TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Machine Learning Accelerators for 3D Physics Simulations

Authors	Yi-Han Hsieh, Aman Kumar, Lennart Röstel, Saad Shamsi
Mentor(s)	Dr. Dirk Hartmann, Siemens AG
	Theo Papadopoulos, MSc. Siemens AG
	Mohamed Khalil, MSc. Siemens AG
Co-Mentor	Laure Vuaille
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Feb 2021

Abstract

Accurately predicting the behavior of fluids is fundamental to a variety of scientific and engineering tasks. Capturing all physical phenomena on small scales is, however, notoriously difficult and requires large computational effort. Advancements in the field of Computational Fluid Dynamics (CFD) therefore aim to increase the accuracy of existing solver methods by modelling the effects of unresolved physical processes on a larger scale. The goal of this project was to develop methods that enable *learning* such correction models from data. Specifically, in this work, we explore a family of correction models that is tightly coupled with classical numerical CFD solver schemes.

On the one hand, the learned model intervenes in each solver step, leading to an improvement in accuracy over the uncorrected solution. On the other hand, error signals are propagated *through the solver*, providing the model with information about the physical evolution of the system.

For this purpose we implement a pipeline that is able to train these kinds of correction models from an external data source. In the course of that, we empirically assess the suitability of two different numerical solver schemes in the context of augmenting them by learned corrections. To represent the corrections, we employ local variants of fully convolutional neural networks, forcing the model to base its predictions only on local features of the flow field. Our experiments show that this method leads to promising generalization properties.

We point out the practical challenges associated with learning models that tightly interact with classical solver schemes in a recurrent manner. Based on our findings, we develop requirements on datasets and solver methods for future work in this direction.

Contents

1 Introduction

Computational Fluid Dynamics (CFD) simulations are an essential tool in a multitude of scientific and engineering disciplines. Advances in high-performance computing have made CFD an indispensable tool for rapid prototyping, design analysis, and material testing in academic and industrial applications. The most prominent uses include: **aerodynamic analysis** in the automotive and aerospace industries, **reaction flow and thermal management** for combustion engines, and **channel flow simulations**.
Typically, CFD simulations require a fundamental trade-off:

- Lower resolution simulations (with fewer degrees of freedom) are unable to capture micro-scale processes, and can miss out their cumulative effects on the macro-scale.
- Higher resolution simulations are able to capture the cumulative effects of micro-scale phenomena, but are computationally expensive and time consuming.

An ideal simulation should thus be detailed enough to capture the required physical details for the desired application whilst being computationally inexpensive and time efficient.

1.1 Problem Definition and Goals of the Project

The crux of this project was to develop a CFD pipeline that operates on coarse discretisations but is able to capture the cumulative effects of micro-scale phenomena on the macro-scale picture whilst maintaining computational cost.
Our task was to achieve this by employing a differentiable solver pipeline, where the intermediate computational steps of the solver are subject to a learned correction, represented by deep neural networks (DNN). Thus, the differentiable CFD solver would enable us to incorporate future information about the dynamics of the system into the training of the corrections, leading to potentially more accurate predictions than in the purely supervised setting [22]. Another project objective was therefore to assess the suitability of different types of solvers for this approach and to report on the practical implications and obstacles involved in the use of varying methodologies.
The dataset provided by Siemens consisted of steady-state flow fields for 5,005 different channel geometries (i.e. boundary conditions) with 4 different sets of initial conditions for each channel. These simulations were produced by the state-of-the art CFD software **Simcenter STAR-CCM+** [20] developed by Siemens Digital Industries Software.

1.2 Related Work

With the rise of deep learning (DL), researchers from the CFD and machine learning communities are trying to leverage DL methods in order to accelerate CFD simulations. One line of work employs so-called super resolution approaches, a group of methods that has been very successful in the image domain. [3] employ super-resolution in fluid simulations in order to reconstruct small scale turbulent features from down-scaled, high-resolution simulations. In engineering applications however, the detailed features of the flow field are typically of less interest. Instead, a different line of research aims to learn the discretization of the solver scheme itself in order to increase the accuracy of predictions on the larger scale [8]. The approach we take directly builds upon the works of [22] and [6], which propose methods in which the solver schemes closely interacts with a learned correction model; the *"solver in the loop"* approach enables gradients to flow through the solver, providing the model with information about the physical evolution of the system. Unlike DNN architectures employed in the previously mentioned works, we restrict the input of our DNNs to a local neighbourhood of the flow field. Furthermore, whereas [22] deal with *transient* simulations, our model is only allowed to learn from steady-state data. This is because the stationary case is often of greater importance in engineering tasks.

This report is structured as follows: In section 2 we introduce the Navier-Stokes equations and the idea of turbulence modelling. Based on that: in section 3 we explain our approach to learning correction models. In sections 4 and 5 we discuss aspects of data preparation and setup of the plane simulations respectively. Section 6 focuses on the implementation of the pipeline for training correction models. In section 7 we present our experimental results and discuss the challenges faced when training these models in section 8.

2 Theoretical Background

The Navier-Stokes equations are non-linear PDEs that govern the dynamics of viscous fluids. The equations are essentially statements of conservation laws for viscous fluids.

2.1 Derivation of Navier-Stokes Equations

Conservation of mass for a fluid can be expressed in the form of a continuity equation:

$$\frac{\partial \rho}{\partial t} - \rho \nabla \cdot \boldsymbol{v} = 0.$$

Since the density ρ for an incompressible fluid is constant, we obtain the first equation:

$$\nabla \cdot \boldsymbol{v} = 0 \tag{1}$$

The conservation of momentum equation can be obtained by expressing Newton's second law in terms of intensive quantities equated to body forces:

$$\rho \frac{d}{dt} \left(\boldsymbol{v} \left(x, y, z, t \right) \right) = \boldsymbol{b}.$$

Applying the chain rule, and assuming that the velocity-field $\boldsymbol{v} = (v_x, v_y, v_z) = \left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt} \right)$ we arrive at the momentum conservation equation:

$$\rho \left(\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v} \cdot \nabla \boldsymbol{v} \right) = \boldsymbol{b}. \tag{2}$$

The body force can be exressed as a sum of internal and external forces. In the case of an incompressible *Newtonian fluid*, we have $\boldsymbol{b} = -\nabla p + \mu \nabla^2 \boldsymbol{v} + \boldsymbol{f}$, where $\boldsymbol{f}$ represents the external forces. The Navier-Stokes equations for an incompressible Newtonian fluid are:

$$\nabla \cdot \boldsymbol{v} = 0 \tag{3}$$

$$\rho \left(\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v} \cdot \nabla \boldsymbol{v} \right) = -\nabla p + \mu \nabla^2 \boldsymbol{v} + \boldsymbol{f} \tag{4}$$

2.2 Turbulence Modelling and Closure

The Navier-Stokes equations constrain the velocity and pressure of fluid flow. Turbulence is indicative of a loss of regularity in fluid flow, and is thus characterised by *chaotic* changes in the the pressure and flow velocity. Consequently, turbulence problems are typically studied within a **stochastic framework**. Turbulent flows are characterised by:

- aperiodic motion,
- random (3D) spatial variations,
- strong dependence upon initial data,
- wide range of relevant length scales.

A characteristic feature of turbulent flow is the emergence of structure at different length scales. We therefore cast the Navier-Stokes equations into a dimensionless form to obtain a scale independent model. This dimensionless model then contains one parameter:

$$Re = \frac{vL}{\nu}. \tag{5}$$

The **Reynolds number** Re is comprised of the speed of the flow $v = |\boldsymbol{v}|$, the characteristic linear dimension (e.g. the diameter of a pipe) L and the kinematic viscosity $\nu = \frac{\mu}{\rho}$. Whereas the maximal scale of fluid motion is set by the overall geometry (e.g. the diameter of a smoke stack or water pipe), the minimal scale is determined by the Reynolds number. As apparent from the equation above, the Reynolds number represents the relative influence of inertial versus viscous forces on the motion of the fluid, and thus determines the relative importance of convection and diffusion mechanisms governing the flow of the fluid. Increasing the Reynolds number increases the influence of smaller scale structure on the flow characteristics. Flows with a high Reynolds number are thus turbulent.
The objective of direct numerical simulation is to solve the time-dependent Navier-Stokes equations resolving all relevant length scales for a sufficient time interval such that the fluid properties reach a statistical equilibrium. **Therefore, direct numerical simulations of turbulent flows need to be resolved to a greater precision (finer meshing) in order to obtain an accurate picture and are thus computationally costly**. In engineering and design applications, wherein often the finest spatio-temporal details are irrelevant, a *time-averaged* and *coarse-scale* approach is sufficient.
In order to extract coarse-scale time-averaged features, we reformulate the time-dependent Navier-Stokes equations in terms of Reynolds-Averaged quantities. The flow velocity is decomposed into a time-averaged component $\bar{\boldsymbol{v}}$ and a fluctuating component $\breve{\boldsymbol{v}}$:

$$\boldsymbol{v}(\boldsymbol{x}, t) = \bar{\boldsymbol{v}}(\boldsymbol{x}) + \breve{\boldsymbol{v}}(\boldsymbol{x}, t) \tag{6}$$

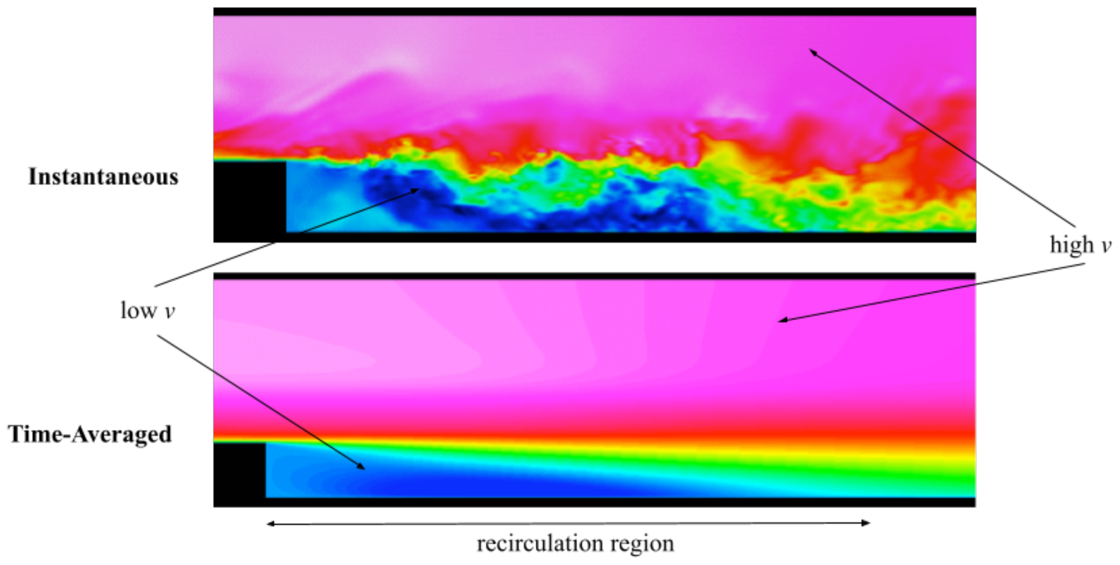


Figure 1: Instantaneous and Time-Averaged flows over a backstep. Only the recirculation region is of interest in engineering applications [14].

This then yields the **Reynolds-Averaged Navier Stokes** equations (in tensor notation):

$$\frac{\partial \bar{v}_i}{\partial x_i} = 0 \tag{7}$$

$$\frac{\partial \bar{v}_i}{\partial t} + \bar{v}_j \frac{\partial \bar{v}_i}{\partial x_j} = \bar{f}_i - \frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial^2 \bar{v}_i}{\partial x_j \partial x_j} - \frac{\partial \overline{\breve{v}_i \breve{v}_j}}{\partial x_j} \tag{8}$$

Note that the velocity fluctuations still appear in the RANS equations in the (non-linear) $-\overline{\breve{v}_i \breve{v}_j}$ term from the convective acceleration. This is a consequence of the nonlinearity of the Navier-Stokes equations. This term $R_{ij} = -\overline{\breve{v}_i \breve{v}_j}$ is known as the Reynolds stress since its effect on the mean flow is like that of a stress term. In order to obtain equations containing only time-averaged quantities, we need to *close* the RANS equations by modelling R_{ij} exclusively in terms of functions of time-averaged quantities. This is the **closure problem**, and lies at the heart of turbulence modelling.

3 General Approach

To compute numerical solutions to the Navier-Stokes equations, the physical domain has to be discretised in the spatial and temporal dimensions. The discretisation introduces numerical errors which are directly proportional to the *coarseness* of the discretisation. We aim to learn *correction funcionals* that operate on coarse spatial discretisations of the flow field to reduce the numerical error over the naive solution. The error itself is computed with respect to a simulation employing a *finer* spatial discretisation.

3.1 Notation

Given an initial velocity-field, $\boldsymbol{v}^{(0)}$ at time $t = 0$, on $\Omega \subset \mathbb{R}^2$ with prescribed boundary conditions, the forward-stepping scheme is:

$$\boldsymbol{v}^{(n_t+1)} = \boldsymbol{\mathcal{F}}(\boldsymbol{v}^{(n_t)}, \Delta t, \theta), \tag{9}$$

where $\boldsymbol{v}^{(n_t)}$ denotes the velocity field after n_t time steps of size Δt each. The number of timesteps $n_t = \frac{t}{\Delta t}$ is a function of the elapsed time t and the (constant) step-size Δt. Therefore $\boldsymbol{\mathcal{F}}(\boldsymbol{v}^{(n_t)}, \Delta t, \theta)$ denotes the propagation of the solution in time by the CFD solver, subject to corrections parametrized by θ (see 6.1 for the forms the correction can take). For all further considerations we will ommit the dependency on Δt and without loss of generality assume that Δt is constant.
After recursively applying the timestepping scheme T times, we obtain the solution of the velocity field at time $T\Delta t$ as:

$$\begin{aligned} \boldsymbol{v}^{(T)} &= \boldsymbol{\mathcal{F}}_T \circ \boldsymbol{\mathcal{F}}_{T-1} \circ ... \circ \boldsymbol{\mathcal{F}}(\boldsymbol{v}^{(0)}, \theta) \\ &:= \tilde{\boldsymbol{\mathcal{F}}}(\boldsymbol{v}^{(0)}, \theta) \end{aligned} \tag{10}$$

where we introduced the shorthand notation $\boldsymbol{\mathcal{F}}_{n_t}$ for the n_t-th application of the solver.

3.2 Optimization Objective

Suppose we are given a target velocity field $\hat{\boldsymbol{v}}$ with the same spatial discretization as $\boldsymbol{v}^{(T)}$ (that is, $\hat{\boldsymbol{v}}$ was *produced* by a higher resolution approach, but *downscaled* to the coarse discretization afterwards). We aim to find correction parameters θ as to minimize some loss measure $\mathcal{L}(\hat{\boldsymbol{v}}, \boldsymbol{v}^{(T)})$ between the two velocity fields. For the rest of this work we will set the loss measure to be the *mean squared error* between the discretized velocity fields, however, we only consider grid points inside the flow domain[1]:

$$\theta^* = \arg\min_\theta \mathcal{L}(\hat{\boldsymbol{v}}, \tilde{\boldsymbol{\mathcal{F}}}(\boldsymbol{v}^{(0)}, \theta)) = \arg\min_\theta \mathcal{L}(\hat{\boldsymbol{v}}, \boldsymbol{v}^{(T)}) \tag{11}$$

$$= \arg\min_\theta \frac{1}{n} \sum_i (\hat{v}_i - v_i^{(T)})^2 \tag{12}$$

where i runs over all grid points and n denotes the number of grid points inside the flow domain.
We want to emphasize that in the primary setting we consider, no *intermediate* velocity targets $\hat{\boldsymbol{v}}^{(n_t)}$ are available to us. Only after T applications of $\boldsymbol{\mathcal{F}}$ do we compute the loss measure for the final velocity field.
We find a (local) minimum of $\mathcal{L}$ by gradient-based methods on the learnable parameters θ. I.e., we iteratively update the correction parameters according to

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial \mathcal{L}}{\partial \theta_j} \qquad \forall \theta_j \in \theta, \tag{13}$$

[1]In general, the computational domains we consider can contain grid points that lie inside *boundary regions* and therefore have a constant velocity value of 0. As the number of grid points in the exterior varies between the different designs in the dataset, we exclude those points in the calculation of the loss as to obtain a more comparable performance measure between simulation with different geometries.

where α is the learning rate hyperparameter (can be adaptive). In eq. 13, first order partial derivatives of the loss function w.r.t individual correction parameters θ_j have to be computed. Because the same correction parameters θ appear in each application of eq. 9, the gradient $\nabla_\theta \mathcal{L}$ contains contributions from each timestep t.
Intuitively this means when corrections to the state at time step $n_t < T$ influence the state of the system at time step T, the gradient signal takes into account the effects of this perturbation over time. The partial derivatives can be derived by applying the chain rule to eq. 10 and 11:

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}^{(T)}} \frac{\partial \tilde{\boldsymbol{\mathcal{F}}}(\boldsymbol{v_0}, \theta)}{\partial \theta_j} \tag{14}$$

$$= \sum_{0 \le n_t \le T} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}^{(T)}} \frac{\partial \boldsymbol{v}^{(T)}}{\partial \boldsymbol{v}^{(n_t)}} \frac{\partial \boldsymbol{\mathcal{F}}}{\partial \theta_j}\bigg|_{\boldsymbol{v}^{(n_t-1)}} \right) \tag{15}$$

The terms $\frac{\partial \boldsymbol{v}^{(T)}}{\partial \boldsymbol{v}^{(n_t)}}$are products of jacobians that propagate the error from step T to step n_t (compare to [12]). Specifically, in our case, these terms require us to be able to calculate derivatives *through the solver*, creating the necessity for a differentiable CFD solver. Rather then explicitly deriving the forms of all intermediate jacobians, in this project we choose to handle these calculations by automatic differentiation (AD) as discussed later in section 6.2.3.

3.3 Correction Functionals

The Navier-Stokes momentum equation is:

$$\frac{\partial \boldsymbol{v}}{\partial t} = -(\boldsymbol{v} \cdot \nabla)\boldsymbol{v} - \frac{1}{\rho} \cdot \nabla p + \nu \nabla^2 \boldsymbol{v} + \boldsymbol{f} \tag{16}$$

We implement and analyse two types of corrections:

1. The ***effective viscosity model*** extends the diffussion term $\nu \nabla^2 \boldsymbol{v}$ by replacing the constant kinematic visocsity ν with a functional of the velocity field such that $\nu \to \nu_0 + \nu_\theta(\boldsymbol{v})$. Here ν_0 is the material viscosity, and θ denotes the *learnable* parameters. The *effective viscosity* augments the *eddy viscosity model* by introducing a velocity dependent viscosity. Thus, we are able to recover the cumulative effects of micro-scale vortices (or *eddies*) on the macro-scale dynamics. Unlike the kinematic viscosity, which is a material parameter, the eddy viscosity can be negative [18, 23].
2. The ***residual model*** replaces the (constant) external body force with a velocity dependent force field: $\boldsymbol{f} \to \boldsymbol{f}_0 + \boldsymbol{f}_\theta(\boldsymbol{v})$. Note that although the *effective viscosity model* implements an explicit correction to the vector Laplacian of the velocity field $\nabla^2 \boldsymbol{v}$, given sufficiently many parameters θ, it is equivalent to the *residual model* and vice versa. The residual model is thus analogous to the *control-term* in [6].

Our approach to these corrections differs from related methods as follows:

- The corrections are local and act only on the neighbourhood of a given point.
- The corrections are time-independent and are applied between solver steps.

- The corrections are strongly coupled to the solver. They incorporate information about future dynamics into the forward evolution of the solver.

The manner in which these corrections interact with the solver depends on the actual discretization and the time-stepping scheme of the solver. In general, we always correct some (intermediate) *physical quantity* as a functional of the flow field.

4 Data Preparation

For the given problem, we are provided with steady state flow field data from 5,005 designs of different nozzle geometries. For each of the designs, we have 4 inlet velocities, resulting in a set of 20,020 targets. Before implementing the subsequent parts of the pipeline, we need to analyse the datasets to gain further insights into the designs and the distribution of the coordinates within the domain of the design. In the following subsections, we will describe the structure of the data and the processing required to produce a target dataset for the training stage.

4.1 Structure of the Training Data

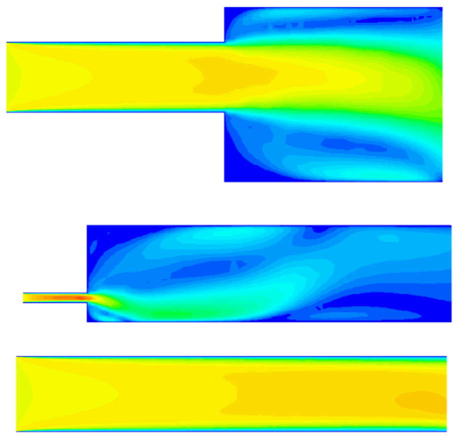
Table 1: Ranges of design parameters, determining the geometry of the simulation. A is the design identifier, B is the height of the channel. C,D and E determine the shape of the nozzle. F denotes the inflow velocity.

A	B	C	D	E	F
1-5005	10-40 mm	0.15-0.4	0.2-0.4	0-0.5	0.5-2 m/s

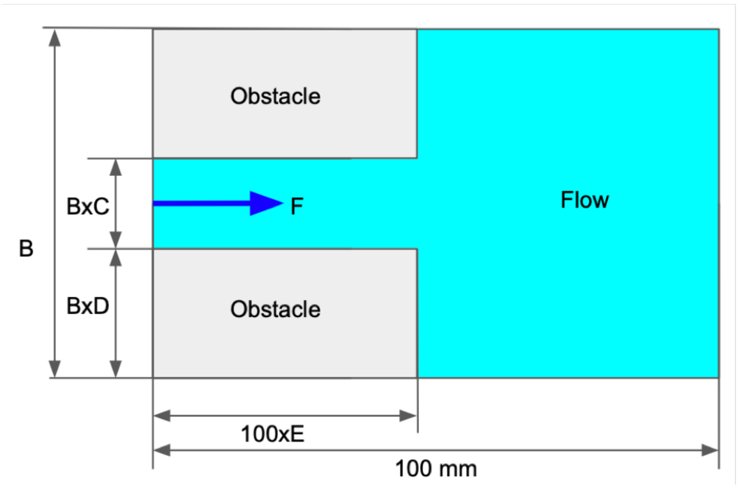
All the nozzle designs provided are based on the parameters shown in the table 1. We were provided an excel file containing the design parameters for all the designs. Varying combinations of these parameters correspond to different designs as shown in the figure 2a. The more about the structure of the data can be found in appendix D).The file most important for our pipeline is the CSV file as will be used as a target for the optimization problem. It stores the coordinates of the data points within the domain of the design and velocity and pressure values at these coordinates. A sample dataframe is shown in the figure 3. If we further analyze the CSV file, we see the coordinates within the domain are unstructured, which means the simulation result is obtained using an unstructured grid. We have a non-uniform distribution of points in the grid, more points in the boundary regions at the walls. The same can be seen in the left part of figure 4.

4.2 Computing Training Targets

We saw in the previous subsection that the simulation result are obtained using the mesh of unstructured triangular grids. The velocity values (either magnitude or vector components) $y_i, \quad i = 1...n$ contained in the CSV file are given at unstructured arbitrary coordinates x_i in the designs as seen in left part of the figure 4. The simulation schemes we employ (section 6.2), are using meshes of structured square grids. Hence, we need to process our target velocities to obtain the velocity data on these grid points.

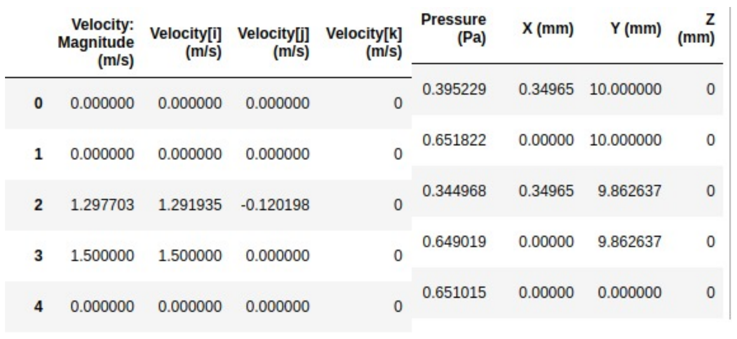


(a) Design parameters, determining the geometry of the problem



(b) Channel and nozzle geometry design. See table 1.

Figure 2



	Velocity: Magnitude (m/s)	Velocity[i] (m/s)	Velocity[j] (m/s)	Velocity[k] (m/s)	Pressure (Pa)	X (mm)	Y (mm)	Z (mm)
0	0.000000	0.000000	0.000000	0	0.395229	0.34965	10.000000	0
1	0.000000	0.000000	0.000000	0	0.651822	0.00000	10.000000	0
2	1.297703	1.291935	-0.120198	0	0.344968	0.34965	9.862637	0
3	1.500000	1.500000	0.000000	0	0.649019	0.00000	9.862637	0
4	0.000000	0.000000	0.000000	0	0.651015	0.00000	0.000000	0

1

Figure 3: A sample dataframe obtained from the simulation CSV file.

To tackle the above problem we solve a weighted least squares (WLS) [4] problem for each point $\hat{x}$ in the regular grid. That is, we try to find a fitting function $g(x) = \boldsymbol{b}(x)^T \boldsymbol{c}$ with polynomial basis vectors $\boldsymbol{b}(x) = [b_1(x), b_2(x) ..., b_k(x)]^T$ of degree k and unknown coefficient $\boldsymbol{c} = [c_1, ..., c_k]^T$.
We can write down the WLS optimization problem as:

$$\min_c \sum_i^n W(d_i)||g(x_i) - y_i||^2$$

where d_i is the euclidean distance between $\hat{x}$ and any other point x_i. We give a more detailed description of WLS in appendix E). A common choice for the weighting function is a Gaussian $W(d) = e^{-\frac{d^2}{\sigma^2}}$, where σ controls the localization of the weighting function around $\hat{x}$. The choice of these types of weighting function ensures that more weight is given to the data points near to $\hat{x}$. In our case this is extremely important so that the velocities at the structured grid points are very close to nearest unstructured grid data that we have.

This fitting task has two parameters. One is the degree of the basis function and another one is the σ. We tried to find the best σ and degree k for the fitting using grid search approach. We implement a function and pass various values of σ and degree to calculate the velocity values at the grid points. Once we get the velocity values at the grid points, we again fit the velocity values back to the unstructured grid points with the same σ and degree. We then calculate the mean squared error of the velocities at the original unstructured grid point coordinates. We ran this function for a subset of the design that covered the parameter range in the dataset. We then calculate the errors for each combination of σ and degree. The figure 5a shows the error value for all the combinations. From the figure, it is quite evident that the best σ is around 1.0 and the best degree is 2. The figure 5b shows the study of the fitting accuracy around $\sigma = 1.0$ for degree 2.
The right side of the figure 4 shows the result of the fitting using the best sigma and the degree on the structured grid. We can see that we are able to produce the velocity values at the structured grid quite close to the original data that we had. This will now act as a target for the pipeline.

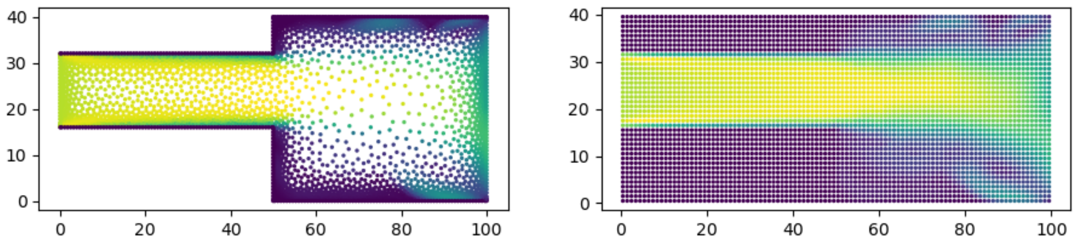
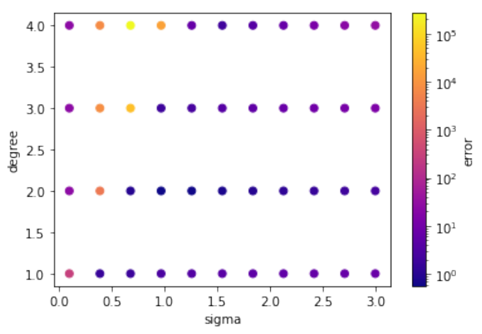
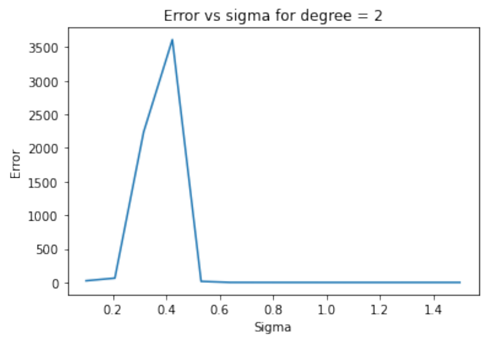


Figure 4: Velocity representation on the unstructured grid. On the left, Original velocity obtained from the *STAR-CCM+* solver. On the right, Interpolated velocity using the best sigma and degree, used as target for the model.



(a) Error distribution



(b) Error for various sigma for a fixed degree 2

Figure 5

5 Simulation Setup

In the following section we will elaborate on setting up the *plain simulation*. By *plain* we refer to the *uncorrected* simulation using the PP SOLVER or Φ_{Flow} [13] solver in order to solve Navier-Stokes equations (3) and (4). We will give a description of both methods in section in 6.2. Here, we focus in the simulation setup in Φ_{Flow} as it is the framework in which we conduct most of our experiments, we reason about this choice in section 7.2. For the later phases of the project it is crucial to verify the correct simulation setup, including boundary conditions and geometry, as well as to choose appropriate time stepping parameters. In this section we are going to answer the following questions:

- How many time-steps will the solver take to reach a steady state?
  $\rightarrow$ It is desirable for use to reach the steady state in as few time steps as possible.
- Does the simulation converge? Will it cause perturbations?
  $\rightarrow$ We aim to find the largest feasible time-step size.
- Is the velocity field obtained from the untrained simulation close to the reference?
  $\rightarrow$ We can verify the correctness of the simulation setup and physical properties.

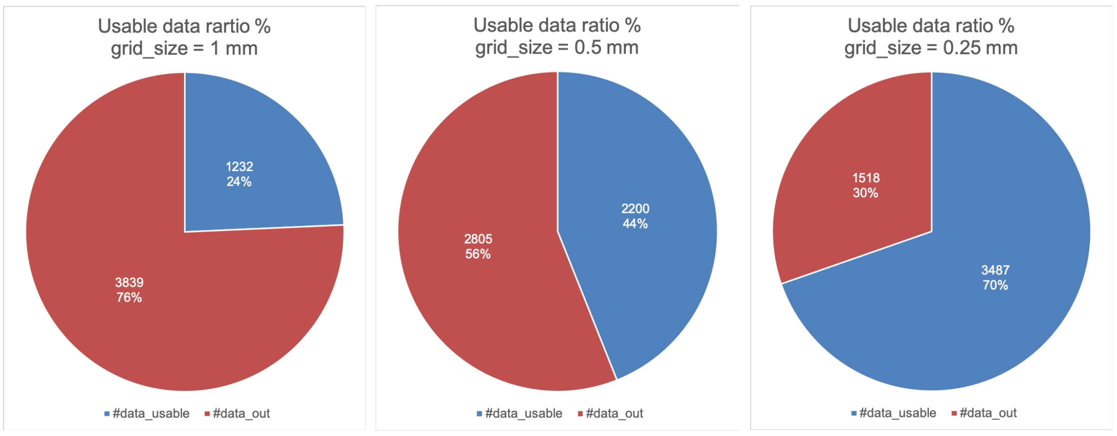
5.1 Meshing

Before elaborating on the setup of the plain simulations, we will discuss a challenge, which we call the *meshing problem*. Meshing is a crucial pre-processing step in CFD simulations to generate grid points. Grids are constructed in the solution domain and every computation happens on a grid point. The solver schemes we employ use structured grids, which are equally spaced in 2D. Ideally, we would like all geometries to be placed precisely in the grid, i.e. that the boundary regions lie *on* the staggered grid points in the case of Φ_{Flow}. If the geometry requires the boundaries to lie *in between* grid points, we should make an additional effort to solve this. This problem is one of the disadvantages and a common issue associated with structured grids. We can address it with three possible strategies:

1. We find the largest grid resolution, which is able to resolve all the boundary regions for all geometries (i.e. the greatest common divisor). This largest resolution is 0.125 mm.
2. We use Shortley-Weller scheme [2], which is used in complicated geometries in finite-difference methods.
3. We fix an appropriate resolution and classify all designs into two groups, based on wether or not their geometries locate *on* or *in between* the grid points. Only the designs lying *on* the grid points are usable for us.

Given that the point of this work is to operate on *coarse* grids, we reject the 1st method. It would lead to impractical computational effort, i.e., the total number of grid points in the largest domain is 256,000. We also reject the Shortley-Weller scheme due to its considerable complexity. We choose the third strategy because we can select the coarse resolution to save computational effort but keep the chosen designs located precisely on the grid points. Figure 6 indicates the amount of usable geometry designs for different resolutions. It is intuitive to consider the higher resolution, however, increasing the amount of usable data significantly increases computational effort. This is a trade off. Finally, we select 1 mm resolution according to three reasons:

1. 1232 designs are presumable enough for training.
2. The smallest nozzle length is 1 mm, so the grid can't be larger than 1mm.[2]
3. The simulation employing the 1 mm resolution has the fastest computational time. See table 2.



(a) grid size 1mm (b) grid size 0.5mm (c) grid size 0.25mm

Figure 6: Statistic pie charts of geometry design.

Table 2: Computation time required for 1000 time-steps in design 5005, employing structured grids of different resolutions.

resolution (mm)	1	0.5	0.25
time (s)	68	152	312

Our ultimate goal is to use the simulation with coarse grids as precise as a reference solution by correction from deep learning. Therefore, we can consider the coarse resolution, which makes the simulation fastest as long as it converges instead of the fined resolution. 1 mm is pretty suitable in both plain simulation and deep learning.

5.2 Plain Simulation

We implement functionalities for setting up parameterized simulations in Φ_{Flow}[13], corresponding to the designs encountered in the dataset. Based on the design number and the specified resolution, we automatically generate the computational domain, including the

[2]In general, only one cell is not enough to resolve the inflow nozzle. Considering the computational time in whole approach, we make a trade off here and exclude these designs from the dataset.

geometry of the flow field and boundary conditions. We then use the results from section 5.3 to set appropriate values for Δt and T and let the simulation run for appropriate number of timesteps.
By producing plots of the resulting flow fields and visually comparing both the velocity field and the velocity variation against the reference solution, we can further debug and check the stability of our setup. We note that we do not expect the simulation to produce an output very similar to that of the reference. We rather check whether or not the plain simulation produces reasonable results and roughly conforms to the reference, based on the given physical properties and boundary conditions.

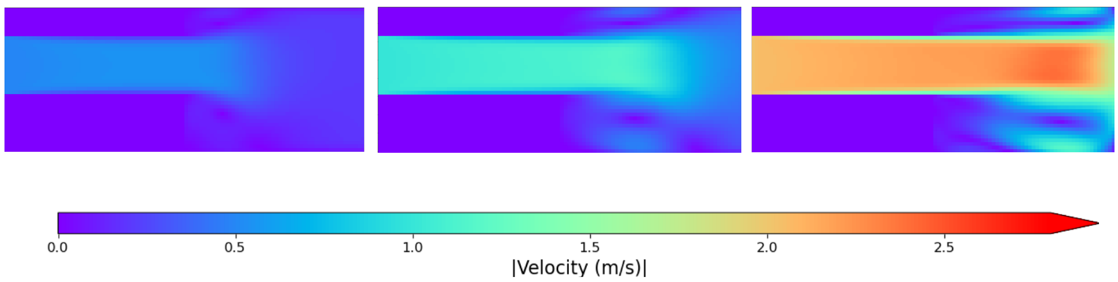


Figure 7: The visualization of the velocity field helps us to check the physical plausibility of the simulation. Depicted are simulations for different inflow velocities.

5.3 Time stepping

The choice of time-step size is crucial in our approach: In the training phase, each forward pass involves a full execution of the plain simulation. To reduce the computational cost, it is therefore important to determine the time stepping that leads to the fastest convergence to the steady state.
We choose the time step size Δt and the total number of time steps T based on the following strategy:

1. We utilize the *Courant-Friedrichs-Levy* (CFL) condition [17]

$$|v_{x,max}|\Delta t < \Delta x, \quad |v_{y,max}|\Delta t < \Delta y \tag{17}$$

   to find an appropriate timestep size. In CFD simulations, the CFL conditions help to ensure the numerical stability in an explicit scheme. In eq.17, Δt is determined by the maximum absolute velocity in the flow field $|v_{max}|$ and the distance between two adjacent grid points Δx (named *resolution* in the following). In the following we will assume that $\Delta x = \Delta y$, as it is the case for Φ_{Flow}.

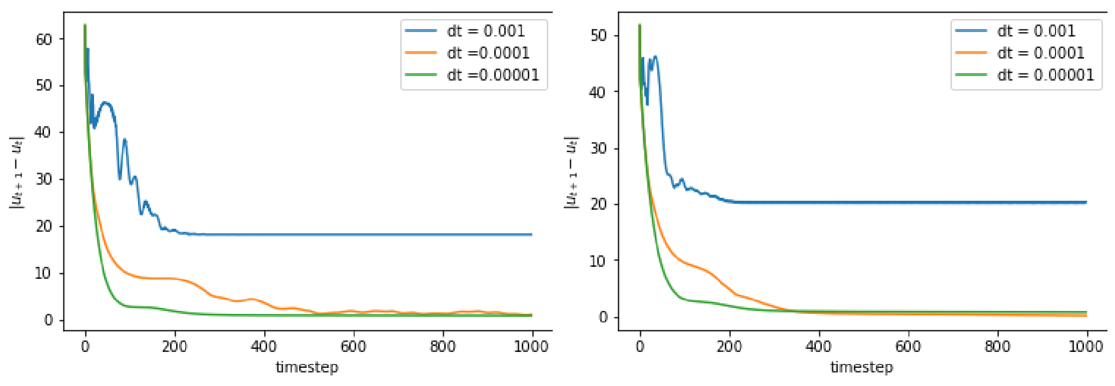
2. Then, we directly compute the difference between the velocity fields for two consecutive time steps n_t and $n_t + 1$ to determine the point of convergence, i.e. we define the condition for the steady state as : $|\boldsymbol{v}^{(n_t+1)} - \boldsymbol{v}^{(n_t)}| \simeq 0$. We denote the timestep after which this condition is reached as convergence time T.

To illustrate the process, we demonstrate one example of choosing appropriate timestepping parameters. Firstly, given an inlet velocity $u_{inlet} = 1$m/s and resolution $\Delta x = 1$ mm, we have the following condition according to CFL:

$$\Delta t < \frac{\Delta x}{|u_{max}|} \leq \frac{\Delta x}{|u_{inlet}|} = 0.001s \tag{18}$$

Secondly, to limit the overall computational effort in finding appropriate settings for the whole dataset, we only test a set of “edge-case” designs which we found to be likely to generate high amounts of turbulence, requiring more timesteps to reach convergence. These cases generally have the largest computational domain with different nozzle geometries, e.g. designs 5005 and 4956. By testing these designs we assess the upper bound on T, enabling us to use the same settings for all geometries. In fig. 8, we observe that with $\Delta t = 0.01$, both cases fail to reach a stationary state. With $\Delta t = 0.001$, both cases improve significantly, but there’s a small perturbation in Design 4956. With $\Delta t = 0.0001$, both case converge after around 400 time-steps. The running time of the simulation is approximately 30 seconds.
In summary, we choose the following settings for Δt and *convergence time* T in the plain simulation as well as the training procedure throughout the rest of the report:

- $\Delta t = 10^{-4}$ s , which fulfills the CFL conditions for the inflow velocities 0.5 to 2 m/s.
- $T = 500$ steps, which is a rather conservative estimate and includes a “safety buffer”.



(a) Design 4956 (b) Design 5005

Figure 8: Velocity variations $|\boldsymbol{u}^{(n_t+1)} - \boldsymbol{u}^{(n_t)}|$ for different step sizes Δt. Larger step sizes result in oscillations and lead to no stationary state.

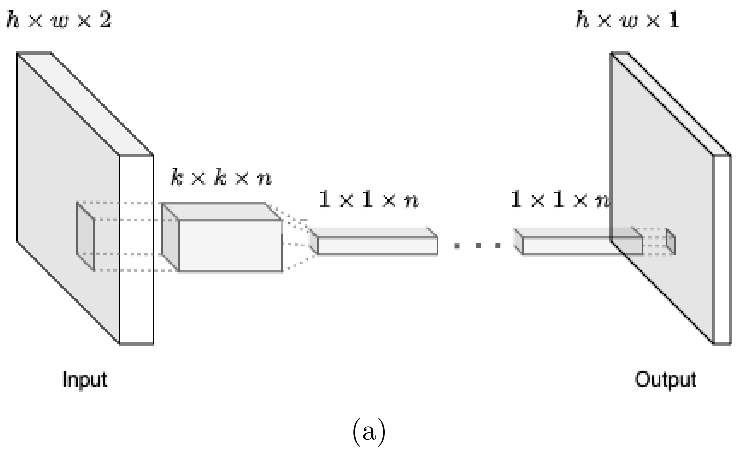
6 Methods and Implementation

In the following section we will elaborate on implementations of the computational components needed for training the correction models studied in this work. We will first motivate the architecture of the neural networks used in this work. After that we will introduce the differentiable CFD solvers employed and point out their differences.

6.1 Network Architecture

We employ deep neural networks (DNN) as universal function approximators to represent the correction functions. Two considerations regarding the architecture of the DNN have to be made:

1. As the network operates on regular grids only, we can use classical convolutional operations to capture spatial correlations.



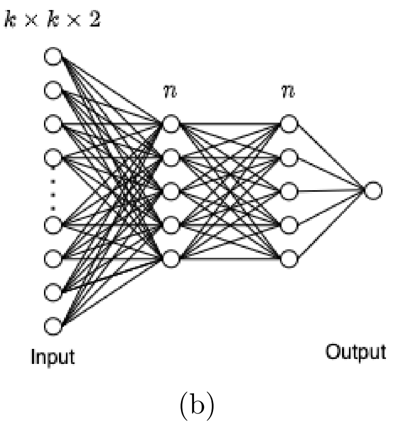


Figure 9: a) Schematic of the local, fully convolutional architecture. The convolutional operations preserve the spatial dimensions $h \times w$ and the perceptive field of each point in the output is restricted to the local 3×3 neighbourhood in the input. b) Predictions of individual points in the output made by the CNN are equivalent to a "moving" MLP where the input consist of the local neighbourhood of $k \times k$ grid points.

2. The architecture should represent the spatial invariances on a global scale. I.e. it should fulfill the condition that the output is only a function of the local neighbourhood of the corresponding input region.

To satisfy both of these requirements we employ a local variant of a fully convolutional neural network (FCNN). The first layer of the network is always a convolution of kernel size $k \times k \times n$, where $k \in \{3, 5\}$ and n is a variable feature size for all future considerations. This layer is able to capture the local neighbourhood in the input of the 9 or 25 grid points respectively. All following operation are $1 \times 1 \times n$ convolutions which do not incorporate any more spatial information then provided by the first layer. A schematic visualization of the architecture is shown in figure 9. The architecture is computationally equivalent to "moving" a *multi layer perceptron* (MLP) with $k \cdot k \cdot d$ input neurons over the input to predict individual scalar values in the output plane as indicated in 9 b). In this case the feature size n corresponds to the number of hidden neurons and d corresponds to the channel size of the input. We want to stress that, unlike common FCNN architectures from the field of image processing (see e.g. [16] or [9]), our network variant is agnostic to "global features". For the purpose of the different experiments performed during the project, we implement this architecture in TensorFlow [11], PyTorch as well as the Julia programming language.
We show the computational steps required to actually apply the networks on the flow field data in appendix B).

6.2 Differentiable Physics Solvers

At the core of our method is a differentiable CFD solver that is able to propagate error signals through its computational steps.
We employ and compare two implementations for solving the incompressible Navier-Stokes

equations: PP-SOLVER [5] and Φ_{Flow}[13]. The two schemes represent two contrasting methods for solving the governing equations: While the PP-SOLVER is a finite-element scheme, Φ_{Flow} employs finite-differences.
Therefore, part of the goal for this project was to assess and compare the suitability for both of these solvers in the context of augmenting them by learned corrections.
Besides these differences in discretization schemes, we also use the opportunity to assess advantages and disadvantages of two different ways of calculating derivatives through the solvers; While we reimplement the PP-SOLVER to use so-called forward-mode automatic differentiation, Φ_{Flow} [13] comes with the ability for reverse mode automatic differentiation (backpropagation).

6.2.1 PP-Solver

Siemens provided us with an MATLAB implementation of a finite-element solver for solving the incompressible Navier-Stokes equations, the so called PP-SOLVER [5]. Due to the limited scope of this report we will omit a description of the details of the scheme itself but, for the interested reader, say that it employs Q2-P1 finite-elements, Van-Kan pressure correction and a Crank-Nicolson time stepping scheme. A detailed theoretical description can be found in [21]. This original ”plain” MATLAB implementation naturally does not allow for gradient propagation through the solver. Therefore our task was to produce a *differentiable* implementation of the scheme.
For this purpose, based on suggestions by Siemens, we first completely re-implement the PP-SOLVER code base in the JULIA programming language. We validate the output of our implementation against the MATLAB reference. Importantly, we implement taking derivatives through the computational steps of the solver by means of forward automatic differentiation. We elaborate on the practical reasoning leading to this decision in section 7.2. We validate our implementation in section 7.1 and assess its suitability regarding the augmentation by learned corrections in section 7.2

6.2.2 PhiFlow

As described on the official github page [13] “ Φ_{Flow} is a research-oriented, open-source PDE solving toolkit that is fully differentiable [...]”. It is a finite-differences solver based on the *stable fluids* algorithm [19] for solving the incompressible Navier Stokes equations. The scheme involves 4 main computational steps to advance the state of the fluid to the next time step, as shown in figure 10.
It deviates from the original stable fluids algorithm [19] in some points. I.a. Φ_{Flow} employs a staggered grid for the velocity field and calculates the diffusion in an explicit manner.

6.2.3 Modes of Automatic Differentiation

Automatic Differentiation (AD) describes an algorithmic form of calculating derivatives of numeric functions implemented as computer programs [1]. It uses the fact that *any*, arbitrary complex, computation is composed of simple arithmetic operation, for which derivatives are known.
We choose to employ AD in our experiments because:

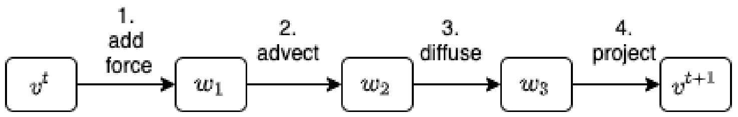


Figure 10: Computational steps in the stable fluid scheme [19] for advancing the flow field to the next time step 1.) External forces are applied as $\boldsymbol{w_1} = \boldsymbol{v^t} + \Delta t \boldsymbol{f}$. For our purposes, the *residual correction model* is applied in this step 2.) The advection of the fluid by itself is calculated by the method of characteristics [19] 3.) The effect of diffusion is taken into account by solving an implicit form of the diffusion equation. The Φ_{Flow} implementation deviates at this point by explicitely computing the vector Laplacian by finite-difference discretization. Our *effective viscosity model* comes in here. 4.) $\boldsymbol{w_3}$ is projected onto the space of divergence free fields by solving the associated Poisson problem.

- In most applications, AD is more accurate and more efficient then calculating derivatives by finite differences [1].
- Manually deriving the expressions for the derivatives for complicated operations is prone to error and no analytical solution may exist for problems with implicit solutions.

At its core, AD presents ways of algorithmically applying the chain rule to compound functions. There are two main variants of Automatic Differentiation that roughly correspond to traversing the chain rule *from the inside out* or *from the outside in* respectively. We refer the reader to [1] and [10] for more comprehensive reviews of the different AD methods. We give a short illustration of both methods in appendix A). In our differentiable PP-solver we implement forward mode AD. For this purpose, we employ the JULIA package FORWARDDIFF [15], where we had to make sure that each function and operation is is able to process the dual number types introduced for differentiability. Φ_{Flow} [13] relies on reverse mode AD (backpropagation) in TENSORFLOW [11].

6.3 Pipeline

We combine the differentiable solver, the local neural network variant and the computation of target velocity field in a *pipeline* for training the different models. The training process for each sample in the dataset can be summarized in short as follows:

1. We obtain the target velocity field $\hat{\boldsymbol{v}}$ by computing the WLS fit of the data provided by Siemens to the regular grid used in our simulation as described in section 4.2. This step is precomputed in advance to the training procedure.
2. We setup the simulation geometry and boundary conditions corresponding to the datasample as described in section 5.
3. We perform T steps of the augmented solver scheme $\boldsymbol{\mathcal{F}}$ and obtain the final velocity field $\boldsymbol{v^{(T)}}$.
4. We compute the loss $\boldsymbol{\mathcal{L}}$ on the magnitude of the velocity field and calculate gradients with respect to model weights $\nabla_\theta \boldsymbol{\mathcal{L}}$ *through the solver* steps. The model weights are updated by gradient descent.

7 Results

This section presents the experimental results and is structured as follows: We begin by verifying the correctness of our implementations and provide a proof of concept on some test cases in 7.1. In section 7.2 we elaborate on the practical implications of the different solver methods and their empirical suitability for the proposed method. In section 7.3 we outline the training experiments in the original setting and point out the challenges associate with it. Based on this, in section 7.4 we will present promising, alternative experimental settings.

7.1 Proof of Implementation

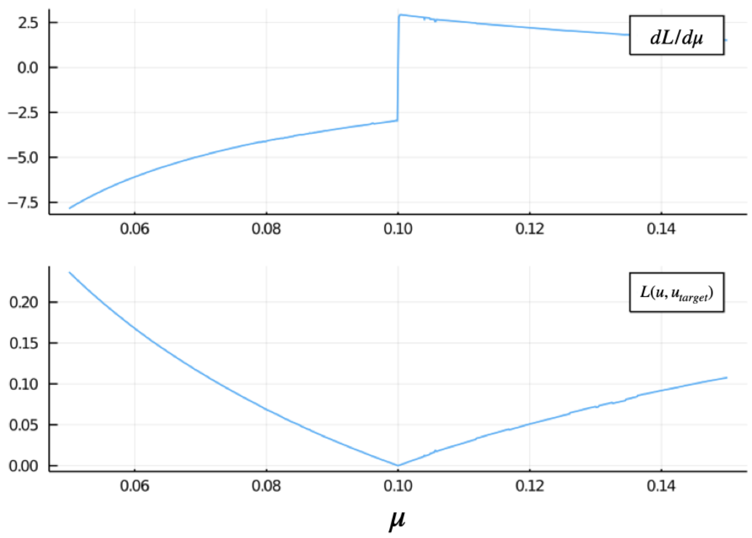


Figure 11: Proof of implementation for the differentiable implementation of the PP-SOLVER. Derivative of the loss function $L{:=}\sqrt{\mathcal{L}}$ w.r.t. viscosity (top) and the shape of the loss function (bottom), both as a function of viscosity μ. The minimum of the loss is located at the viscosity of the target simulation with $\mu = 0.10$. All dimensionless quantities for testing purposes.

To confirm the correctness of our implementations of the differentiable solver as well as the training procedure, we perform the following tests:

1. We begin by verifying that derivatives are correctly propagated through our differential implementation of the PP-SOLVER. For that purpose we first produce a reference simulation with a grid of 4x4 elements using a *target* viscosity of $\mu = 0.1$. We then run 1000 individual simulations where we vary the viscosity value in the range $[0.05, 0.15]$ in between simulations. For each simulation we compute the loss between the velocity field u at timtetstep 10 and the velocity field of the reference simulation u_{target}. We obtain the derivative $\frac{dL}{d\mu}$ at the same time from forward mode AD. We verify our solution by plotting the resulting "loss landscape" as well as its derivative obtained from forward mode AD in fig 11.
2. For verifying the training procedure, we design an experiment in a similar manner to 1.; we compute a reference velocity field after $T = 10$ timesteps with $\Delta t = 0.0001$ and a viscosity value which is 10 times higher then the kinematic viscosity ν_0 used during optimization. This time, however, we calculate gradients w.r.t. the model

weights, rather then derivatives w.r.t. a scalar. We then try to overfit our model to the reference state. The correction model in this case is the *effective viscosity model*, using a 4-layer CNN with an architecture as described in section 6.1 (k=3, n=8, totalling 377 learnable parameters). For reasons explained in the next section we used Φ_{Flow} for this and further experiments. For optimization we use the Adam optimizer [7] with a learning rate of 10^{-3}. As seen in fig 12 a), after ~ 125 iterations the optimization converges to a state close to the reference, indicating that the correction successfully learned to reproduce the target viscosity function. While the optimization easily converges for the case of a laminar pipe flow in figure 12 b), for the general case of non-laminar flow the optimization on a single sample is more prone to converge to local minima.

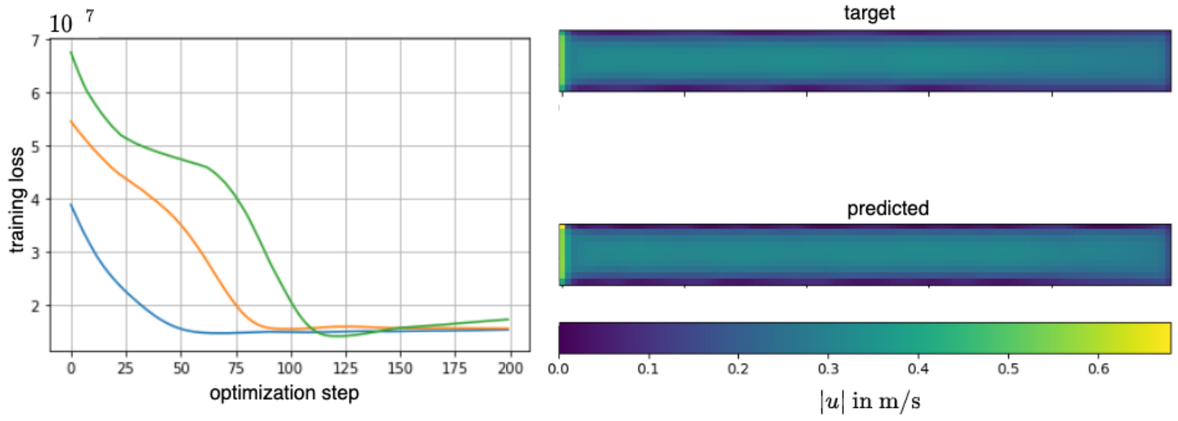


Figure 12: Exemplary results from overfitting to a single target state. On the left: Training loss during optimization for 3 different random initializations (glorot uniform). On the right: Reference state and predicted solution after convergence after $T = 10$ steps. The simulation geometry is that of a plain pipe, resulting in a laminar flow field.

7.2 Assessment of Solver Suitability

Based on our experience in implementing and working with the corrections schemes in both Φ_{Flow} as well as our differentiable implementation of the PP-SOLVER, we would like to discuss the empirical advantages and disadvantages associated with each method both from a computational point of view, as well as regarding its practicality and convergence properties.

Computational aspects: In each solver step of the PP-SOLVER, internally several nested loops and iterative methods are called. Storing all intermediate values for the backward pass in reverse mode AD quickly leads a an explosion in memory requirements, even for relatively small simulations. Therefore, in our differentiable PP-SOLVER implementation we decided to employ forward mode AD, as it drastically reduced the memory

requirements of the computations over naively employing reverse mode AD. The reason for this is that fewer intermediate results have to be stored in memory with forward AD, as there is no backward pass. On the other hand, the run time costs of forward mode AD increase with the number of input parameters (i.e. weights to be optimized) [1]. In our implementation, the main performance hurdle was that core operations like matrix multiplication were not optimized for use with *dual numbers*. Together with Siemens we decided that further optimizing our differentiable implementation of the PP-SOLVER on a computational level was beyond the scope of the project. In general, we conclude that, in order to implement a performant differentiable CFD solver, automatic differentiation methods have to be *combined* with either 1) manually derivation of expressions or 2) derivatives by finite differences for substituting otherwise heavy computational operations.
Regarding the implementation of the pipe-flow simulations and correction models in Φ_{Flow}, we find that with “graph mode” in TENSORFLOW [11], a large computational overhead is introduced compared to executing the *plain* simulation (section 5). The reason for this is that, contrary to classical machine learning tasks, in our case a new computation graph corresponding to the new simulation for each training sample has to be created. We experience a 14 fold decrease in training time when switching to “eager”-execution for training.

Convergence: When augmenting the CFD solvers by learned corrections, one inevitably perturbs the intermediate states. As the same correction functional is applied for all timesteps, we find that the convergence properties of the solver schemes are affected even by minor perturbations when integrated in time. This behaviour poses a major challenge when trying to optimize corrections by gradient-based methods; each update step can potentially lead to divergence of the solver scheme, effectively leading the training run into a dead end.
In the case of the PP-SOLVER scheme, even with careful initialization and small learning rates, we were not able to consistently train for longer periods without ”blowing up” the solver scheme. Although formally studying the convergence properties of the PP-SOLVER is beyond the scope of this project, we think that major adjustments to the scheme have to be made in order to be robust to the kind of perturbation applied by our method.
On the contrary, we find that the *stable fluids* algorithm [19] as employed in Φ_{Flow} is in general well suited for applying perturbations to the intermediate states. In fact, the scheme was originally developed to be able to handle external ”interactions” with the flow field, without leading to ”blow-ups” [19] of the scheme (i.e. even if the resulting behaviour may be non-physical, it does not lead to divergence of the scheme). We argue that this property is essential for choosing an appropriate solver scheme accompanying the correction method studied in this work.
In summary, because of the difficulties regarding both the performance, as well as the convergence of the PP-SOLVER, we decided to focus on the implementation employing Φ_{Flow} for all experiments to follow in this report.

7.3 Training Experiments

Like discussed previously, the original goal for the project is to learn intermediate correction functionals from only the steady state data provided by Siemens. This is equivalent to optimizing the objective 11 i.e. only the target state $\hat{\boldsymbol{u}}$ corresponding to the final timestep T is given. Using the results from section 5.3 we know that $T \approx 500$ is required to reach the steady state in the plain simulations. Hence, starting from initial state $\boldsymbol{v}^{(0)}$ (set to 0 by default), 500 recurrent steps $\boldsymbol{\mathcal{F}}$ have to be computed before the loss and corresponding gradients can be calculated. We find that this is not possible in practice. Below, we outline the experiments leading to this finding and study its failure modes. Based on this, in section 7.4 we will present promising, more suitable experimental settings.

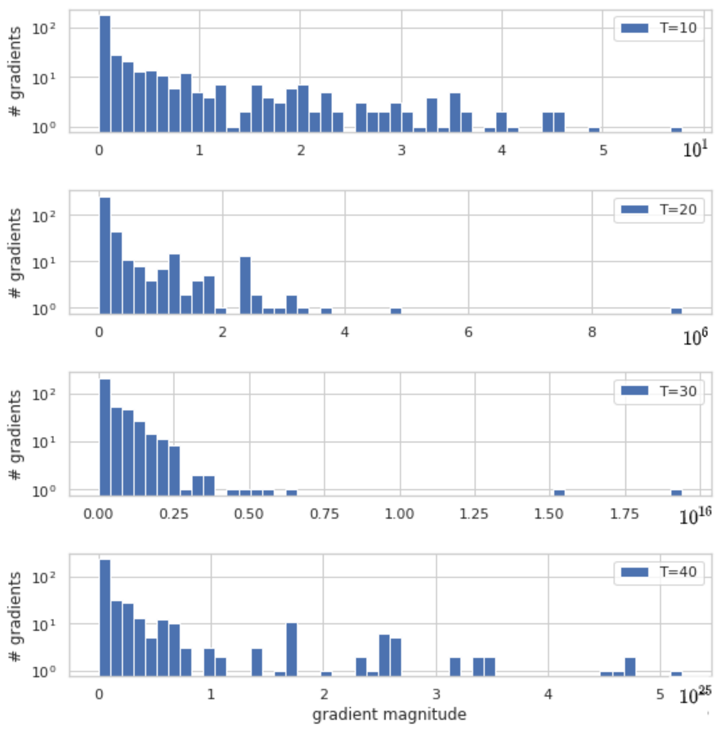


Figure 13: Distribution of absolute gradient components for different maximum number of timesteps $T \in \{10, 20, 30, 40\}$. The entries of the histograms are the 377 weights of a 4-layer convolutional architecture respectively. The distributions correspond to the very first iteration of gradient descent i.e. before any weight update is applied. Note the increasing scale on the x-axis.

Exploding Gradients When trying to calculate gradients corresponding to long temporal components (i.e. $T \gg 1$ in eq. 15), we observe the problem of *exploding gradients*. To illustrate the issue we repeat the experiment from section 7.1 point 2 with targets at different number of timesteps T. Fig 13 shows the distribution of absolute components of the gradients for increasing T. The gradient magnitude clearly increases with T, leading to numerical overflow in some components for $T \geq 50$.

We experimented with addressing this issue by *gradient clipping* [12] (i.e. normalizing gradient if its magnitude goes above a certain threshold). However, in the setting discussed here, we were unable to consistently obtain valid gradients with this method, mainly because for large T, increasing numbers of gradient components are invalidated by numerical overflow.

We want to emphasize that we find the issue of exploding gradients to occur independently of the weight initialization or learning rate. We also found no connection to the size of the neural network, i.e. the issue remains present even for a network with a single layer.

In fact, we believe that the issue is closely related to the exploding gradients problem encountered when training recurrent neural networks (RNN)[12]. We will discuss the analogies in appendix C).

7.4 Alternative Settings

In the following we present ways of circumventing the problems elucidated in the previous sections. All of the proposed experimental settings involve calculating *intermediate* error signals with respect to targets states $\hat{\boldsymbol{v}}^{(\boldsymbol{n_t})}$ at times $n_t \leq T$:

$$\overline{\mathcal{L}}(\hat{\boldsymbol{v}}, \boldsymbol{v}^{(T)}) := \frac{1}{n} \sum_{1 \leq n_t \leq T} \sum_i (\hat{v}_i{}^{(n_t)} - v_i^{(n_t)})^2 \tag{19}$$

This setting enables us to control the contributions of long term temporal components to the gradients.
Siemens offered to provide us with the a new set of target simulations where intermediate state at times n_t (in intervals of $\Delta t = 10^{-4}s$) would be available. However, we have found that the new dataset was not compatible for our purposes: The transient simulation data provided by Siemens was obtained using the StarCCM+ software which implements a finite volume scheme, while Φ_{Flow} employs a finite difference scheme with an explicit advection step. This leads to very different dynamics of the velocity fields, especially in the beginning of the simulation, where the advection of the fluid from the inflow outwards is limited by the finite step size.
To still be able to present a proof of concept of the method, we consider the following alternative settings:

1. We *create* a small scale dataset, consisting of 5 high resolution transient simulations using Φ_{Flow}. For this purpose we choose a grid resolution of 0.2mm, resulting in 1 grid point on the coarse scale for every 25 grid points on the fine scale. Similarly, according to the CFL criteria, we adjust the timestepping size by a factor of 5. The increased spatial and temporal resolution increases the computational time by a factor of $\sim$ 125. We choose 4 out of 5 simulations as training data and use the remaining simulation for evaluation. Fig 14 shows an exemplary training run for $T = 10$ with intermediate error signals, achieving up to 40% improvement over the plain simulation in the unseen simulation used for evaluation. We find the results

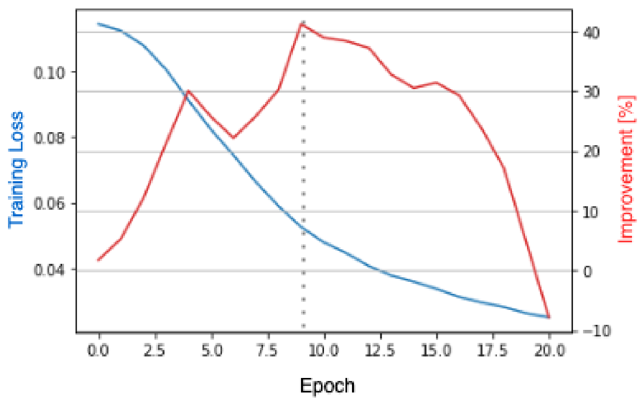


Figure 14: Results from training on the self-produced high-resolution transient data. The improvement is calculated w.r.t. the coarse plain simulation. The dashed line at epoch 9 marks the point after which overfitting to the training data is observed.

on the small dataset to be promising, as they show that good generalization can be achieved with relatively small amounts of data. We argue that this is because each simulation actually contains information for thousands of individual samples, considering the spatial and temporal dimensions. A caveat is that this small-scale study was performed in the regime in the beginning of the simulation, where the progression of the flow field is limited by the finite advection step.

2. We study the case where the initial velocity field is set to the target velocity field corresponding to the steady state i.e. $\boldsymbol{v}^{(0)} = \hat{\boldsymbol{v}}^{(T)}$. This is justified by the fact that $\hat{\boldsymbol{v}}^{(T)}$ is assumed to be a flow field in steady state, which does not change in time by definition. This enables us to also set all intermediate targets to $\hat{\boldsymbol{v}}^{(T)} = \hat{\boldsymbol{v}}^{(n_t)}$. So, while the loss will be 0 in the very first iteration, every consecutive (untrained) solver step is expected to bring $\boldsymbol{v}^{(n_t)}$ *away* from $\hat{\boldsymbol{v}}^{(T)}$, while the trained model should keep $\boldsymbol{v}^{(n_t)}$ *near* $\hat{\boldsymbol{v}}^{(T)}$ for all timesteps.
   We find that the *effective viscosity model* is able to generalize well beyond timeframes seen during training, leading us to the conclusion that the viscosity functional is a good prior on the type of correction. We describe the experiments leading up to this finding in more detail in appendix F). A noteworthy detail in implementing this setting is that the target and initial state $\hat{\boldsymbol{v}}^{(T)}$ will in general not be divergent free in the discretization of the coarse simulation (although being divergent free in the original discretization). We therefore first project $\hat{\boldsymbol{v}}^{(T)}$ onto the space of divergent free fields in order to obtain consistent initial and target states.

8 Discussion

Challenges We find that optimizing the objective eq. (11) is very challenging in practice. We hypothesise that this is mainly because of two reasons: 1) the interaction of the correction model with the CFD solver introduces a large non-linear contribution into the objective problem and 2) the recursive nature of the correction model influences the gradient flow while training. We think that these circumstances add a large amount of complexity to the training procedure when compared to more classical, fully supervised machine learning approaches.
Specifically, we find that the optimization process becomes more difficult, the *sparser* the error signal becomes; in the case where there is only a single target state after T steps, we observe that the optimization becomes "brittle" in a sense that it is very sensitive to initialization, easily converges to "bad" local minima or even diverges for larger T. Introducing intermediate error signals (eq. 19) helps in "smoothing" the optimization and addressing some of the previous issues as discussed in section 7.4.

Generalization By design of the correction model, the interaction with the flow field is limited to a local neighbourhood. Because the same correction functional is applied across the whole domain, this can be interpreted as a strong regularization on the learned parameter space, hampering the risk of overfitting. We could verify this hypothesis in our experiments and find that, with relatively small amounts of training data, good generalization within the training domain can be accomplished. On the other hand, in the scope of the project the parameter space of simulations shown to the correction models was limited: The dataset only consisted of pipe-flows with an inflow on the left side, we therefore expect that our models has a strong bias towards this setting. Nonetheless we expect the method to potentially exhibit better generalization properties then more classical "global" architectures, an assumption that could be verified in future work.

9 Conclusions and Outlook

In this work we studied the learning of closure models with the goal of increasing the accuracy of coarsely discretized CFD solvers. In the course of the project we created training targets from an external data source, we implemented a differential CFD solver, automated the setup of parameterized pipe- flow simulations and experimented with different correction models.
We have found that the type of models studied in this work are, in principle, able to capture physical phenomena based on only local predictions. In our experiments, we show that training recurrent correction models is challenging in practice. In the following we conclude our findings and, based on that, define requirements on chosen solver methods and training data for future work:

1. The interaction of the correction models with the fluid solver perturbs the intermediate physical states, influencing the convergence properties of the original scheme. *When choosing a solver method, one should pay attention to the robustness of the scheme w.r.t. these kinds of perturbations. We found the stable fluids algorithm [19] as implemented in Φ_{Flow} to work well in practice.*
2. We think that intermediate error signals/ targets are necessary in order to control the influence of long term contributions to the gradients. *For iterative methods, the training data should contain information of the flow field for the intermediate steps.*
3. In general, it is difficult to learn corrections corresponding to time series of arbitrary length. In particular, we find that is is not feasible to learn general correction functionals for an iterative model from only the steady state data. *The training data should be obtained by a method that is "compatible" with the scheme used for training (i.e. the flow field should be propagated in time in a similar manner).*
4. When trained in the right setting, the models show nice generalization properties, even with relatively small amounts of training data. *We found the effective viscosity model together with the local NN architecture to be good priors on the form of correction.*

For training, we found it useful to experiment with lowering the magnitude of the weight initialization and generally found gradient clipping to have a positive effect on performance (see appendices C) and F)). In section 7.4 we showed that with a suitable dataset of transient simulations, the method studied in this work exhibits promising properties. We expect that this result is transferable to a larger dataset of high-resolution transient simulations, which has to be shown in future work.

References

[1] Atilim Gunes Baydin et al. “Automatic differentiation in machine learning: a survey”. In: *Journal of Machine Learning Research* 18 (Feb. 2015), pp. 1–43. arXiv: 1502.05767. URL: http://arxiv.org/abs/1502.05767.

[2] Rainer Callies. *Numerical Programming 2 CSE [MA3306]*. Technical University of Munich, 2020 S, p. 70.

[3] Kai Fukami, Koji Fukagata, and Kunihiko Taira. “Super-resolution reconstruction of turbulent flows with machine learning”. In: *Journal of Fluid Mechanics* 870 (July 2019), pp. 106–120. ISSN: 14697645. DOI: 10.1017/jfm.2019.238. arXiv: 1811.11328.

[4] Rao Garimella and Rao Garimella. “Title: A Simple Introduction to Moving Least Squares and Local Regression Estimation Intended for: A Simple Introduction to Moving Least Squares and Local Regression Estimation”. In: (June 2017).

[5] Thomans Helmich et al. *Kurzanleitung fuer den Matlab-PP-Loeser fuer die inkompressible Navier-Stokes Gleichung.* Tech. rep. Technical University Dortmund, Department of Mathematics, Sept. 2018.

[6] Philipp Holl, Vladlen Koltun, and Nils Thuerey. “Learning to Control PDEs with Differentiable Physics”. In: (Jan. 2020). arXiv: 2001.07457. URL: https://arxiv.org/abs/2001.07457v1.

[7] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[8] “Learning data-driven discretizations for partial differential equations”. In: (). DOI: 10.1073/pnas.1814058116. URL: www.pnas.org/cgi/doi/10.1073/pnas.1814058116.

[9] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 3431–3440.

[10] Charles C Margossian. *A Review of Automatic Differentiation and its Efficient Implementation Graphical table of content Automatic Differentiation.* Tech. rep. 2019. arXiv: 1811.05031v2. URL: www.autodiff.org..

[11] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[12] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks.* Tech. rep. arXiv: 1211.5063v2.

[13] Nils Thuerey Philipp Holl. *PhiFlow.* https://github.com/tum-pbs/PhiFlow. 2020.

[14] Ramsai. “RANS Derivation and Analysis”. In: *SKILL LYNC* (2020). URL: https://skill-lync.com/projects/week-8-literature-review-rans-derivation-and-analysis-3.

[15] J. Revels, M. Lubin, and T. Papamarkou. "Forward-Mode Automatic Differentiation in Julia". In: *arXiv:1607.07892 [cs.MS]* (2016). URL: `https://arxiv.org/abs/1607.07892`.

[16] 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.* Springer. 2015, pp. 234–241.

[17] Benjamin Ruth and Gerasimos Chourdakis. *CFD Lab CSE [IN2186].* Technical University of Munich, 2020 S, p. 43.

[18] G. Sivashinsky and V. Yakhot. "Negative viscosity effect in large-scale flows". In: *Physics of Fluids* 28.4 (Apr. 1985), pp. 1040–1042. DOI: `10.1063/1.865025`.

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

[20] *STAR-CCM+.* `https://www.plm.automation.siemens.com/global/de/products/simcenter/STAR-CCM.html`. Accessed: 2010-02-12.

[21] Stefan Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approache.* Vol. 6. Springer Science & Business Media, 1999.

[22] Kiwon Um et al. "Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers". In: *arXiv preprint arXiv:2007.00016* (2020).

[23] Hong-Yi Xie and Alex Levchenko. "Negative viscosity and eddy flow of the imbalanced electron-hole liquid in graphene". In: *Phys. Rev. B* 99 (4 Jan. 2019), p. 045434. DOI: `10.1103/PhysRevB.99.045434`. URL: `https://link.aps.org/doi/10.1103/PhysRevB.99.045434`.

Acknowledgements

We would like to thank our mentors from Siemens AG: Dr. Dirk Hartmann, Theo Papadopoulos and Mohamed Khalil. For proposing the concept underlying this work, readily providing us with multiple iterations of simulation training data and guiding us through the project. We also would like to thank our TUM Co-Mentor Laure Vuaille for providing us with helpful input and feedback throughout the project. Lastly, we want to thank Philipp Holl for readily answering our questions regarding Φ_{Flow}[13].

Appendix

A) Modes of Automatic Differentiation

In the following we will introduce the two distinct modes of automatic differentiation, namely forward mode and backward mode. To illustrate the conceptual differences, we will closely follow [10] and consider the case where we are interested in calculating elements of a jacobian $J_{ij} = \frac{\partial \mathcal{F}_i}{\partial \theta_j}$, where $\mathcal{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a nested nested function similar to eq. (10).

Forward Mode AD: Given a "seed" vector $u \in \mathbb{R}^n$ in the input space of $\mathcal{F}$, Forward Mode AD evaluates the effect of the jacobian $J \cdot u$. It computes the directed partial derivates w.r.t to all outputs *in one forward sweep*. This is achieved by employing a so-called *algebra of dual numbers*. In our differentiable PP-solver we implement forward mode AD. For this, we employ the JULIA package FORWARDDIFF [15], where we had to make sure that each function and operation is is able to process the dual number types introduced for differentiability.

Reverse Mode AD: Given a vector in the output space $w \in \mathbb{R}^m$, reverse mode AD evaluates the effect of the transposed jacobian $J^T \cdot w$. I.e. for complex operation it actually implements the adjoint jacobian. For this reason, calculating derivatives *through the solver* in eq. 15, can be considered as solving for the *adjoint* of $\tilde{\mathcal{F}}$.
Reverse mode AD comprises two distinct phases: In the *forward pass*, $\mathcal{F}$ is evaluated as usual, however, the operations leading up to the final result are recorded and intermediate results are saved. In the *backward pass*, starting from the output, error adjoints are computed by traversing the function in reverse. Backpropagation, as employed in most machine learning frameworks, is a sepcial case of reverse mode AD for scalar outputs (i.e. loss functions). Φ_{Flow}[13] relies on reverse mode AD in tensorflow [11].

B) Applying CNNs to staggered grids

In the case of Φ_{Flow}, the velocity field is given in form of a staggered grid. To be able to integrate the CNN into the solver step, we do the following prep-processing steps: We first center the x and y vector components respectively by linear interpolation. We then concatenate the components as channel dimensions and feed the resulting tensor of shape $h \times w \times 2$ to the network. In the case of the *effective viscosity model* the output of the NN is of channel dimension 1, such that we can feed it as a centered scalar field to the diffusion step. In the case of the *residual model* the output is of shape $h \times w \times 2$ and has to be converted back to a staggered grid. We do this by linear interpolation to shape $h - 1 \times w - 1 \times 2$, followed by a zero padding operation along the appropriate dimensions to obtain the outputs of shapes $h \times w + 1 \times 1$ and $h + 1 \times w \times 1$ for the x and y components respectively.

C) Recurrent View of Correction Models

The correction models described in section 3.3 are applied in a recurrent manner at each solver step. Although the model is not a Recurrent Neural Network (RNN) the classical sense, in the following we will show the analogies to RNNs point out the emergence of the exploding gradients problem. For this purpose, in our explanations below we will repeatedly relate our expressions to the derivations shown in [12]. Consider the most simple, one unit recurrent neural network:

$$\mathbf{x_{n_t}} = F(\mathbf{x_{n_t-1}}, \mathbf{u_{n_t}}, \theta) = \mathbf{W}\sigma(\mathbf{x_{n_t-1}}) + \mathbf{W_{in}}\mathbf{u_{n_t}} + \mathbf{b} \tag{20}$$

Where $\mathbf{x_{n_t}}$ is the internal "hidden state" at step n_t, and θ denote learnable parameters, in this case the weight matrices $\mathbf{W}$, $\mathbf{W_{in}}$ and bias term $\mathbf{b}$. $\mathbf{u_{n_t}}$ is the input at step n_t and σ is an element-wise function introducing the non-linearity. Transferring this formulation to our problem, we can identify the following correspondences for the example of the *effective viscosity model*:
$\mathbf{u_{n_t}} = 0$ (no input), $\mathbf{x_{n_t}} = \nu_{\mathbf{n_t}}$ (scalar viscosity field). The non-linearity σ in this case encapsulated the solver scheme. It takes in a scalar viscosity field and a vector field of velocity $\mathbf{v}$ and computes the velocity field at the next timestep $\sigma(\mathbf{v_{n_t-1}}, \mu_{\mathbf{n_t-1}}) = \mathbf{v_{n_t}}$. Choose dimensions of $\mathbf{W}$ and $\mathbf{b}$ accordingly, such that

$$\mu_{\mathbf{n_t}} = \mathbf{W}\sigma(\mathbf{v_{n_t-1}}, \mu_{\mathbf{n_t-1}}) + \mathbf{b}. \tag{21}$$

compare this to equation 8 in [12]. We showed that training the correction model can be considered as training an RNN with no input and a very complex non-linearity. For the case of the simple RNN model, [12] show that, depending on the eigenvalues of $\boldsymbol{W}$, gradients through the model tend to *vanish* or *explode*. This property is grounded in the nature of the product of jacobians, propagating the error *through time*. In our case (eq. 15), this corresponds to the terms $\frac{\partial \boldsymbol{v}^{(T)}}{\partial \boldsymbol{v}^{(n_t)}} = \prod_{n_t<i<T} \frac{\partial \boldsymbol{v}^{(i)}}{\partial \boldsymbol{v}^{(i-1)}}$. The applicability of the proof of exploding gradients to the case of correction models studied in this work depends on the properties of the solver non-linearity, specifically wether its derivative is bounded. Instead of formally proving the applicability, we will argue from an experimental point of view and observe the temporal components of the gradients. In figure 15 we show the magnitudes of the temporal components $(\hat{v}_i^{(n_t)} - v_i^{n_t})^2$ increasing exponentially. This will lead the overall gradient to be generally dominated by the loss component corresponding to the largest temporal component. By gradient clipping, this explosion of gradients can be mitigated to a certain degree, which is why we employ gradient clipping throughout our experiments. As mentioned in section 6.2 however, very large T lead to numerical overflow in some components, making it impractical to calculate loss functions over arbitrary large time intervals.

D) Structure of training data

The simulation data of the different designs are contained in four folders which correspond to four the inlet velocities. Each folder contains 5,005 sub-folders corresponding to each design. Again moving into a simulation folder of a design, we have the result of the simulation stored in a *.CSV file as well as the visualization of the result in a *.png file. We can visualize the structure in the figure 16.

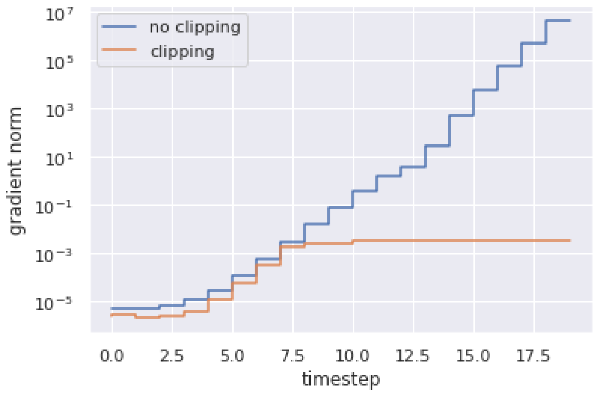


Figure 15: Temporal contributions of different timesteps to the gradient with and without gradient clipping. The cumulative sum of the shown contributions equals the loss with *intermediate* temporal targets (eq. 19.)

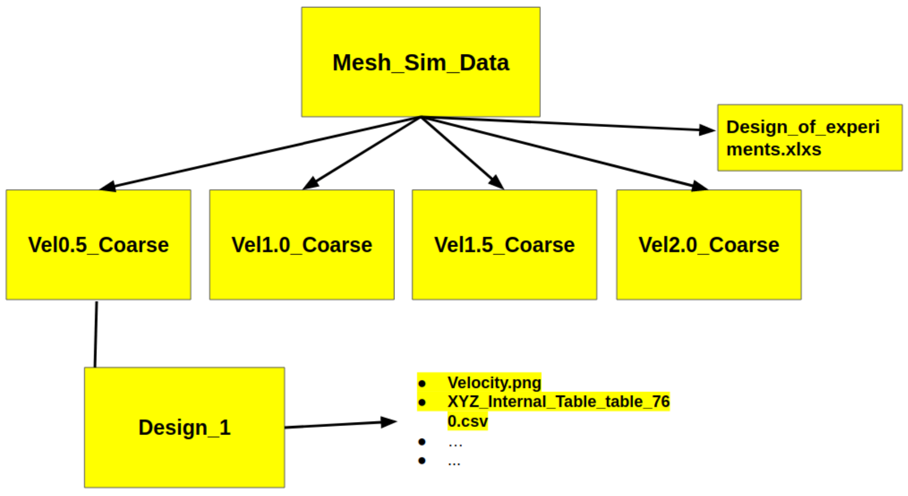


Figure 16: Structure of the training data folder

E) Weighted Least Squares

Let us consider the problem of fitting a function to a data set in 1D. The point coordinates are given by x_i, where $i = 1...n$, each of which is associated with a scalar value y_i. We try to find a fitting function $g(x)$ given by equation 22

$$g(x) = \sum_{j=1}^{k} b_j(x)c_j \tag{22}$$

It can also be written in the vector form as

$$g(x) = \boldsymbol{b}(x)^T\boldsymbol{c}$$

with polynomial basis vectors $\boldsymbol{b}(x) = [b_1(x), b_2(x)..., b_k(x)]^T$ of degree k and unknown coefficient $\boldsymbol{c} = [c_1, ..., c_k]^T$. The Least Squares (LS) approach finds these coefficient by solving the optimization problem

$$\min_{\boldsymbol{c}} \sum_{i=1}^{n} ||g(x_i) - y_i||^2.$$

The solution of the problem is

$$\boldsymbol{c} = (\boldsymbol{B}^T\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{g}$$

where matrix $\boldsymbol{B}$ is given by $\begin{bmatrix} \mathrm{b}_1(x_1) & \mathrm{b}_2(x_1) & \ldots & \mathrm{b}_k(x_1) \\ \vdots & \ddots & & \vdots \\ \vdots & & & \vdots \\ \mathrm{b}_1(x_n) & \mathrm{b}_2(x_n) & \ldots & \mathrm{b}_k(x_n) \end{bmatrix}$ and $\boldsymbol{g}$ is a vector of known function values.

In WLS, instead of finding a global function $g(x)$, we tend to go local. We are interested in finding the function values at points $\hat{x}$, which in our case correspond to the coordinates of the regular grid. Between this fixed point and any other point x_i, we will define a weighting function $W(d_i)$ as a function of the distance between these two points points $d_i = ||\hat{x} - x_i||$. The objective function is slightly changed and added with weight vector as shown in the equation 23

$$\min_{\boldsymbol{c}} \sum_{i=1}^{n} W(d_i)||g(x_i) - g_i||^2 \tag{23}$$

F) Supplementary Results

Training from Initial Steady State

The following section presents more detailed experimental results for the training setting in which the initial velocity field is set to the target velocity field. An optimal model is therefore expected to keep the flow field close to the initial state for all times. As described in section 7.4, the initial velocity field and target are first projected onto the space of divergence free fields. Figure 17 shows the velocity fields before and after this step.

Figure 18 shows two exemplary *roll-outs* of the two trained correction models. Both models were trained in the same setting with the same hyper-parameters i.e. Adam optimizer, a learning rate of $0.5 \cdot 10^{-3}$ and an architecture as described in 6.1 with parameters $k = 5$, $n = 8$, and three hidden layers. Layer wise gradient clipping with a threshold of 0.001 was applied. We don't expect these hyper-parameters to be the most optimal in general, although we found them to work reliably across training runs. To limit the computational effort, the dataset used herein consisted of a subset of the large dataset discussed in section (4), which we found to include a representative cross section of training samples encountered in the main dataset. During training time, only the first 5 consecutive time-steps for each training sample were shown to the model. The *roll-outs* in figure (18) are therefore extrapolating, both in the temporal dimension ($n_t > 5$) as well as regarding the actual geometry from the evaluation set. In these experiments we find that the *effective viscosity model* tends to exhibit better generalization properties then the *residual model.*

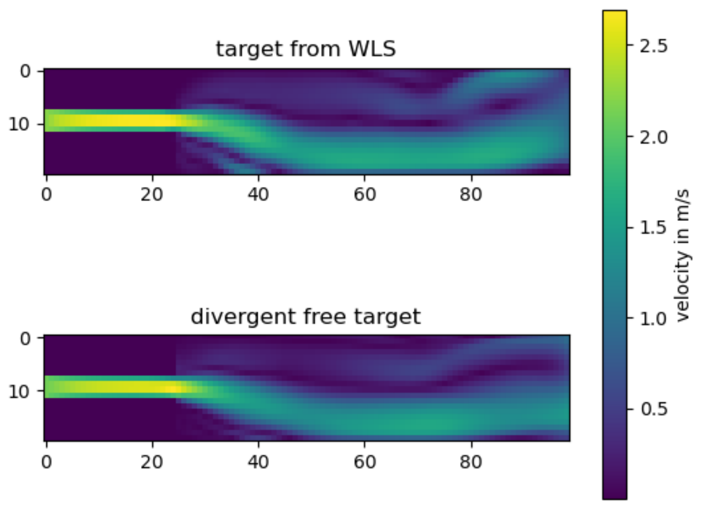
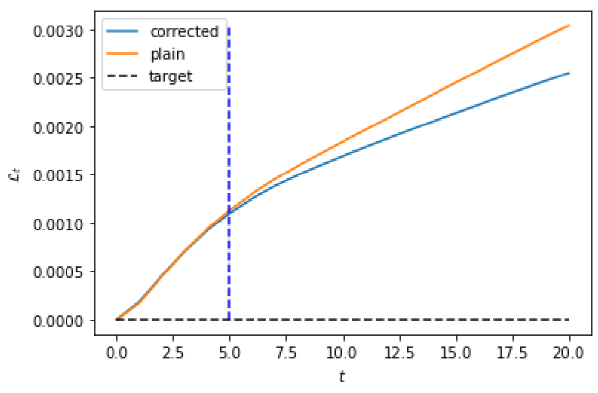
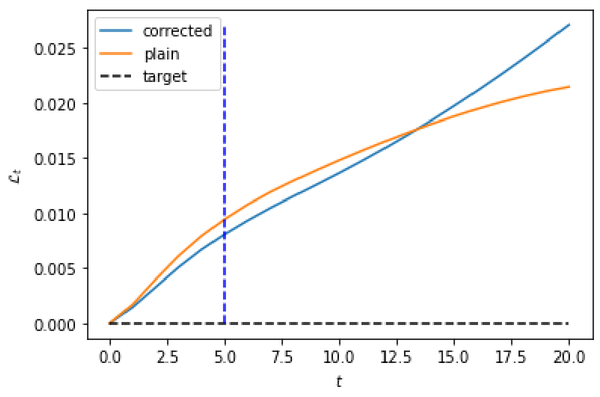


Figure 17: Velocity fields obtained directly as the weighted least squares fit (top) and divergent free target (bottom), which is also the initial velocity field for the experiments in this section.

As seen in figure 18, while the *residual model* is able to reduce the evaluation loss over the plain simulation for time-steps up to ~ 13, it generally fails to extrapolate beyond that. The *effective viscosity model* on the other hand improves the predictions far beyond the time-frames seen during training.
Figure 19 shows the resulting flow fields for the case of the *effective viscosity model*. By observing the difference between the plain and the corrected solutions (bottom), we see that the model predominantly learned to correct the boundary regions around the nozzle and inflow. This is actually expected, as the plain coarse simulation is unable to resolve these boundary regions with the same accuracy as the target solution. As seen in figure 4, the method which produced the target data introduces more mesh elements around these regions to accurately resolve the boundary layers. Although the trained model clearly improves the accuracy of the solver over the plain simulation, we think that the training setting described here is in general sub-optimal, considering that the model still notably deviates from the target state.



(a) correction by *effective viscosity model*



(b) correction by *residual model*

Figure 18: Roll-outs of the *effective viscosity model* (a) and the *residual model* (b) on two evaluation cases unseen during training. Both models were trained on the same setting, i.e. same hyper-parameters and training set. The training data included $T = 5$ consecutive time-steps for both cases only, as indicated by the dashed blue line.

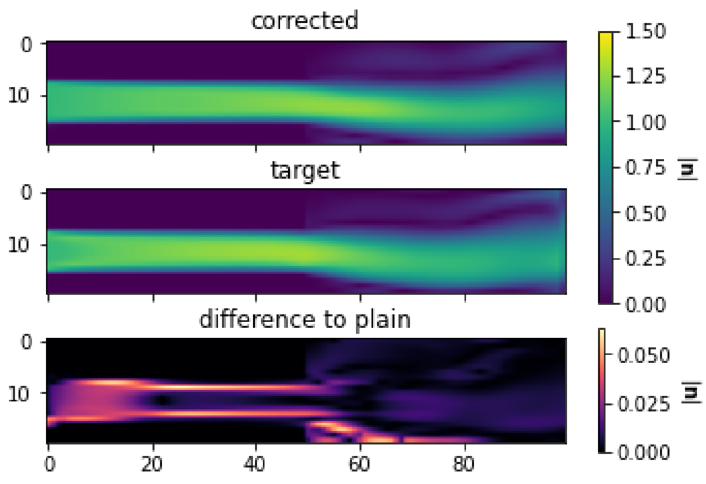


Figure 19: Exemplary final flow fields from the corrected simulation (*effective viscosity model*), the target and the absolute difference in flow fields between the corrected and the plain simulation.