Machine Learning Accelerators for 3D Physics Simulations

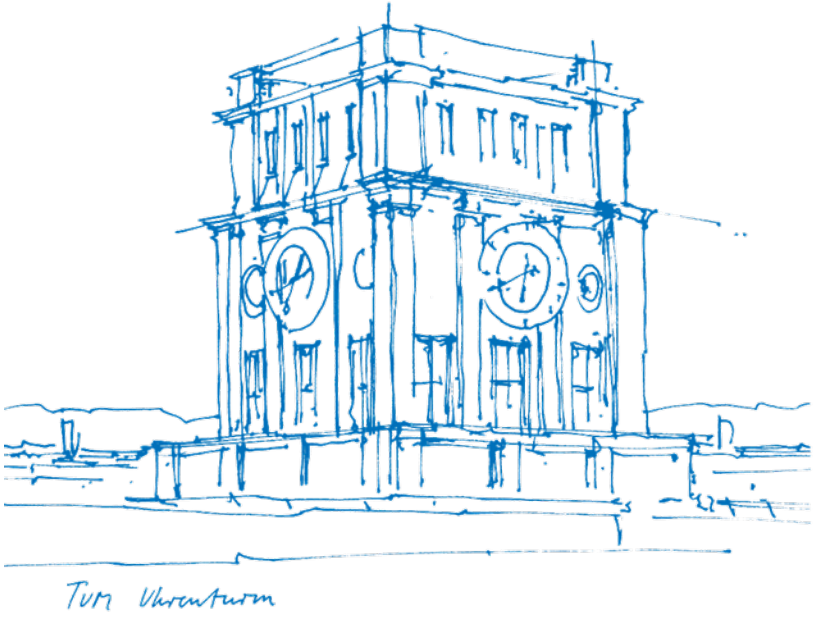
TUM Data Innovation Lab

Yi-Han(Kimi) HSIEH[1], Aman KUMAR[1], Lennart RÖSTEL[1], Saad SHAMSI[2]

[1]Department of Informatics, Technical University of Munich (TUM)

[2]Department of Mathematics, Technical University of Munich (TUM)

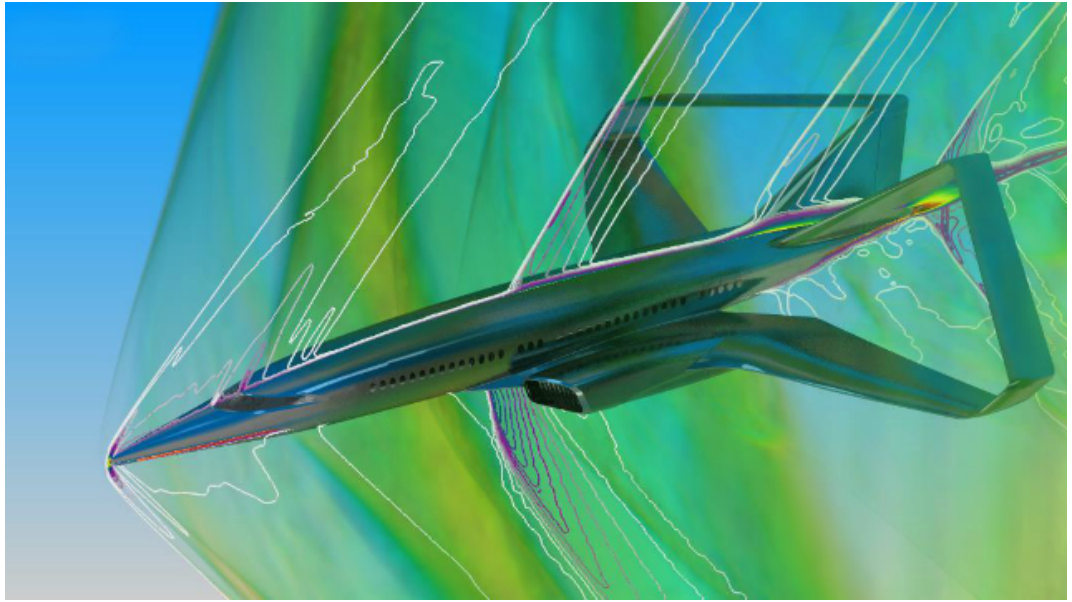
February 25, 2021



Roadmap

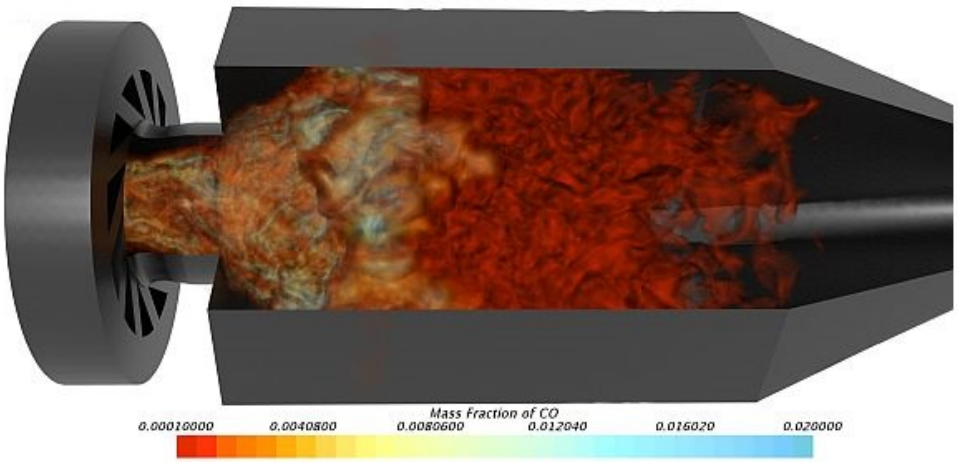
- **Motivation**
- Navier-Stokes Equations and Turbulence
- Correction Functionals
- Computational Strategy
- Data Preparation
- Differentiable Solvers
- Simulation of Navier-Stokes Flow
- Final Experiments
- Conclusions

Aerodynamic Analysis



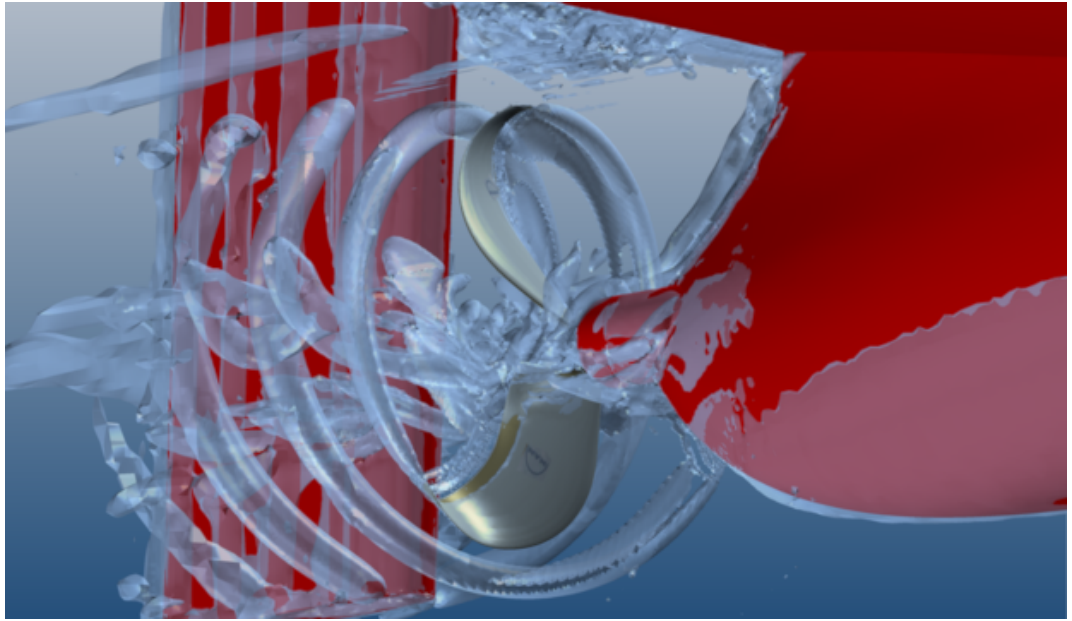
Source: Simcenter STAR-CCM+

Reacting Flows



Source: Simcenter STAR-CCM+

Cavitation Analysis



Source: Simcenter STAR-CCM+

Motivation

CFD simulations require a trade-off – **Resolution vs. Speed**:

- **Low-Res + High Speed**: No Micro-Scale Details
- **High-Res + Low Speed**: Computationally Expensive

Ideal Simulation = **High Resolution + High Speed**

Plan from **SIEMENS**

Goals:

- Develop a CFD pipeline that operates on coarse discretisations
- Capture the cumulative effects of micro-scale phenomena on the macro-scale
- Lower computational cost

Computational Approach:

- Construct **Differentiable Solver Pipeline**
- Learn **Correction Functionals**
- Assess **Solver Suitability**

Roadmap

- Motivation
- **Navier-Stokes Equations and Turbulence**
- Correction Functionals
- Computational Strategy
- Data Preparation
- Differentiable Solvers
- Simulation of Navier-Stokes Flow
- Final Experiments
- Conclusions

Theory: Navier-Stokes Equations

Conservation of Mass

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

Conservation of Momentum

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

The Trouble with Turbulence

Most problems exhibit **turbulence** characterised by:

- **aperiodic motion**
- **random spatial variations**
- **instability**
- **phenomena at multiple length-scales**

Turbulence problems are typically studied within a **stochastic framework**

The Trouble with Turbulence

- Direct Simulations: **High Resolution** in *space* and *time*
- Applications: *Coarse Picture* + *Time-Averaged*

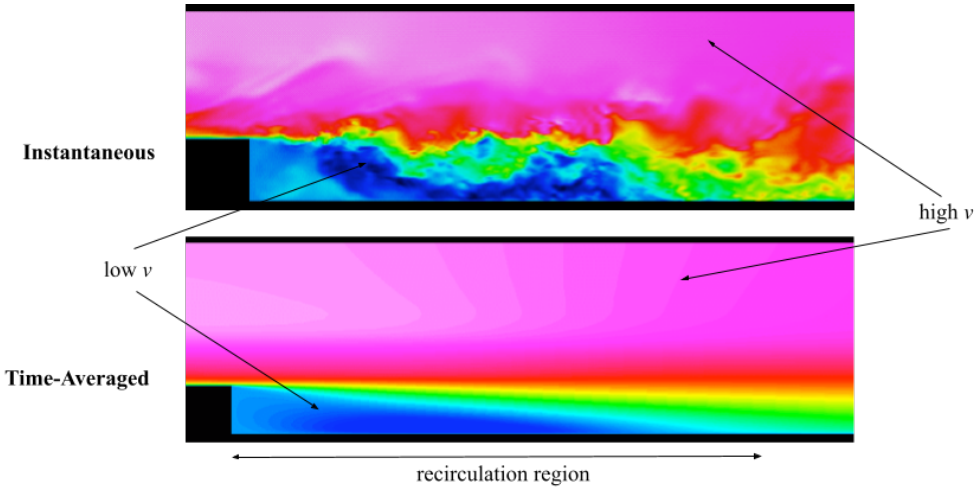


Figure: Instantaneous and Time-Averaged flows over a backstep. Only the recirculation region is of interest. Ramsai 2020.

The Trouble with Turbulence

To extract *coarse-scale* and *time-averaged* features use Reynolds-Averaged Quantities:

$$v(x,t) = \bar{v}(x) + \breve{v}(x,t) \tag{3}$$

This yields the **Reynolds-Averaged Navier Stokes** (RANS) equations:

$$\frac{\partial \bar{v}_i}{\partial t} + \bar{v}_j \frac{\partial \bar{v}_i}{\partial x_j} = \bar{f}_i - \frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial^2 \bar{v}_i}{\partial x_j \partial x_j} + \frac{\partial \boxed{-\overline{\breve{v}_i \breve{v}_j}}}{\partial x_j} \tag{4}$$

Closure Problem

- Non-Linearity $\rightarrow -\check{v}_i\check{v}_j$ in RANS equations $\rightarrow$ **Closure Problem**
- $R_{ij} = -\overline{\check{v}_i\check{v}_j}$ is the Reynolds stress
- *Turbulence Modelling*: $R_{ij} = R_{ij}\left(\bar{v}, \bar{p}\right)$
- *Turbulence Modelling*: **correction functionals** *correct* for $R_{ij} = -\overline{\check{v}_i\check{v}_j}$

Can we *learn* the turbulence model?

Roadmap

- Motivation
- Navier-Stokes Equations and Turbulence
- **Correction Functionals**
- Computational Strategy
- Data Preparation
- Differentiable Solvers
- Simulation of Navier-Stokes Flow
- Final Experiments
- Conclusions

Turbulence Models: Correction Functionals

Effective Viscosity Model

- Extend the diffusion term $\nu \nabla^2 v$ by introducing a correction: $\nu_\theta(v)$
- Reproduces the cumulative effects of small scale vortices

$$\boxed{\nu \rightarrow \nu_0 + \nu_\theta(v)} \tag{5}$$

Residual Model

- Augment the external forces f by introducing a correction: $f_\theta(v)$
- Analogous to the control term introduced in Holl, Koltun, and Thuerey 2020

$$\boxed{f \rightarrow f_0 + f_\theta(v)} \tag{6}$$

Turbulence Models: Our Approach

Our implementation of correction functionals distinguishes itself in terms of:

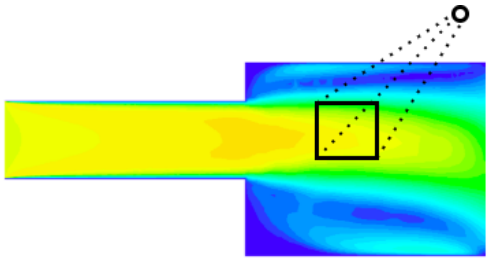
- **Spatial Locality**: Corrections act on the *neighbourhood* of a point
- **Temporal Independence**: Corrections propagate through the entire time domain
- **Strong Coupling**: Future dynamics incorporated into corrections to previous times

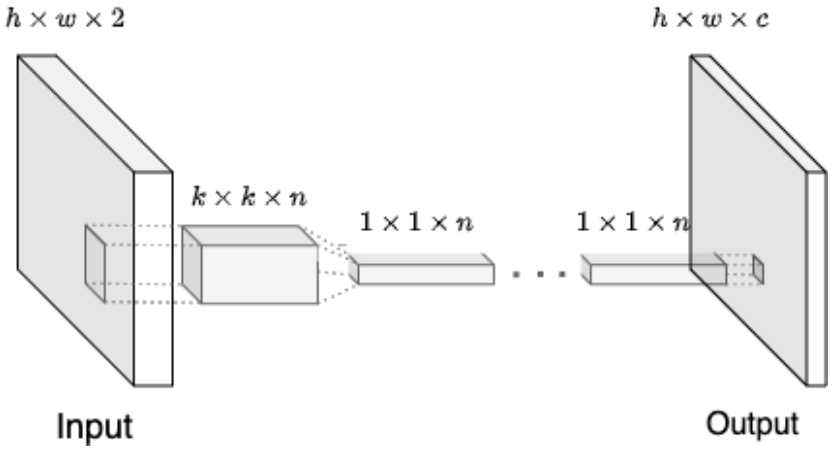
Roadmap

- Motivation
- Navier-Stokes Equations and Turbulence
- Correction Functionals
- **Computational Strategy**
- Data Preparation
- Differentiable Solvers
- Simulation of Navier-Stokes Flow
- Final Experiments
- Conclusions

Network Architecture

Implementing the **locality assumption** into the NN architecture (inductive bias).
We propose a *local* variant of a fully convolution neural network (FCNN):



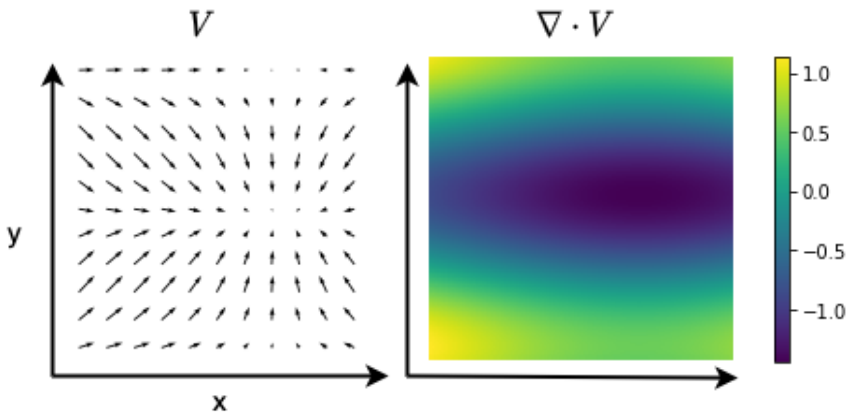


Learning Operators on Vector Fields

Employ local FCNN for *learning stencils*.
Preliminary study on “mockup” vector field:

$$V : \mathbb{R}^2 \to \mathbb{R}^2 \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} \sin(\omega_x x + \phi_x) \\ \cos(\omega_y y + \phi_y) \end{pmatrix}$$

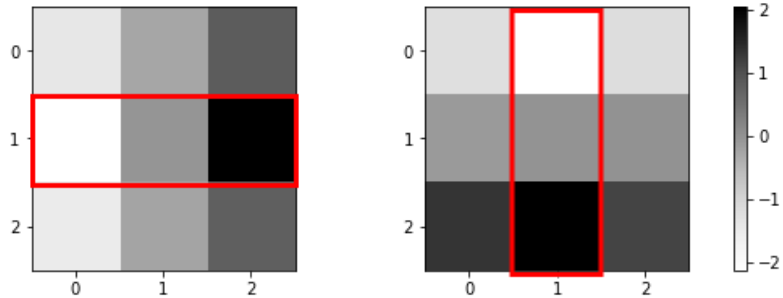
For this simple case, we can analytically calculate the effect of different operators. By varying ω and ϕ we create a training dataset.



Learning Operators on Vector Fields

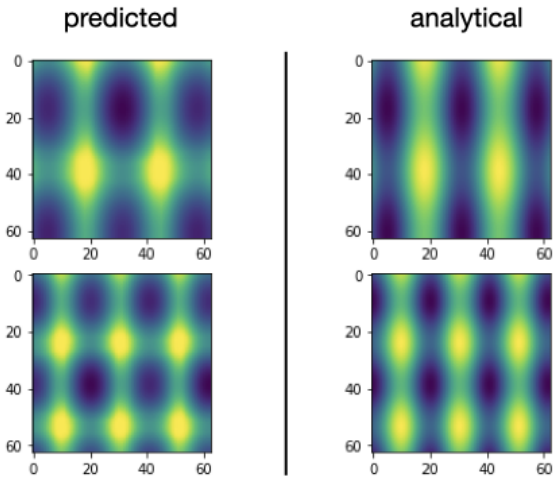
Divergence: $\nabla \cdot V$

Learned kernels resemble central differences:



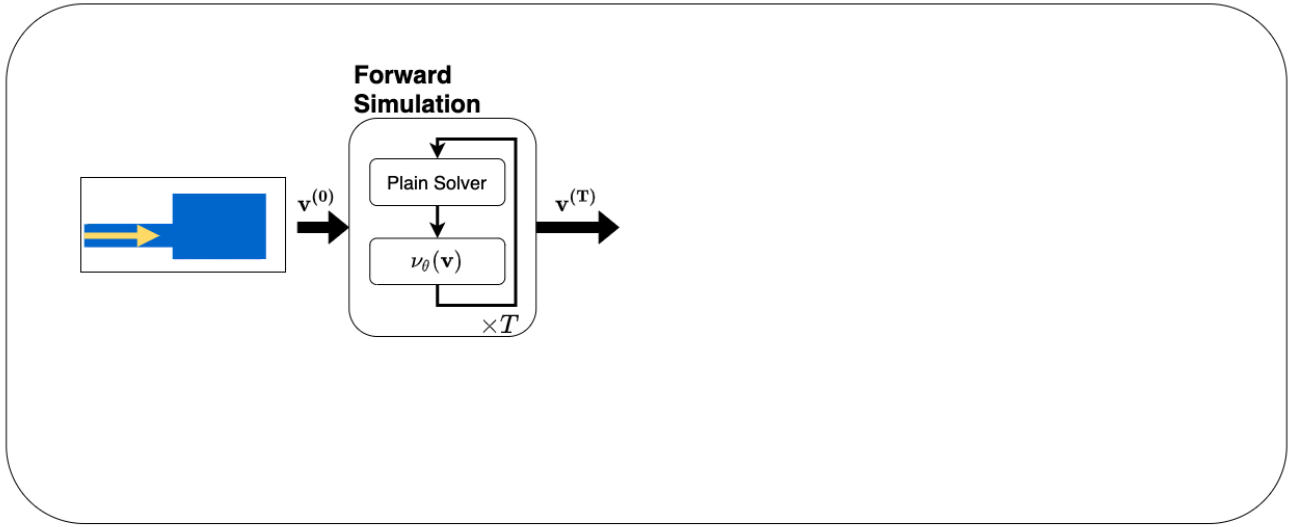
$$\approx \frac{dV}{dx} \qquad \approx \frac{dV}{dy}$$

Non-linear Operators, e.g. $V \cdot \nabla^2 V$

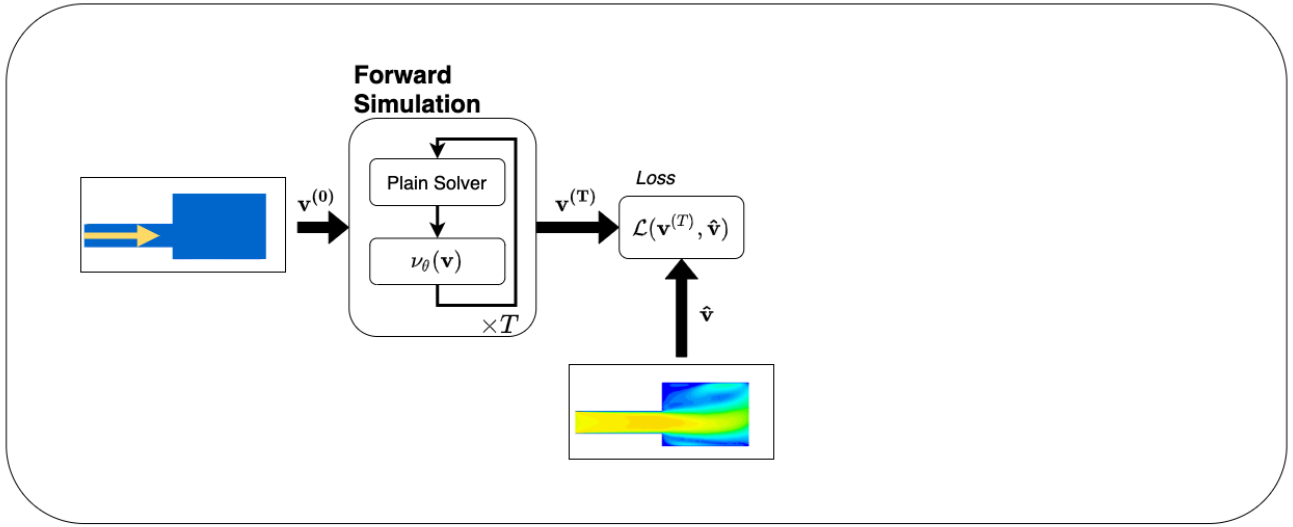


Conclusion: Local FCNN is able to resemble operators based on **local predictions**.

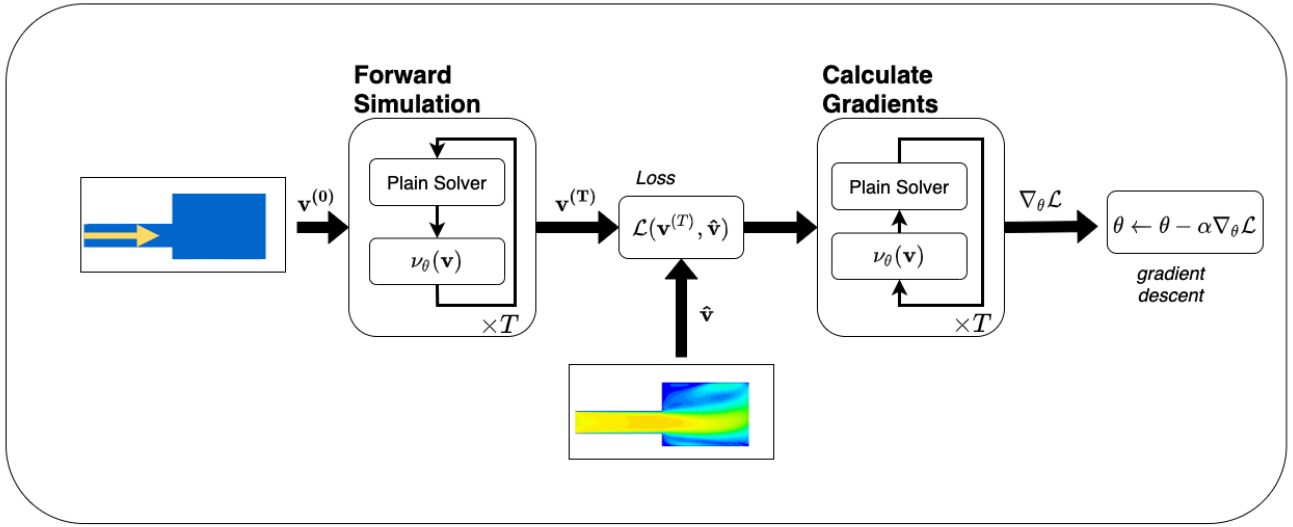
Parameter Optimization - Pipeline



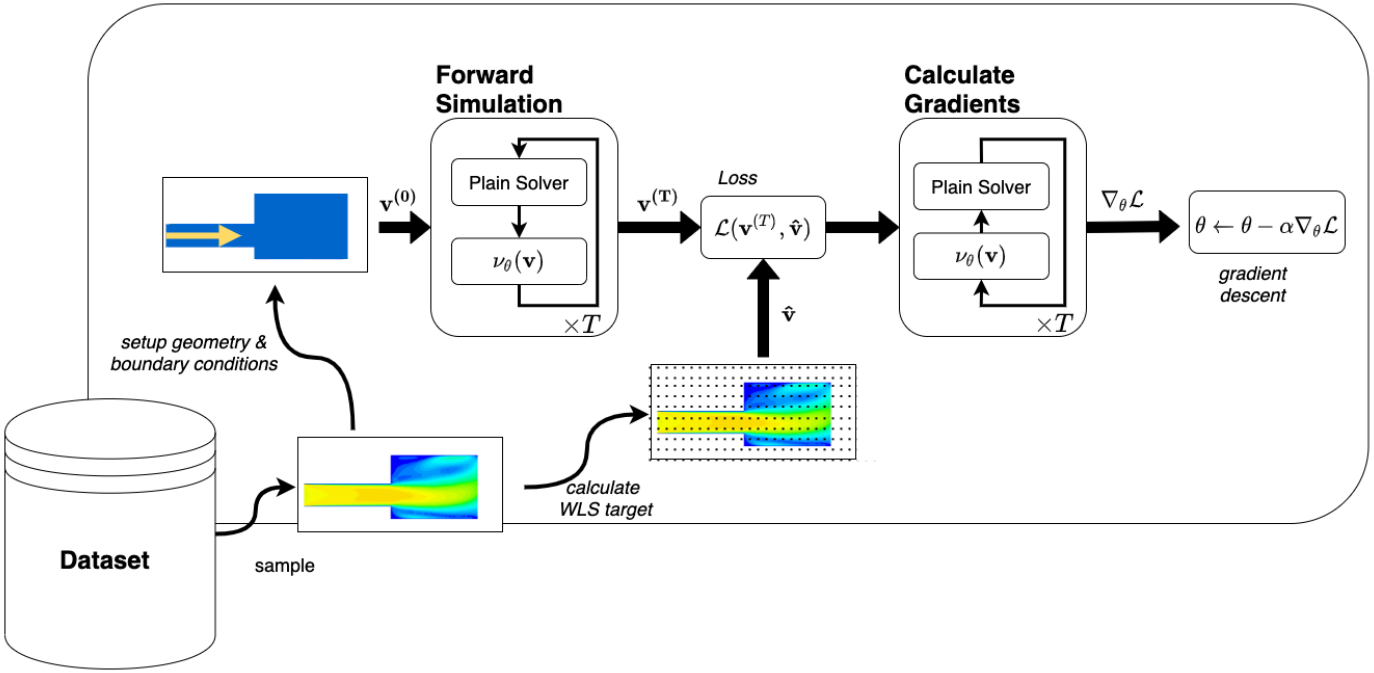
Parameter Optimization - Pipeline



Parameter Optimization - Pipeline



Parameter Optimization - Pipeline



Roadmap

- Motivation
- Navier-Stokes Equations and Turbulence
- Correction Functionals
- Computational Strategy
- **Data Preparation**
- Differentiable Solvers
- Simulation of Navier-Stokes Flow
- Final Experiments
- Conclusions

Data Preparation

A	B	C	D	E	F
Name	channel height	ratio_height_chl_nzl	ratio_nzl_to_floor	ratio_width_chl_nzl	Vin
Design 1	10	0.1	0.2	0	1
Design 2	10	0.1	0.2	0.05	1
Design 3	10	0.1	0.2	0.1	1
Design 4	10	0.1	0.2	0.15	1
Design 5	10	0.1	0.2	0.2	1
Design 6	10	0.1	0.2	0.25	1
Design 7	10	0.1	0.2	0.3	1
Design 8	10	0.1	0.2	0.35	1
Design 9	10	0.1	0.2	0.4	1
Design 10	10	0.1	0.2	0.45	1

Figure: Design parameters to generate various designs

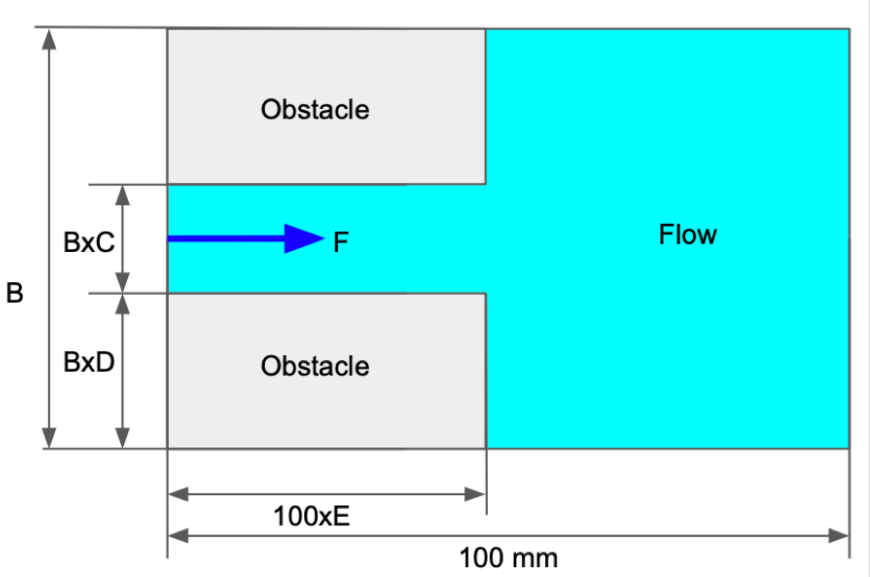


Figure: Corresponding Geometry

Design Visualization

Simulations obtained from the Simcenter STAR-CCM+ by Siemens

Data Structure

	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

Figure: A sample dataframe containing the result of simulation

$\rightarrow$ Design folder containing simulation results contained in files named **Velocity.png, *.csv** $\rightarrow$ CSV file containing the results at the points in the simulation domain

$\rightarrow$ Processing required to get the underlying structure of data

Data Processing

→ CSV file contains velocity values at unstructured grid points

→ Need to retrieve velocity values for the structured grid points as required by the solver

→ Weighted Least Square approach to fit the data

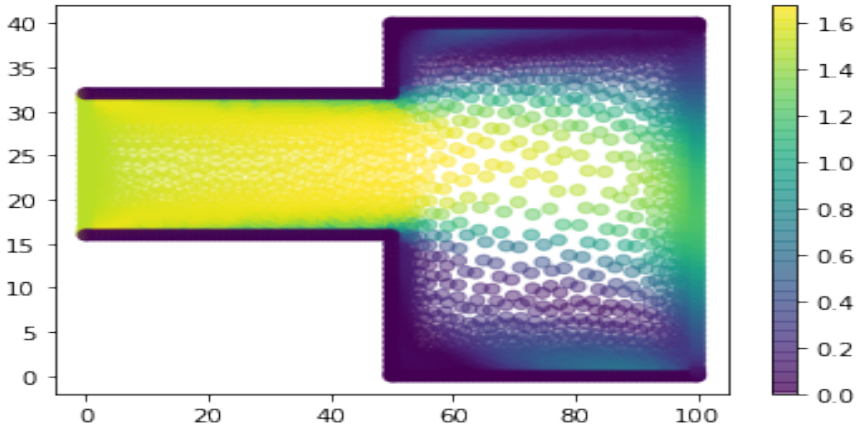


Figure: Actual velocity on the unstructured grid

Weighted Least Square

$\rightarrow$ More suitable as the velocity at the structured grid needs to be close to nearest unstructured point

$\rightarrow$ Optimization function is

$$\min_{c} \sum_{i}^{n} W(d_i)||g(x_i) - y_i||^2$$

$\rightarrow$ d_i is distance of structured grid point from ith point in the unstructured grid

$\rightarrow$ $g(x) = \mathbf{b(x)}^T\mathbf{c}$ where b(x) is polynomial basis function of degree k

$\rightarrow$ Weight function is

$$W(d) = e^{-\frac{d^2}{\sigma^2}}$$

$\rightarrow$ Two hyper-parameters to optimize : **degree** and σ

Parameters of WLS

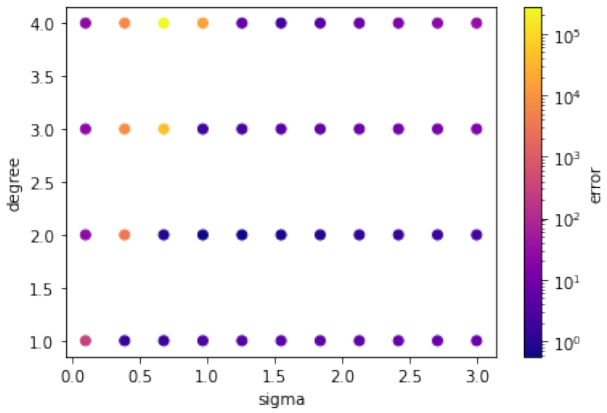


Figure: Error plots for various combinations of degree and σ

$\rightarrow$ Hyper parameter search for degree and σ

$\rightarrow$ Best combination is degree = 2 and $\sigma = 1$

Target computation

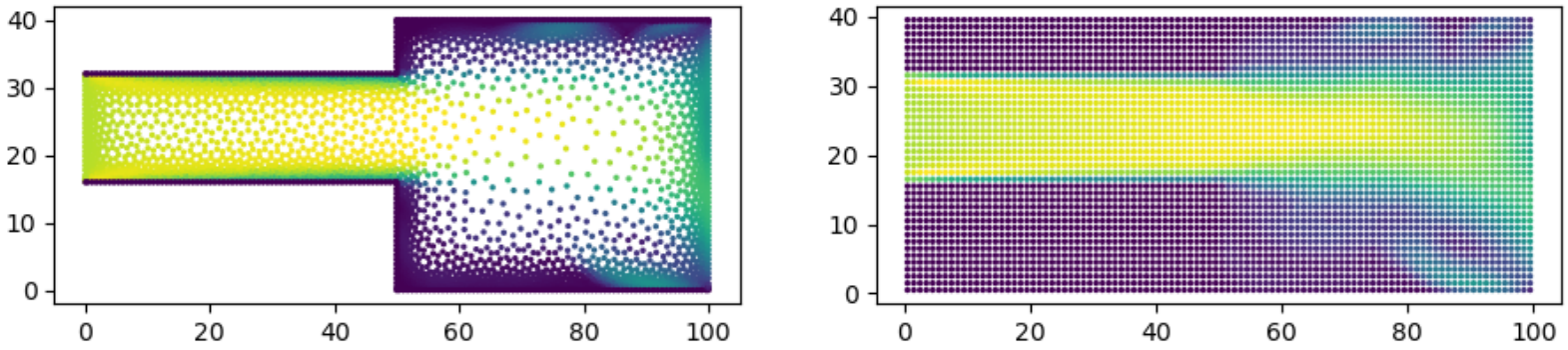
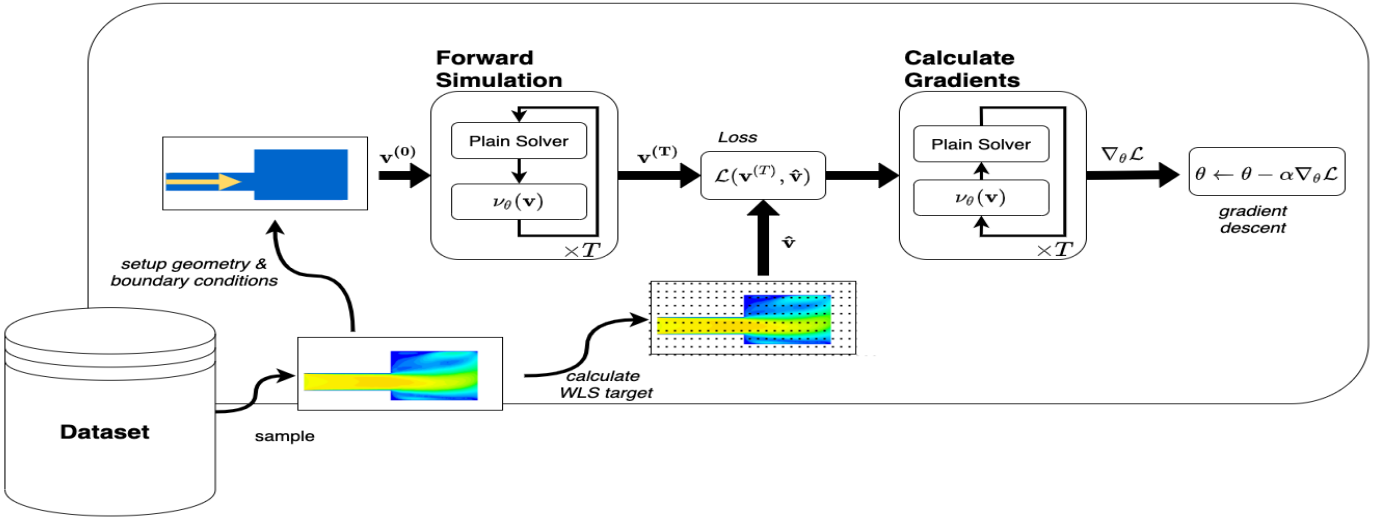


Figure: In left we see the actual velocity provided and on the right interpolated velocity using the best sigma and degree, used as target for the model.

Integration with the pipeline

→ Processing provides the target velocity for the learning process

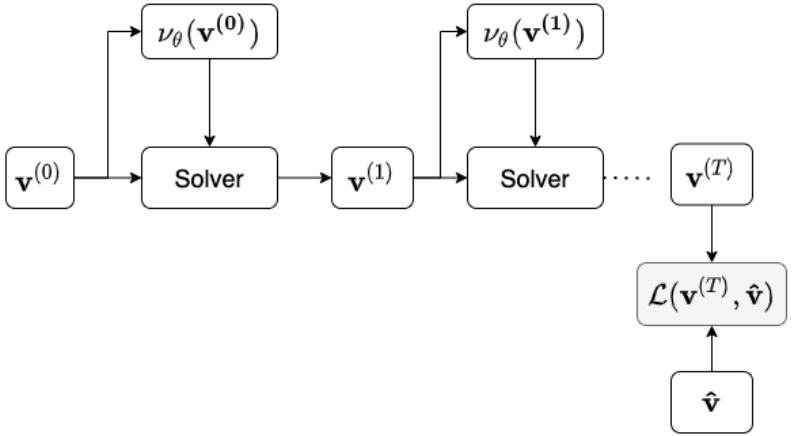
→ Provides the parameters and boundary conditions for a particular design as shown in the figure



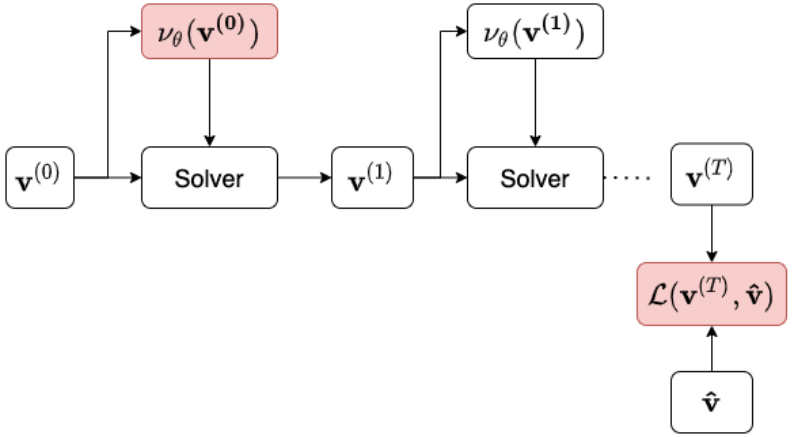
Roadmap

- Motivation
- Navier-Stokes Equations and Turbulence
- Correction Functionals
- Computational Strategy
- Data Preparation
- **Differentiable Solvers**
- Simulation of Navier-Stokes Flow
- Final Experiments
- Conclusions

Timestepping with Correction

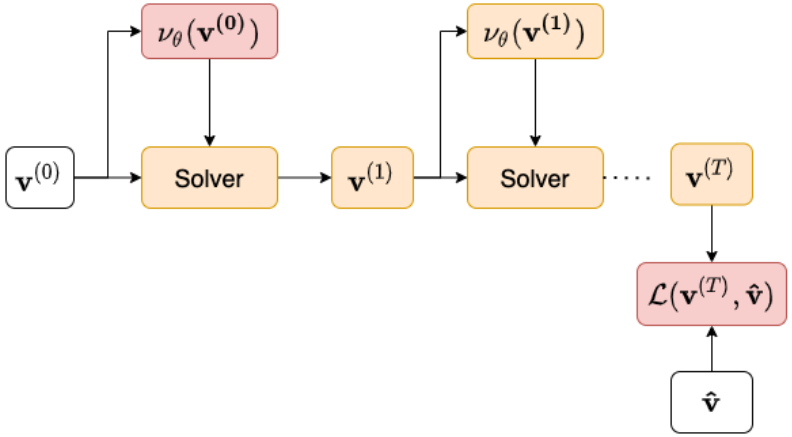


Timestepping with Correction

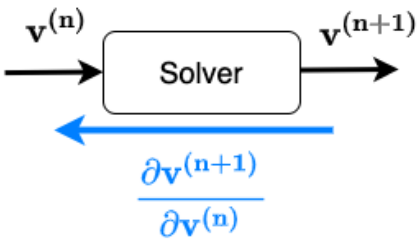


We optimize parameters θ by gradient-based methods.
$\rightarrow$ find $\nabla_\theta \mathcal{L}$!
Or: How does the correction at step n influence the loss at step T?

Timestepping with Correction



We optimize parameters θ by gradient-based methods.
$\rightarrow$ find $\nabla_\theta \mathcal{L}$!
Or: How does the correction at step n influence the loss at step T?
$\rightarrow$ Gradients ***through the solver!***



Differentiable CFD solvers

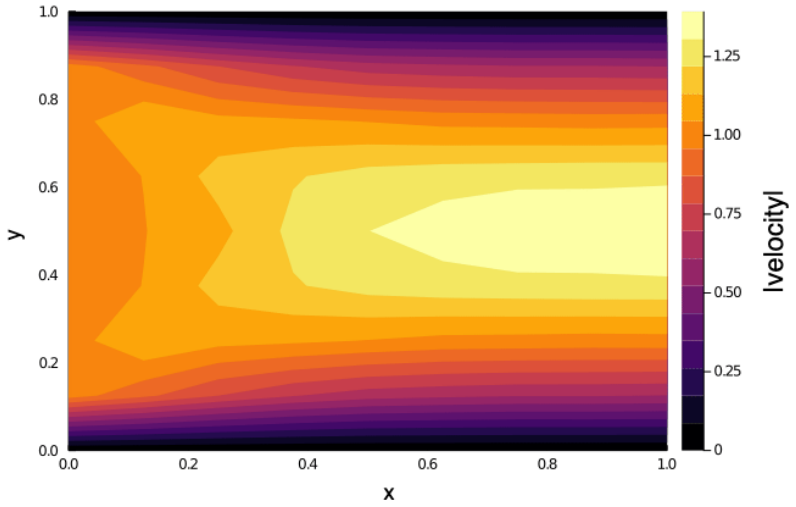
We need to calculate derivatives *through the solver* $\rightarrow$ differentiable CFD solver.

We **employ and test** two solver implementations:

	Φ_{Flow}	PP-solver
Method	Finite-Difference	Finite-elements
Algorithm	$\sim$*stable fluids* [Stam 1999]	PP-algorithm [Helmich et al. 2018]
Differentiable?	yes	no

PP-solver

- MATLAB reference implementation provided by Siemens.
- We **completely re-implement** the solver in JULIA.
- We employ **automatic differentiation** (AD)
- Forward Mode AD, due to memory efficiency.

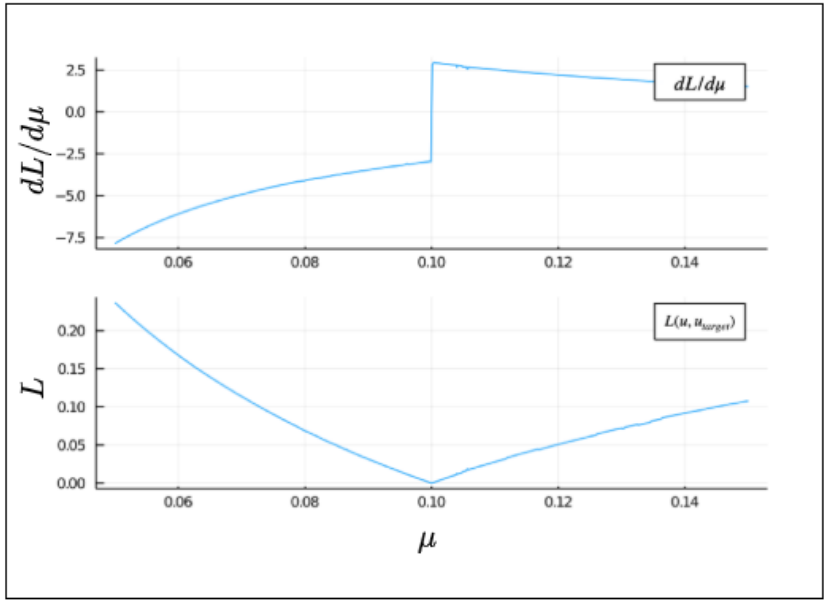


$\rightarrow$ We produce an end-to-end differentiable implementation.

PP-solver - Proof of Implementation

We **verify our implementation** of the *forward differentiable* solver:

- Set up small pipe flow simulation.
- Run *target simulation* with viscosity $\mu = 0.10$. Obtain target velocity field $\hat{\mathbf{v}}$ at $T = 10$.
- Run 1000 simulations with varying μ. For each simulation calculate:
  - Final state $\mathbf{v}^{(T)}$
  - Loss $L = ||\mathbf{v}^{(T)} - \hat{\mathbf{v}}||_2$
  - Derivative $dL/d\mu$



PP-solver - Challenges

- We implement the (residual) correction model into PP-solver.
- We optimize weights *through the solver* by gradient descent.
- We face two main challenges:

1. **Computational Inefficiencies**
2. **Convergence** depends on current correction parameters.
   Correction introduces **perturbance of the physical state**.
   $\rightarrow$ Optimization by gradient descent empirically difficult.

Conclusion: We concentrate our efforts on Φ_{Flow}.

Φ_{flow}

Philipp Holl 2020

- Invented by Nils Thuerey Group in TUM I15
- A differentiable and open-source physics simulation library.
- A Navier-Stokes stable fluids-type solver.
- Using the Structured-Staggered-Grid system.
- Integration with TensorFlow allowing for straightforward neural network.

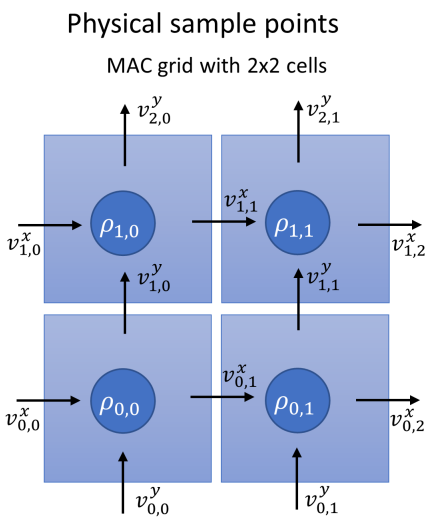


Figure:

Roadmap

- Motivation
- Navier-Stokes Equations and Turbulence
- Correction Functionals
- Computational Strategy
- Data Preparation
- Differentiable Solvers
- **Simulation of Navier-Stokes Flow**
- Final Experiments
- Conclusions

Simulation of Navier-Stokes Flow

plain simulation

- **Input**: geometry, boundary conditions, physical properties, grid resolution and timestep *dt*
- **Output**: flow field in steady state.

In general, we try to find the reasonable gird resolution, *dt* and *T*(#timesteps to reach steady state), we can apply them to forward simulation.

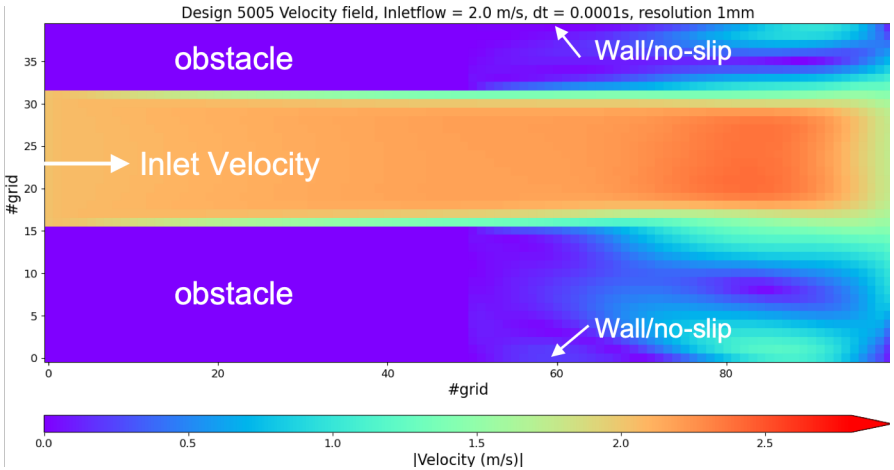


Figure: Velocity field plot

Meshing and resolution

All computations happen in the grid points. We would like all geometries to be placed precisely in the grid.

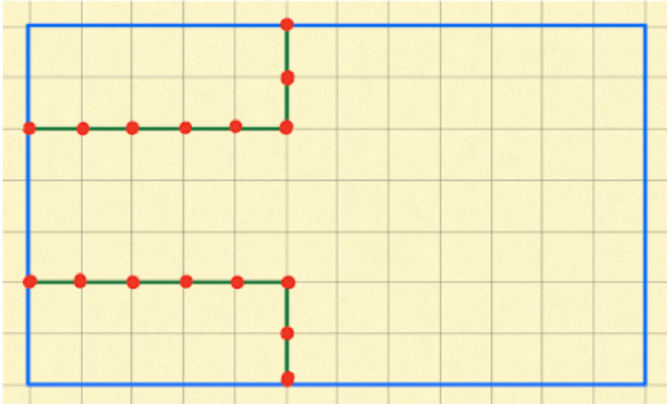


Figure: *on* grid

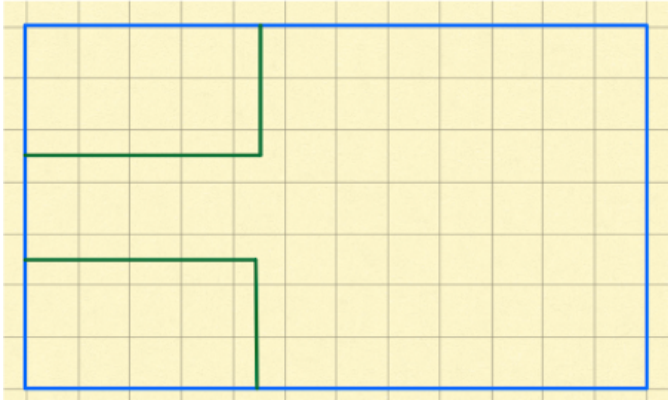
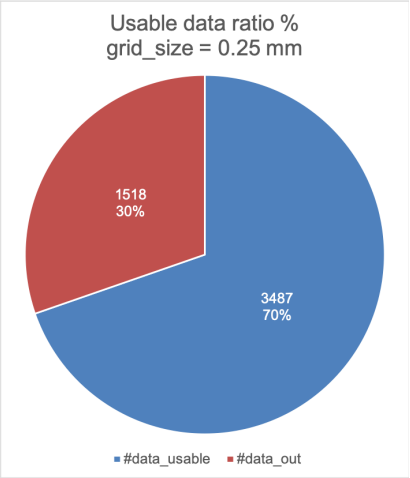
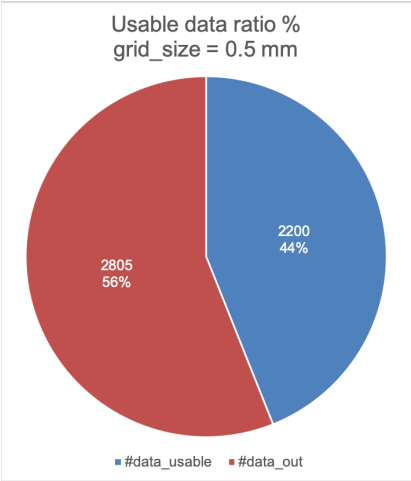


Figure: *in between* grid

1. Find the largest feasible grid resolution. i.e. the greatest common divisor, which is 0.125 mm.
2. Shortley-Weller scheme.
3. Fix a resolution and classify all designs into two groups,*on* or *in between* the grid points.

Meshing and resolution cont'd





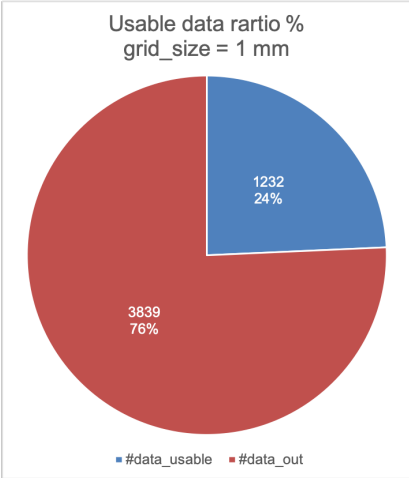


Table: Computation time required for 1000 time-steps in design 5005

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

Time stepping

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

$$|v_{x,max}|dt < \Delta x, \quad |v_{y,max}|dt < \Delta y \tag{7}$$

2. We define the condition for the steady state as : $|v^{(n_t+1)} - v^{(n_t)}| \simeq 0$.

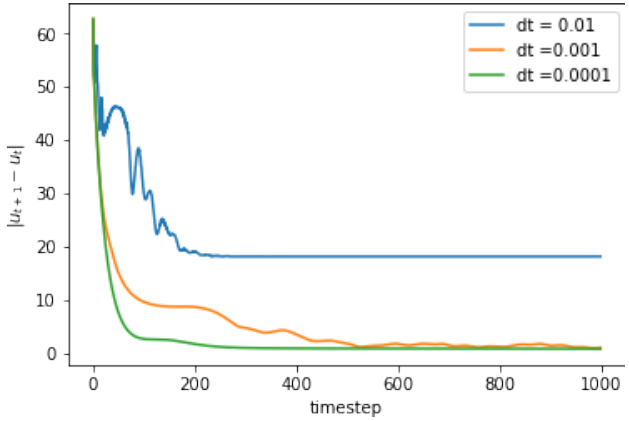


Figure: Design 4956

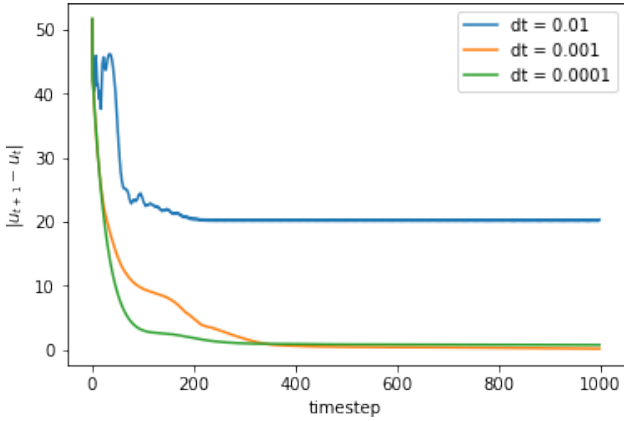


Figure: Design 5005

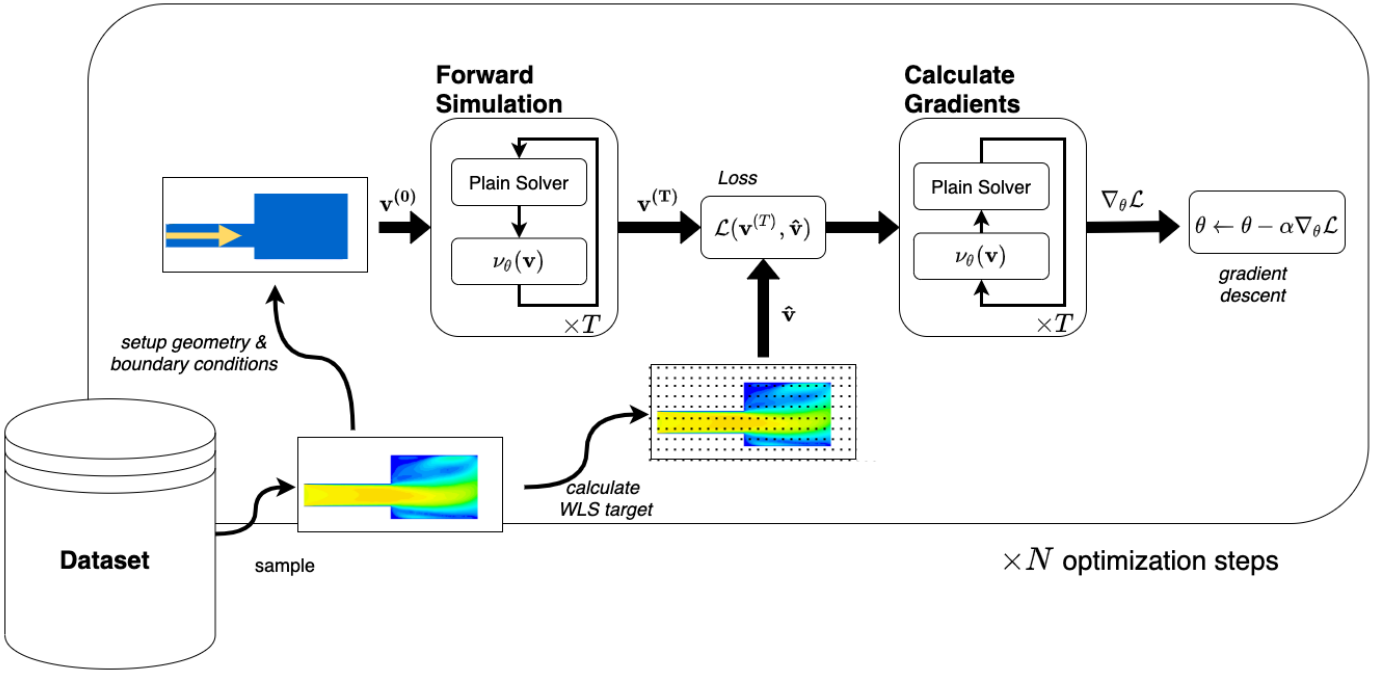
Summary of Simulation of Navier-Stokes Flow

- We have 1232 test desighs with resolution = 1mm, which saves compuataional effoet.
- $dt = 10^{-4}$ 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”.

Roadmap

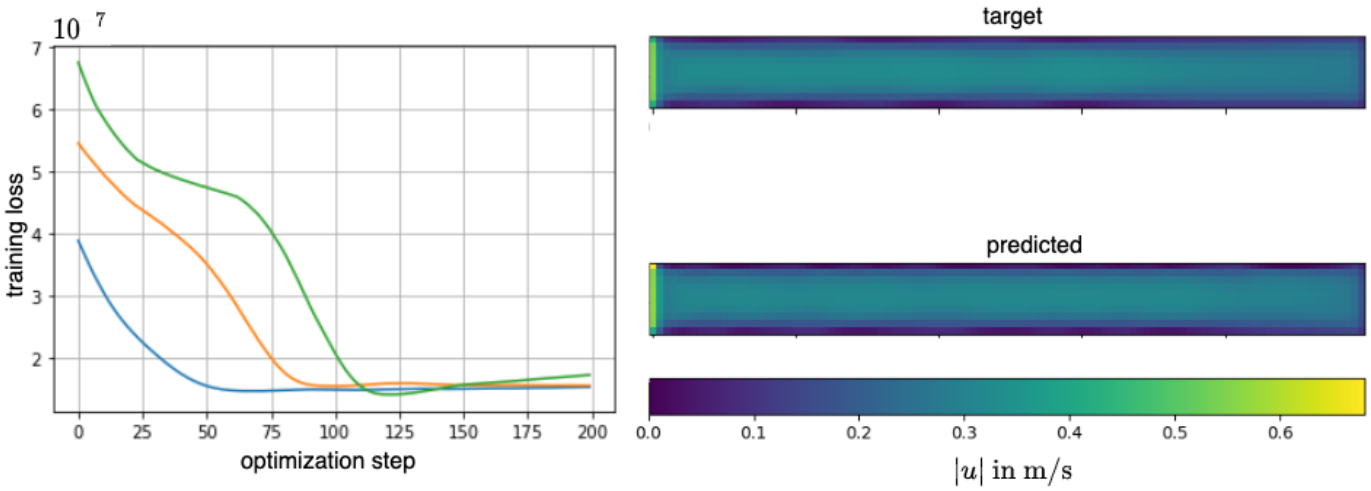
- Motivation
- Navier-Stokes Equations and Turbulence
- Correction Functionals
- Computational Strategy
- Data Preparation
- Differentiable Solvers
- Simulation of Navier-Stokes Flow
- **Final Experiments**
- Conclusions

Pipeline


Forward Simulation
Calculate Gradients
Plain Solver
$\nu_\theta(\mathbf{v})$
$\times T$
$\mathbf{v}^{(0)}$
$\mathbf{v}^{(T)}$
Loss
$\mathcal{L}(\mathbf{v}^{(T)}, \hat{\mathbf{v}})$
$\hat{\mathbf{v}}$
Plain Solver
$\nu_\theta(\mathbf{v})$
$\times T$
$\nabla_\theta \mathcal{L}$
$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
gradient descent
setup geometry & boundary conditions
calculate WLS target
Dataset
sample
$\times N$ optimization steps

Training Experiments - Proof of concept

We test the **training setup** (including simulation setup, solver, FCNN and optimization) by overfitting to a made-up training sample with fixed target viscosity and $T = 10$.



Training Experiments - Exploding Gradients

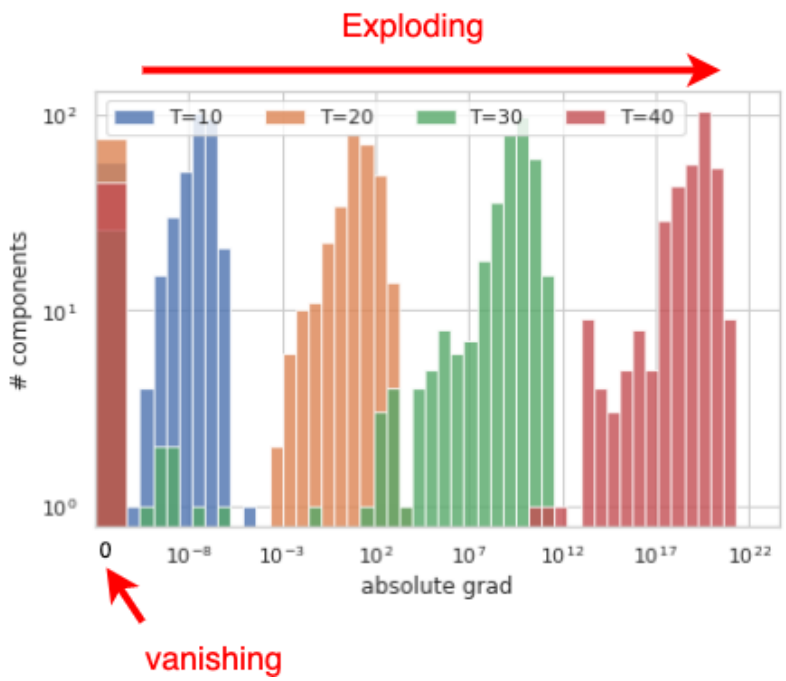
For increasing T, gradients tend to *vanish* or *explode*.

Problem occurs even with:

- careful weight initialization
- small learning rates
- very small networks

Gradient clipping only helps to a certain degree.

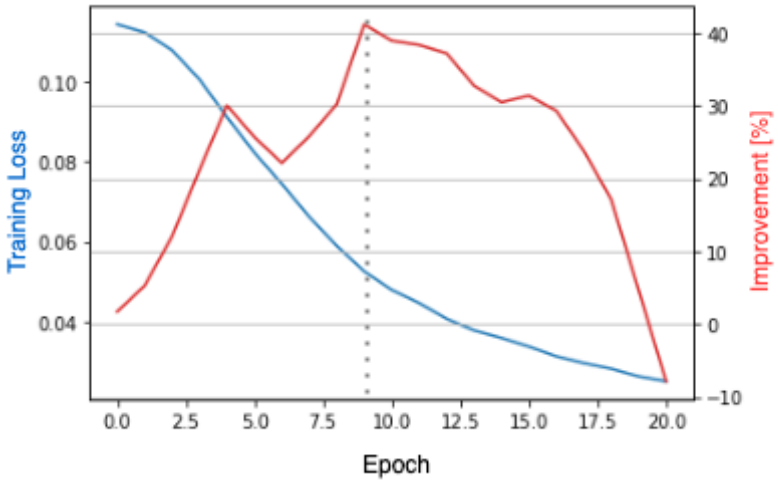
$\rightarrow$ training on only steady state ($T = 500$) not feasible

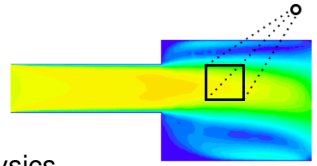


Alternative Settings - Transient Training

Intermediate target states are required:

- We *create* a small dataset of 5 high-resolution **transient simulations**
- Target simulation 125*computation time
- **Promising generalization** even with few training samples.
- Actually: Every simulations contains thousands of samples!



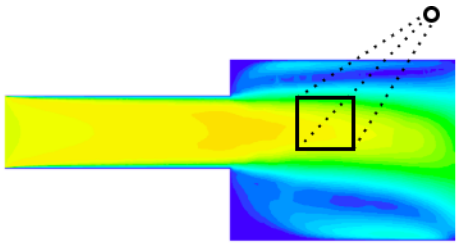


Roadmap

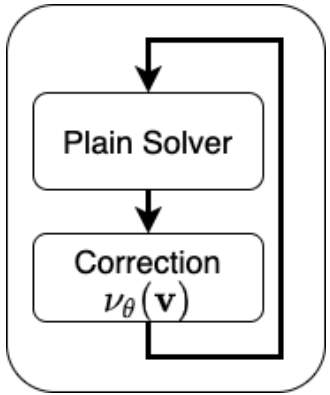
- Motivation
- Navier-Stokes Equations and Turbulence
- Correction Functionals
- Computational Strategy
- Data Preparation
- Differentiable Solvers
- Simulation of Navier-Stokes Flow
- Final Experiments
- **Conclusions**

Summary

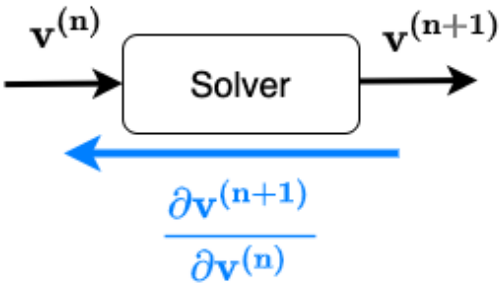
Locality (assumption): Correction depends only on the *local neighbourhood* in the flow field.



Recurrence: The correction is applied in each solver step.



Interaction with solver: Training of correction parameters *informed by* future evolution of physical states through differentiable solver.



Conclusions

1. Correction functionals *perturb* intermediate physical states.
   $\rightarrow$ empirically, optimization via gradient descent is delicate.

2. Optimization of the recurrent model not feasible for arbitrary time periods.

3. For correction of iterative methods, intermediate state information needs to be available.

4. In the right setting: *Effective viscosity model* and *local* FCNN lead to nice generalization.

Our Mentors

Dr. Dirk Hartmann
Senior Principal Key Scientist
CT RDA SDT
München, Deutschland
E-mail hartmann.dirk@siemens.com

Theo Papadopoulos
Engineer – Business Developer
CT RDA SDT
München, Deutschland
E-mail theodoros.papadopoulos@siemens.com

Mohamed Khalil
Engineer
CT RDA SDT
München, Deutschland
E-mail khalil.mohamed@siemens.com

SIEMENS

Acknowledgments

We would like to thank...

- our Mentors from Siemens
- our TUM Co-Mentor Laure Vuaille

- **... you for your attention!**

References

- Baydin, A. G. et al. (2015). “Automatic differentiation in machine learning: a survey”. In: *Journal of Machine Learning Research* 18, pp. 1–43. arXiv: 1502.05767.
- Helmich, T. et al. (2018). *Kurzanleitung fuer den Matlab-PP-Loeser fuer die inkompressible Navier-Stokes Gleichung*. Tech. rep. Technical University Dortmund, Department of Mathematics.
- Holl, P., V. Koltun, and N. Thuerey (2020). “Learning to Control PDEs with Differentiable Physics”. In: arXiv: 2001.07457.
- Margossian, C. C. (2019). *A Review of Automatic Differentiation and its Efficient Implementation Graphical table of content Automatic Differentiation*. Tech. rep. arXiv: 1811.05031v2.
- Pascanu, R., T. Mikolov, and Y. Bengio (n.d.). *On the difficulty of training Recurrent Neural Networks*. Tech. rep. arXiv: 1211.5063v2.
- Philipp Holl, N. T. (2020). *PhiFlow*. https://github.com/tum-pbs/PhiFlow.
- Ramsai (2020). “RANS Derivation and Analysis”. In: *SKILL LYNC*.
- Revels, J., M. Lubin, and T. Papamarkou (2016). “Forward-Mode Automatic Differentiation in Julia”. In: *arXiv:1607.07892 [cs.MS]*.
- Stam, J. (1999). “Stable fluids”. In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 121–128.

Backup Slides

Contrasting to “Global” Architectures

Contrasting our approach, it is comparatively *easy* to directly predict steady states from the input geometry (i.e. no solver in the loop):

- Custom Unet Variant ($\sim$ 2M parameters)
- Inference time 0.05s
- Fast and accurate, real-time ready

But:

- *Global* architecture do not generalize well;
- Predictions limited to training distribution
- No control over physical quantities (viscosity, etc.)

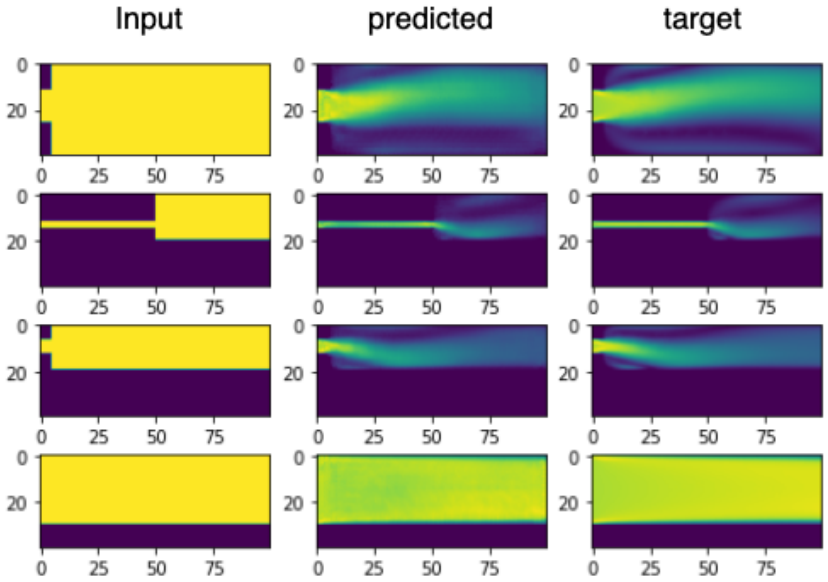


Figure: Our results on steady state prediction with a Unet Variant

Automatic Differentiation (AD) [Margossian 2019]

Automatic differentiabtion uses the fact that any computation can be decomposed into elementary operations.

Consider the function $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^m$. In general, we are interested in computing the jacobian $J_{ij} = \frac{\partial \mathcal{F}_i}{\partial \theta_j}$.

If $\mathcal{F}$ is decomposable as $\mathcal{F} = \mathcal{F}_T \circ \mathcal{F}_{T-1} \circ ... \circ \mathcal{F}_1$, then by the chain rule it follows $J = J_T \cdot J_{T-1} \cdot ... \cdot J_1$. Two main variants:

- **Forward Mode:** Given a "seed" vector $u \in \mathbb{R}^n$ in the input space of $\mathcal{F}$, Forward Mode AD evaluates $J \cdot u$.
- **Reverse Mode:** Given a vector in the output space $w \in \mathbb{R}^m$, reverse mode AD evaluates $J^T \cdot w$.

The relevant difference between the variants lies in the implications regarding computational implementation

Reverse Mode AD

Two distinct phases

1. **Forward pass**: Evaluate $\mathcal{F}$, store intermediate results (activations) and computations (computation graph).
2. **Backward pass**: Starting from the output, compute error (adjoints) by traversing $\mathcal{F}$ in reverse.

Calculate derivatives w.r.t all inputs (i.e. whole gradients) in two passes!
Backpropagation is a special case of reverse mode AD for scalar outputs (m=1). Most machine learning frameworks, including Φ_{Flow} employ reverse AD.
However: Reverse mode requires storing of intermediate results! Large memroy overhead for our application (internal iterative methods, convergence criteria ...).

Forward Mode AD

- Computes the directed partial derivates w.r.t to all outputs *in one forward sweep*.
- One forward pass required for each input (weight to be optimized).
- No backward pass needed $\rightarrow$ no storage of intermediate activation required.

This is achieved by augmenting operations with a *dual number* type (see Baydin et al. 2015):

$$x \rightarrow (v + \epsilon \dot{v}) \quad , v, v \in \mathbb{R}, \quad \epsilon^2 = 0$$

e.g. multiplication becomes $(v_1 + \epsilon \dot{v}_1) \cdot (v_2 + \epsilon \dot{v}_2) = (v_1 v_2) + (v_1 \dot{v}_2 + \dot{v}_1 v_2)\epsilon$.
For differentiation, "just" evaluate $\mathcal{F}(v + 1\epsilon) = \frac{d\mathcal{F}(\theta)}{d\theta}|_v$ and $\epsilon = 0$ for any other number.
In our application: We employ ForwardDiff [Revels, Lubin, and Papamarkou 2016] and make sure dual number types are understood by all operations.

Alternative Setting - Initialization with steady state

Exploit the fact that the given target $\hat{\mathbf{v}}$ is in steady state:
$\rightarrow$ set $\mathbf{v}^{(0)} = \hat{\mathbf{v}}$
and use intermediate targets $\hat{\mathbf{v}}^{(n)}$ for computing the loss.

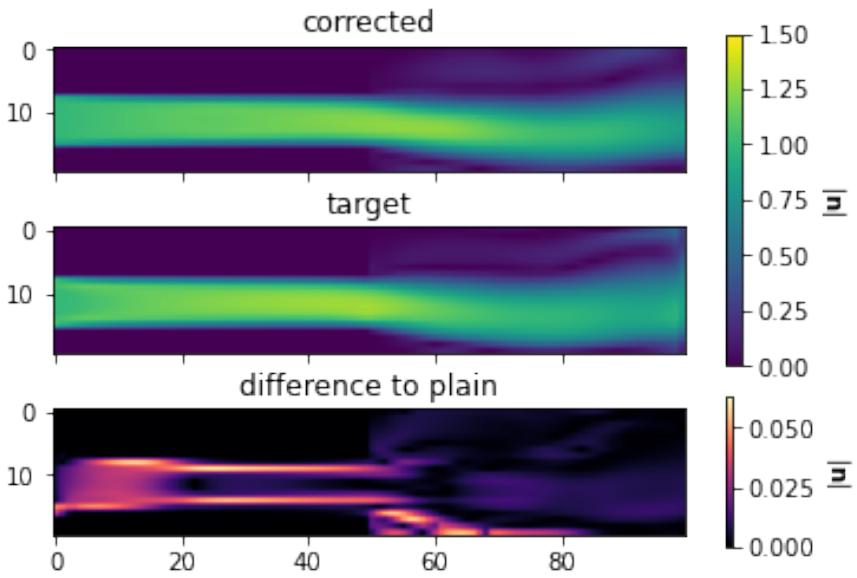


Figure: Correction by effective viscosity model

Alternative Setting - Initialization with steady state

In roll-outs in unseen simulations: Effective viscosity generalizes well beyond time periods seen during training.

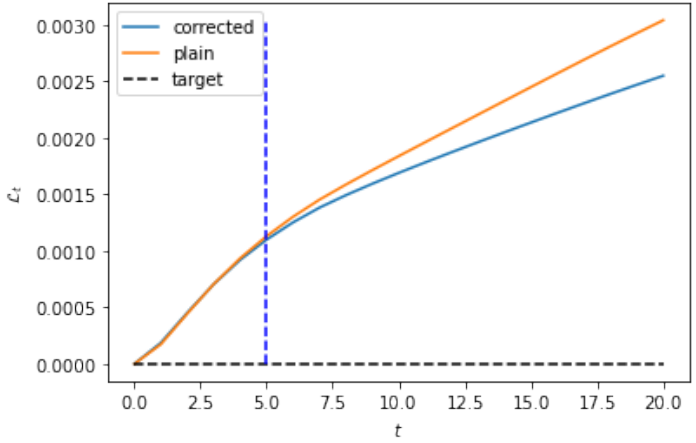


Figure: Effective viscosity model

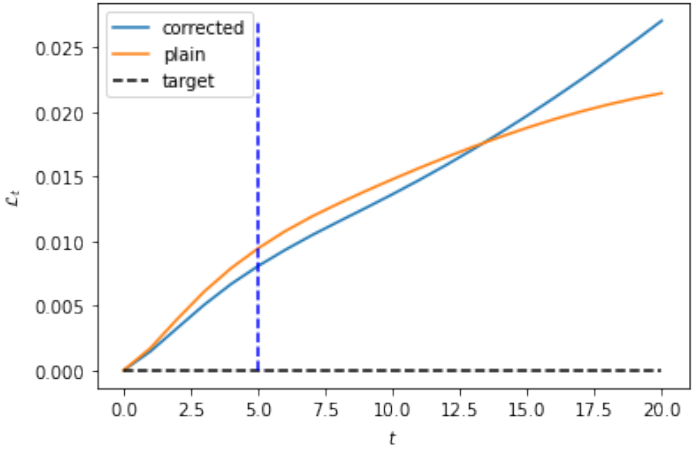


Figure: Residual model

Recurrence and Exploding Gradients

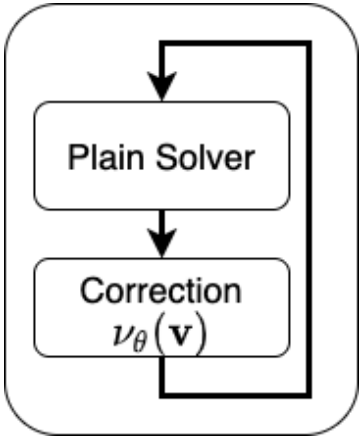
Exploding and vanishing gradients are properties of recurrent neural networks.

For a one-layer NN with parameters **W**, **b**, we can rewrite our scheme as.

$$\nu^{(\mathbf{n})} = \mathbf{W}\sigma(\mathbf{v}^{(\mathbf{n}-\mathbf{1})}, \nu^{(\mathbf{n}-\mathbf{1})}) + \mathbf{b}. \tag{8}$$

where $\sigma(\mathbf{v}^{(n-1)}, \nu^{(n-1)}) = \mathbf{v}^{(n)}$ resembles the (highly non-linear) solver.

Under some assumptions about σ, gradients for RNN's *vanish* or *explode*, depending on the eigenvalues of **W** [Pascanu, Mikolov, and Bengio n.d.].



Recurrence and Exploding Gradients

The gradient components can be calculated as:

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \frac{\partial \mathcal{L}}{\partial v^{(T)}} \frac{\partial \tilde{\mathcal{F}}(v^{(0)}, \theta)}{\partial \theta_j} \tag{9}$$

$$= \sum_{0 \leq n_t \leq T} \left(\frac{\partial \mathcal{L}}{\partial v^{(T)}} \frac{\partial v^{(T)}}{\partial v^{(n_t)}} \frac{\partial \mathcal{F}}{\partial \theta_j} \bigg|_{v^{(n_t-1)}} \right) \tag{10}$$

where the error is propagated *through time* by the product of jacobians:

$$\frac{\partial v^{(T)}}{\partial v^{(n_t)}} = \prod_{n_t < i < T} \frac{\partial v^{(i)}}{\partial v^{(i-1)}} \tag{11}$$