).

that the provided derivatives might not be as accurate. We conclude that a comparison of such loss values is very difficult and one has to be very careful with drawing conclusions - especially on a Marker and Cell grid as the grid itself might introduce errors due to finite difference approximations. Furthermore, local loss values do not translate directly into physical accuracy for more global phenomena such as drag and lift forces.

Acknowledgement

This work has been supported by the German Research Foundation (DFG), FOR 2535 Anticipating Human Behavior.

References

[1] The cfd benchmarking project. http://www.mathematik.tu-dortmund.de/ feat-flow/en/benchmarks/cfdbenchmarking.html, 2021. (accessed at 08-Sep-2021).

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[3] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020.

[4] Nils Wandel, Michael Weinmann, and Reinhard Klein. Learning incompressible fluid dynamics from scratch - towards fast, differentiable fluid models that generalize. *9th International Conference on Learning Representations, ICLR 2021*, 2021.