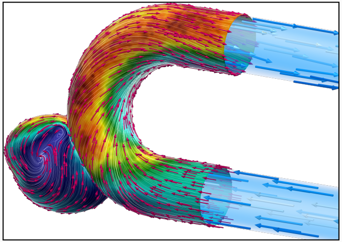
Neural networks with physical constraints

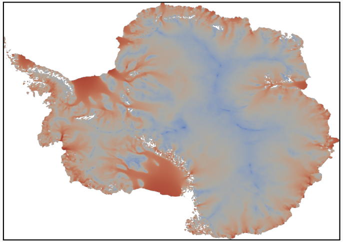
Domain decomposition-based network architectures and model order reduction

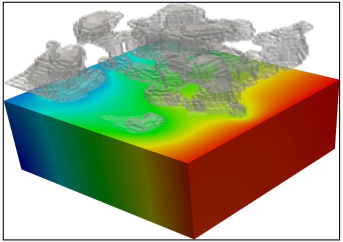
Alexander Heinlein[1]

Seminar talk, Technical University Munich, July 13, 2023

[1]Delft University of Technology







Numerical methods

Based on physical models

+ Robust and generalizable
− Require availability of mathematical models

Machine learning models

Driven by data

+ Do not require mathematical models
− Sensitive to data, limited extrapolation capabilities

Scientific machine learning (SciML)

Combining the strengths and **compensating the weaknesses** of the individual approaches:

numerical methods **improve** machine learning techniques
machine learning techniques **assist** numerical methods

Scientific Machine Learning as a Standalone Field

N. Baker, A. Frank, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, K. Willcox, and S. Lee.
Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence.
USDOE Office of Science (SC), Washington, DC (United States), 2019.

Priority Research Directions

Foundational research themes:

- Domain-awareness
- Interpretability
- Robustness

Capability research themes:

- Massive scientific data analysis
- Machine learning-enhanced modeling and simulation
- Intelligent automation and decision-support for complex systems

Outline

1. Physics-informed machine learning

2. Domain decomposition-based network architectures for PINNs

   Based on joint work with **Victorita Dolean** (U Strathclyde, U Côte d'Azur), **Ben Moseley**, and **Siddhartha Mishra** (ETH Zürich)

3. Surrogate models for CFD simulations – Data-based approach

   Based on joint work with **Mattias Eichinger**, **Viktor Grimm**, and **Axel Klawonn** (University of Cologne)

4. Surrogate models for CFD simulations – Physics-aware approach

   Based on joint work with **Viktor Grimm** and **Axel Klawonn** (University of Cologne)

5. Surrogate models for CFD simulations – GAN-based training

   Based on joint work with **Mirko Kemna** and **Kees Vuik** (TU Delft)

Physics-informed machine learning

Artificial Neural Networks for Solving Ordinary and Partial Differential Equations

Isaac Elias Lagaris, Aristidis Likas, *Member, IEEE,* and Dimitrios I. Fotiadis

Published in IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 9, NO. 5, 1998.

Approach

Solve a general differential equation subject to boundary conditions

$$G(\boldsymbol{x}, \Psi(\boldsymbol{x}), \nabla\Psi(\boldsymbol{x}), \nabla^2\Psi(\boldsymbol{x})) = 0 \quad \text{in } \Omega$$

by solving an **optimization problem**

$$\min_{\theta} \sum_{\boldsymbol{x}_i} G(\boldsymbol{x}_i, \Psi_t(\boldsymbol{x}_i, \theta), \nabla\Psi_t(\boldsymbol{x}_i, \theta), \nabla^2\Psi_t(\boldsymbol{x}_i, \theta))^2$$

where $\Psi_t\left(\boldsymbol{x}, \theta\right)$ is a **trial function**, $\boldsymbol{x}_i$ **sampling points inside the domain** Ω and θ are **adjustable parameters**.

Construction of the trial functions

The trial functions **explicitly satisfy the boundary conditions**:

$$\Psi_t(\boldsymbol{x}, \mathbf{p}) = A(\boldsymbol{x}) + F(\boldsymbol{x}, N(\boldsymbol{x}, \mathbf{p}))$$

- N is a **feedforward neural network** with **trainable parameters** θ and input $x \in \mathbb{R}^n$
- A and F are **fixed functions**, chosen s.t.:
  - A **satisfies the boundary conditions**
  - F **does not contribute to the boundary conditions**

Approach

Solve a general differential equation subject to boundary conditions

$$G(\boldsymbol{x}, \Psi(\boldsymbol{x}), \nabla\Psi(\boldsymbol{x}), \nabla^2\Psi(\boldsymbol{x})) = 0 \quad \text{in } \Omega$$

by solving an **optimization problem**

$$\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}_i} G(\boldsymbol{x}_i, \Psi_t(\boldsymbol{x}_i, \boldsymbol{\theta}), \nabla\Psi_t(\boldsymbol{x}_i, \boldsymbol{\theta}), \nabla^2\Psi_t(\boldsymbol{x}_i, \boldsymbol{\theta}))^2$$

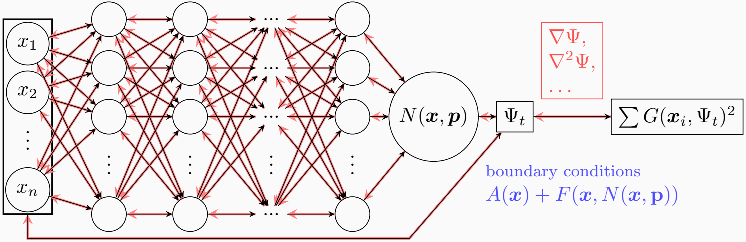
where $\Psi_t\left(\boldsymbol{x}, \boldsymbol{\theta}\right)$ is a **trial function**, $\boldsymbol{x}_i$ **sampling points inside the domain** Ω and $\boldsymbol{\theta}$ are **adjustable parameters**.

Construction of the trial functions

The trial functions **explicitly satisfy the boundary conditions**:

$$\Psi_t(\boldsymbol{x}, \mathbf{p}) = A(\boldsymbol{x}) + F(\boldsymbol{x}, N(\boldsymbol{x}, \mathbf{p}))$$

- N is a **feedforward neural network** with **trainable parameters** $\boldsymbol{\theta}$ and input $x \in \mathbb{R}^n$
- A and F are **fixed functions**, chosen s.t.:
  - **A satisfies the boundary conditions**
  - **F does not contribute to the boundary conditions**



Physics-Informed Neural Networks (PINNs)

In the **physics-informed neural network (PINN)** approach introduced by Raissi et al. (2019), a **neural network** is employed to **discretize a partial differential equation**

$$\mathcal{N}[u](\mathbf{x}, \mathbf{t}) = f(\mathbf{x}, \mathbf{t}), \qquad (\mathbf{x}, \mathbf{t}) \in [0, T] \times \Omega \subset \mathbb{R}^d.$$

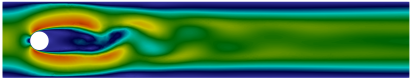
It is based on the approach by Lagaris et al. (1998). The main novelty of PINNs is the use of a **hybrid loss function**:

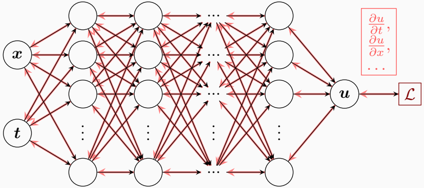
$$\mathcal{L} = \omega_{\text{data}} \mathcal{L}_{\text{data}} + \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}},$$

where ω_{data} and ω_{PDE} are **weights** and

$$\mathcal{L}_{\text{data}} = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left(u(\hat{\mathbf{x}}_i, \hat{\mathbf{t}}_i) - u_i \right)^2,$$

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \left(\mathcal{N}[u](\mathbf{x}_i, \mathbf{t}) - f(\mathbf{x}_i, \mathbf{t}_i) \right)^2.$$





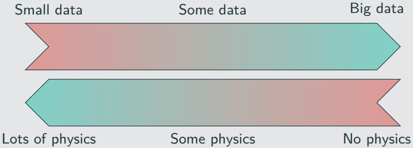
Advantages

- **"Meshfree"**
- **Small data**
- **Generalization properties**
- **High-dimensional problems**
- **Inverse** and **parameterized problems**

Drawbacks

- **Training cost** and **robustness**
- **Convergence not well-understood**
- **Difficulties with scalability** and **multi-scale problems**

Hybrid loss



- **Known solution values** can be included in $\mathcal{L}_{\text{data}}$
- **Initial and boundary conditions** are also included in $\mathcal{L}_{\text{data}}$

Available Theoretical Results for PINNs – An Example

Mishra and Molinaro. *Estimates on the generalisation error of PINNs*, **2022**

Estimate of the generalization error

The generalization error (or total error) satisfies

$$\mathcal{E}_G \leq C_{\text{PDE}} \mathcal{E}_T + C_{\text{PDE}} C_{\text{quad}}^{1/p} N^{-\alpha/p}$$

where

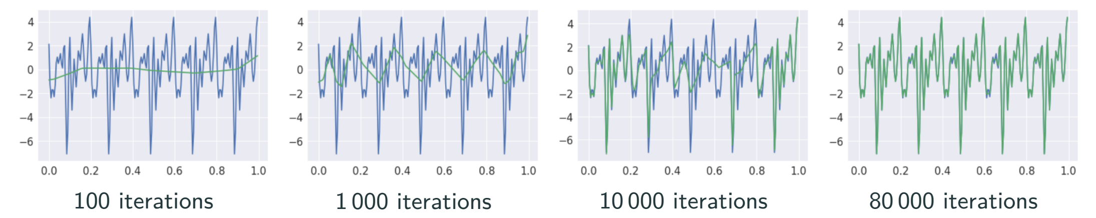
- $\mathcal{E}_G = \mathcal{E}_G(\theta; \boldsymbol{X}) := \|\mathbf{u} - \mathbf{u}^*\|_V$ (V Sobolev space, $\boldsymbol{X}$ training data set)
- $\mathcal{E}_T$ is the training error (l^p loss of the residual of the PDE)
- C_{PDE} and C_{quad} constants depending on the PDE resp. the quadrature
- N number of the training points and α convergence rate of the quadrature

Rule of thumb:

"As long as the PINN is trained well, it also generalizes well"

Scaling Issues in Neural Network Training

- **Spectral bias: neural networks prioritize learning lower frequency functions** first irrespective of their amplitude



Rahaman et al., *On the spectral bias of neural networks*, **ICML (2019)**

- Solving solutions on **large domains and/or with multiscale features** potentially requires **very large neural networks**.
- Training may **not sufficiently reduce the loss** or take **large numbers of iterations**.
- Significant **increase on the computational work**

Convergence analysis of PINNs via the **neural tangent kernel**: **Wang, Yu, Perdikaris,** *When and why PINNs fail to train: A neural tangent kernel perspective*, **JCP (2022)**

Domain decomposition-based network architectures for PINNs

Solve

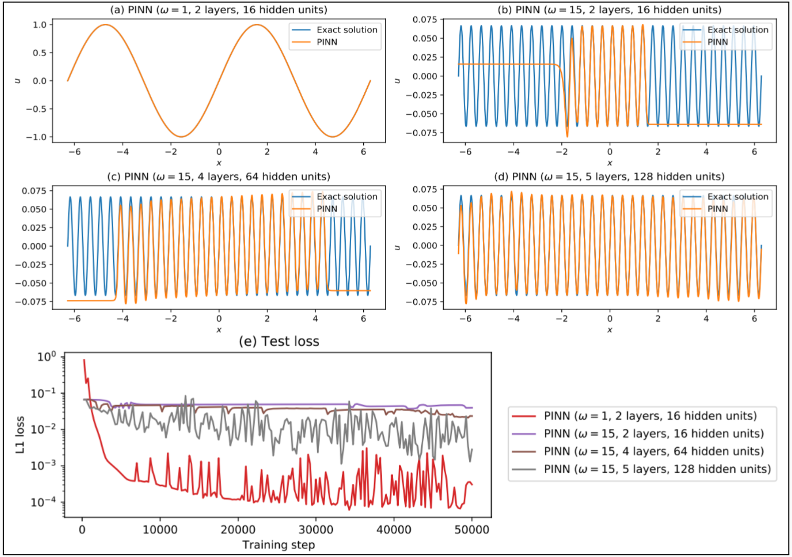
$$\begin{aligned} u' &= \cos(\omega \boldsymbol{x}), \\ u(0) &= 0, \end{aligned}$$

for different values of ω using **PINNs with varying network capacities**.

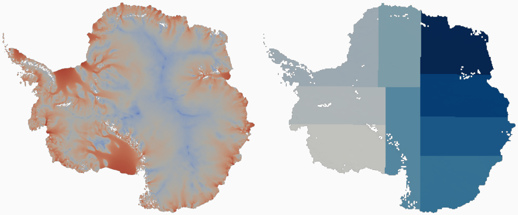
Scaling issues

- Large computational domains
- Small frequencies

Cf. Moseley, Markham, and Nissen-Meyer (arXiv 2021)



(a) 321 free parameters (d) 66 433 free parameters



Images based on Heinlein, Perego, Rajamanickam (2022)

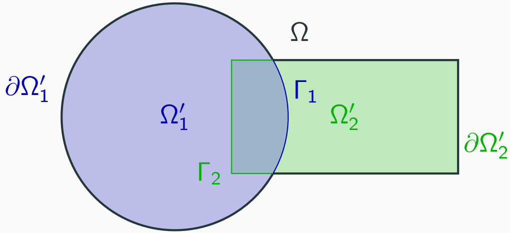
Historical remarks: The **alternating Schwarz method** is the earliest **domain decomposition method (DDM)**, which has been invented by **H. A. Schwarz** and published in **1870**:

- Schwarz used the algorithm to establish the **existence of harmonic functions** with prescribed boundary values on **regions with non-smooth boundaries**.

Idea

Decomposing a large **global problem** into smaller **local problems**:

- Better **robustness** and **scalability** of numerical solvers
- Improved **computational efficiency**
- Introduce **parallelism**



Machine Learning and Domain Decomposition Methods

A **non-exhaustive overview**:

- **Machine Learning for adaptive BDDC, FETI–DP, and AGDSW**: Heinlein, Klawonn, Lanser, Weber (2019, 2020, 2021, 2021, 2021, 2022); Klawonn, Lanser, Weber (preprint 2022)
- **Domain decomposition for CNNs**: Gu, Zhang, Liu, Cai (2022); Lee, Park, Lee (2022); Klawonn, Lanser, Weber (arXiv 2023)
- **D3M**: Li, Tang, Wu, and Liao (2019)
- **DeepDDM**: Li, Xiang, Xu (2020); Mercier, Gratton, Boudier (arXiv 2021); Li, Wang, Cui, Xiang, Xu (2023)
- **FBPINNs**: Moseley, Markham, and Nissen-Meyer (arXiv 2021); Dolean, Heinlein, Mishra, Moseley (accepted 2023, submitted 2023/arXiv:2306.05486)
- **Schwarz Domain Decomposition Algorithm for PINNs**: Kim, Yang (2022, arXiv 2022)
- **cPINNs**: Jagtap, Kharazmi, Karniadakis (2020)
- **XPINNs**: Jagtap, Karniadakis (2020)

An overview of the state-of-the-art in early 2021:

A. Heinlein, A. Klawonn, M. Lanser, J. Weber.
Combining machine learning and domain decomposition methods for the solution of partial differential equations — A review.
GAMM-Mitteilungen. 2021.

Finite Basis Physics-Informed Neural Networks (FBPINNs)

In the **finite basis physics informed neural network (FBPINNs) method** introduced in **Moseley, Markham, and Nissen-Meyer (arXiv 2021)**, we solve the boundary value problem

$$\begin{aligned} \mathcal{N}[u](\boldsymbol{x}) &= f(\boldsymbol{x}), & \boldsymbol{x} \in \Omega \subset \mathbb{R}^d, \\ \mathcal{B}_k[u](\boldsymbol{x}) &= g_k(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma_k \subset \partial\Omega. \end{aligned}$$

using the **PINN** approach and **hard enforcement of the boundary conditions**, similar to **Lagaris et al. (1998)**.

FBPINNs use the **network architecture**

$$u(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \mathcal{C} \sum_{j=1}^{J} \omega_j u_j (\boldsymbol{\theta}_j)$$

and the **loss function**

$$\mathcal{L}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \frac{1}{N} \sum_{i=1}^{N} \Big(\mathcal{N}[\mathcal{C} \sum_{\boldsymbol{x}_i \in \Omega_j} \omega_j u_j](\boldsymbol{x}_i, \boldsymbol{\theta}_j) - f(\boldsymbol{x}_i) \Big)^2.$$

- **Overlapping DD**: $\Omega = \bigcup_{l=1}^{J} \Omega_j$
- **Window functions** ω_j with $\mathrm{supp}(\omega_j) \subset \Omega_j$ and $\sum_{j=1}^{J} \omega_j \equiv 1$ on Ω

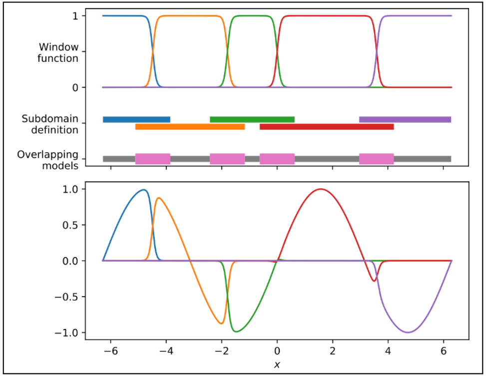
Hard enforcement of boundary conditions

Loss function

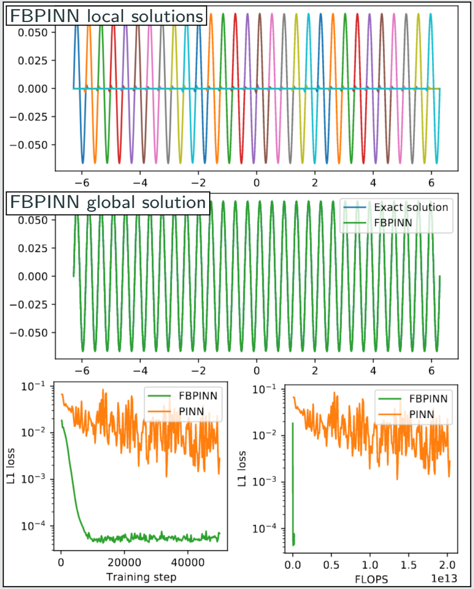
$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left(\mathcal{N}[\mathcal{C} u](\boldsymbol{x}_i, \boldsymbol{\theta}) - f(\boldsymbol{x}_i) \right)^2,$$

with constraining operator $\mathcal{C}$, which **explicitly enforces the boundary conditions**.

$\rightarrow$ Often **improves training performance**



PINN Vs FBPINN (Moseley et al. (arXiv 2021))



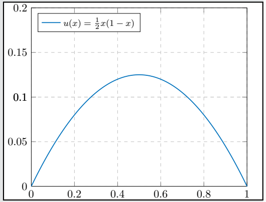
Scalability of FBPINNs

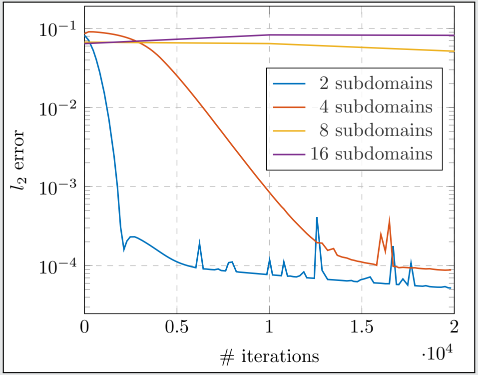
Consider the **simple boundary value problem**

$$-u'' = 1 \quad \text{in } [0,1],$$
$$u(0) = u(1) = 0,$$

which has the **solution**

$$u(x) = 1/2\, x(1-x).$$





Two-Level FBPINN Algorithm

Coarse correction and spectral bias

Questions:

- Scalability requires **global transport of information**. This can be done via **coarse global problem**.
- What does this mean in the **context of network training**?

Idea:

→ Learn **low frequencies** using a **small global network**, train **high frequencies** using **local networks**.

Two-level FBPINN network architecture:

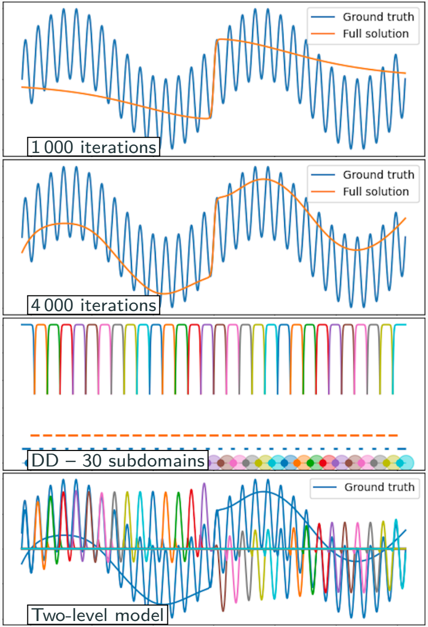
$$u(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \mathcal{C}\left(u_0(\boldsymbol{\theta}_0) + \sum_{j=1}^{J} \omega_j u_j(\boldsymbol{\theta}_j)\right)$$

Consider a **simple model problem** with **two frequencies**

$$\begin{cases} u' = \omega_1 \cos(\omega_1 \boldsymbol{x}) + \omega_2 \cos(\omega_2 \boldsymbol{x}) \\ u(0) = 0. \end{cases}$$

with $\omega_1 = 1$, $\omega_2 = 15$.

Cf. Dolean, Heinlein, Mishra, Moseley (accepted 2023).



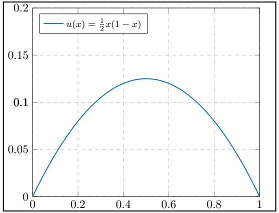
Numerical Results for FBPINNs – One Versus Two Levels

Consider, again, the **simple boundary value problem**

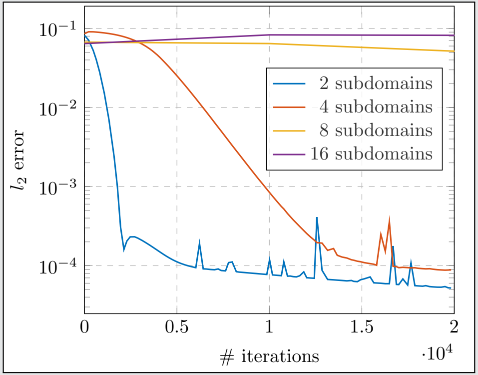
$$-u'' = 1 \quad \text{in } [0,1],$$
$$u(0) = u(1) = 0,$$

which has the **solution**

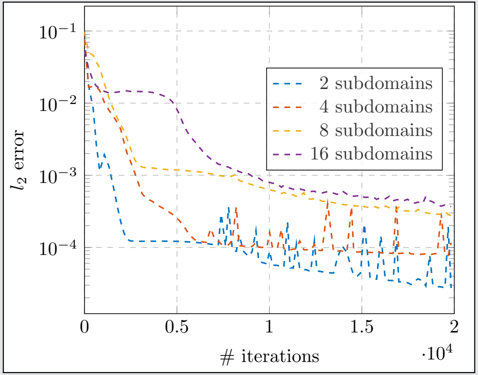
$$u(x) = 1/2\, x(1-x).$$



One-Level FBPINNs



Two-Level FBPINNs



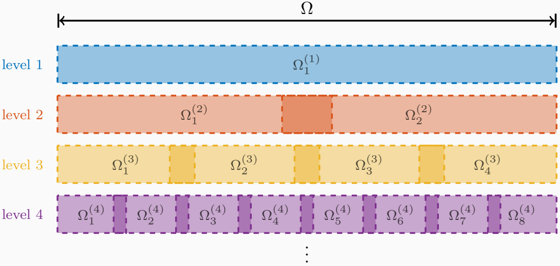
Multi-Level FBPINN Algorithm

We introduce a **hierarchy of L overlapping domain decompositions**

$$\Omega = \bigcup_{j=1}^{J^{(l)}} \Omega_j^{(l)}$$

and corresponding window functions $\omega_j^{(l)}$ with

$$\operatorname{supp}\left(\omega_j^{(l)}\right) \subset \Omega_j^{(l)} \text{ and } \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \equiv 1 \text{ on } \Omega.$$



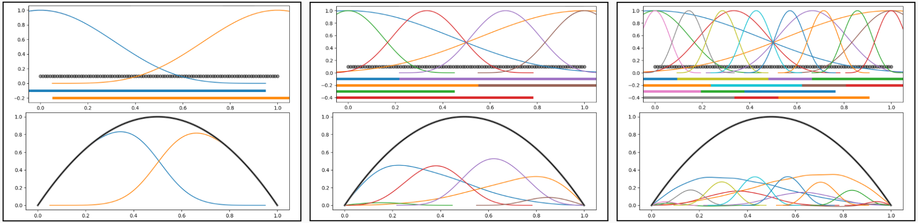
This yields the **L-level FBPINN algorithm**:

L-level network architecture

$$u\left(\theta_1^{(1)}, \ldots, \theta_{J^{(L)}}^{(L)}\right) = \mathcal{C}\left(\sum_{l=1}^{L} \sum_{i=1}^{N^{(l)}} \omega_j^{(l)} u_j^{(l)}\left(\theta_j^{(l)}\right)\right)$$

Loss function

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(n[\mathcal{C} \sum_{x_i \in \Omega_j^{(l)}} \omega_j^{(l)} u_j^{(l)}](x_i, \theta_j^{(l)}) - f(x_i)\right)^2$$



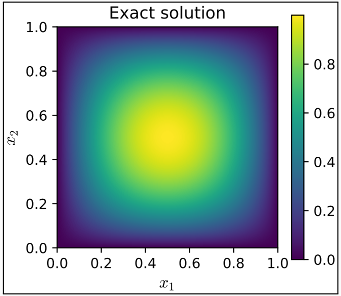
Multilevel FBPINNs – 2D Laplace

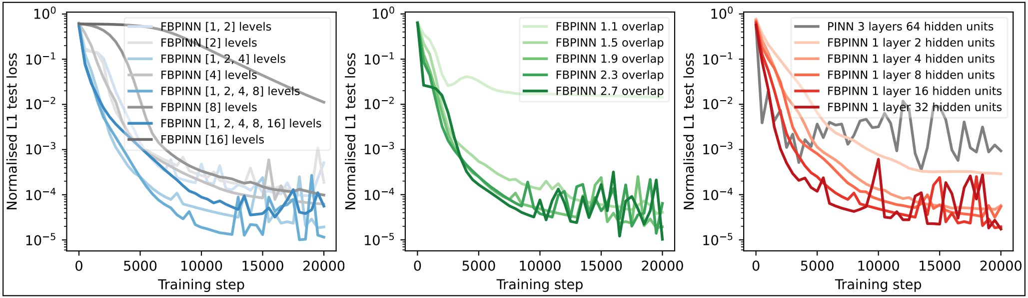
Let us consider the **simple two-dimensional boundary value problem**

$$
\begin{aligned}
-\Delta u &= 32(\boldsymbol{x}(1-\boldsymbol{x}) + \boldsymbol{y}(1-\boldsymbol{y})) \quad && \text{in } \Omega = [0,1]^2, \\
u &= 0 && \text{on } \partial\Omega,
\end{aligned}
$$

which has the **solution**

$$
u(\boldsymbol{x},\boldsymbol{y}) = 16\left(\boldsymbol{x}(1-\boldsymbol{x})\boldsymbol{y}(1-\boldsymbol{y})\right).
$$





Cf. Dolean, Heinlein, Mishra, Moseley (submitted 2023/arXiv:2306.05486).

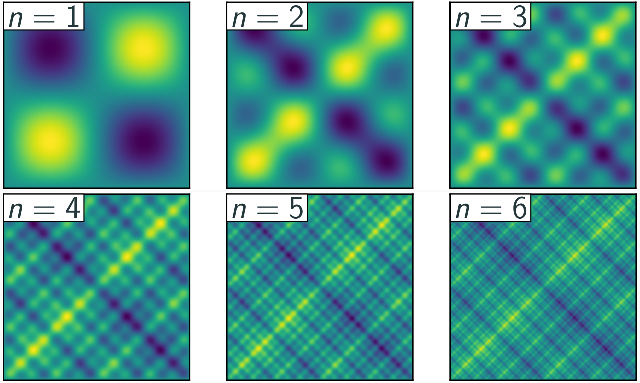
Multi-Frequency Problem

Let us now consider the **two-dimensional multi-frequency Laplace boundary value problem**

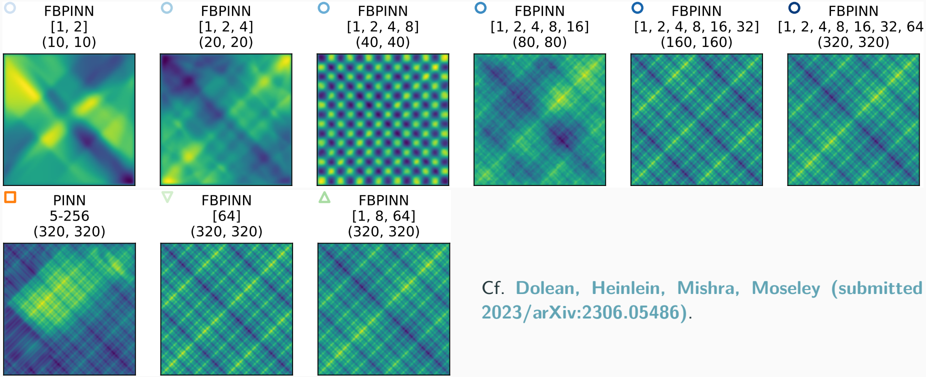
$$
\begin{aligned}
-\Delta u &= 2 \sum_{i=1}^{n} (\omega_i \pi)^2 \sin(\omega_i \pi \boldsymbol{x}) \sin(\omega_i \pi \boldsymbol{y}) && \text{in } \Omega = [0,1]^2, \\
u &= 0 && \text{on } \partial\Omega,
\end{aligned}
$$

with $\omega_i = 2^i$.

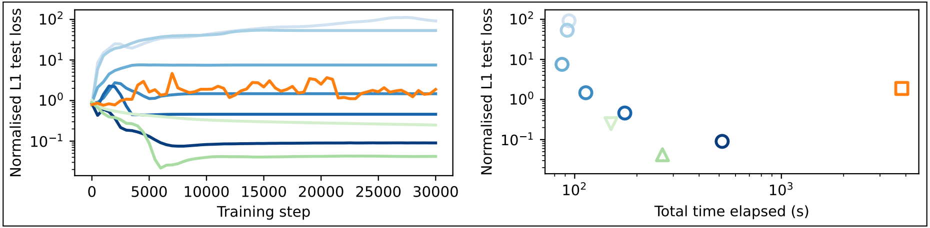
For increasing values of n, we obtain the **analytical solutions**:

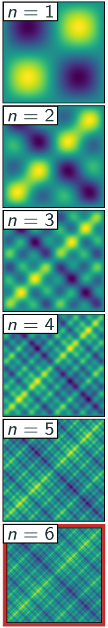


Multi-Level FBPINNs for a Multi-Frequency Problem – Strong Scaling

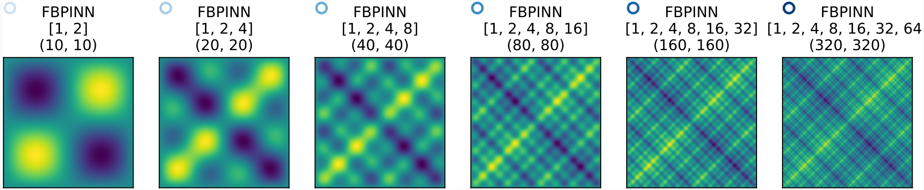


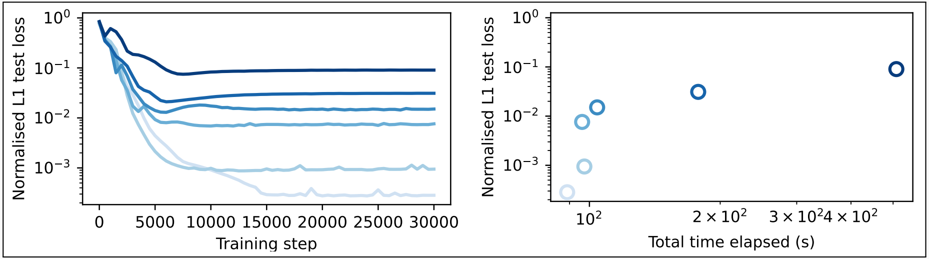
Cf. **Dolean, Heinlein, Mishra, Moseley (submitted 2023/arXiv:2306.05486)**.

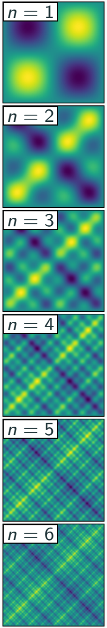




Multi-Level FBPINNs for a Multi-Frequency Problem – Weak Scaling







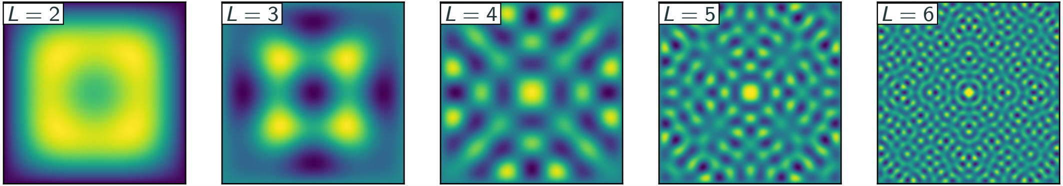
Cf. **Dolean, Heinlein, Mishra, Moseley (submitted 2023/arXiv:2306.05486)**.

Helmholtz Problem

Finally, let us consider the **two-dimensional Helmholtz boundary value problem**

$$\begin{aligned} \Delta u - k^2 u &= f \quad \text{in } \Omega = [0,1]^2, \\ u &= 0 \quad \text{on } \partial\Omega, \\ f(\mathbf{x}) &= e^{-\frac{1}{2}(\|\mathbf{x}-0.5\|/\sigma)^2}. \end{aligned}$$

With $k = 2^L \pi / 1.6$ and $\sigma = 0.8/2^L$, we obtain the **solutions**:

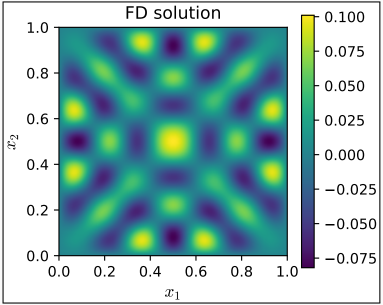


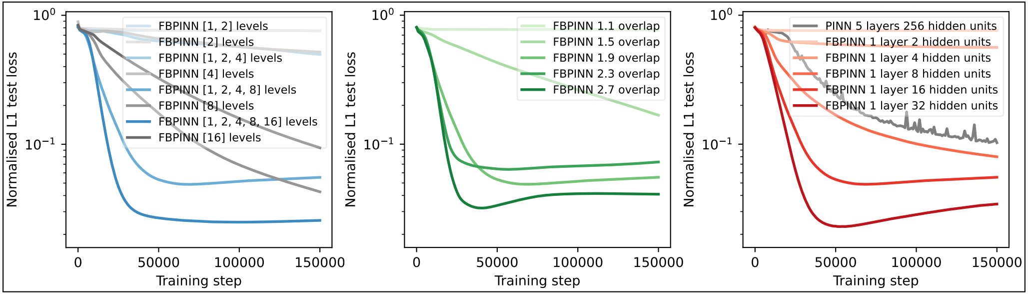
Let us consider the **two-dimensional Helmholtz boundary value problem**

$$\Delta u - k^2 u = f \quad \text{in } \Omega = [0,1]^2,$$
$$u = 0 \quad \text{on } \partial\Omega,$$
$$f(\mathbf{x}) = e^{-\frac{1}{2}(\|\mathbf{x}-0.5\|/\sigma)^2}.$$

with $k = 2^4\pi/1.6$ and $\sigma = 0.8/2^4$.

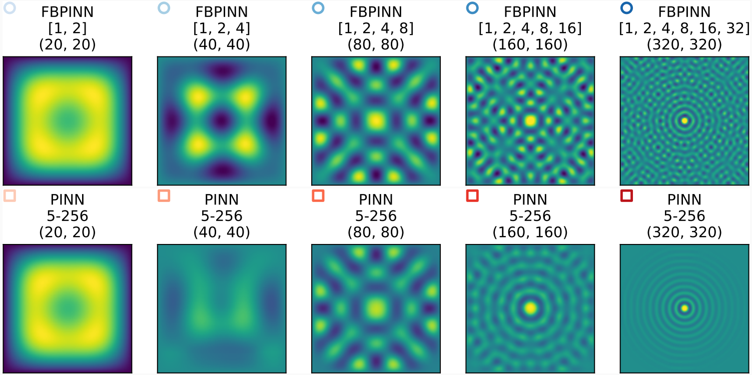
We compute a **reference solution** using **finite differences** with a 5-point stencil on a 320×320 grid.

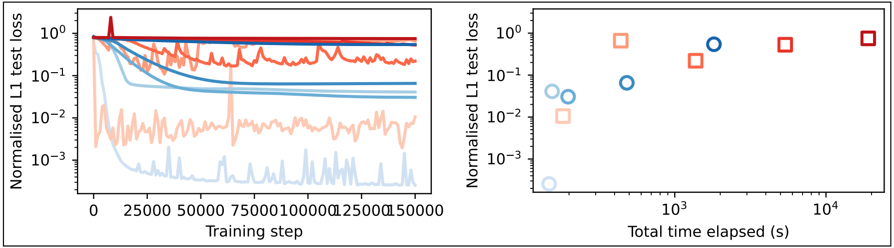


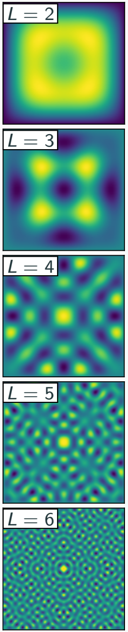


Cf. **Dolean, Heinlein, Mishra, Moseley (submitted 2023/arXiv:2306.05486)**.

Multi-Level FBPINNs for the Helmholtz Problem – Weak Scaling

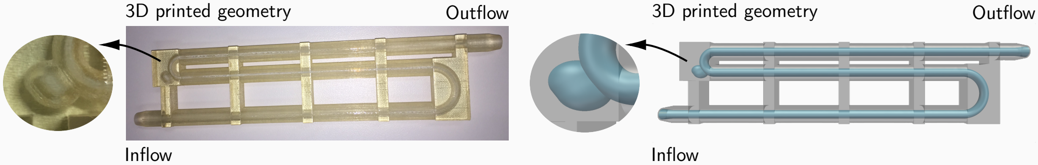




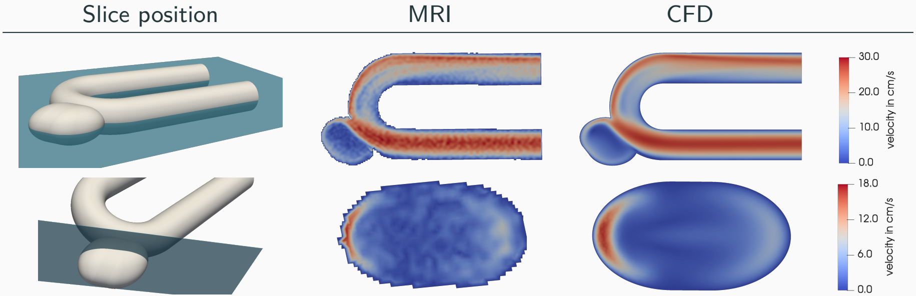


Surrogate models for CFD simulations
Data-based approach

In Giese, Heinlein, Klawonn, Knepper, Sonnabend (2019), a benchmark for **comparing MRI measurements and CFD simulations** of hemodynamics in **intracranial aneurysms** was proposed.

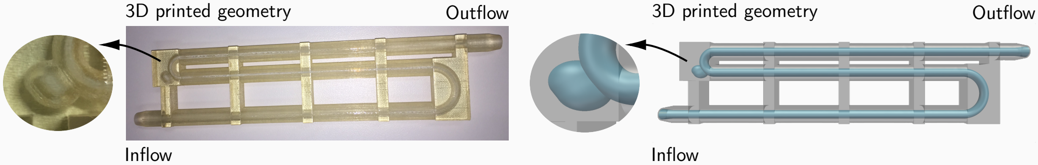


To obtain accurate simulation results, a simulation with $\approx 10\,\text{m}$ d.o.f.s has been carried out. On $O(100)$ MPI ranks, the computation of a steady state took $O(1)\,\text{h}$ on CHEOPS supercomputer at UoC.

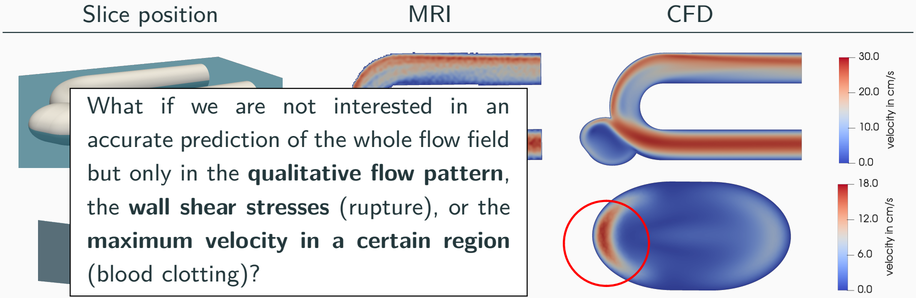


Computational Fluid Dynamics (CFD) Simulations are Time Consuming

In **Giese, Heinlein, Klawonn, Knepper, Sonnabend (2019)**, a benchmark for **comparing MRI measurements and CFD simulations** of hemodynamics in **intracranial aneurysms** was proposed.



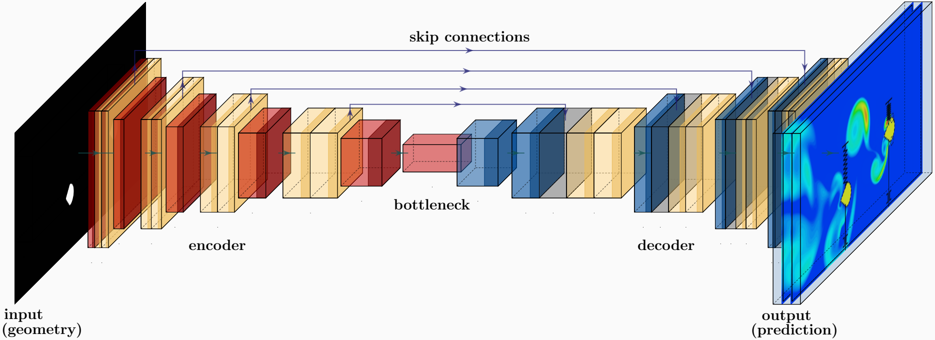
To obtain accurate simulation results, a simulation with ≈ 10 m d.o.f.s has been carried out. On $O(100)$ MPI ranks, the computation of a steady state took $O(1)$ h on CHEOPS supercomputer at UoC.



What if we are not interested in an accurate prediction of the whole flow field but only in the **qualitative flow pattern**, the **wall shear stresses** (rupture), or the **maximum velocity in a certain region** (blood clotting)?

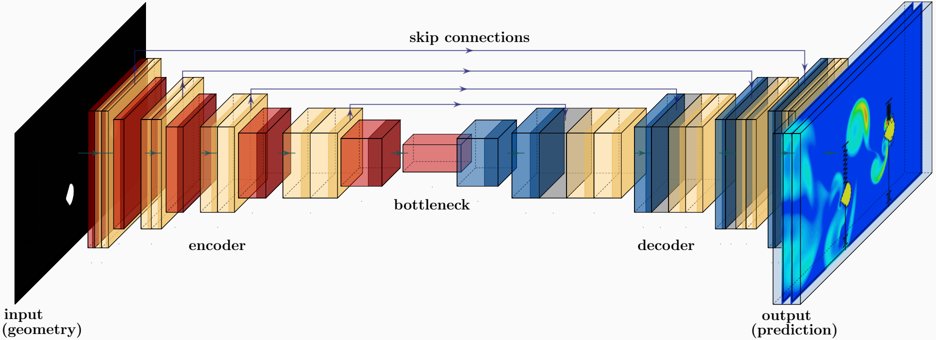
Our approach is inspired by the work Guo, Li, Iorio (2016), in which **convolutional neural networks (CNNs)** are employed to predict the flow in channel with an obstacle.

In particular, we use a pixel image of the **geometry as input** and predict an image of the resulting **stationary flow field as output**:



Other related works: E.g.

- Guo, Li, Iorio (2016)
- Niekamp, Niemann, Schröder (2022)
- Stender, Ohlsen, Geisler, Chabchoub, Hoffmann, Schlaefer (2022)



We learn the **nonlinear map** between a **representation space of the geometry** and the **solution space** of the stationary Navier–Stokes equations → **Operator learning**.

Operator learning

Learning **maps between function spaces**, e.g.,

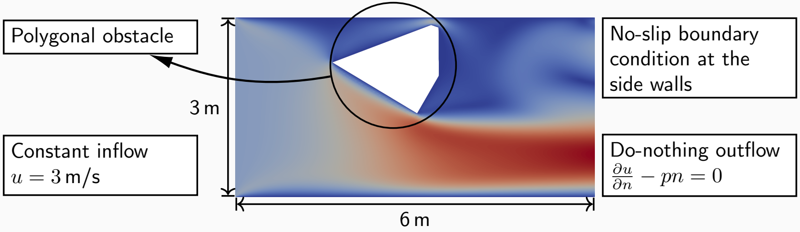
- between the right-hand side and the solution of a BVP.

Other operator learning approaches

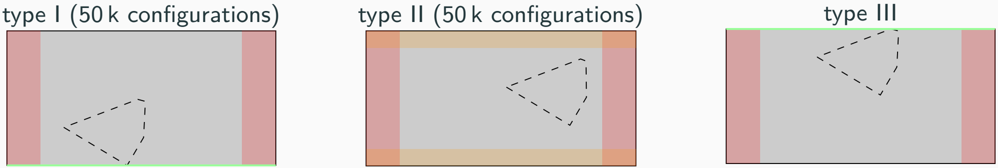
- **DeepOnet:** Lu, Jin, and Karniadakis. (arXiv preprint 2019).
- **Neural operators:** Kovachki, Li, Liu, Azizzadenesheli, Bhattacharya, Stuart, and Anandkumar (arXiv preprint 2021).

Model Problem – Flow Around an Obstacle in Two Dimensions

We propose a **simple model problem** to investigate predictions of a **steady flow in a channel with an obstacle**; this setup is also inspired by **Guo, Li, Iorio (2016)**.



Data: randomly generated geometries (star-shaped polygons with 3, 4, 5, 6, and 12 edges)



90 k training data & 10 k test data

(transfer learning; cf. **Eichinger, Heinlein, Klawonn (2022)**)

We solve the **steady Navier–Stokes equations**

$$-\nu\Delta\vec{u} + (u\cdot\nabla)\,\vec{u} + \nabla p = 0 \text{ in } \Omega,$$
$$\nabla\cdot u = 0 \text{ in } \Omega,$$

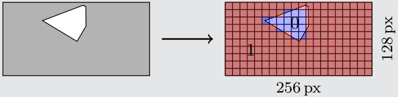
where $\vec{u}$ and p are the velocity and pressure fields and ν is the viscosity. Furthermore, we prescribe the previously described boundary conditions.

Software pipeline

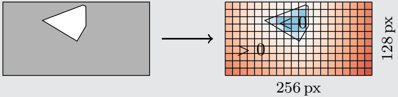
1. Define the boundary of the polygonal obstacle and **create the corresponding STL (standard triangulation language) file**.
2. **Generate a hexahedral compute grid** (`snappyHexMesh`).
3. Run the **CFD simulation** (`simpleFoam`).
4. **Interpolate geometry information and flow field** onto a pixel grid.
5. **Train the CNN**.

Input data

Binary

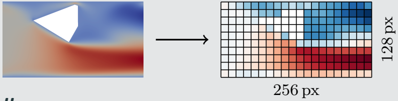


SDF (Signed Distance Function)

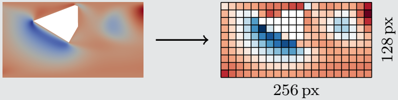


Output data

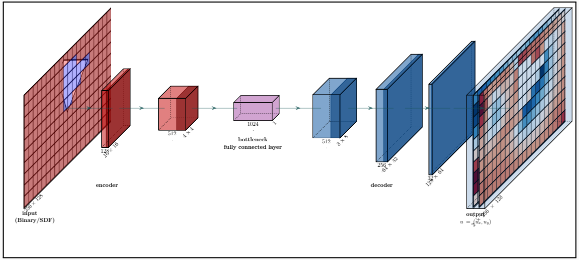
u_x



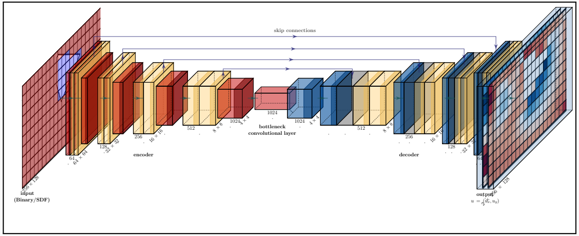
u_y



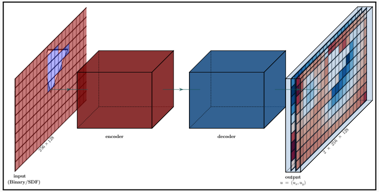
Bottleneck CNN (Guo, Li, Iorio (2016))



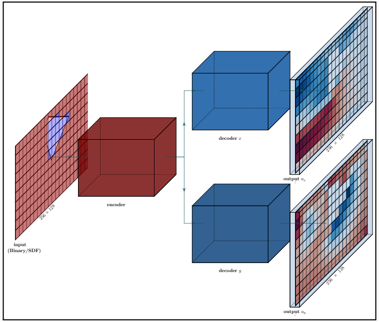
U-Net (Ronneberger, Fischer, Brox (2015))



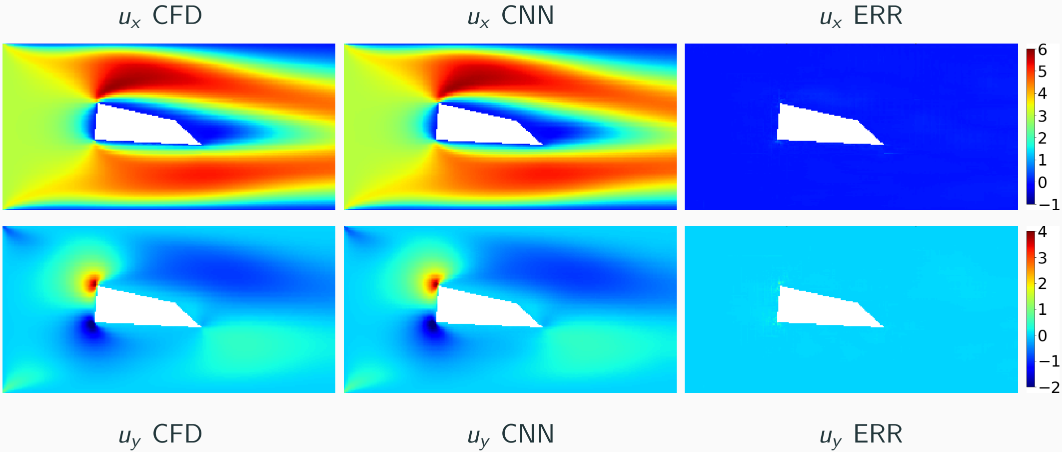
One decoder



Two decoders



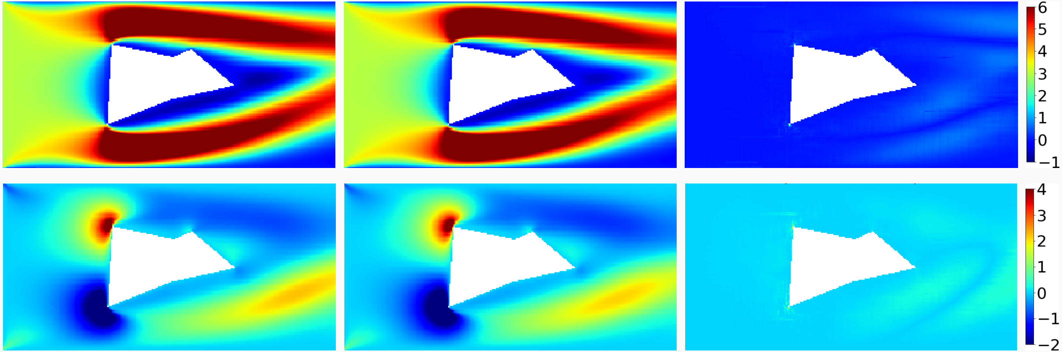
Comparison CFD Vs NN (Relative Error 2 %)



Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**

Comparison CFD Vs NN (Relative Error 14 %)

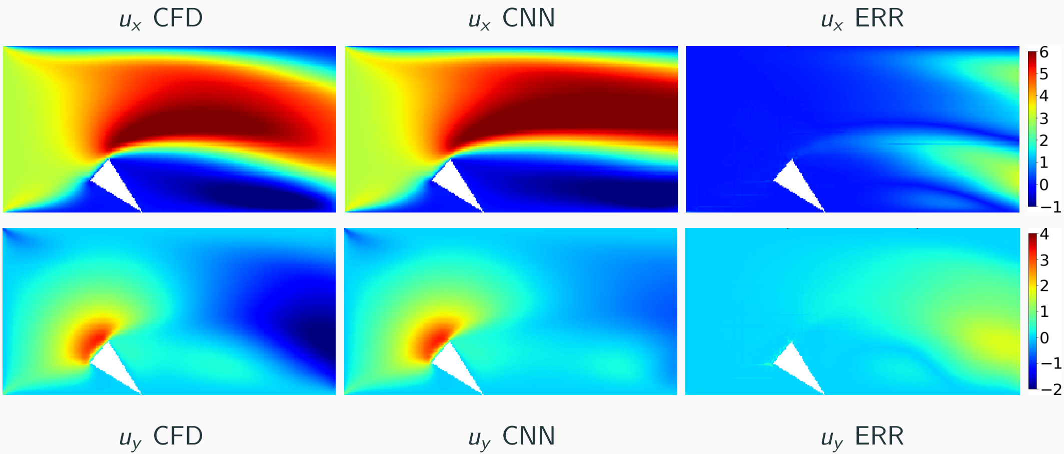
u_x CFD | u_x CNN | u_x ERR



u_y CFD | u_y CNN | u_y ERR

Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**

Comparison CFD Vs NN (Relative Error 31 %)



Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**

First Results (Eichinger, Heinlein, Klawonn (2021, 2022))

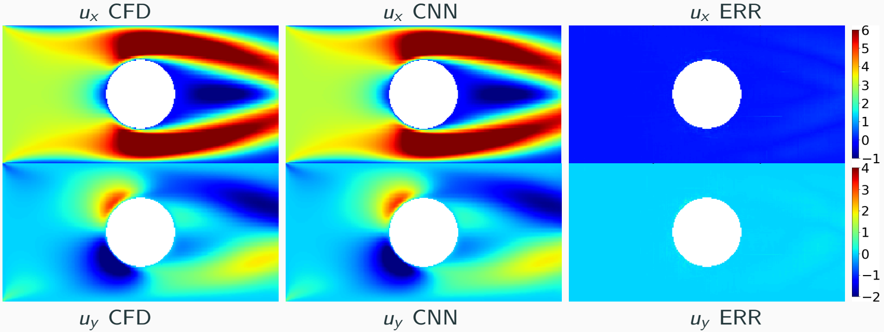
We compare the **relative error (RE)** $\frac{\|u_{i,j}-\hat{u}_{i,j}\|_2}{\|u_{i,j}\|_2+10^{-4}}$ **averaged over all non-obstacle pixels** and **all validation data configurations**. Furthermore: **MSE = mean squared error**; **MAE = mean absolute error**.

			Bottleneck CNN (Guo, Li, Iorio (2016))			U-Net (Ronneberger, Fischer, Brox (2015))		
input	# dec.	loss	total	type I	type II	total	type I	type II
SDF	1	MSE	**61.16 %**	110.46 %	11.86 %	17.04 %	29.42 %	4.66 %
		MSE + RE	3.97 %	3.31 %	**4.63 %**	2.67 %	2.11 %	3.23 %
		MAE	25.19 %	41.52 %	8.86 %	9.10 %	13.89 %	4.32 %
		MAE + RE	4.45 %	3.84 %	5.05 %	2.48 %	1.87 %	**3.10 %**
	2	MSE	49.82 %	89.12 %	10.51 %	13.01 %	21.59 %	4.42 %
		MSE + RE	**3.85 %**	**3.05 %**	4.64 %	**2.43 %**	**1.78 %**	3.23 %
		MAE	45.23 %	81.38 %	9.08 %	5.47 %	7.06 %	3.89 %
		MAE + RE	4.33 %	3.74 %	4.91 %	2.57 %	1.98 %	3.17 %
Binary	1	MSE	49.78 %	88.28 %	11.28 %	27.15 %	49.15 %	5.15 %
		MSE + RE	10.12 %	11.44 %	8.80 %	5.49 %	6.25 %	4.74 %
		MAE	39.16 %	64.77 %	13.54 %	15.69 %	26.36 %	5.02 %
		MAE + RE	10.61 %	12.34 %	8.87 %	**4.48 %**	**5.05 %**	**3.90 %**
	2	MSE	**51.34 %**	91.20 %	11.48 %	24.00 %	43.14 %	4.85 %
		MSE + RE	10.03 %	11.37 %	8.69 %	5.56 %	6.79 %	4.33 %
		MAE	37.16 %	62.01 %	12.32 %	21.54 %	38.12 %	4.96 %
		MAE + RE	**9.53 %**	**10.91 %**	**8.15 %**	6.04 %	7.88 %	4.20 %

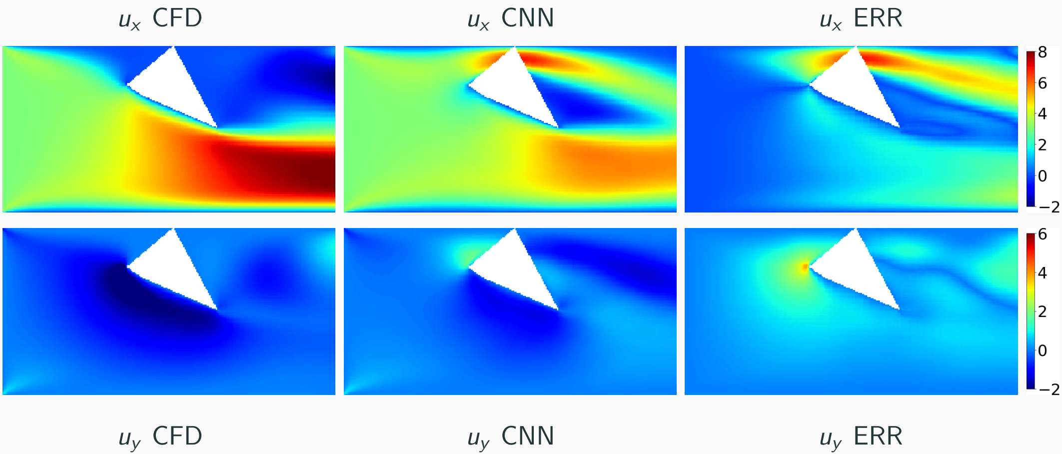
Generalization Properties (Eichinger, Heinlein, Klawonn (2021, 2022))

We test the **generalization properties** of our **previously trained U-Net**. In particular, we predict the flow for new geometries of **Type I** and **Type II**; 1 000 geometries each (500 Type I & 500 Type II).

# polygon edges	SDF input			Binary input		
	total	type I	type II	total	type I	type II
7	2.71 %	1.89 %	3.53 %	4.39 %	4.61 %	4.16 %
8	2.82 %	1.98 %	3.65 %	4.67 %	4.89 %	4.44 %
10	3.21 %	2.32 %	4.10 %	5.23 %	5.51 %	4.94 %
15	4.01 %	3.16 %	4.86 %	7.76 %	7.85 %	6.66 %
20	5.08 %	4.22 %	5.93 %	9.70 %	10.43 %	8.97 %



Generalization Issues – Type III Geometry (Relative Error 158 %)



Cf. **Eichinger, Heinlein, Klawonn (2022)**

Transfer Learning – Type III Geometries

The **best model (U-Net, one decoder, MAE+RE loss)** trained on type I and type II geometries **performs poorly on 2 500 type III geometries**:

	SDF Input	binary Input
type III	22 985.89 %	4 134.69 %

We compare the following approaches to generalize to type III geometries:

- **Approach 1:** Train a new model from scratch on type III geometries (2 500 training + 2 500 validation data)
- **Approach 2:** Train the previous model on type III geometries
- **Approach 3:** Train the previous model on a data set consisting of the old data (type I & type II) and type III data

learning approach	# training epochs	type I & II		type III	
		SDF input	binary input	SDF input	binary input
1	100	-	-	98.02 %	111.75 %
2	100	208.02 %	105.43 %	7.18 %	11.81 %
3	3	**3.33 %**	**7.06 %**	**4.94 %**	**11.28 %**

Neural networks **forget if data is removed from the training data**. However, **new geometries (type III: symmetric to Type I) can be learned quickly** if they are **added to the existing training data**.

Computing Times

Data:

	Avg. Runtime per Case (Serial)
Create STL	0.15 s
snappyHexMesh	37 s
simpleFoam	13 s
Total Time	$\approx$ 50 s

Training:

	Bottleneck CNN		U-Net	
# decoders	1	2	1	2
# parameters	$\approx$ 47 m	$\approx$ 85 m	$\approx$ 34 m	$\approx$ 53.5 m
time/epoch	**180 s**	**245 s**	**195 s**	**270 s**

Comparison CFD Vs NN:

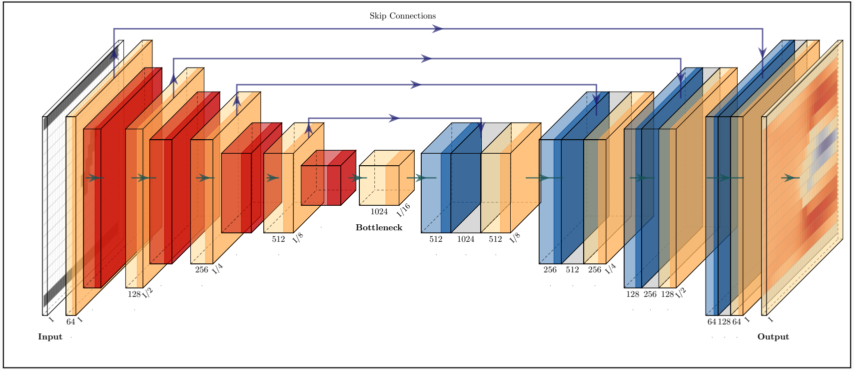
	CFD (CPU)	NN (CPU)	NN (GPU)
Avg. Time	**50 s**	**0.092 s**	**0.0054 s**

$\Rightarrow$ Flow predictions using neural networks may be less accurate and the **training phase expensive**, but the **flow prediction is** $\approx 5 \cdot 10^2 - 10^4$ **times faster**.

CPU: AMD Threadripper 2950X (8 $\times$ 3.8 Ghz), 32GB RAM; **GPU:** GeForce RTX 2080Ti

Surrogate models for CFD simulations
Physics-aware approach

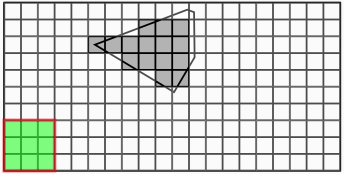
U-Net Revisited & Action of a Discrete Convolution

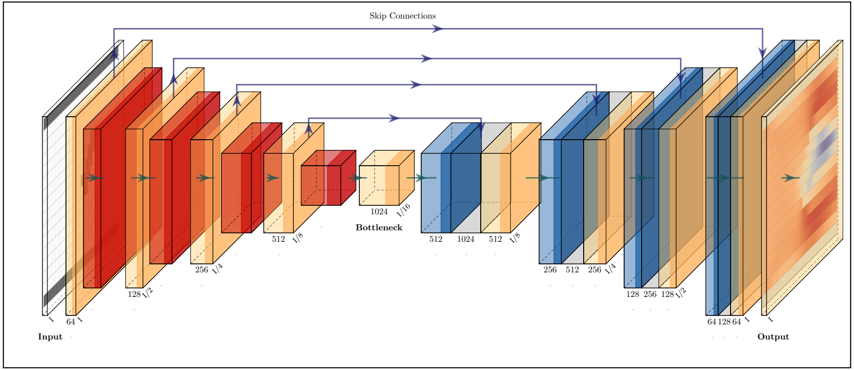


Convolution

The action of a **convolutional layer** corresponds to **going over the image with a filter** (matrix):

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

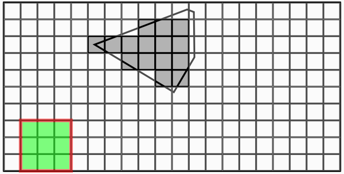


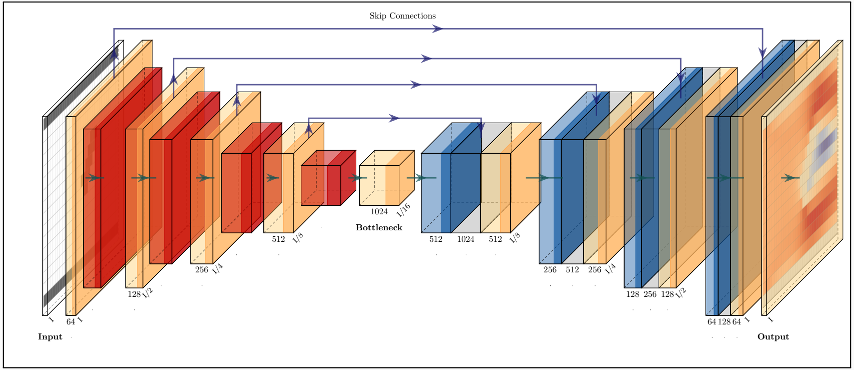


Convolution

The action of a **convolutional layer** corresponds to **going over the image with a filter** (matrix):

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

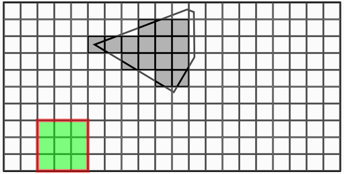


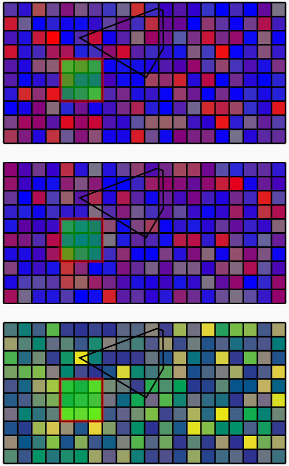


Convolution

The action of a **convolutional layer** corresponds to **going over the image with a filter** (matrix):

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$





Minimization of the mean squared residual of the Navier-Stokes equations

$$\min_{u_{NN}, p_{NN}} \frac{1}{\#\text{pixels}} \sum_{\text{pixels}} \left\| \begin{matrix} F_{\text{mom}}(u_{NN}, p_{NN}) \\ F_{\text{mass}}(u_{NN}, p_{NN}) \end{matrix} \right\|^2$$

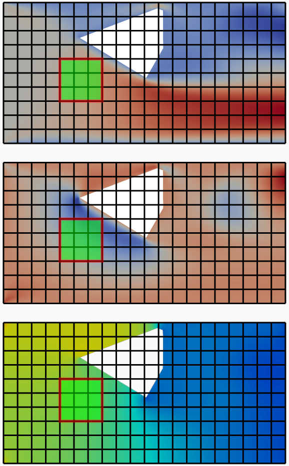
where u_{NN} and p_{NN} are the output images of our CNN and

$$\begin{aligned} F_{\text{mom}}(u, p) &:= \quad -\nu \Delta \vec{u} + (u \cdot \nabla)\, \vec{u} + \nabla p, \\ F_{\text{mass}}(u, p) &:= \quad \nabla \cdot u. \end{aligned}$$

We use a **finite difference discretization on the output pixel image** by defining filters on the last layer of the CNN-based on the stencils:

$$\frac{\partial}{\partial x} \quad \begin{matrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{matrix} \quad \frac{\partial}{\partial y}$$

$$\frac{\partial^2}{\partial x^2} \quad \begin{matrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{matrix} \quad \frac{\partial^2}{\partial y^2}$$



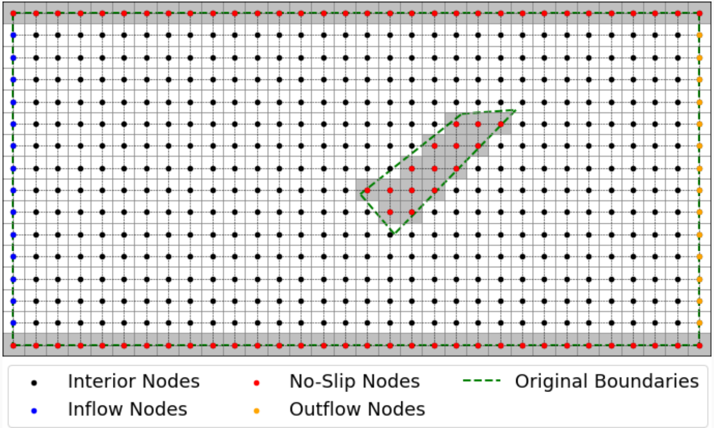
$$\left\| \begin{matrix} F_{\text{mom}}(u_{NN}, p_{NN}) \\ F_{\text{mass}}(u_{NN}, p_{NN}) \end{matrix} \right\|^2 >> 0$$

$$\left\| \begin{matrix} F_{\text{mom}}(u_{NN}, p_{NN}) \\ F_{\text{mass}}(u_{NN}, p_{NN}) \end{matrix} \right\|^2 \approx 0$$

Cf. **Grimm, Heinlein, Klawonn**

The PDE loss can be minimized **without using simulation results as training data**.

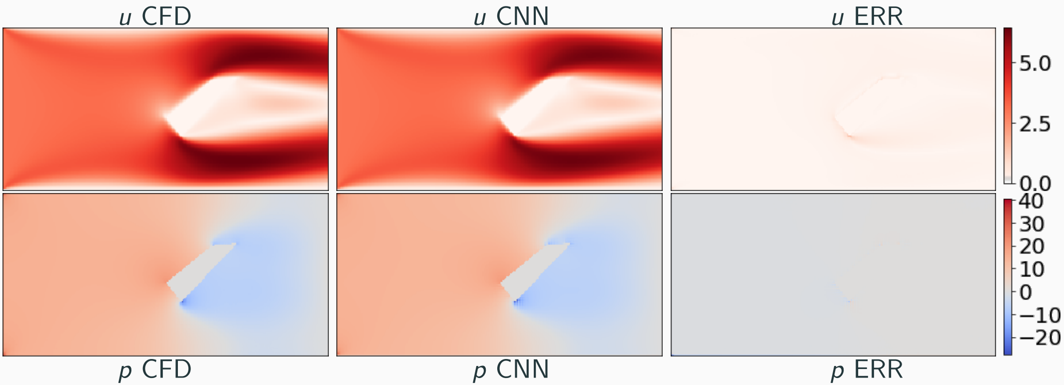
→ On a single geometry, this training of the neural network just corresponds to an **unconventional way of solving the Navier-Stokes equations using finite differences**.

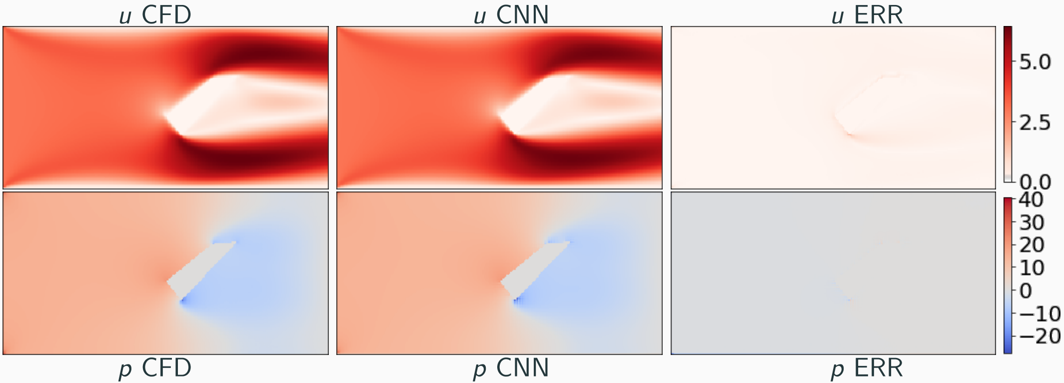


Boundary conditions

- Computing the correct solution requires **enforcing the correct boundary conditions**.
- Therefore, we additionally encode **flags for the different boundary conditions** in the **input image**.

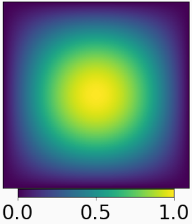
Physics-Informed Approach – Single Geometry





⇒ We can **solve the boundary value problem using a neural network**. Let us briefly discuss why, for a **single geometry**, this is **not an efficient solver**.

Convergence Comparison – CNN Versus FDM



Solve

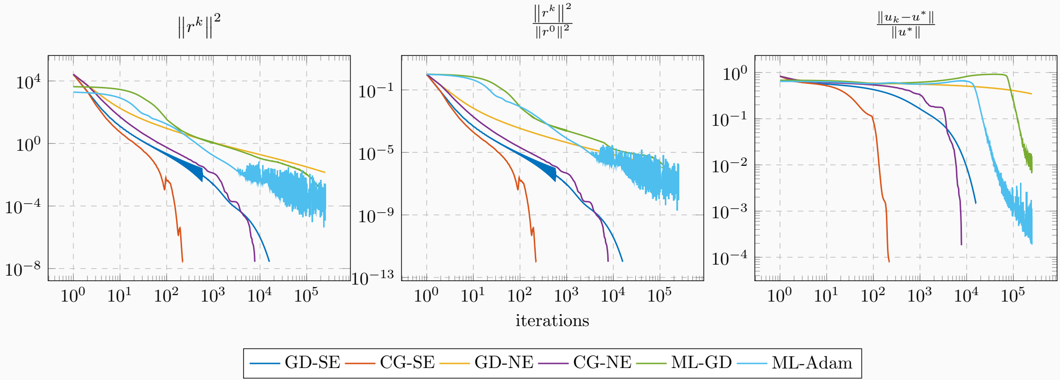
$$-\Delta u = f$$

using

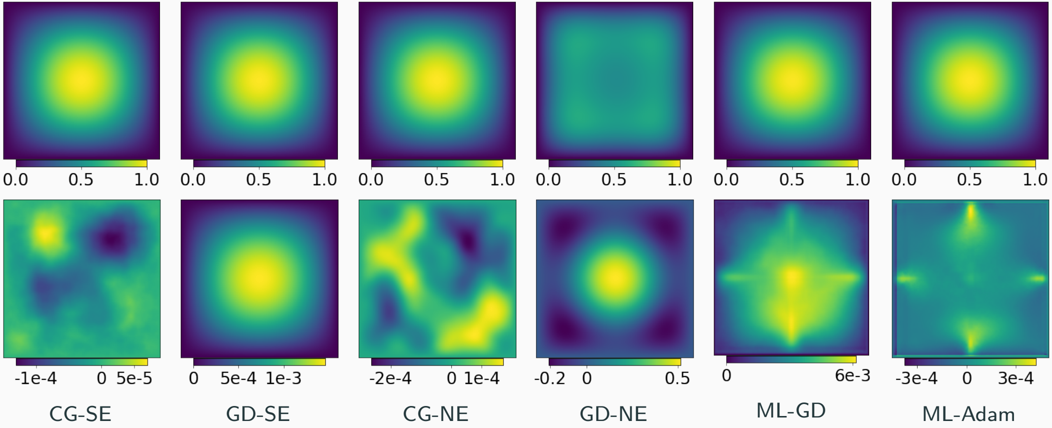
- classical finite differences
- **ML**: CNN

- **GD**: gradient descent
- **CG**: conjugate gradient method

- **SE**: $Ax = b$
- **NE**: $\|Ax - b\|^2$



Convergence Comparison – CNN Versus FDM

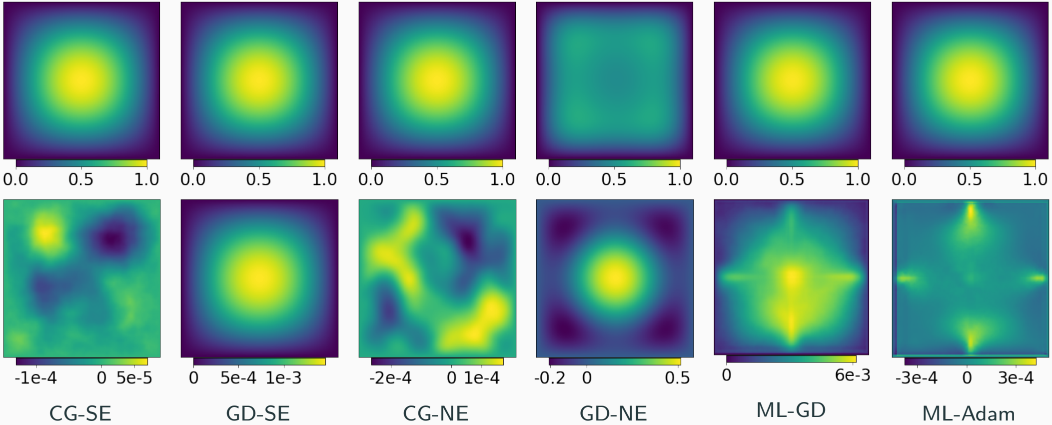


The results are in alignment with the **spectral bias of neural networks**. The neural network approximations yield a low error norm compared with the residual (MSE loss).

$$Ae = A(u^* - u) = b - Au = r$$

Cf. **Grimm, Heinlein, Klawonn (accepted 2023)**.

Convergence Comparison – CNN Versus FDM



The results are in alignment with the **spectral bias of neural networks**. The neural network approximations yield a low error norm compared with the residual (MSE loss).

$$Ae = A(u^* - u) = b - Au = r$$

Cf. Grimm, Heinlein, Klawonn (accepted 2023).

→ **Next:** surrogate model for multiple geometries

Results on $\approx 5\,000$ Type II Geometries

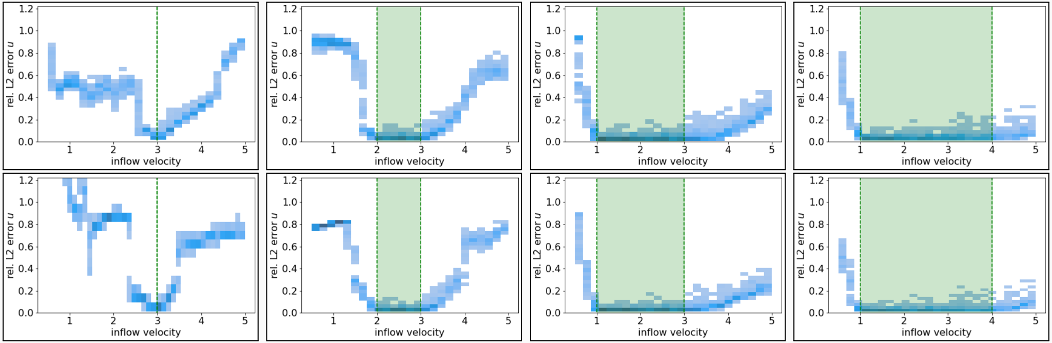
	training data	error	$\frac{\lVert u_{NN}-u \rVert_2}{\lVert u \rVert_2}$	$\frac{\lVert p_{NN}-p \rVert_2}{\lVert p \rVert_2}$	mean residual momentum	mean residual mass	# epochs trained
data-based	10%	train.	2.07%	10.98%	$1.1 \cdot 10^{-1}$	$1.4 \cdot 10^{0}$	500
		val.	4.48 %	15.20 %	$1.6 \cdot 10^{-1}$	$1.7 \cdot 10^{0}$	
	25%	train.	1.93%	8.45%	$9.1 \cdot 10^{-2}$	$1.2 \cdot 10^{0}$	500
		val.	3.49 %	10.70 %	$1.2 \cdot 10^{-1}$	$1.4 \cdot 10^{0}$	
	50%	train.	1.48%	8.75%	$9.0 \cdot 10^{-2}$	$1.1 \cdot 10^{0}$	500
		val.	2.70 %	10.09 %	$1.1 \cdot 10^{-1}$	$1.2 \cdot 10^{0}$	
	75%	train.	1.43%	7.30%	$1.0 \cdot 10^{-1}$	$1.5 \cdot 10^{0}$	500
		val.	**2.52 %**	**8.67 %**	$1.2 \cdot 10^{-1}$	$1.5 \cdot 10^{0}$	
physics-informed	10%	train.	5.35%	12.95%	$3.5 \cdot 10^{-2}$	$7.8 \cdot 10^{-2}$	**5 000**
		val.	6.72%	15.39%	$6.7 \cdot 10^{-2}$	$2.0 \cdot 10^{-1}$	
	25%	train.	5.03%	12.26%	$3.2 \cdot 10^{-2}$	$7.3 \cdot 10^{-2}$	**5 000**
		val.	5.78 %	13.38 %	$5.3 \cdot 10^{-2}$	$1.4 \cdot 10^{-1}$	
	50%	train.	5.81%	12.92%	$3.9 \cdot 10^{-2}$	$9.3 \cdot 10^{-2}$	**5 000**
		val.	5.84 %	12.73 %	$4.8 \cdot 10^{-2}$	$1.2 \cdot 10^{-1}$	
	75%	train.	5.03%	11.63%	$3.2 \cdot 10^{-2}$	$7.7 \cdot 10^{-2}$	**5 000**
		val.	**5.18 %**	**11.60 %**	$4.2 \cdot 10^{-2}$	$1.1 \cdot 10^{-1}$	

Results on $\approx$ 5 000 Type II Geometries

	training data	error	$\frac{\|u_{NN}-u\|_2}{\|u\|_2}$	$\frac{\|p_{NN}-p\|_2}{\|p\|_2}$	mean residual momentum	mean residual mass	# epochs trained
data-based	10%	train.	2.07%	10.98%	$1.1 \cdot 10^{-1}$	$1.4 \cdot 10^{0}$	500
		val.	4.48 %	15.20 %	$1.6 \cdot 10^{-1}$	$1.7 \cdot 10^{0}$	
	25%	train.	1.93%	8.45%	$9.1 \cdot 10^{-2}$	$1.2 \cdot 10^{0}$	500
		val.	3.49 %	10.70 %	$1.2 \cdot 10^{-1}$	$1.4 \cdot 10^{0}$	
	50%	train.	1.48%	8.75%	$9.0 \cdot 10^{-2}$	$1.1 \cdot 10^{0}$	500
		val.	2.70 %	10.09 %	$1.1 \cdot 10^{-1}$	$1.2 \cdot 10^{0}$	
	75%	train.	1.43%	7.30%	$1.0 \cdot 10^{-1}$	$1.5 \cdot 10^{0}$	500
		val.	**2.52 %**	**8.67 %**	$1.2 \cdot 10^{-1}$	$1.5 \cdot 10^{0}$	
physics-informed	10%	train.	5.35%	12.95%	$3.5 \cdot 10^{-2}$	$7.8 \cdot 10^{-2}$	**5 000**
		val.	6.72%	15.39%	$6.7 \cdot 10^{-2}$	$2.0 \cdot 10^{-1}$	
	25%	train.	5.03%	12.26%	$3.2 \cdot 10^{-2}$	$7.3 \cdot 10^{-2}$	**5 000**
		val.	5.78 %	13.38 %	$5.3 \cdot 10^{-2}$	$1.4 \cdot 10^{-1}$	
	50%	train.	5.81%	12.92%	$3.9 \cdot 10^{-2}$	$9.3 \cdot 10^{-2}$	**5 000**
		val.	5.84 %	12.73 %	$4.8 \cdot 10^{-2}$	$1.2 \cdot 10^{-1}$	
	75%	train.	5.03%	11.63%	$3.2 \cdot 10^{-2}$	$7.7 \cdot 10^{-2}$	**5 000**
		val.	**5.18 %**	**11.60 %**	$4.2 \cdot 10^{-2}$	$1.1 \cdot 10^{-1}$	

→ The results for the **physics-informed approach** are **comparable to the data-based approach**; the **errors are slightly higher**. However, no **reference data at all is needed for the training**.

Generalization With Respect to the Inflow Velocity

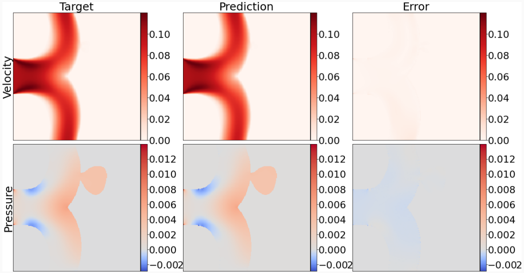


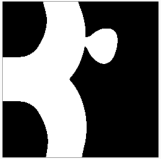
order	# data	range inflow vel.	[0.5, 1.0]	[1.0, 2.0]	[2.0, 3.0]	[3.0, 4.0]	[4.0, 5.0]
2	1 000	[3.0, 3.0]	55.5 %	48.1 %	31.1 %	17.4 %	61.5 %
		[2.0, 3.0]	89.3 %	57.4 %	**4.0 %**	15.5 %	59.1 %
		[1.0, 3.0]	40.2 %	**3.8 %**	**4.3 %**	7.1 %	20.4 %
		[1.0, 4.0]	31.3 %	**4.0 %**	**4.3 %**	**5.8 %**	7.7 %
2	4 500	[3.0, 3.0]	186.8 %	87.1 %	40.5 %	36.9 %	70.6 %
		[2.0, 3.0]	78.4 %	44.3 %	**3.2 %**	16.1 %	68.2 %
		[1.0, 3.0]	38.7 %	**2.9 %**	**3.4 %**	6.7 %	18.5 %
		[1.0, 4.0]	27.7 %	**3.1 %**	**3.4 %**	**4.7 %**	7.2 %

Training: 500 geometries **Validation:** $\approx 1\,200$ geometries

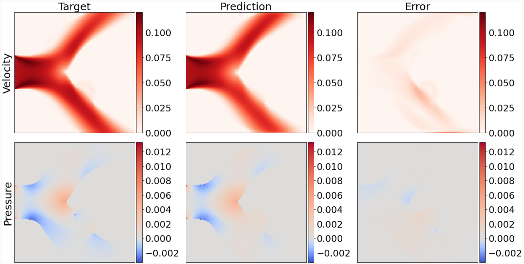
Relative $L2$**-error** on the **validation data set** in u: **4.9** %, in p: **9.5** %.

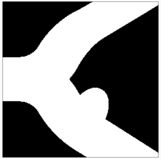
u: 3.3%, p: 7.4%

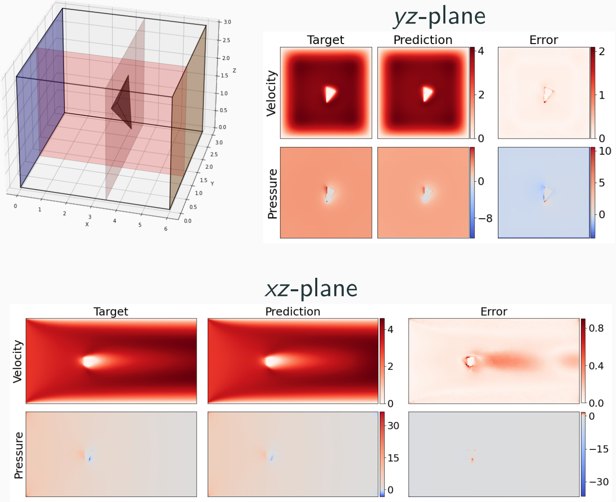




u: 13.2%, p: 35.1%





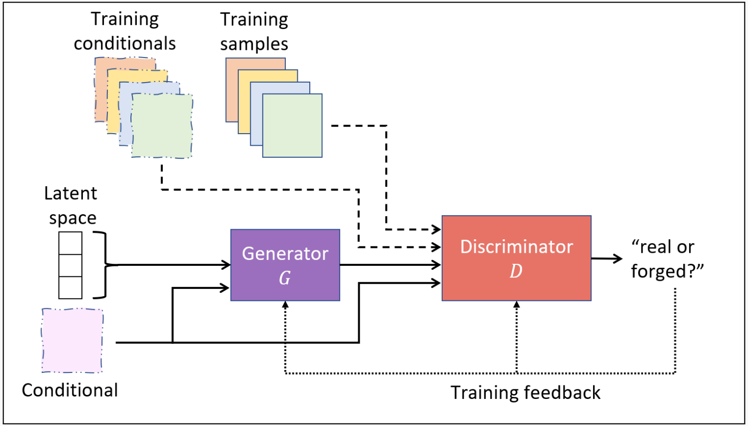


- We extend our approach to three dimensions.
  Pixel images of size $256 \times 128 \times 128 \rightarrow$ increased computation time and memory demand.
- Training on 100 geometries, validation on 83 geometries.
- Rel. $L2$-error on the validation dataset in u: 3.9%, in p: 17.5%.

Surrogate models for CFD simulations
GAN-based training

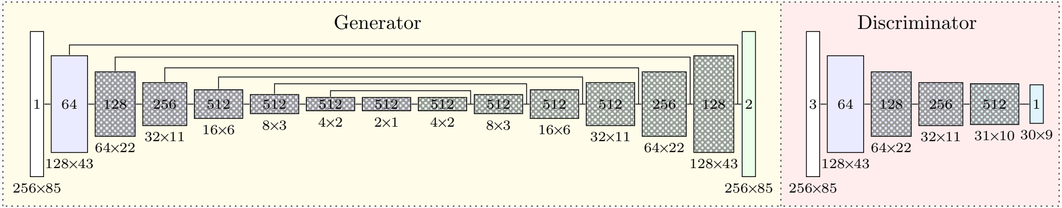
Training the Surrogate Model via GANs

Cf. Kemna, Heinlein, Vuik (2023).

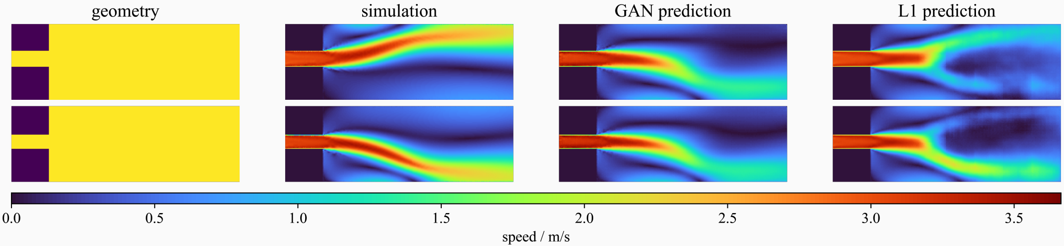


- **Generative adversarial networks (GANs)** based on Goodfellow et al. (2014) consist of two independent neural networks that are trained concurrently in an adversarial setting:
  - **Generator** is trained to *fool* the discriminator into classifying its outputs as training data
  - **Discriminator** is trained to distinguish between generated samples and training data

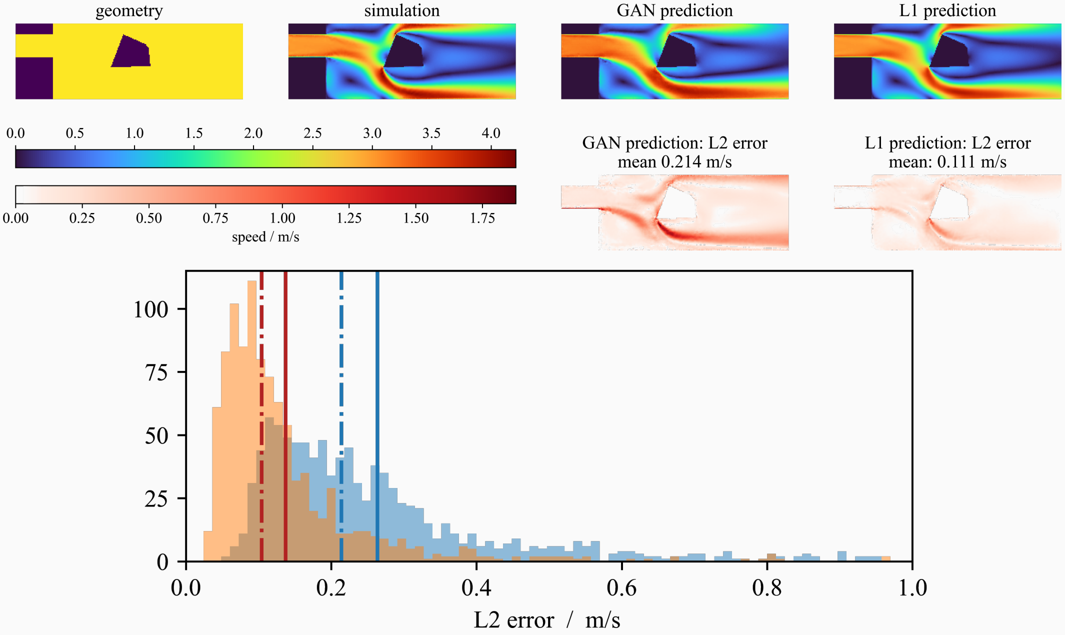
GANs for Fluid Prediction – Bifurcation Example



For investigating the effect of training the surrogate model as a generator of a GAN, consider the following **sudden expansion scenario** (see, e.g., Mullin et al. (2009)), which leads to a **bifurcation if the inlet is centered**.



GANs for Fluid Prediction – Overall Performance

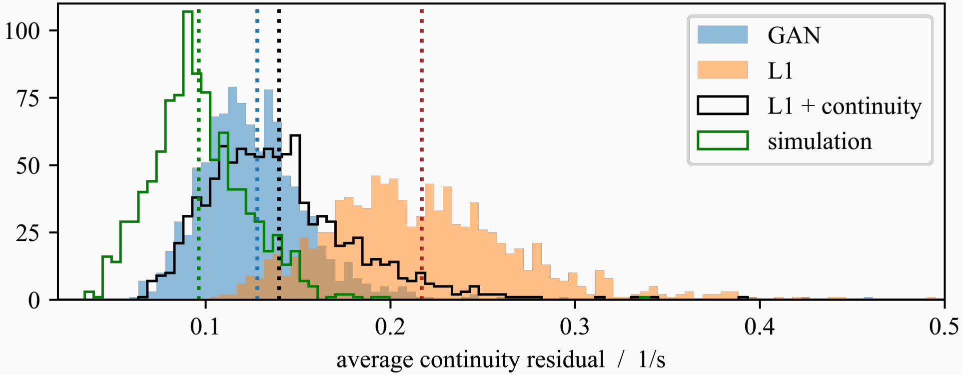


GANs for Fluid Prediction – Divergence Error

Let us investigate how well the predictions satisfy the continuity equation in the Navier–Stokes equations:

$$-\nu \Delta \vec{u} + (u \cdot \nabla)\, \vec{u} + \nabla p = 0 \text{ in } \Omega,$$

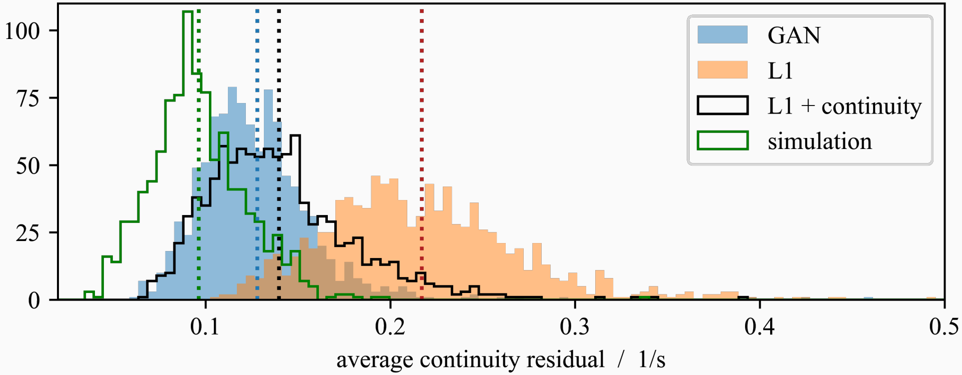
$$\nabla \cdot u = 0 \text{ in } \Omega.$$



GANs for Fluid Prediction – Divergence Error

Let us investigate how well the predictions satisfy the continuity equation in the Navier–Stokes equations:

$$-\nu \Delta \vec{u} + (u \cdot \nabla)\, \vec{u} + \nabla p = 0 \text{ in } \Omega,$$
$$\nabla \cdot u = 0 \text{ in } \Omega.$$



→ The GAN loss seems to **help learning the physics of the system**.

Summary

Scientific machine learning

- The field of **scientific machine learning (SciML)** deals with the **combination of scientific computing and machine learning** techniques; **physics-informed machine learning** models allow for the **combination of physical models and data**.

Finite basis physics-informed neural networks

- **Schwarz domain decomposition methods** can help to **improve the performance of PINNs**, especially for (but not restricted to) **large domains** and/or **multiscale problems**.

Surrogate models

- **CNNs** yield an **operator learning approach for predicting fluid flow** inside **varying computational domains**; the model can be trained using **data and/or physics**.

Acknowledgements

- The **Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE)**
- **German Research Foundation (DFG)** [project number 277972093]

Thank you for your attention!