

# Neural networks with physical constraints

Domain decomposition-based network architectures and model order reduction

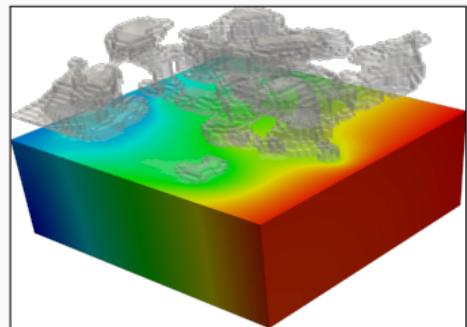
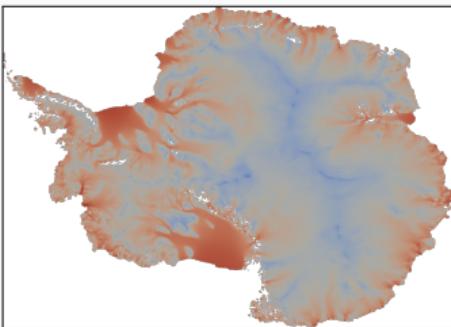
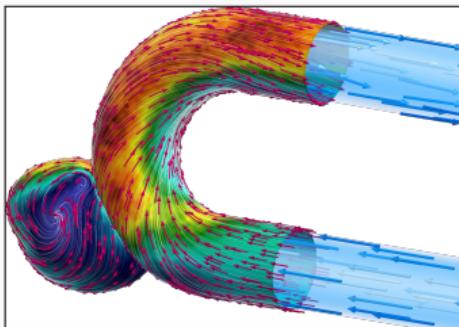
---

Alexander Heinlein<sup>1</sup>

Seminar talk, Technical University Munich, July 13, 2023

<sup>1</sup>Delft University of Technology

# Scientific Machine Learning in Computational Science and Engineering



## 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	<b>improve</b>	machine learning techniques
machine learning techniques	<b>assist</b>	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)

## 6 Reconstruction of nonlinear ocean wave surface elevations

Based on joint work with **Svenja Ehlers**, **Mathies Wedler** (TU Hamburg), **Nicolas Desmars** (TU Hamburg & DLR), **Norbert Hoffmann** (TU Hamburg & Imperial College London), and **Merten Stender** (TU Berlin)

# **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

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

by solving an **optimization problem**

$$\min_{\theta} \sum_{x_i} G(x_i, \Psi_t(x_i, \theta), \nabla\Psi_t(x_i, \theta), \nabla^2\Psi_t(x_i, \theta))^2$$

where  $\Psi_t(x, \theta)$  is a **trial function**,  $x_i$  sampling points inside the domain  $\Omega$  and  $\theta$  are adjustable parameters.

### Construction of the trial functions

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

$$\Psi_t(x, p) = A(x) + F(x, N(x, p))$$

- $N$  is a **feedforward neural network** with **trainable parameters**  $\theta$  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**

# Neural Networks for Solving Differential Equations

## Approach

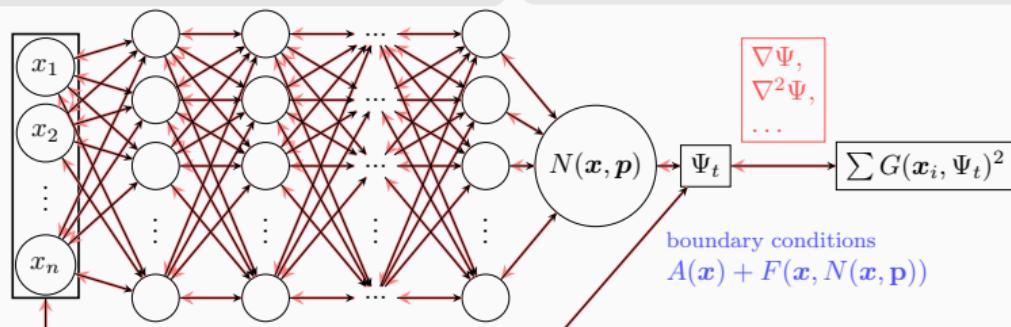
Solve a general differential equation subject to boundary conditions

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

by solving an **optimization problem**

$$\min_{\theta} \sum_{x_i} G(x_i, \Psi_t(x_i, \theta), \nabla\Psi_t(x_i, \theta), \nabla^2\Psi_t(x_i, \theta))^2$$

where  $\Psi_t(x, \theta)$  is a **trial function**,  $x_i$  sampling points inside the domain  $\Omega$  and  $\theta$  are adjustable parameters.



## Construction of the trial functions

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

$$\Psi_t(x, p) = A(x) + F(x, N(x, p))$$

- $N$  is a **feedforward neural network** with **trainable parameters**  $\theta$  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**

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

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  $\omega_{\text{data}}$  and  $\omega_{\text{PDE}}$  are **weights** and

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

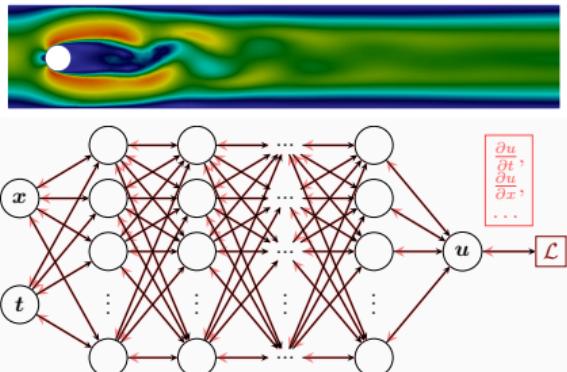
$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} (\mathcal{N}[u](\mathbf{x}_i, t_i) - f(\mathbf{x}_i, t_i))^2.$$

## Advantages

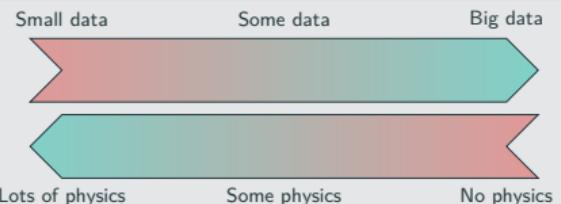
- "Meshfree"
- Small data
- Generalization properties
- High-dimensional problems
- Inverse and parameterized problems

## 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  $\mathcal{L}_{\text{data}}$

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

## Estimate of the generalization error

The generalization error (or total error) satisfies

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

where

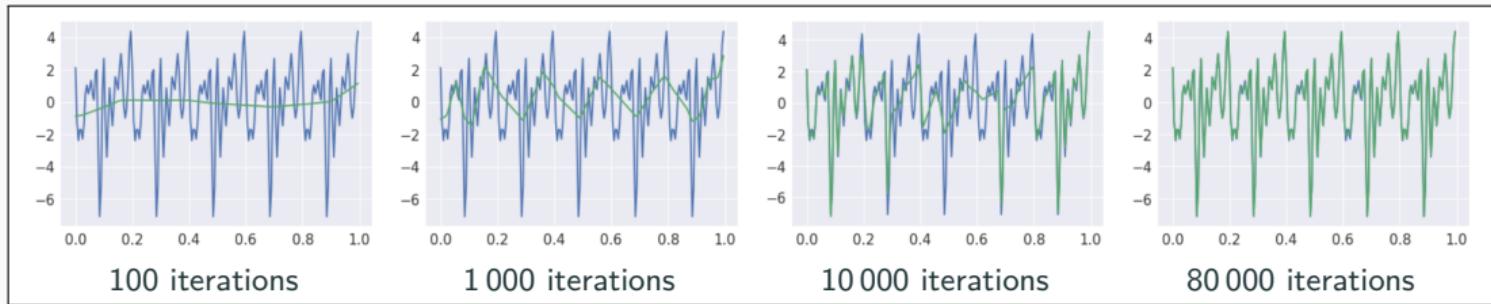
- $\mathcal{E}_G = \mathcal{E}_G(\theta; \mathbf{X}) := \|\mathbf{u} - \mathbf{u}^*\|_V$  ( $V$  Sobolev space,  $\mathbf{X}$  training data set)
- $\mathcal{E}_{\mathcal{T}}$  is the training error ( $l^p$  loss of the residual of the PDE)
- $C_{\text{PDE}}$  and  $C_{\text{quad}}$  constants depending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

---

# Motivation – Some Observations on the Performance of PINNs

Solve

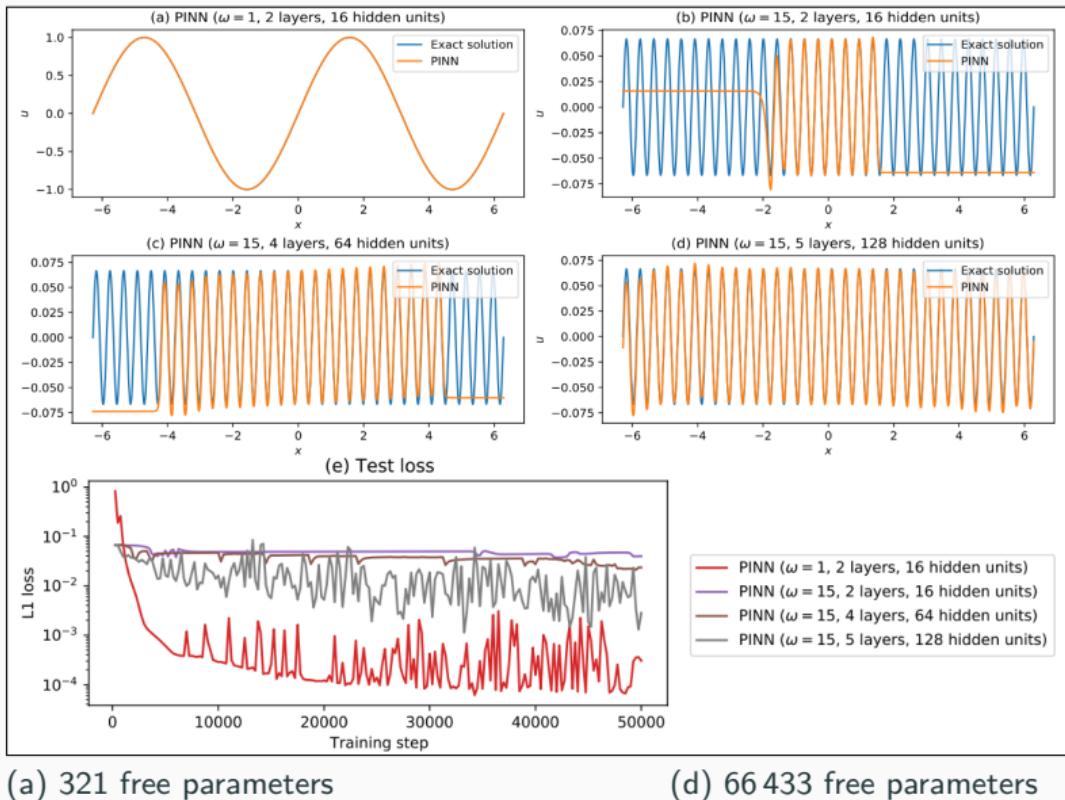
$$\begin{aligned} u' &= \cos(\omega x), \\ u(0) &= 0, \end{aligned}$$

for different values of  $\omega$   
using PINNs with  
varying network  
capacities.

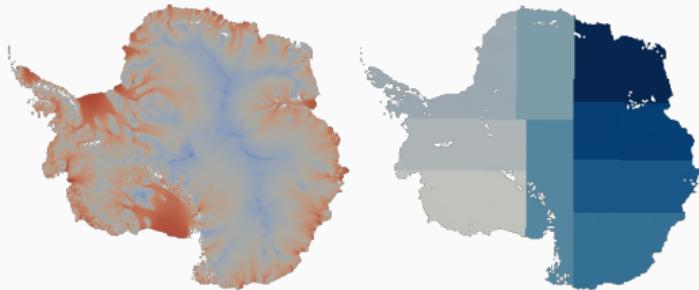
## Scaling issues

- Large computational domains
- Small frequencies

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



# Domain Decomposition Methods



Images based on Heinlein, Perego, Rajamanickam (2022)

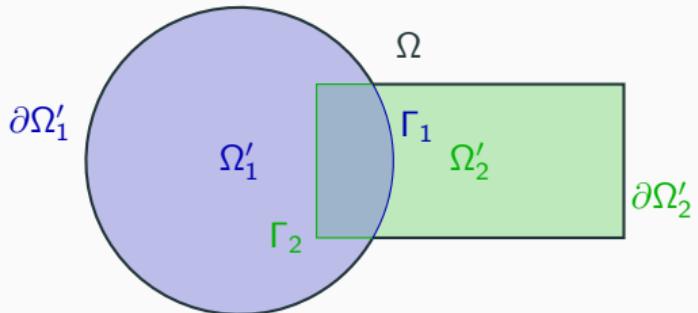
**Historical remarks:** The **alternating Schwarz method** is the earliest **domain decomposition method (DDM)**, which has been invented by **H. A. Schwarz** and published in **1870**:

- 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, Langer, Weber (2019, 2020, 2021, 2021, 2021, 2022); Klawonn, Langer, Weber (preprint 2022)
- Domain decomposition for CNNs: Gu, Zhang, Liu, Cai (2022); Lee, Park, Lee (2022); Klawonn, Langer, 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. Langer, 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

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

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

**FBPINNs** use the **network architecture**

$$u(\theta_1, \dots, \theta_J) = \mathcal{C} \sum_{j=1}^J \omega_j u_j(\theta_j)$$

and the **loss function**

$$\mathcal{L}(\theta_1, \dots, \theta_J) = \frac{1}{N} \sum_{i=1}^N \left( n[\mathcal{C} \sum_{x_i \in \Omega_j} \omega_j u_j](x_i, \theta_j) - f(x_i) \right)^2.$$

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

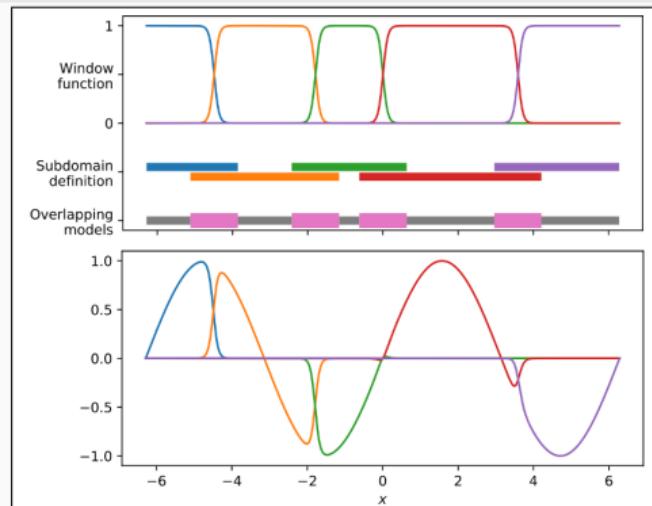
## Hard enforcement of boundary conditions

Loss function

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N (n[\mathcal{C} u](x_i, \theta) - f(x_i))^2,$$

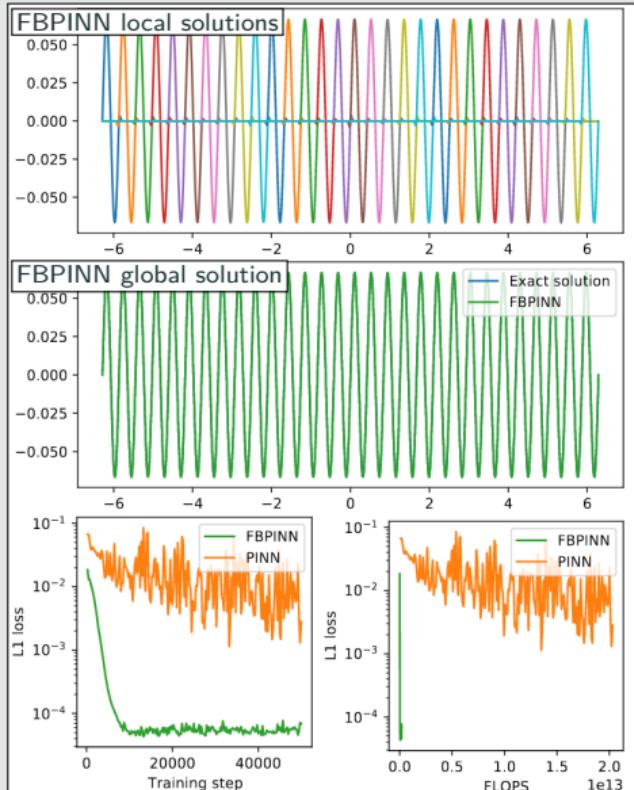
with constraining operator  $\mathcal{C}$ , which **explicitly enforces the boundary conditions**.

→ Often **improves training performance**



# Numerical Results for FBPINNs

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



## Scalability of FBPINNs

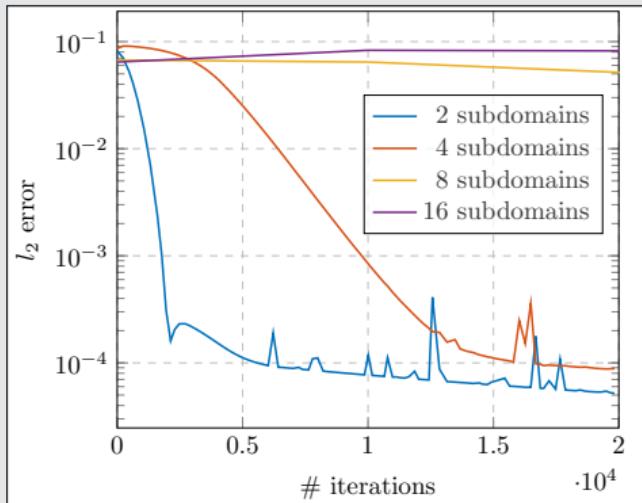
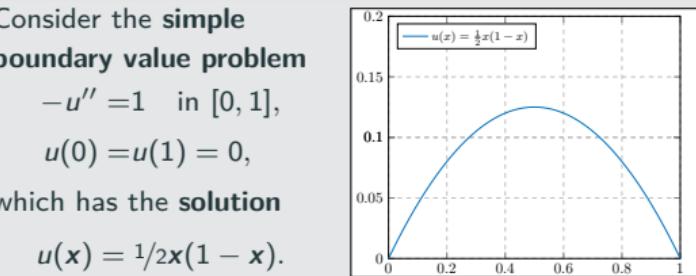
Consider the simple boundary value problem

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

$$u(0) = u(1) = 0,$$

which has the solution

$$u(x) = \frac{1}{2}x(1-x).$$



# 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:

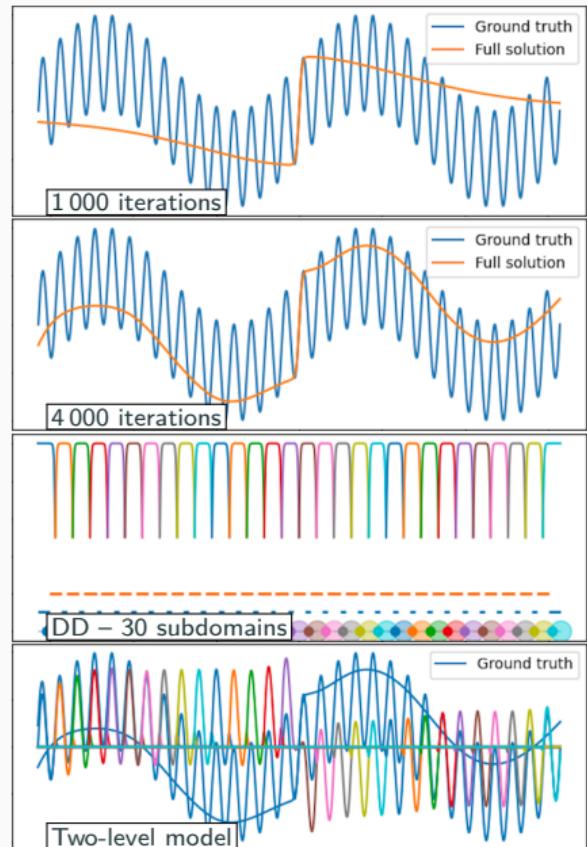
$$u(\theta_0, \theta_1, \dots, \theta_J) = \mathcal{C} \left( u_0(\theta_0) + \sum_{j=1}^J \omega_j u_j(\theta_j) \right)$$

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

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

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

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



# Numerical Results for FBPINNs – One Versus Two Levels

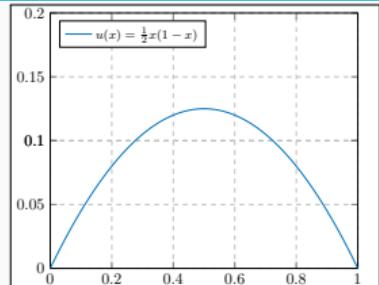
Consider, again, the **simple boundary value problem**

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

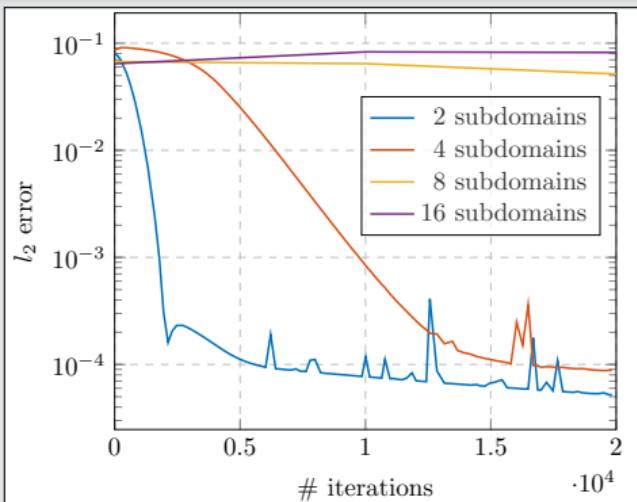
$$u(0) = u(1) = 0,$$

which has the **solution**

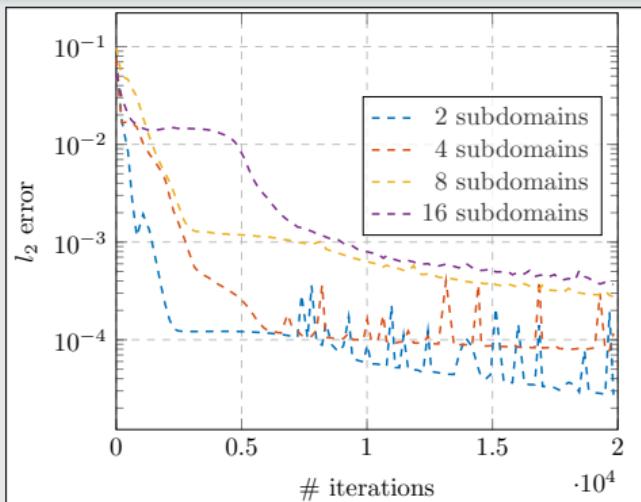
$$u(x) = \frac{1}{2}x(1-x).$$



## One-Level FBPINNs



## Two-Level FBPINNs



# Multi-Level FBPINN Algorithm

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

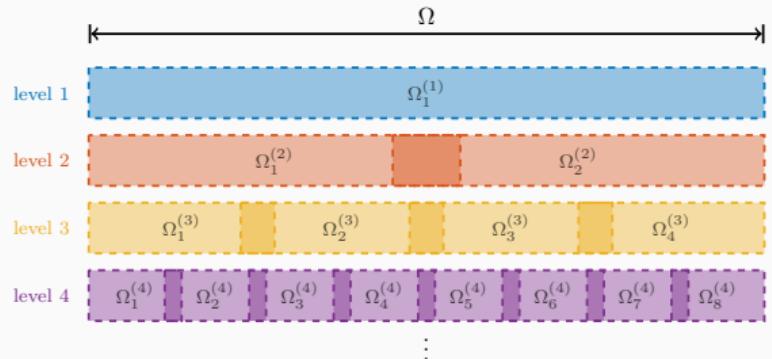
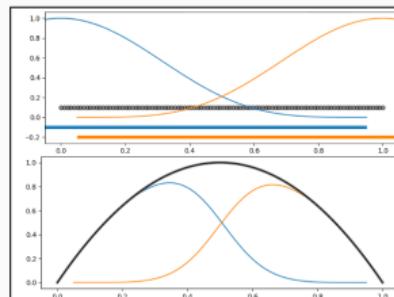
$$\Omega = \bigcup_{j=1}^{J^{(l)}} \Omega_j^{(l)}$$

and corresponding window functions  $\omega_j^{(l)}$  with  $\text{supp}(\omega_j^{(l)}) \subset \Omega_j^{(l)}$  and  $\sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \equiv 1$  on  $\Omega$ .

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

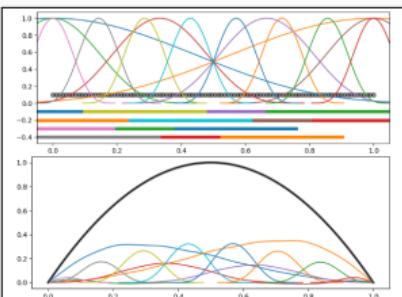
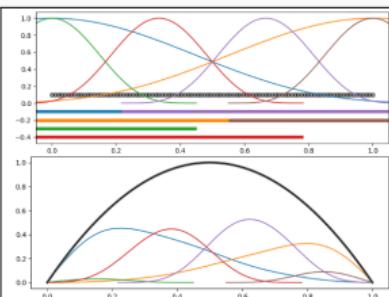
**$L$ -level network architecture**

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



**Loss function**

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



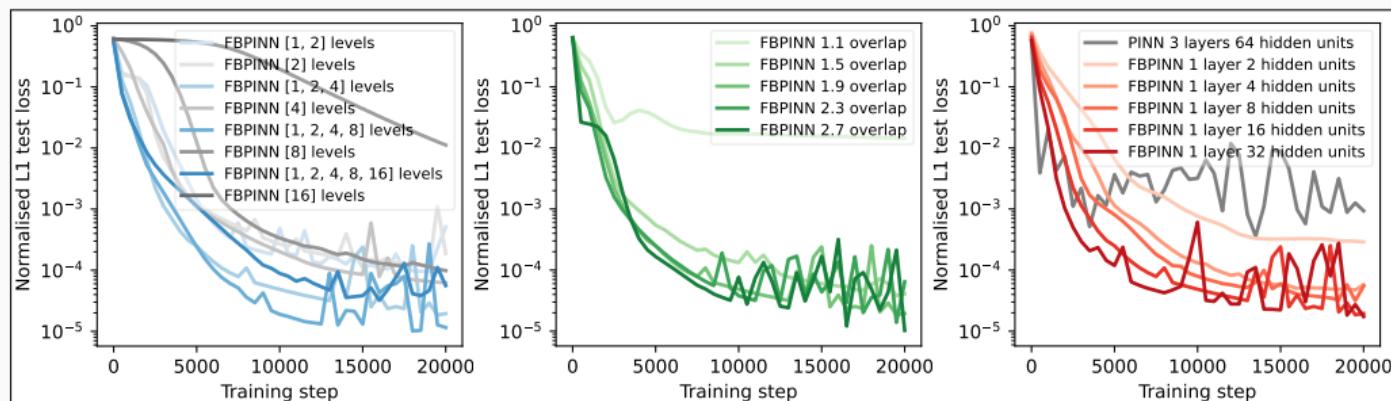
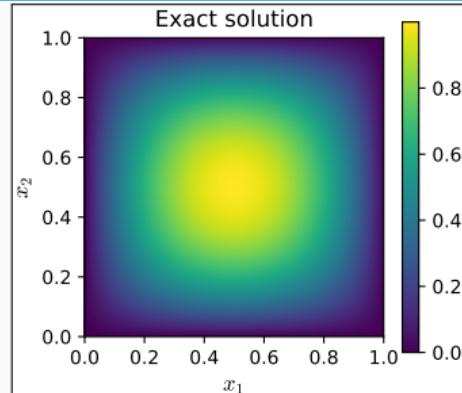
# Multilevel FBPINNs – 2D Laplace

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

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

which has the **solution**

$$u(x, y) = 16(x(1-x)y(1-y)).$$



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

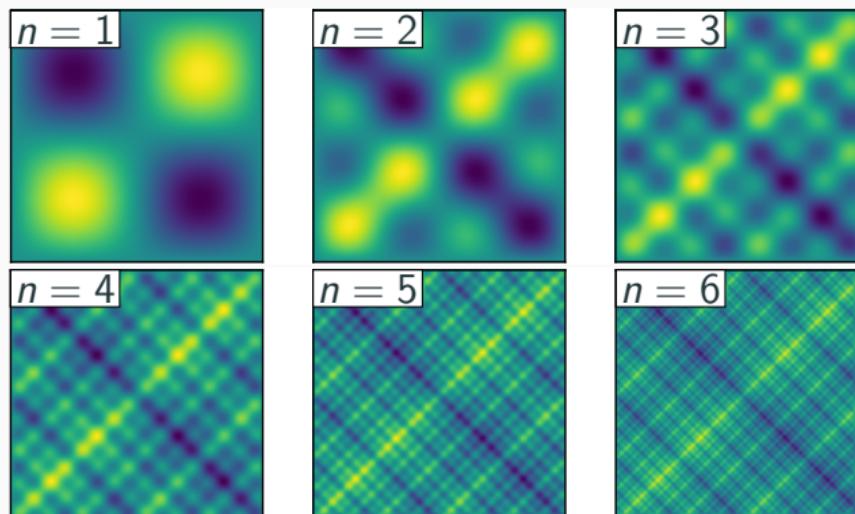
# Multi-Frequency Problem

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

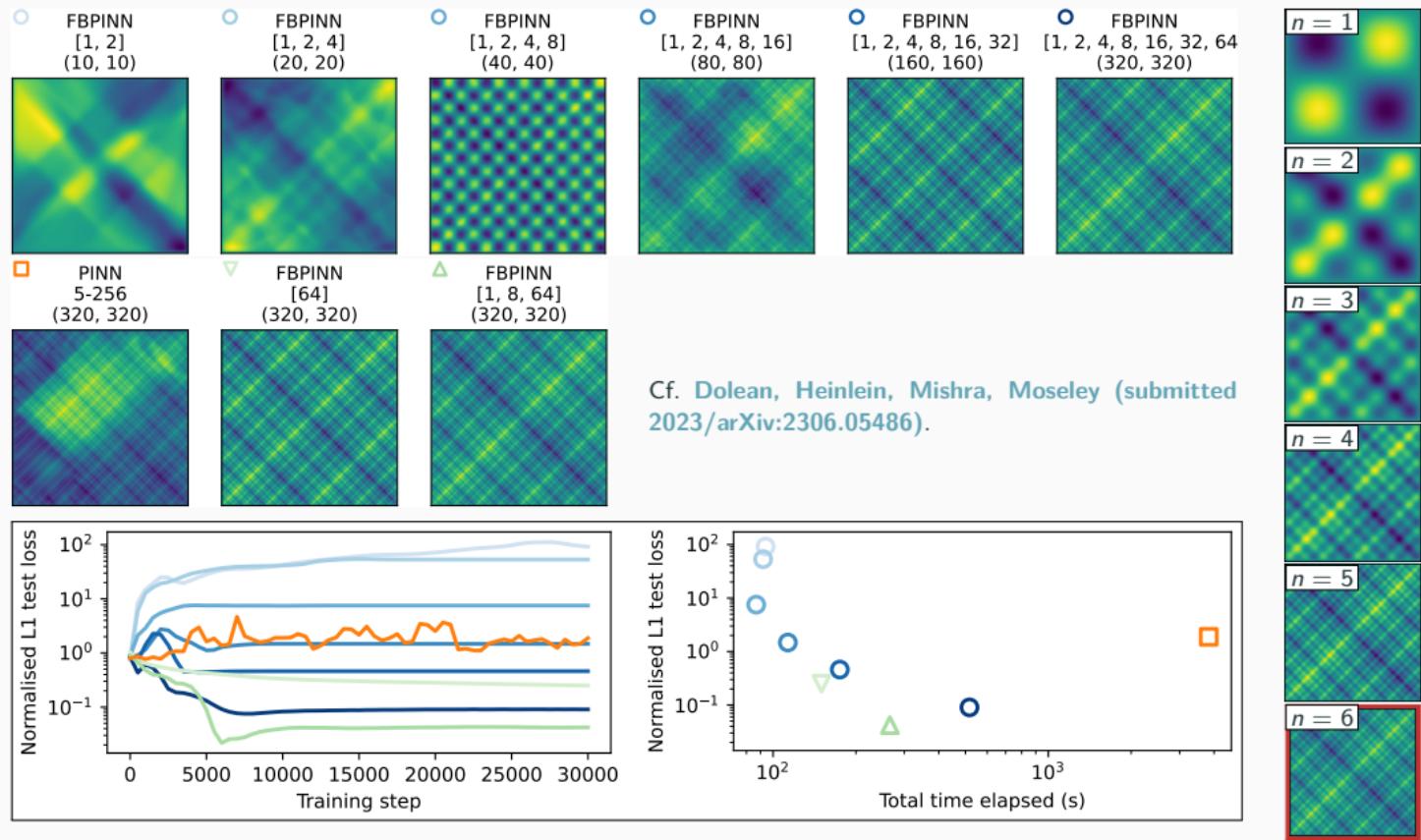
$$\begin{aligned} -\Delta u &= 2 \sum_{i=1}^n (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y) && \text{in } \Omega = [0, 1]^2, \\ u &= 0 && \text{on } \partial\Omega, \end{aligned}$$

with  $\omega_i = 2^i$ .

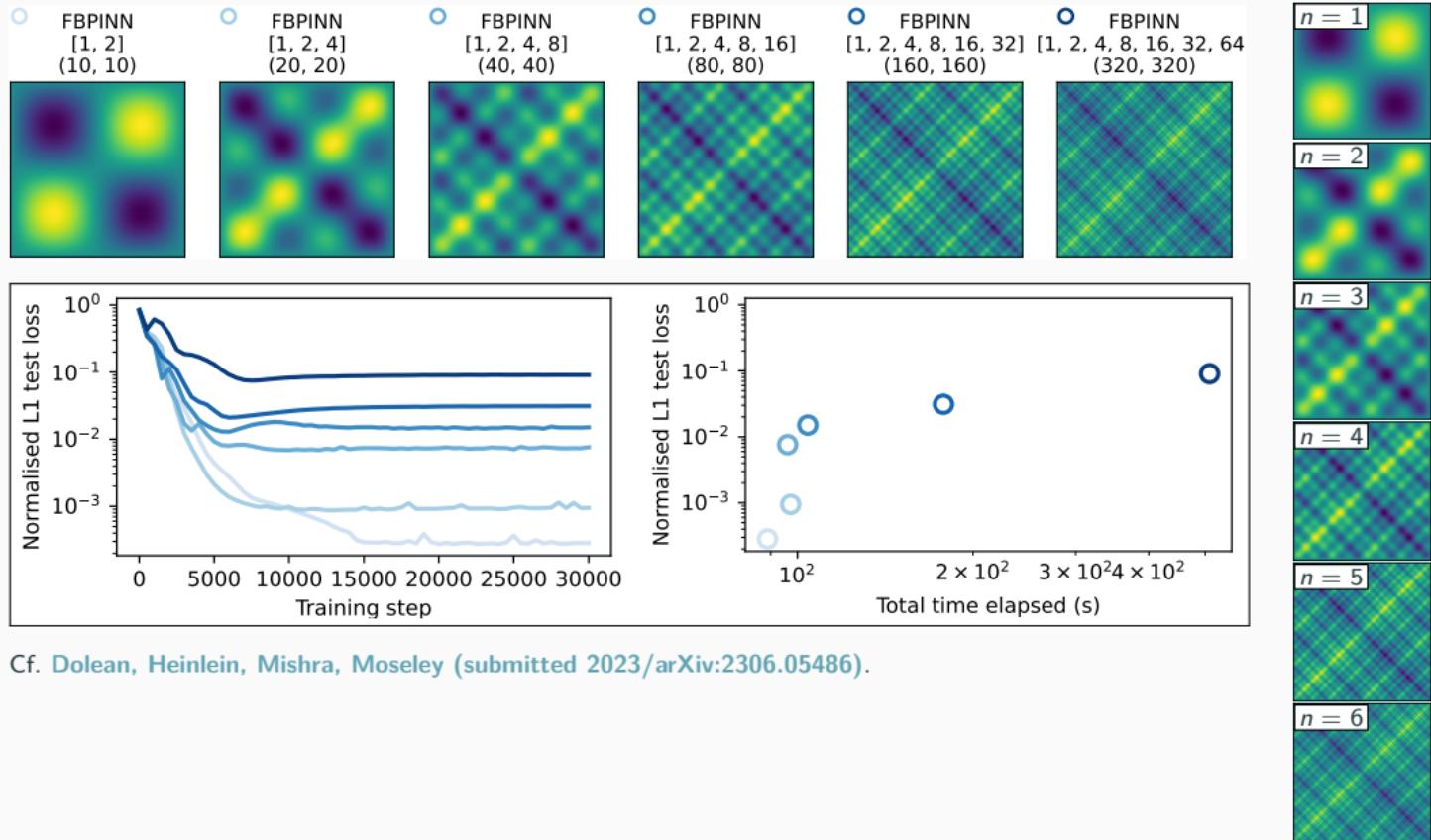
For increasing values of  $n$ , we obtain the **analytical solutions**:



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



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



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

# Helmholtz Problem

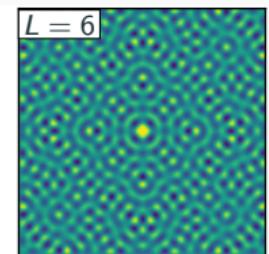
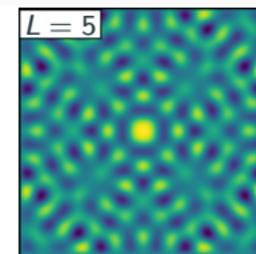
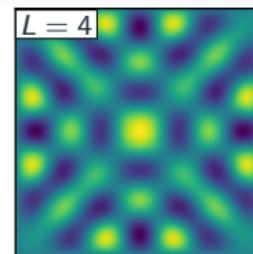
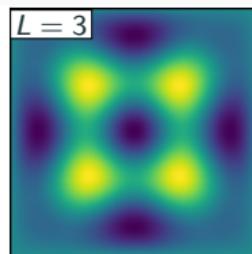
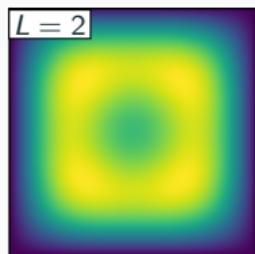
Finally, let us consider the **two-dimensional Helmholtz boundary value problem**

$$\Delta u - k^2 u = f \quad \text{in } \Omega = [0, 1]^2,$$

$$u = 0 \quad \text{on } \partial\Omega,$$

$$f(\mathbf{x}) = e^{-\frac{1}{2}(\|\mathbf{x}-0.5\|/\sigma)^2}.$$

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



# Multilevel FBPINNs – 2D Helmholtz Problem

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

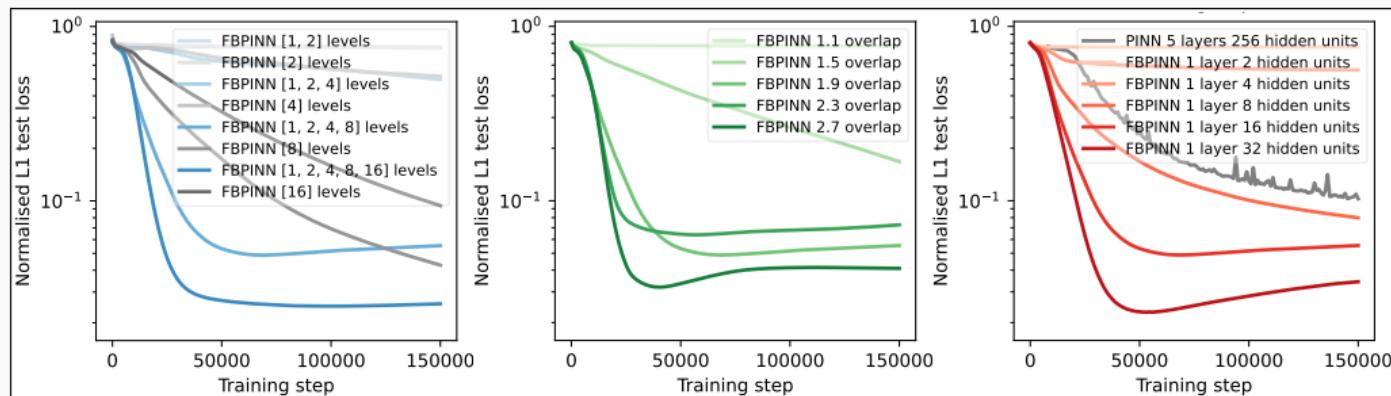
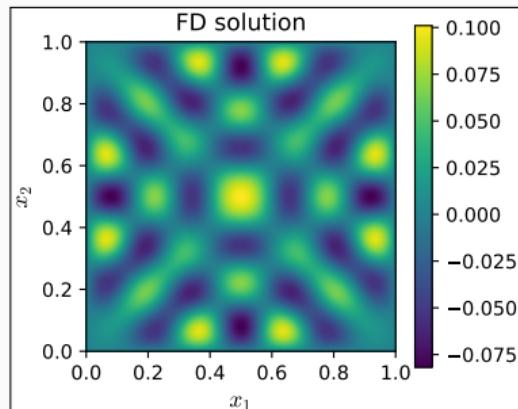
$$\Delta u - k^2 u = f \quad \text{in } \Omega = [0, 1]^2,$$

$$u = 0 \quad \text{on } \partial\Omega,$$

$$f(x) = e^{-\frac{1}{2}(\|x-0.5\|/\sigma)^2}.$$

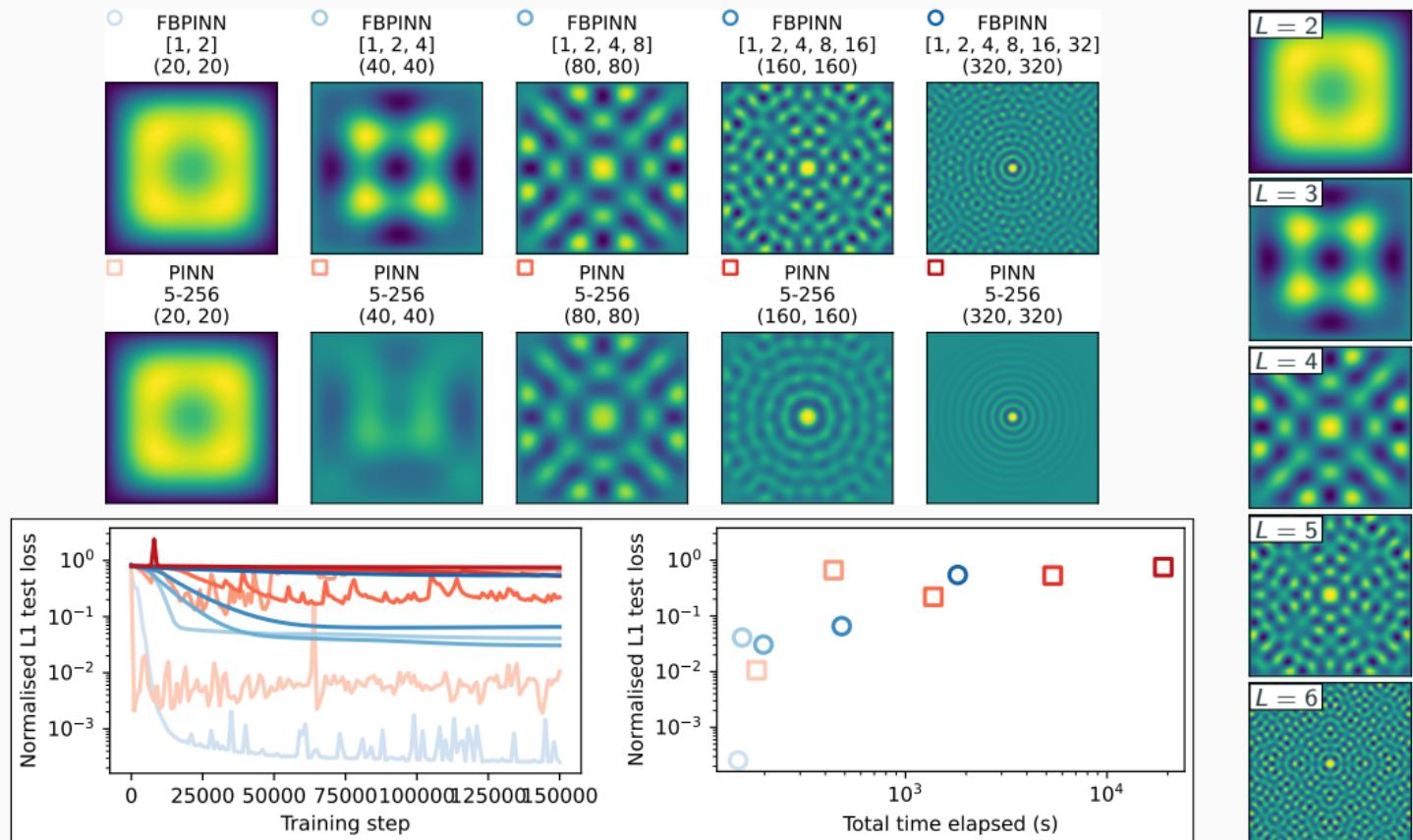
with  $k = 2^4\pi/1.6$  and  $\sigma = 0.8/2^4$ .

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



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

# Multi-Level FBPINNs for the Helmholtz Problem – Weak Scaling



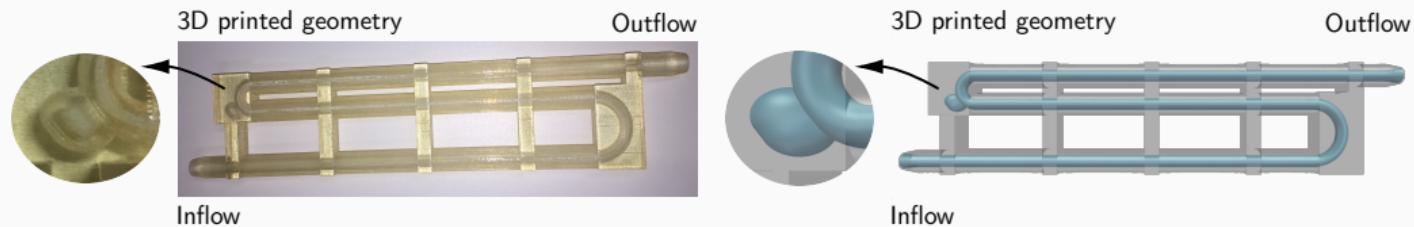
# **Surrogate models for CFD simulations**

## **Data-based approach**

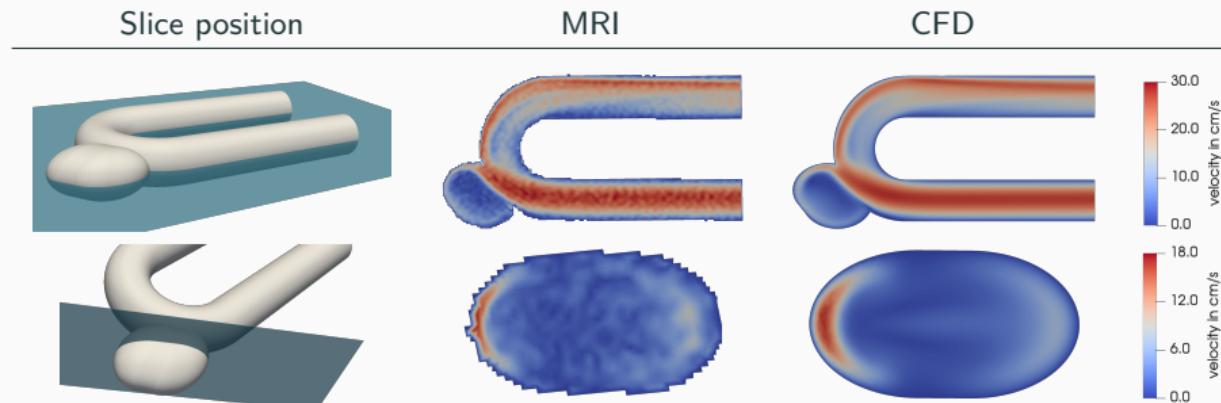
---

# 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.

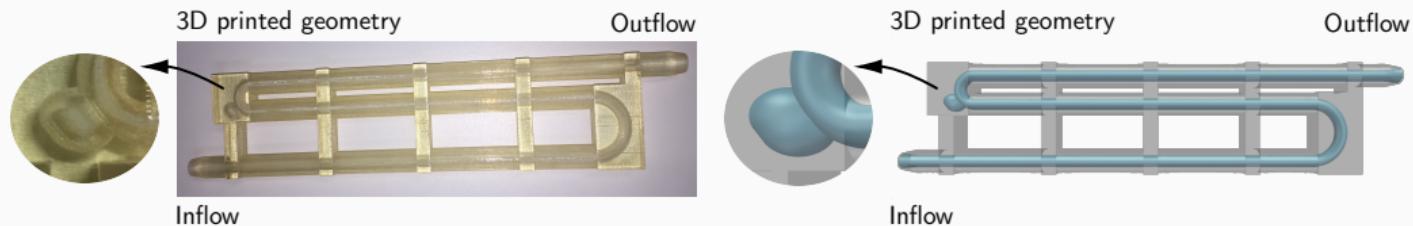


To obtain accurate simulation results, a simulation with  $\approx 10^6$  d.o.f.s has been carried out. On  $O(100)$  MPI ranks, the computation of a steady state took  $O(1)$  h on CHEOPS supercomputer at UoC.

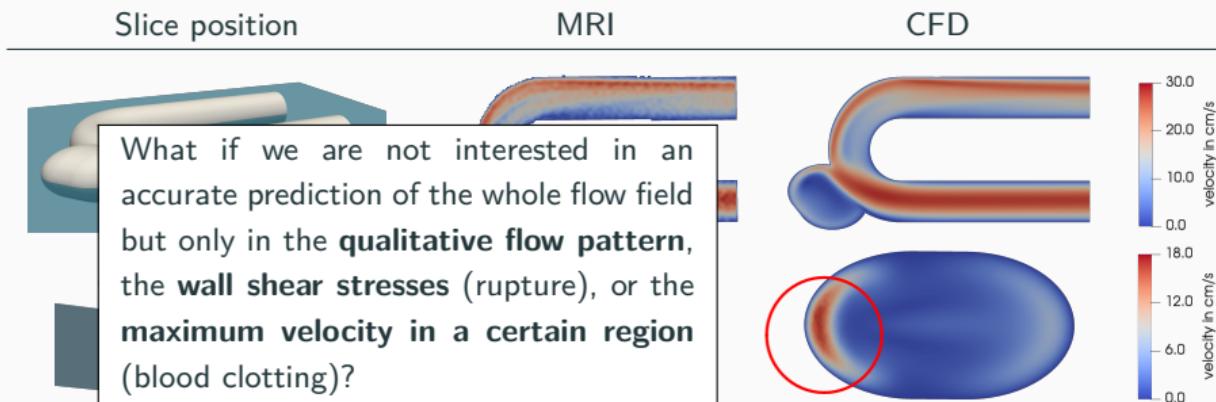


# Computational Fluid Dynamics (CFD) Simulations are Time Consuming

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



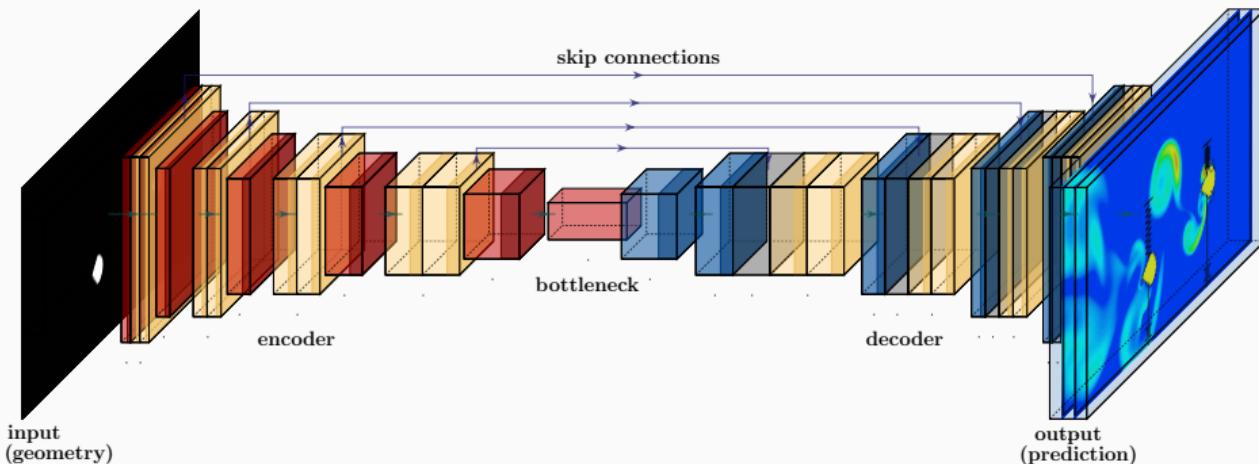
To obtain accurate simulation results, a simulation with  $\approx 10^6$  d.o.f.s has been carried out. On  $O(100)$  MPI ranks, the computation of a steady state took  $O(1)$  h on CHEOPS supercomputer at UoC.



# Operator Learning and Surrogate Modeling

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.

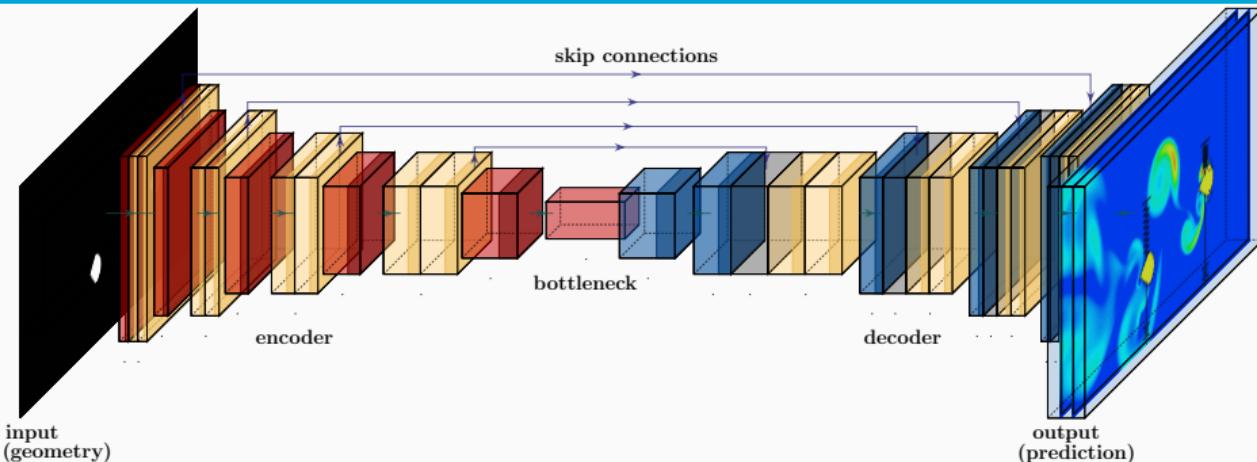
In particular, we use a pixel image of the **geometry as input** and predict an image of the resulting **stationary flow field as output**:



Other related works: E.g.

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

# Operator Learning and Surrogate Modeling



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.,

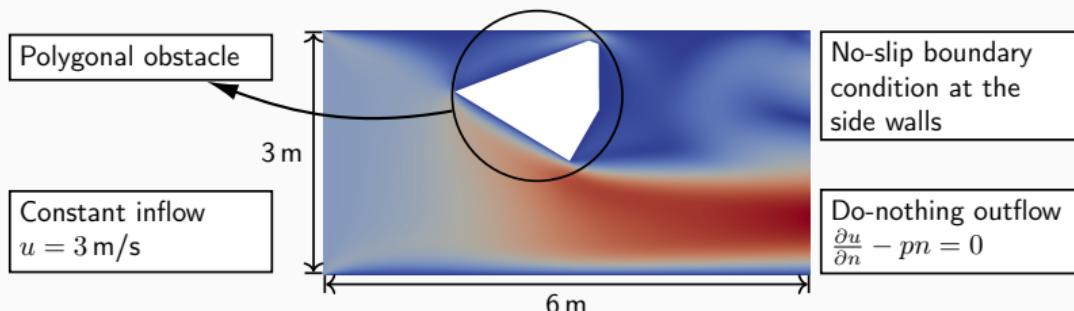
- between the right-hand side and the solution of a BVP.

## 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](#)).

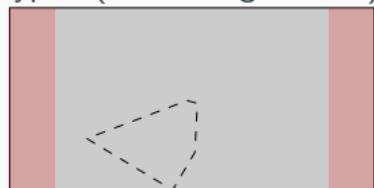
# Model Problem – Flow Around an Obstacle in Two Dimensions

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 \(2016\)](#).



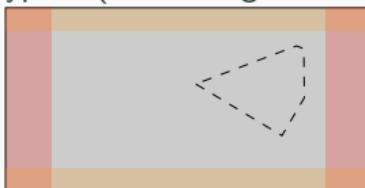
**Data:** randomly generated geometries (star-shaped polygons with 3, 4, 5, 6, and 12 edges)

type I (50 k configurations)

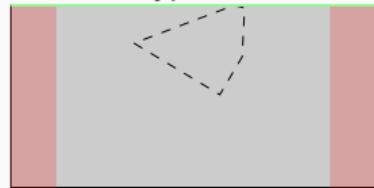


90 k training data & 10 k test data

type II (50 k configurations)



type III



(transfer learning; cf. [Eichinger, Heinlein, Klawonn \(2022\)](#))

# Computation of the Flow Data Using OpenFOAM®

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

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

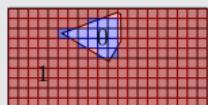
where  $\vec{u}$  and  $p$  are the velocity and pressure fields and  $\nu$  is the viscosity. Furthermore, we prescribe the previously described boundary conditions.

## 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

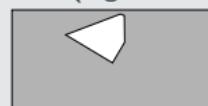
Binary



256 px

128 px

SDF (Signed Distance Function)

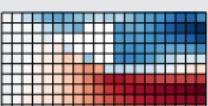


256 px

128 px

## Output data

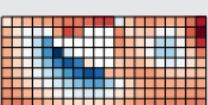
$u_x$



256 px

128 px

$u_y$

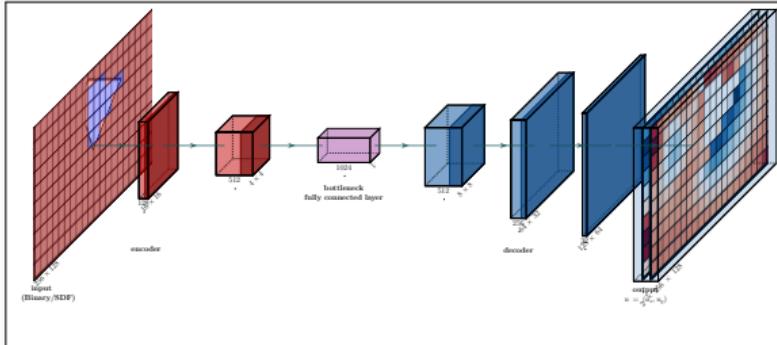


256 px

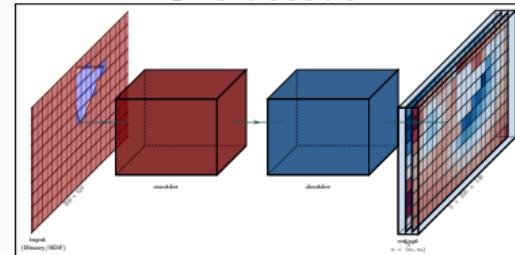
128 px

# Neural Network Architectures

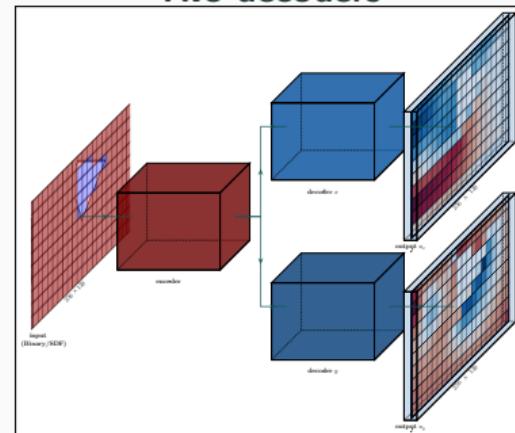
Bottleneck CNN (Guo, Li, Iorio (2016))



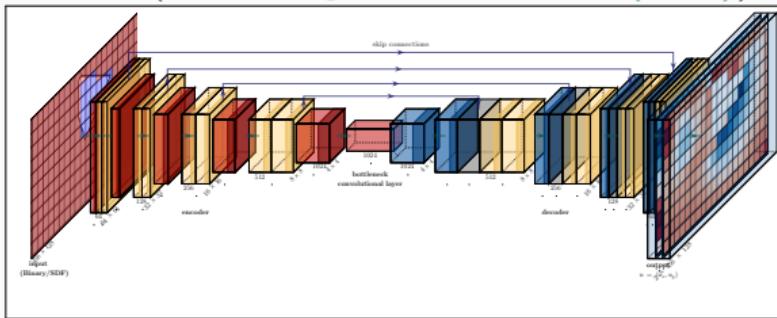
One decoder



Two decoders

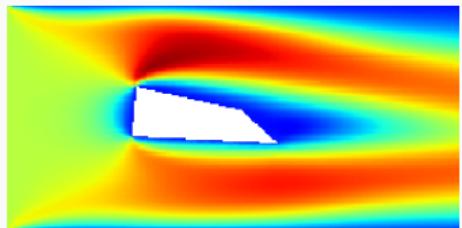


U-Net (Ronneberger, Fischer, Brox (2015))

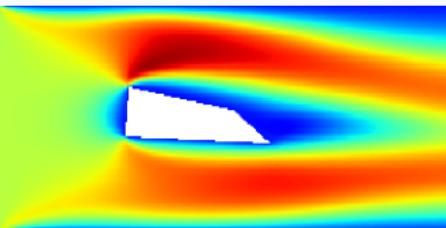


## Comparison CFD Vs NN (Relative Error 2 %)

$u_x$  CFD



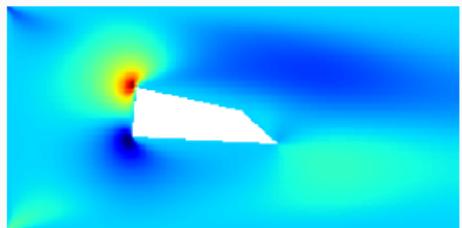
$u_x$  CNN



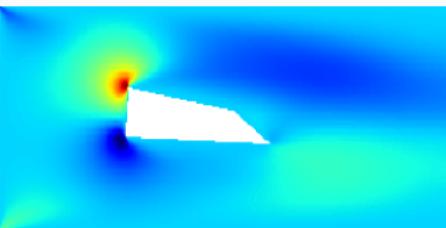
$u_x$  ERR



$u_y$  CFD



$u_y$  CNN

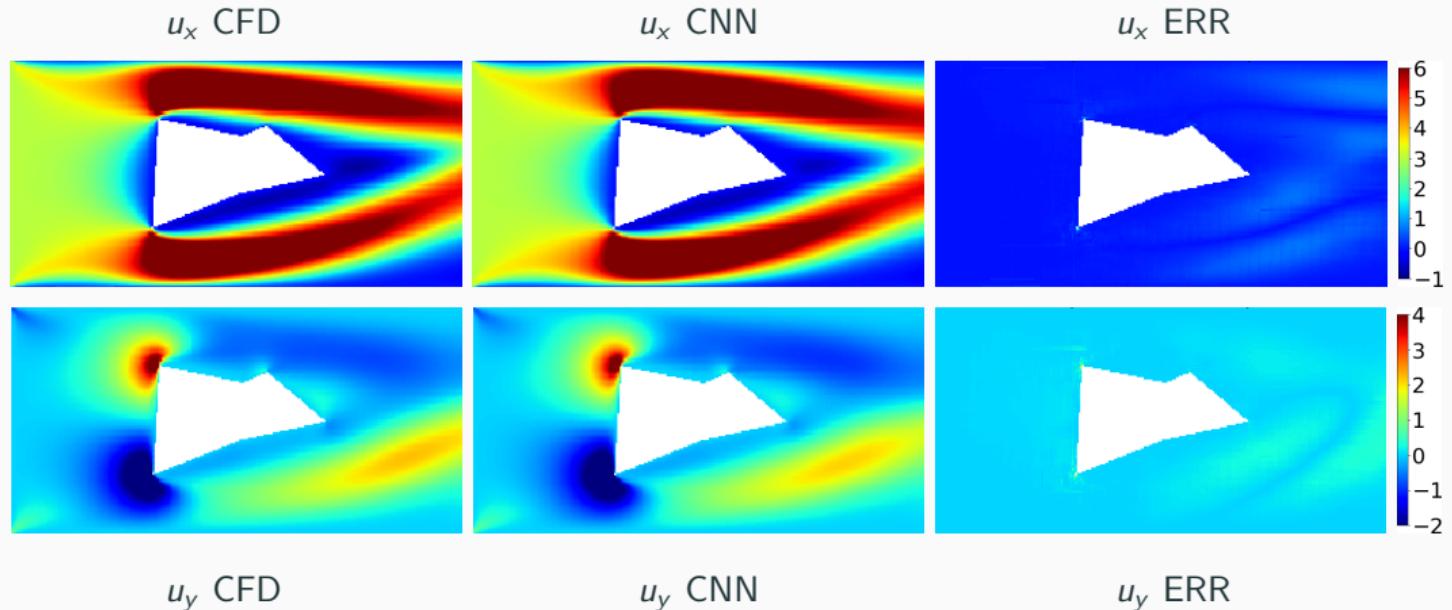


$u_y$  ERR



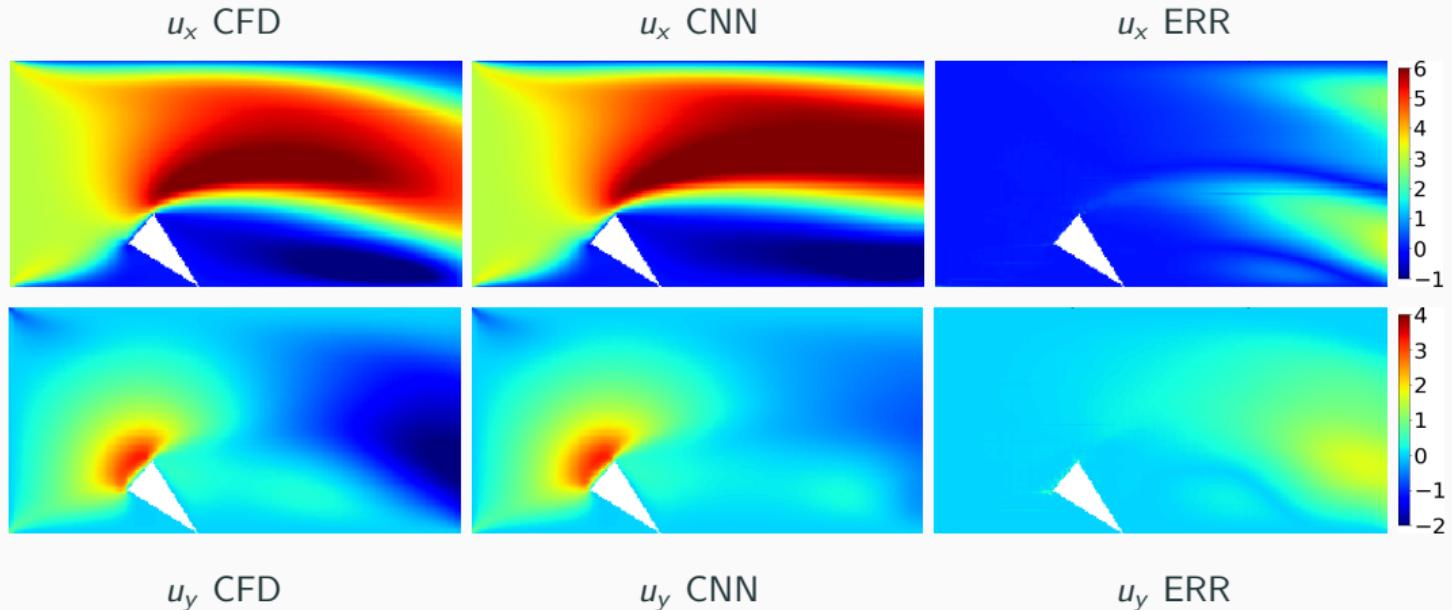
Cf. Eichinger, Heinlein, Klawonn (2021, 2022)

# Comparison CFD Vs NN (Relative Error 14 %)



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))

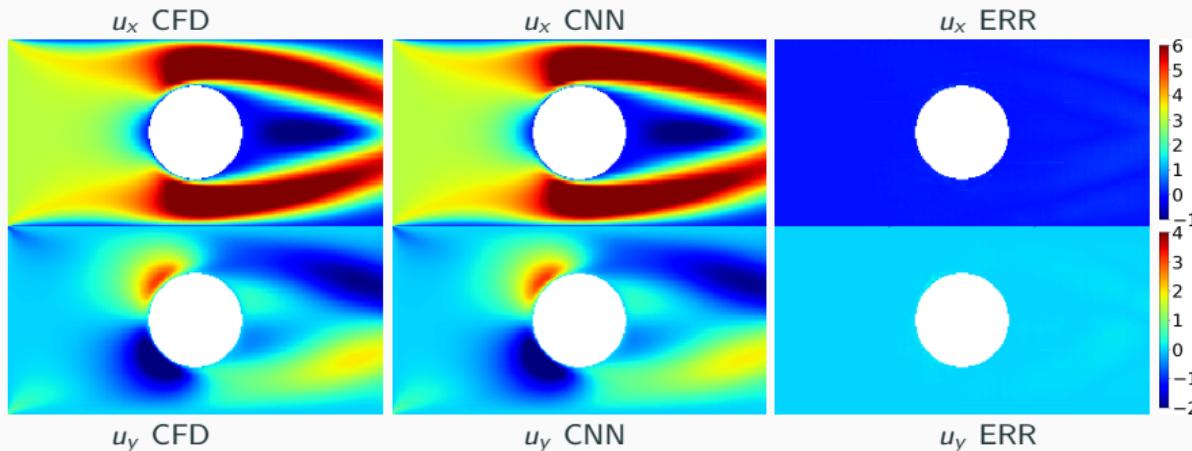
We compare the **relative error (RE)**  $\frac{\|u_{i,j} - \hat{u}_{i,j}\|_2}{\|u_{i,j}\|_2 + 10^{-4}}$  averaged over all non-obstacle pixels and all validation data configurations. Furthermore: **MSE** = mean squared error; **MAE** = mean absolute error.

			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	<b>61.16 %</b>	110.46 %	11.86 %	17.04 %	29.42 %	4.66 %
		MSE + RE	3.97 %	3.31 %	<b>4.63 %</b>	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 %	<b>3.10 %</b>
	2	MSE	49.82 %	89.12 %	10.51 %	13.01 %	21.59 %	4.42 %
		MSE + RE	<b>3.85 %</b>	<b>3.05 %</b>	4.64 %	<b>2.43 %</b>	<b>1.78 %</b>	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 %	<b>4.48 %</b>	<b>5.05 %</b>	<b>3.90 %</b>
	2	MSE	<b>51.34 %</b>	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	<b>9.53 %</b>	<b>10.91 %</b>	<b>8.15 %</b>	6.04 %	7.88 %	4.20 %

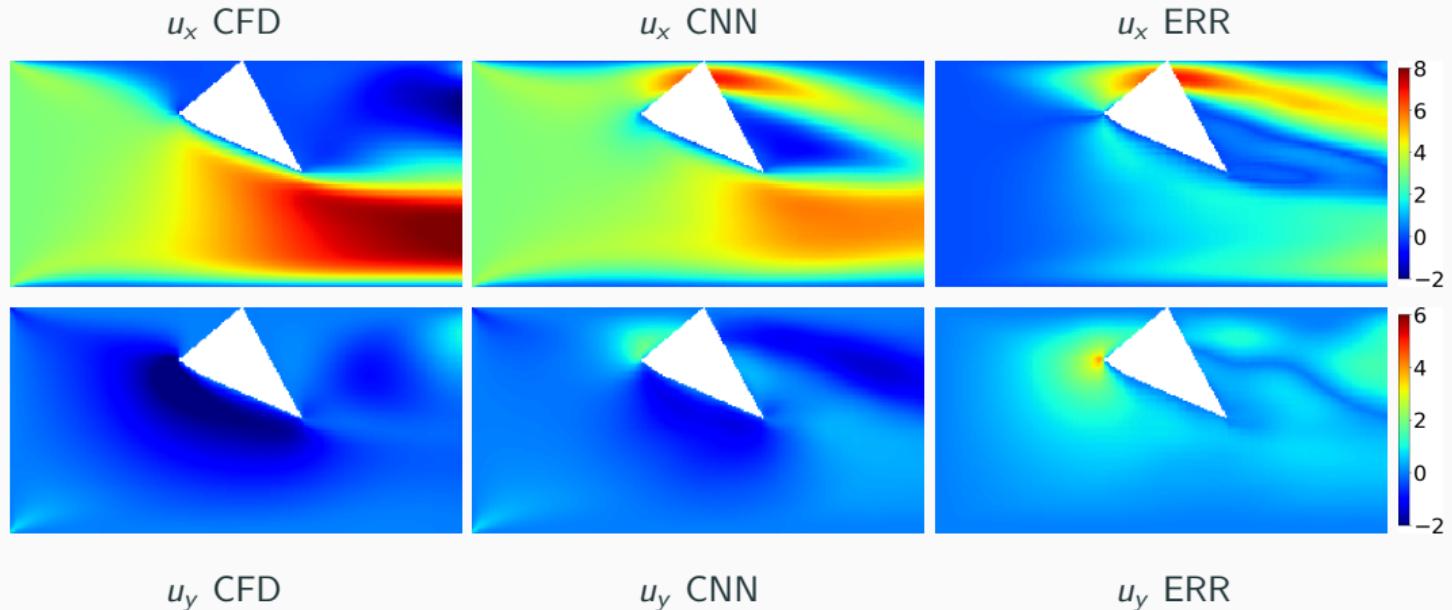
# Generalization Properties (Eichinger, Heinlein, Klawonn (2021, 2022))

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 2500 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 (2500 training + 2500 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
<b>Total Time</b>	$\approx 50$ 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)
<b>Avg. Time</b>	50 s	0.092 s	0.0054 s

⇒ 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 × 3.8 Ghz), 32GB RAM;

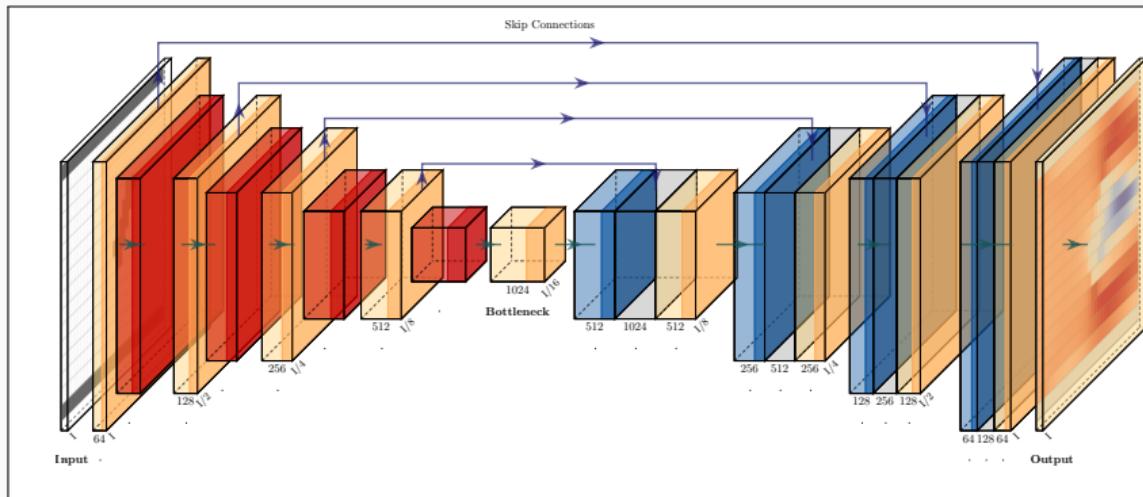
GPU: GeForce RTX 2080Ti

# **Surrogate models for CFD simulations**

## **Physics-aware approach**

---

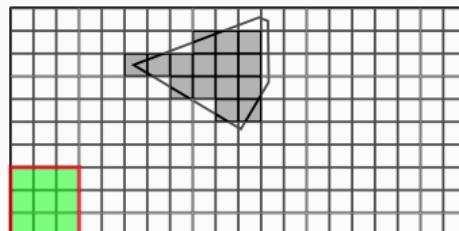
# U-Net Revisited & Action of a Discrete Convolution



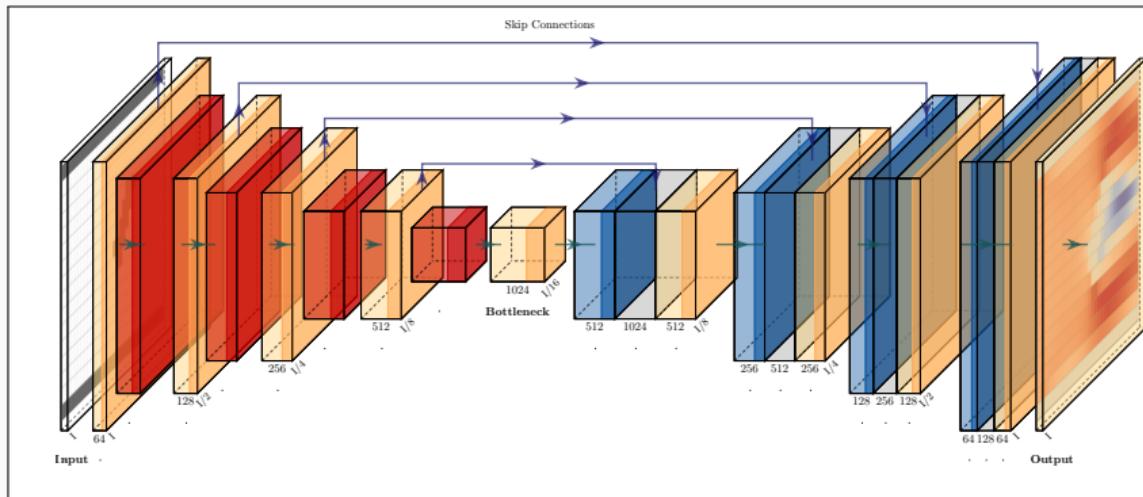
## Convolution

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

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



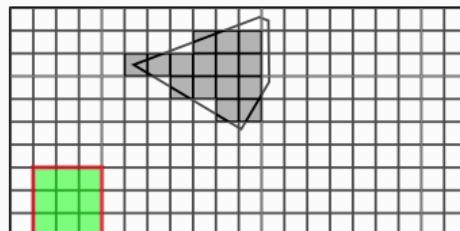
# U-Net Revisited & Action of a Discrete Convolution



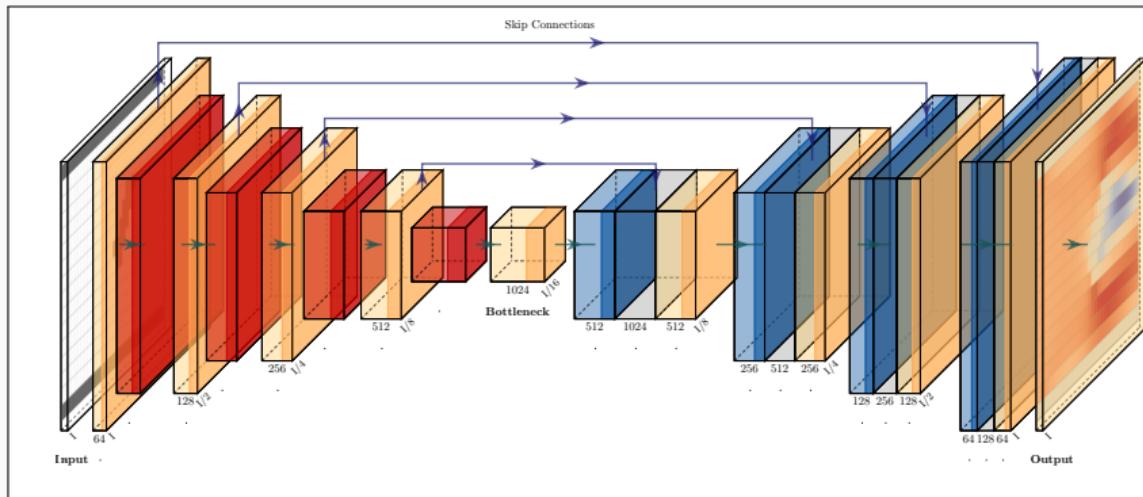
## Convolution

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

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



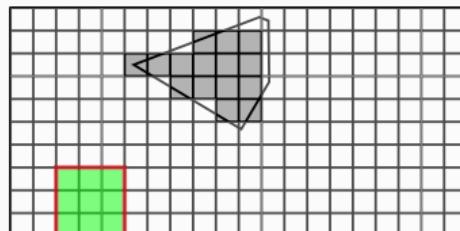
# U-Net Revisited & Action of a Discrete Convolution



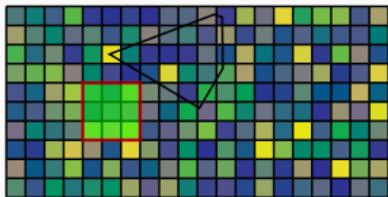
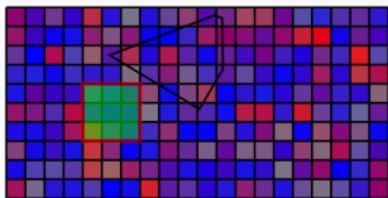
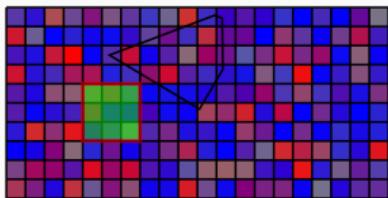
## Convolution

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

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



# Unsupervised Learning Approach – PDE Loss Using Finite Differences



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

Cf. Grimm, Heinlein, Klawonn

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

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

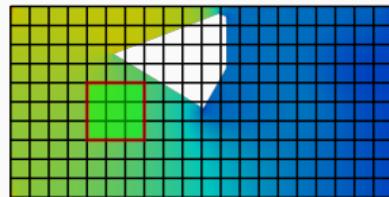
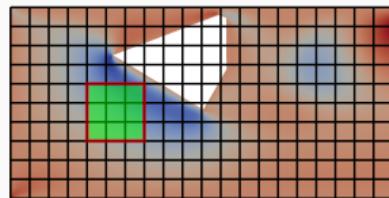
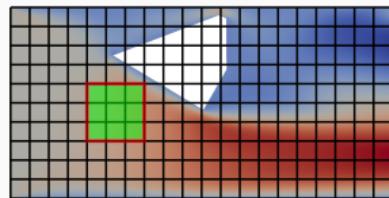
where  $u_{\text{NN}}$  and  $p_{\text{NN}}$  are the output images of our CNN and

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

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:

$$\begin{array}{c} \frac{\partial}{\partial x} \\ \hline \begin{array}{ccc} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{array} \end{array} \quad \begin{array}{c} \frac{\partial}{\partial y} \\ \hline \begin{array}{ccc} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{array} \end{array}$$

$$\begin{array}{c} \frac{\partial^2}{\partial x^2} \\ \hline \begin{array}{ccc} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{array} \end{array} \quad \begin{array}{c} \frac{\partial^2}{\partial y^2} \\ \hline \begin{array}{ccc} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{array} \end{array}$$

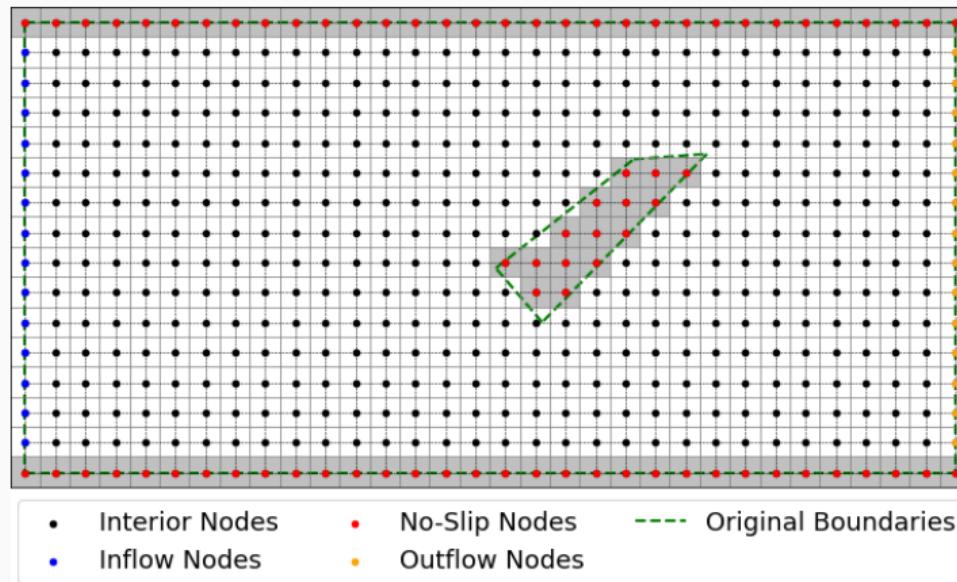


$$\left\| \begin{array}{l} F_{\text{mom}}(u_{\text{NN}}, p_{\text{NN}}) \\ F_{\text{mass}}(u_{\text{NN}}, p_{\text{NN}}) \end{array} \right\|^2 \approx 0$$

# Physics-Informed Approach & Boundary Conditions

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

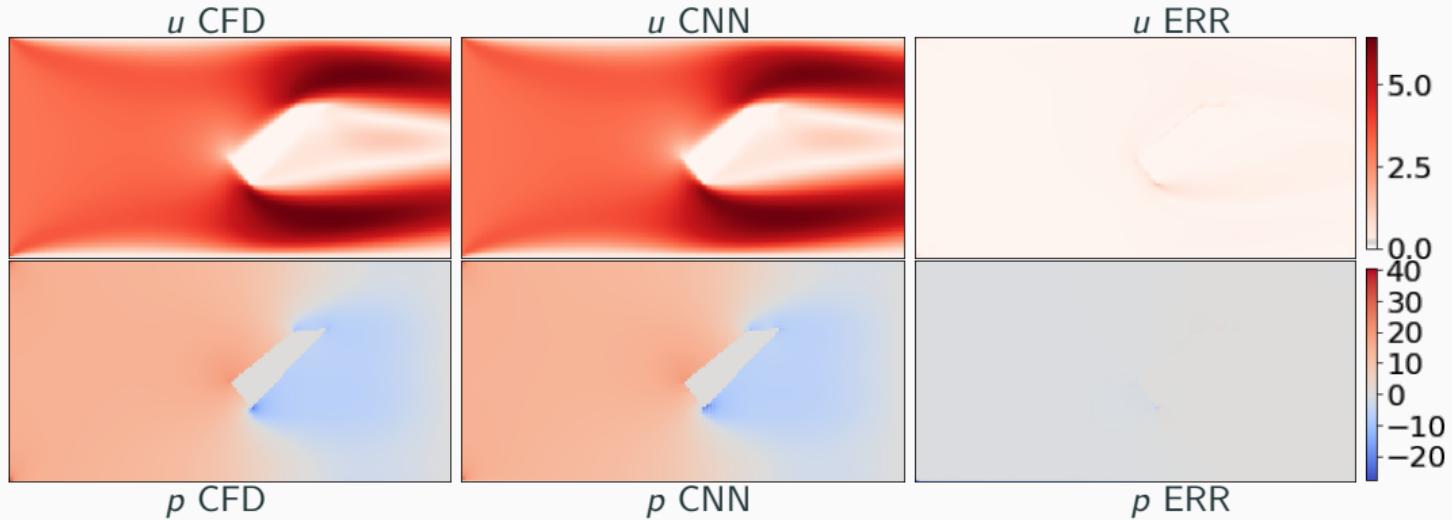
→ On a single geometry, this training of the neural network just corresponds to an **unconventional way of solving the Navier-Stokes equations using finite differences**.



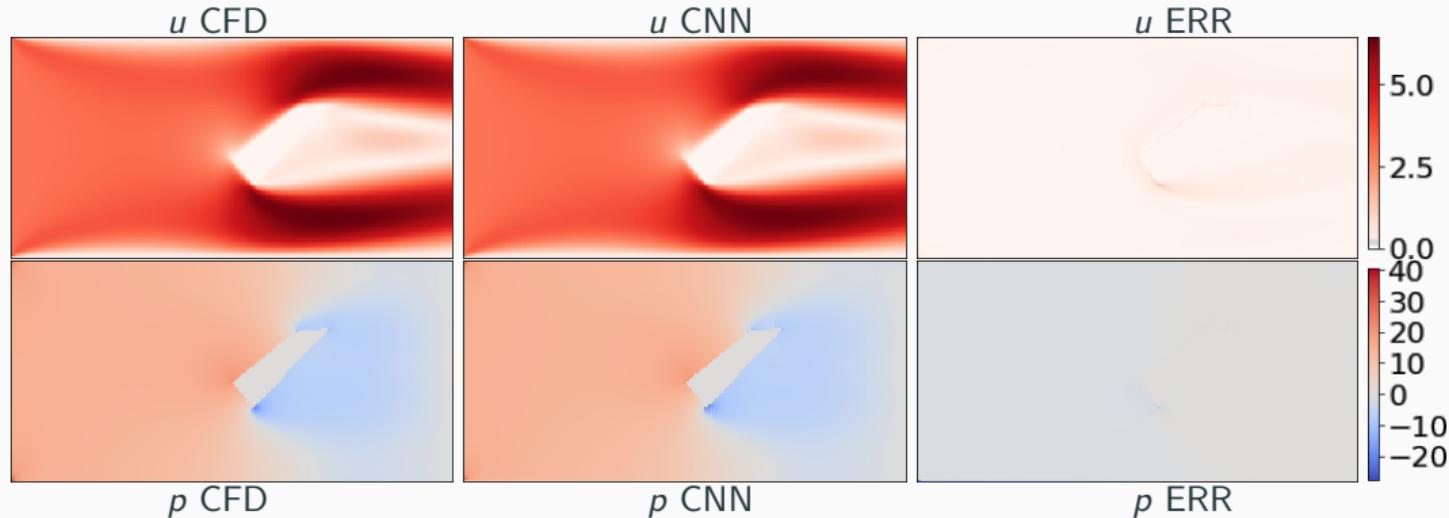
## Boundary conditions

- 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

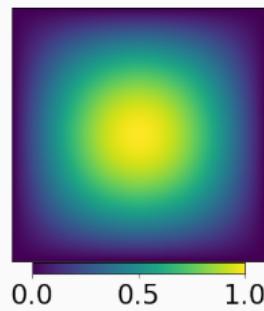


## Physics-Informed Approach – Single Geometry



⇒ 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

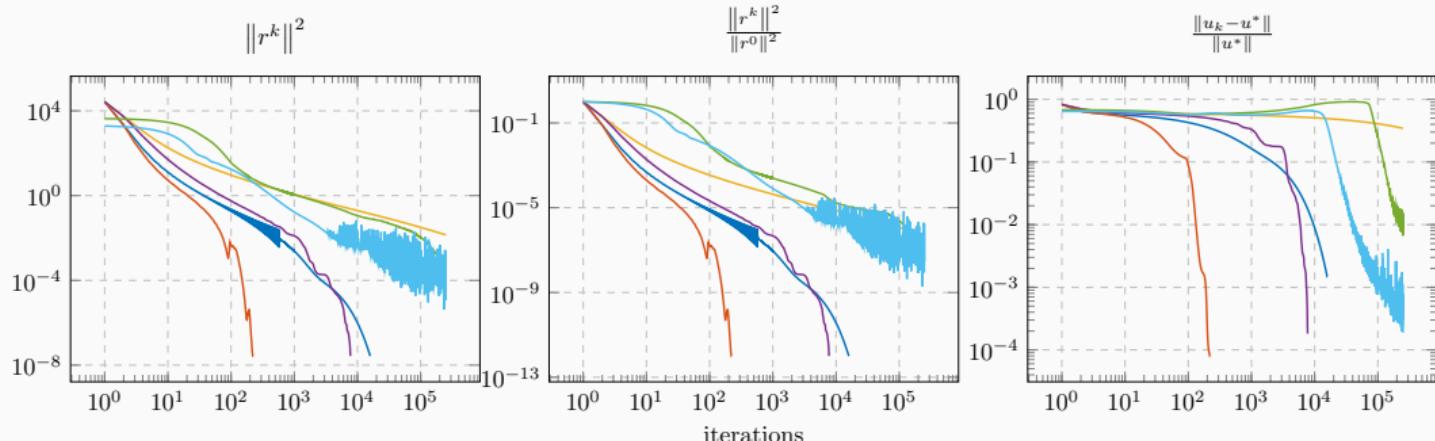


Solve

$$-\Delta u = f$$

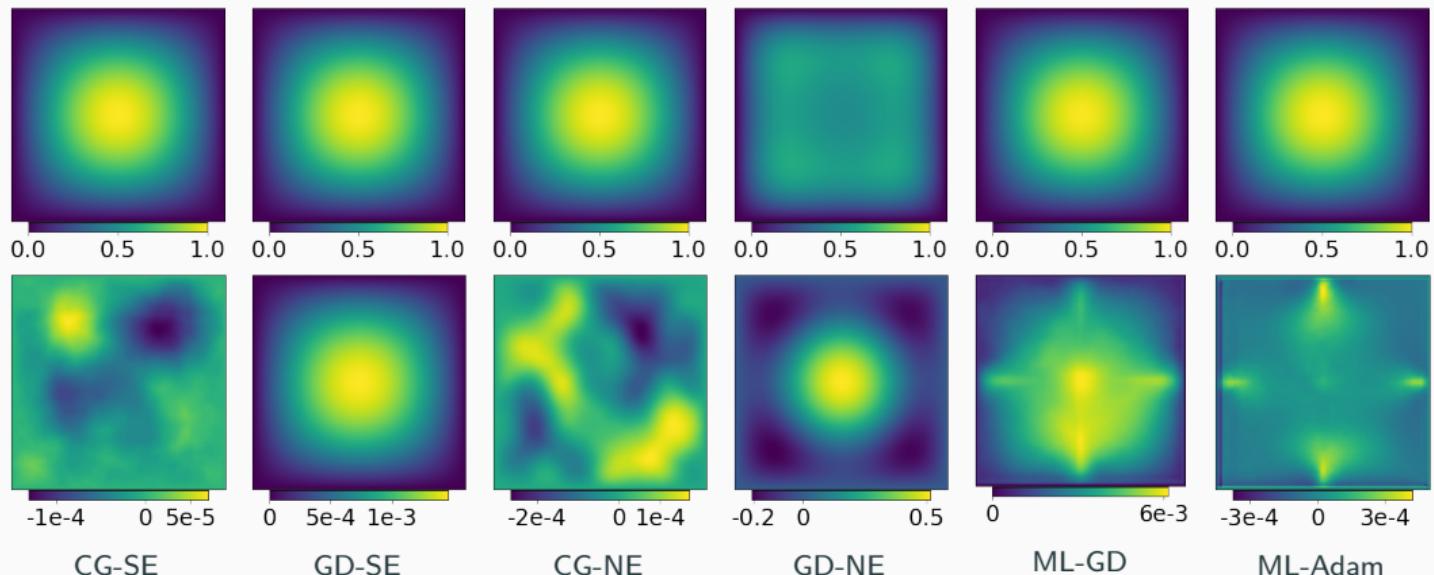
using

- classical finite differences
- **GD**: gradient descent
- **CG**: conjugate gradient method
- **SE**:  $Ax = b$
- **NE**:  $\|Ax - b\|^2$
- **ML**: CNN



— GD-SE — CG-SE — GD-NE — CG-NE — ML-GD — ML-Adam

## Convergence Comparison – CNN Versus FDM

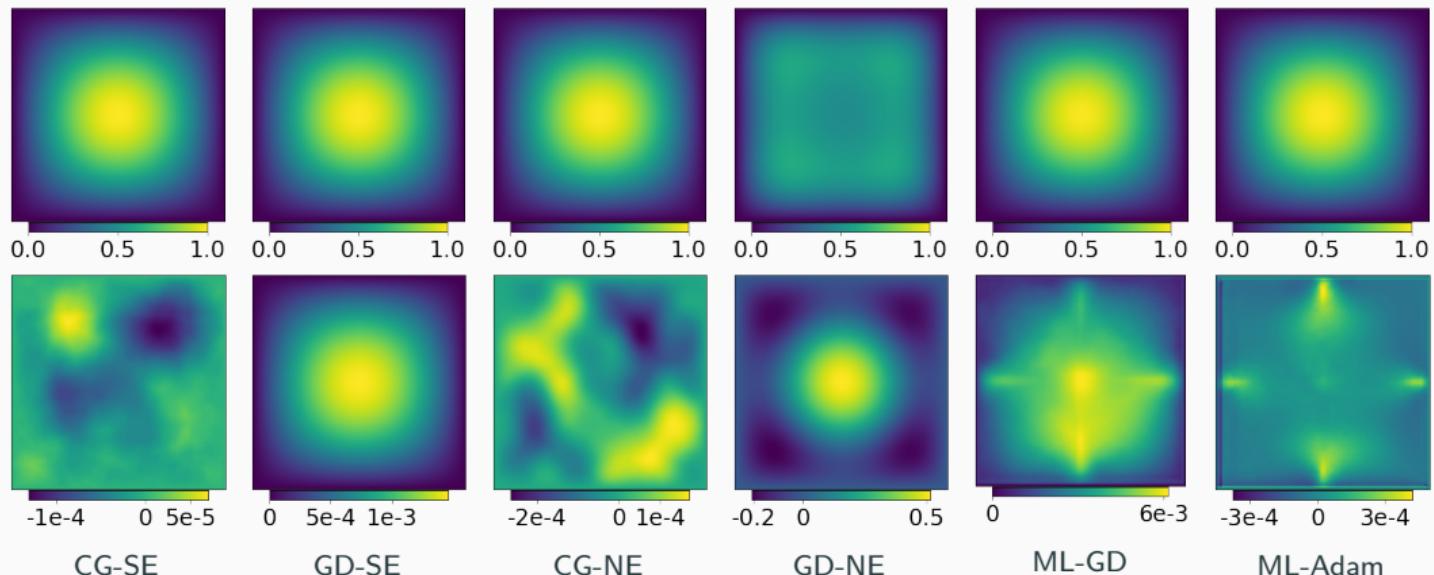


The results are in alignment with the **spectral bias of neural networks**. The neural network approximations yield a low error norm compared with the residual (MSE loss).

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

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

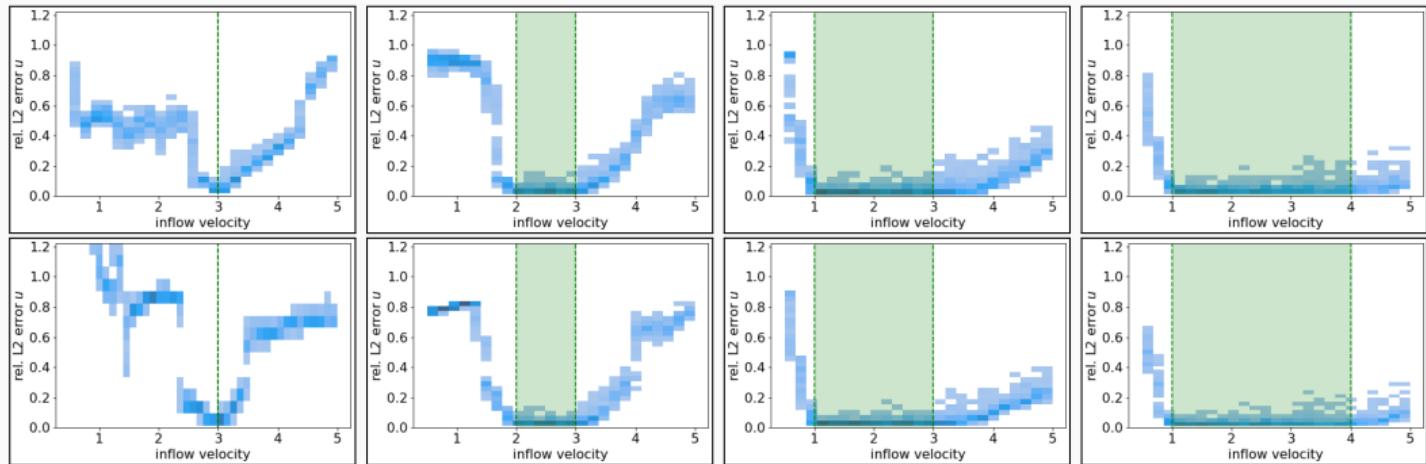
	training data	error	$\frac{\ u_{NN} - u\ _2}{\ u\ _2}$	$\frac{\ P_{NN} - P\ _2}{\ P\ _2}$	mean residual		# epochs trained
					momentum	mass	
data-based	10%	train. val.	2.07% 4.48 %	10.98% 15.20 %	$1.1 \cdot 10^{-1}$ $1.6 \cdot 10^{-1}$	$1.4 \cdot 10^0$ $1.7 \cdot 10^0$	500
	25%	train. val.	1.93% 3.49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

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

→ 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

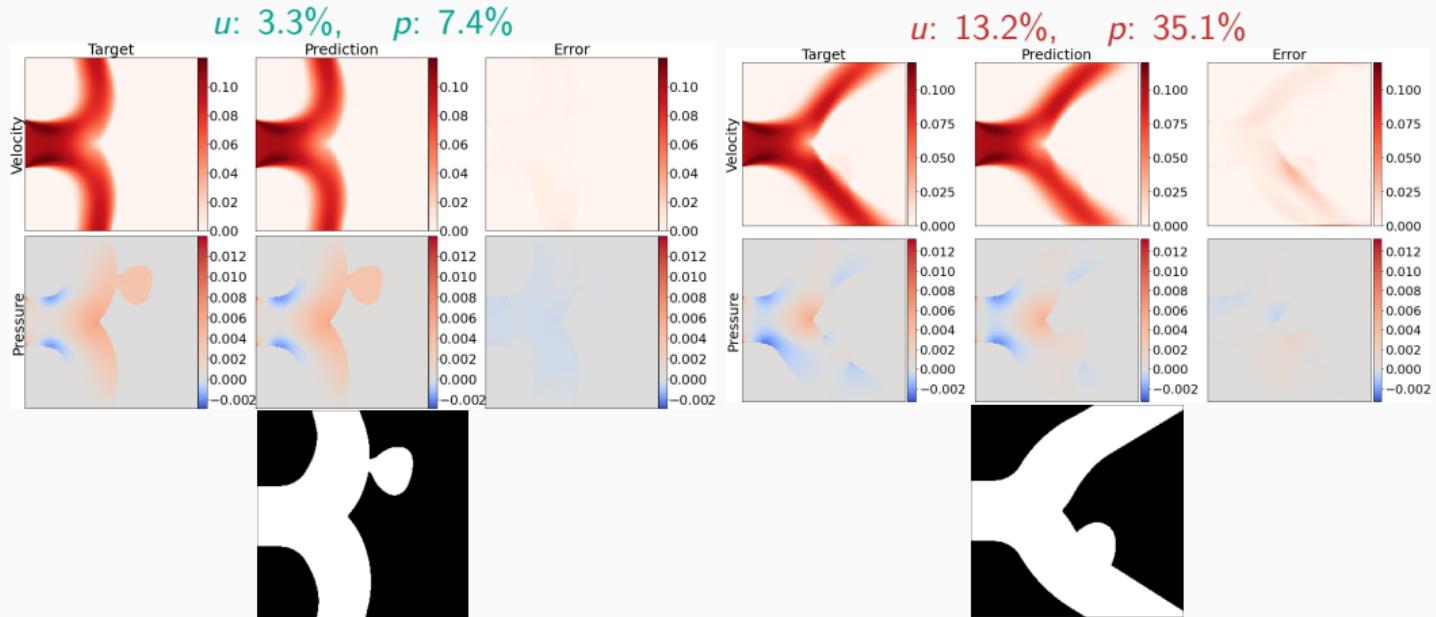


order	# data	range inflow vel.	[0.5, 1.0]	[1.0, 2.0]	[2.0, 3.0]	[3.0, 4.0]	[4.0, 5.0]
2	1 000	[3.0, 3.0]	55.5 %	48.1 %	31.1 %	17.4 %	61.5 %
		[2.0, 3.0]	89.3 %	57.4 %	<b>4.0 %</b>	15.5 %	59.1 %
		[1.0, 3.0]	40.2 %	<b>3.8 %</b>	<b>4.3 %</b>	7.1 %	20.4 %
		[1.0, 4.0]	31.3 %	<b>4.0 %</b>	<b>4.3 %</b>	<b>5.8 %</b>	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 %	<b>3.2 %</b>	16.1 %	68.2 %
		[1.0, 3.0]	38.7 %	<b>2.9 %</b>	<b>3.4 %</b>	6.7 %	18.5 %
		[1.0, 4.0]	27.7 %	<b>3.1 %</b>	<b>3.4 %</b>	<b>4.7 %</b>	7.2 %

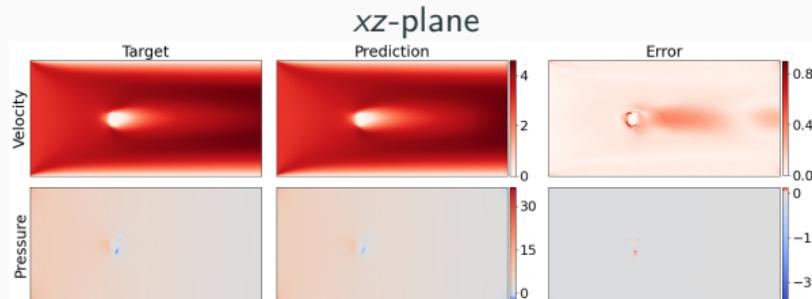
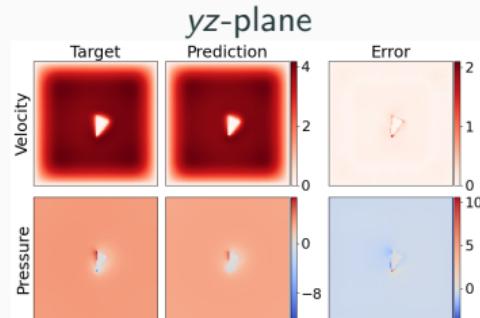
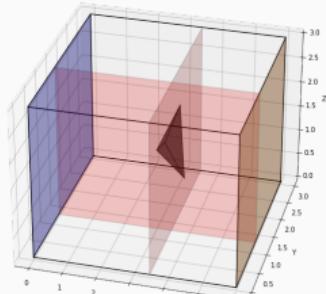
# Aneurysm Geometries

**Training:** 500 geometries    **Validation:**  $\approx$  1200 geometries

**Relative  $L^2$ -error on the validation data set in  $u$ : 4.9 %, in  $p$ : 9.5 %.**



# Extension to Three-Dimensional Geometries



- 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.  $L_2$ -error on the validation dataset in  $u$ : 3.9%, in  $p$ : 17.5%.

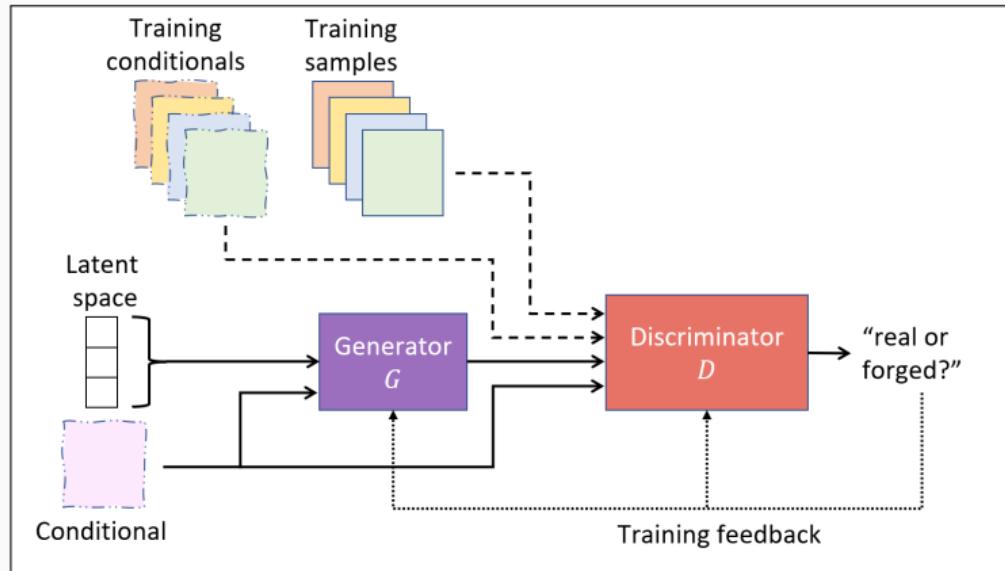
# **Surrogate models for CFD simulations**

## **GAN-based training**

---

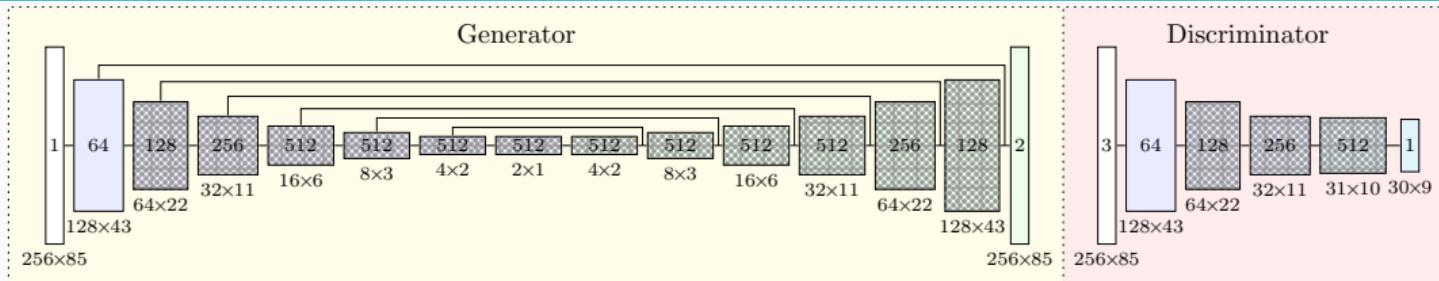
# Training the Surrogate Model via GANs

Cf. Kemna, Heinlein, Vuik (2023).

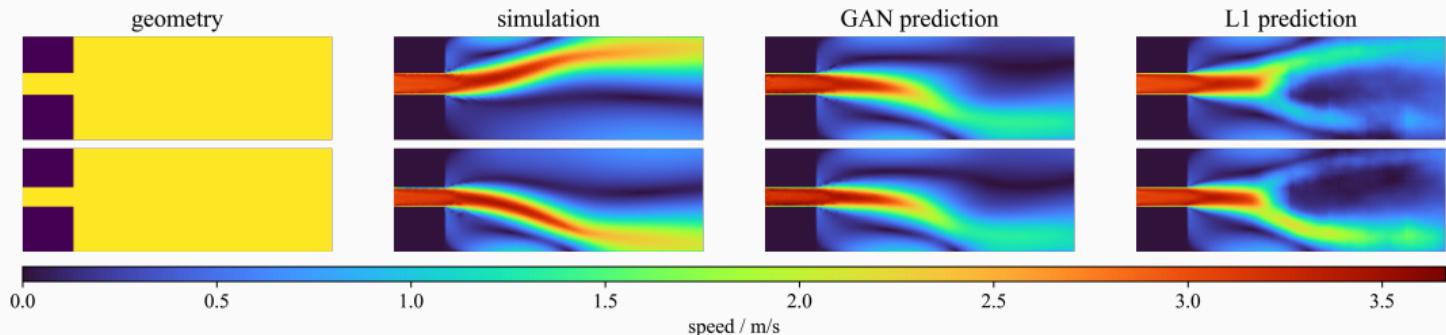


- **Generative adversarial networks (GANs)** based on Goodfellow et al. (2014) consist of two independent neural networks that are trained concurrently in an adversarial setting:
  - **Generator** is trained to *fool* the discriminator into classifying its outputs as training data
  - **Discriminator**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

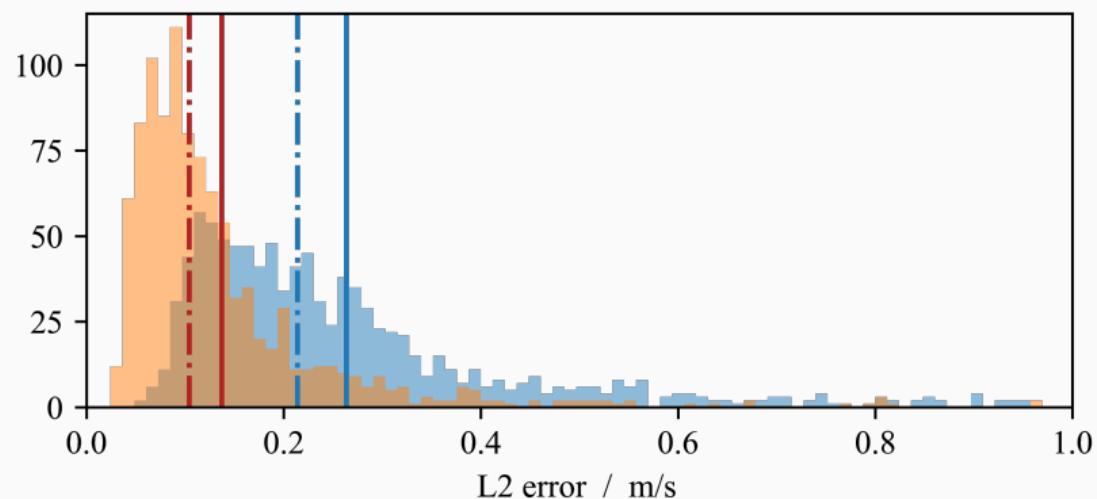
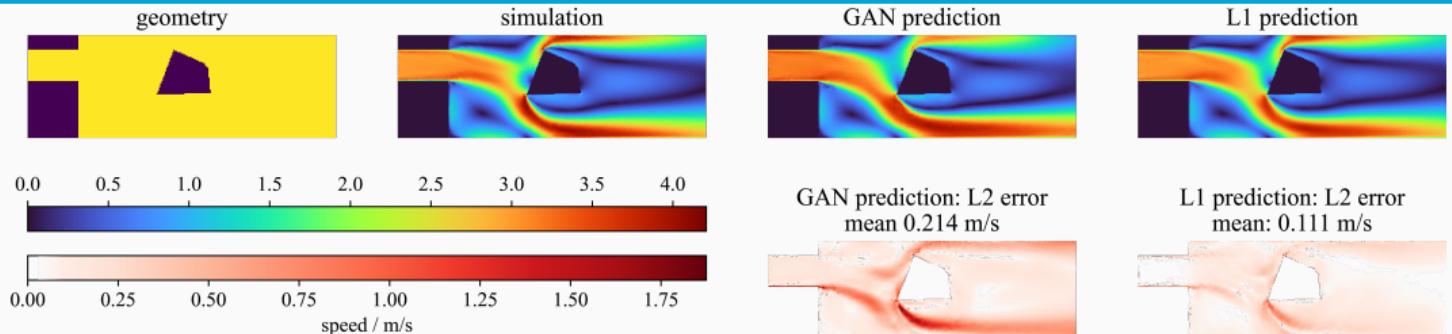
# GANs for Fluid Prediction – Bifurcation Example



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

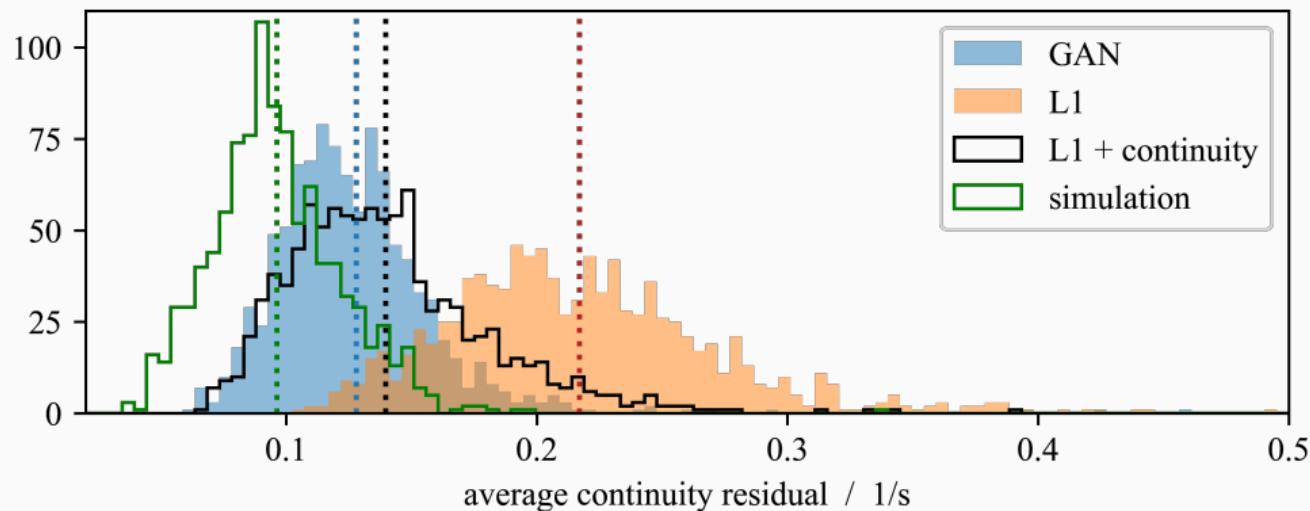


## GANs for Fluid Prediction – Divergence Error

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

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

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

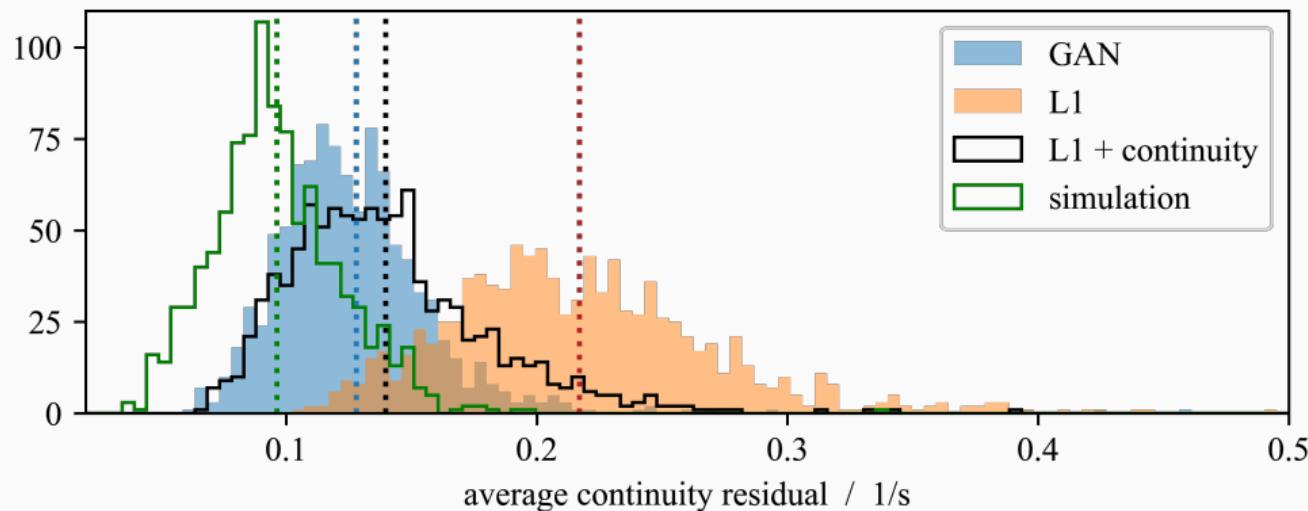


## GANs for Fluid Prediction – Divergence Error

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

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

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



→ 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!**