

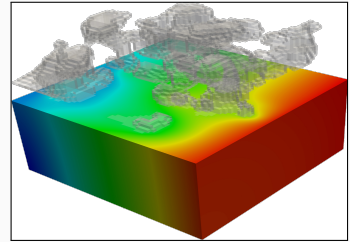
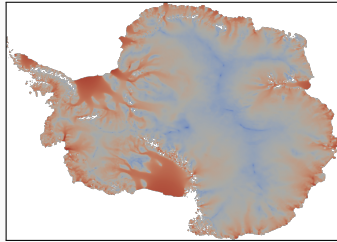
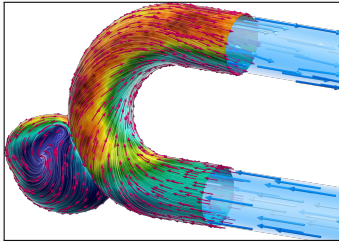
Neural networks with physical constraints, domain decomposition-based training strategies, and model order reduction

Alexander Heinlein

SimTech ML-Session, University of Stuttgart, January 11, 2023

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 **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

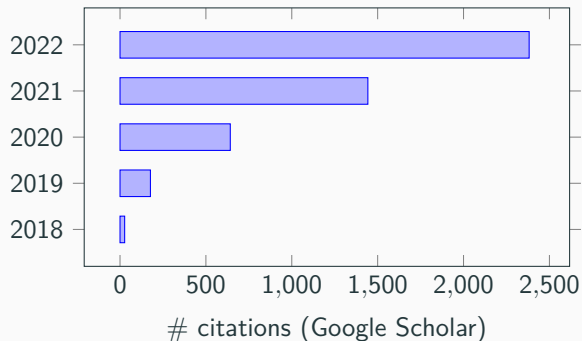
Foundational research themes:


- 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

Development of the Field of Scientific Machine Learning

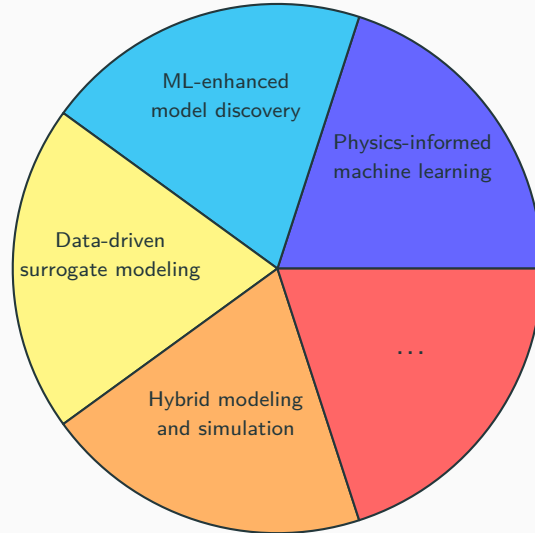


-  M. Raissi, P. Perdikaris, and G. E. Karniadakis.
Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.
Journal of Computational physics, 378, 686-707. 2019.

(and the respective arXiv preprints)

Scientific Machine Learning Examples

Many approaches in scientific machine learning have been developed in the past few years.

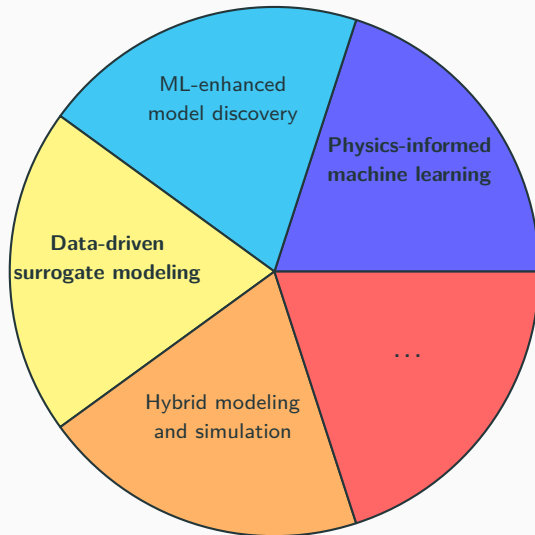


Scientific Machine Learning Examples

Many approaches in scientific machine learning have been developed in the past few years. We will focus on **two types**:

Data-driven surrogate modeling

Replacing a **computationally expensive** numerical simulator by a **fast** data-driven model.



Physics-informed machine learning

Regularizing a data-driven machine learning model using a physics-based model.

1 Physics-informed machine learning

2 Domain decomposition-based training strategies for PINNs

Based on joint work with Victorita Dolean (University of Strathclyde) and Ben Moseley and Siddhartha Mishra (ETH Zürich)

3 Surrogate models for computational fluid dynamics simulations – Data-driven approach

Based on joint work with Mattias Eichinger and Axel Klawonn (University of Cologne)

4 Surrogate models for computational fluid dynamics simulations – GAN-based training

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

5 Surrogate models for computational fluid dynamics simulations – Physics-aware approach

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

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.

Idea: Solve a general differential equation subject to boundary conditions

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

by solving an **optimization problem**

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

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

Construction of the trial functions

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

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

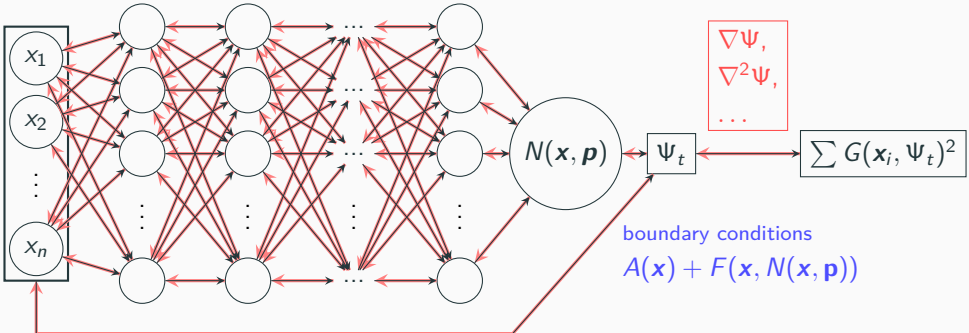
where

- $N(\mathbf{x}, \mathbf{p})$ is a **trainable feedforward neural network** with parameters \mathbf{p} and input $x \in \mathbb{R}^n$ and
- the functions A and F are **fixed functions**, chosen such that
 - A satisfies the boundary conditions, and
 - F does not contribute to the boundary conditions.

From the conclusion of the paper:

*“The **success of the method** can be attributed to **two factors**. The first is the employment of neural networks that are **excellent function approximators** and the second is the form of the **trial solution that satisfies by construction the BC’s** and therefore the constrained optimization problem becomes a substantially simpler unconstrained one.”*

Sketch of the Approach by Lagaris et al. (1998)



Physics-Informed Neural Networks (PINNs)

In **Raissi, Perdikaris, Karniadakis (2019)**, the authors have revisited and modified the approach by **Lagaris et al. (1998)**, denoting their method as **physics-informed neural networks (PINNs)**. Consider the generic partial differential equation

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

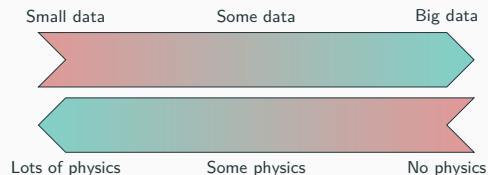
The main novelty of PINNs is that a **hybrid loss function** is used for training the feedforward neural network:

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

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

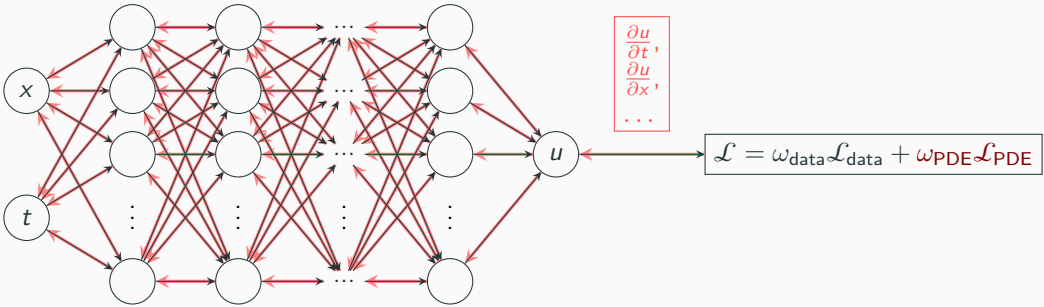
$$\mathcal{L}_{\text{data}} = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} (u(x_i, t_i) - u_i)^2,$$

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



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

Sketch of the PINN approach by Raissi et al.



Advantages

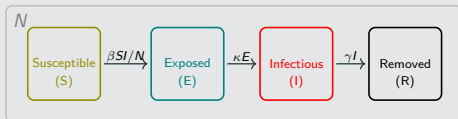
- “Meshfree”.
- **Mostly unsupervised** and work with **incomplete models** (e.g., we learn only the missing physics) and **imperfect data**.
- Strong generalization properties with **small data** due to **embedded physics**.
- **High-dimensional problems** (PDEs like Black-Scholes, Allen-Cahn).
- Solve **inverse and forward problems**, stationary and time-dependent, assimilate data in the **same way**.

Drawbacks

- **Large computational cost** associated with the training of the neural networks.
- Generally, the training process is **not robust** and **depends heavily on well-chosen weights**
- **Convergence** properties **not well-understood** yet.
- **Poor scaling** to large domains.
- Learning high frequencies (**spectral bias**) / multi-scale solutions is **difficult**.

Epidemic Parameter Identification Using Physics-Informed Neural Networks

SEIR model



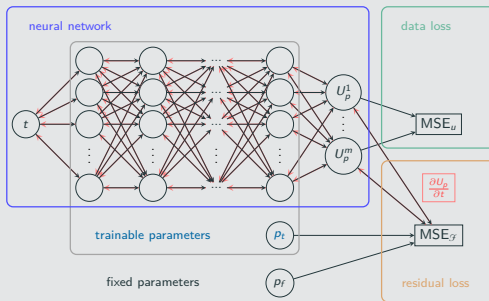
We consider the system of ODEs

$$\begin{aligned}\frac{dS}{dt} &= -\beta \frac{SI}{N} \\ \frac{dE}{dt} &= \beta \frac{SI}{N} - \kappa E \\ \frac{dI}{dt} &= \kappa E - \gamma I \\ \frac{dR}{dt} &= \gamma I\end{aligned}$$

with initial values $S(t_0) \geq 0, E(t_0) \geq 0, I(t_0) \geq 0$, and $R(t_0) \geq 0$ at some initial time t_0 . The infective period γ and the exposed period $1/\kappa$ are given.

→ **Identify the time-dependent contact rate β** from given data for S, E , and I .

Parameter identification using PINNs

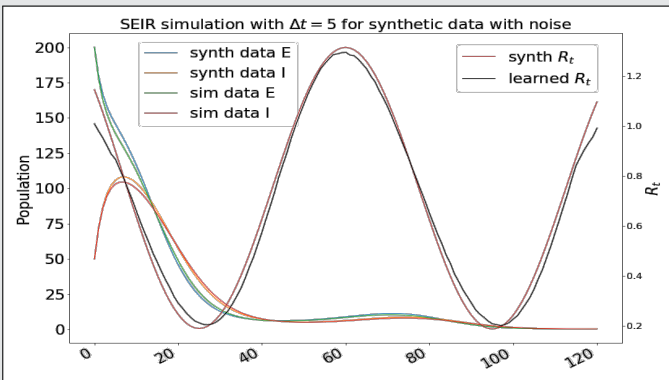
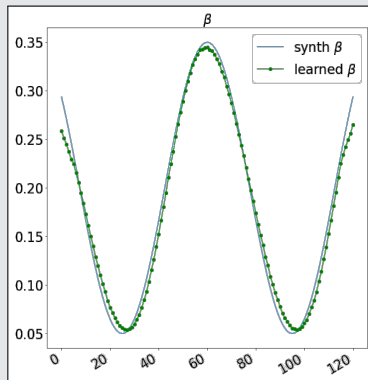


Training the weights W and bias b of the the neural work and the contact rate β by **minimizing the mean-squared data error (MSDE) and the mean-squared residual error (MSRE)**:

$$\arg \min_{W, b, \beta} \left(\underbrace{\mathcal{L}_{\text{data}}(W, b, \beta)}_{\text{MSDE}} + \underbrace{\mathcal{L}_{\text{ODE}}(W, b, \beta)}_{\text{MSRE}} \right)$$

Epidemic Parameter Identification Using PINNS – Results

Results for synthetic data

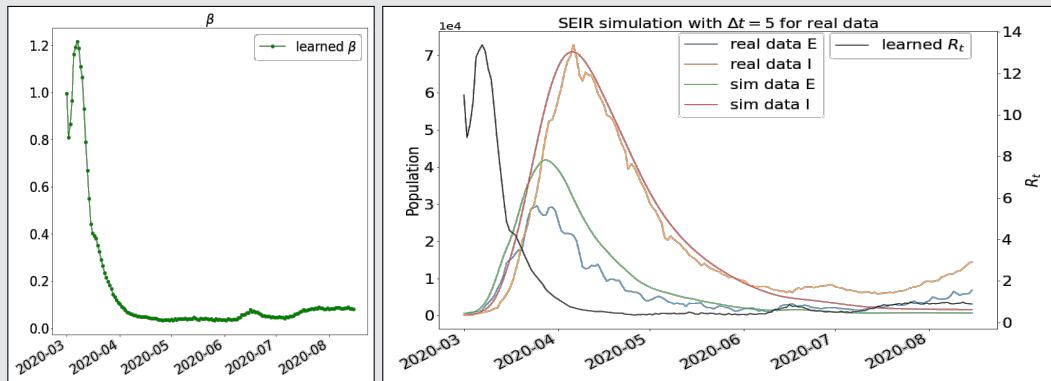


synth data and *sim data* are obtained by simulating the SEIR model with *synth β* and *learned β* , respectively.

Cf. Grimm, Heinlein, Klawonn, Lanser, Weber (2022).

Epidemic Parameter Identification Using PINNS – Results

Results for real data for COVID-19 (Germany)



real data and sim data are obtained by simulating the SEIR model with real β and learned β , respectively.

Cf. Grimm, Heinlein, Klawonn, Lanser, Weber (2022).

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

Estimate of the generalization error

The generalization error (or total error) satisfies

$$\varepsilon_G \leq C_{\text{PDE}} \varepsilon_{\mathcal{T}} + C_{\text{PDE}} C_{\text{quad}}^{1/p} N^{-\alpha/p}$$

where

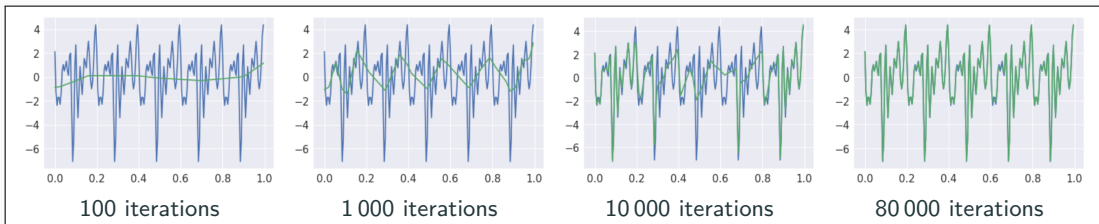
- $\varepsilon_G = \varepsilon_G(\theta; \mathbf{X}) := \|\mathbf{u} - \mathbf{u}^*\|_V$ (V Sobolev space, \mathbf{X} training data set)
- $\varepsilon_{\mathcal{T}}$ is the training error (L^p loss of the residual of the PDE)
- C_{PDE} and C_{quad} constants depending on the PDE resp. the quadrature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

The devil is in the details:

“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, N., et al, *On the spectral bias of neural networks*. 36th International Conference on Machine Learning, 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**

Perdikaris et al, *When and why PINNs fail to train: A neural tangent kernel perspective*, JCP (2022)

Neural tangent kernel (NTK) theory in a nutshell

- Write the gradient descent method at the continuous level (\mathcal{L} is the loss function)

$$\frac{d\theta}{dt} = -\nabla \mathcal{L}(\theta), \quad \mathcal{L}(\theta) = \omega_{\text{data}} \sum_i \mathcal{R}_{\text{data}}(\mathbf{x}_{\text{data}}^i, \theta(t))^2 + \omega_{\text{PDE}} \sum_i \mathcal{R}_{\text{PDE}}(\mathbf{x}_{\text{PDE}}^i, \theta(t))^2$$

- Residual vectors in the collocation points obey an ODE

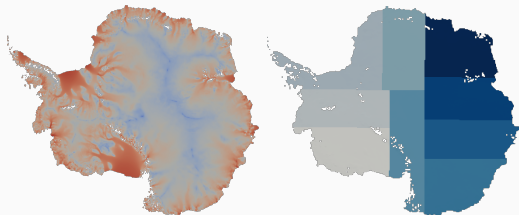
$$\frac{d}{dt} \begin{bmatrix} \mathcal{R}_{\text{data}}(\mathbf{x}_{\text{data}}, \theta(t)) \\ \mathcal{R}_{\text{PDE}}(\mathbf{x}_{\text{PDE}}, \theta(t)) \end{bmatrix} = - \begin{bmatrix} K_{\text{data,data}}(t) & K_{\text{data,PDE}}(t) \\ K_{\text{PDE,data}}(t) & K_{\text{PDE,PDE}}(t) \end{bmatrix} \begin{bmatrix} \mathcal{R}_{\text{data}}(\mathbf{x}_{\text{data}}, \theta(t)) \\ \mathcal{R}_{\text{PDE}}(\mathbf{x}_{\text{PDE}}, \theta(t)) \end{bmatrix}$$

- The NTK $K(t) \rightarrow K^*$ (convergence of the expectation) for infinitely wide and shallow networks
- Spectral properties of K^* explain the speed of training

*“To provide further insight, we analyze the **training dynamics of fully-connected PINNs through the lens of their NTK** and show that **not only they suffer from spectral bias**, but they also exhibit a **discrepancy in the convergence rate among the different loss components** contributing to the total training error”*

Domain decomposition-based training strategies for PINNs

Domain Decomposition Methods



Images based on [Heinlein, Perego, Rajamanickam \(2022\)](#)

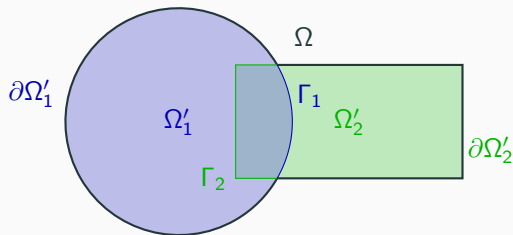
Historical remarks: The **alternating Schwarz method** is the earliest **domain decomposition method (DDM)**, which has been invented by **H. A. Schwarz** and published in **1870**:

- 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**



A non-exhaustive overview:

- **Machine Learning-enhanced adaptive FETI–DP** (finite element tearing and interconnecting – dual primal): [Heinlein, Klawonn, Lanser, Weber \(2019\)](#)
- **D3M** (deep domain decomposition method): [Li, Tang, Wu, and Liao \(2019\)](#)
- **DeepDDM** (deep-learning-based domain decomposition method): [Li, Xiang, Xu \(2020\)](#)
- **Two-Level DeepDDM**: [Mercier, Gratton \(arXiv 2021\)](#)
- **cPINNs** (conservative physics-informed neural networks): [Jagtap, Kharazmi, and Karniadakis \(2020\)](#)
- **XPINNs** (extended physics-informed neural networks): [Jagtap, Karniadakis \(2020\)](#)
- **FBPINNs** (finite basis physics-informed neural networks): [Moseley, Markham, and Nissen-Meyer \(arXiv 2021\)](#)

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.

DeepDDM (deep-learning-based domain decomposition method)

Li, Xiang, Xu. *Deep domain decomposition method: Elliptic problems*. **Mathematical and Scientific Machine Learning (2020)**

Train **local networks** h_s by **batch stochastic gradient descent** (SGD) and transmitting only interface values to the neighboring subdomains.

Local loss functions contain **volume, boundary, and interface jump terms**:

$$\mathcal{L}_s(\theta_s; \mathbf{X}_s) := \mathcal{L}_{\Omega_s}(\theta_s; \mathbf{X}_s) + \mathcal{L}_{\partial\Omega_s \setminus \Gamma_s}(\theta_s; \mathbf{X}_s) + \mathcal{L}_{\Gamma_s}(\theta_s; \mathbf{X}_s),$$

where

$$\mathcal{L}_{\Omega_s}(\theta; \mathbf{X}_{f_s}) := \frac{1}{N_{f_s}} \sum_{i=1}^{N_{f_s}} \left| n(h_s(\mathbf{x}_{f_s}^i; \theta_s)) - f(\mathbf{x}_{f_s}^i) \right|^2,$$

$$\mathcal{L}_{\partial\Omega_s \setminus \Gamma_s}(\theta_s; \mathbf{X}_{g_s}) := \frac{1}{N_{g_s}} \sum_{i=1}^{N_{g_s}} \left| \mathcal{B}(h_s(\mathbf{x}_{g_s}^i; \theta_s)) - g(\mathbf{x}_{g_s}^i) \right|^2,$$

$$\mathcal{L}_{\Gamma_s}(\theta_s; \mathbf{X}_{r_s}) := \frac{1}{N_{r_s}} \sum_{i=1}^{N_{r_s}} \left| \mathcal{D}(h_s(\mathbf{x}_{r_s}^i; \theta_s)) - \mathcal{D}(h_r(\mathbf{x}_{r_s}^i; \theta_s)) \right|^2.$$

Advantages

- The outer-iteration depends on the size overlap and the number of subdomains
- DeepDDM can “easily” handle PDEs with **curved interfaces and heterogeneities**.

Drawbacks

- **No quantitative estimates** of the convergence rate of deep learning solving PDE.
- What is the standard to design the best network architecture?

Li, Wang, Cui, Xiang, Xu. *Deep domain decomposition method: Helmholtz equation. Advances in Applied Mathematics and Mechanics (2023)*

Train **local networks** h_s by transmitting only Robin interface values to the neighboring subdomains.

Use of NNs with **plane wave (PW) activation** to account for oscillatory nature of the solution. The loss function reads

$$\mathcal{L}_s(\theta_s; \mathbf{X}_s) := \mathcal{L}_{\Omega_s}(\theta_s; \mathbf{X}_s) + \mathcal{L}_{\partial\Omega_s \setminus \Gamma_s}(\theta_s; \mathbf{X}_s) + \mathcal{L}_{\Gamma_s}(\theta_s; \mathbf{X}_s),$$

where

$$\mathcal{L}_{\Omega_s}(\theta_s; \mathbf{X}_{f_s}) := \frac{1}{N_{f_s}} \sum_{i=1}^{N_{f_s}} \left| n(h_s(\mathbf{x}_{f_s}^i; \theta_s)) - f(\mathbf{x}_{f_s}^i) \right|^2,$$

$$\mathcal{L}_{\partial\Omega_s \setminus \Gamma_s}(\theta_s; \mathbf{X}_{g_s}) := \frac{1}{N_{g_s}} \sum_{i=1}^{N_{g_s}} \left| \beta(h_s(\mathbf{x}_{g_s}^i; \theta_s)) - g(\mathbf{x}_{g_s}^i) \right|^2,$$

$$\mathcal{L}_{\Gamma_s}(\theta_s; \mathbf{X}_{\Gamma_s}) := \frac{1}{N_{\Gamma_s}} \sum_{i=1}^{N_{\Gamma_s}} \left| \frac{\partial h_s(\mathbf{x}_{\Gamma_s}^i; \theta_s)}{\partial \mathbf{n}_s} + \gamma_s h_s(\mathbf{x}_{\Gamma_s}^i; \theta_s) - g_s(\mathbf{x}_{\Gamma_s}^i) \right|^2.$$

Advantages

- Number of outer iterations comparable to the use of FDM with DDM
- **Competitive solution time/iteration** when the wave number increases.

Drawbacks

- Comparison done with an iterative Schwarz method – what about Krylov acceleration?
- All relies on PW activation function – **number of parameters dependent on k** .

Two-level DeepDDM Acceleration

Mercier, Gratton, Boudier. *A coarse space acceleration of DeepDDM (arXiv 2021)*

Add a **second level** by training a **coarse network** h_c on the top of DeepDDM in order to achieve scalability. Loss of the coarse network:

$$\mathcal{L}_c(\theta_c) := \mathcal{L}_\Omega(\theta_c) + \mathcal{L}_{\partial\Omega}(\theta_c) + \mathcal{L}_f(\theta_c),$$

where

$$\mathcal{L}_\Omega(\theta_c) := \frac{1}{N_f} \sum_{i=1}^{N_f} \left| \mathcal{n} \left(h_c \left(\mathbf{x}_f^{i,c}; \theta_c \right) \right) - f \left(\mathbf{x}_f^i \right) \right|^2,$$

$$\mathcal{L}_{\partial\Omega_s \setminus \Gamma_s}(\theta_c) := \frac{1}{N_g} \sum_{i=1}^{N_g} \left| \mathcal{B} \left(h_c \left(\mathbf{x}_g^{i,c}; \theta_c \right) \right) - g \left(\mathbf{x}_g^i \right) \right|^2,$$

$$\mathcal{L}_{\Gamma_s}(\theta_c) := \frac{1}{N_{f,c}} \sum_{i=1}^{N_{f,c}} \left| h_c \left(\mathbf{x}_f^{i,c}; \theta_c \right) - \sum_s E_s \left(\chi_s h_s \left(\mathbf{x}_f^{i,c}; \theta_s \right) \right) \right|^2.$$

- χ_s partition of unity
- E_s extension by zero operator

Ongoing work & open questions

- What is the best accuracy that one can obtain for a given number of subdomains?
- How does the performance depend on the capacities of the local and coarse problem networks?
- How does the size of the overlap influence the convergence?
- How does the performance depend on how the collocation points are distributed?

XPINNs (extended physics-informed neural networks)

Jagtap, Karniadakis. *Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations.*

Communications in Computational Physics (2020)

$$\mathcal{L}_s(\theta_s, \mathbf{X}_s) = \omega_{u_s} \mathcal{L}_{u_s}(\theta_s, \mathbf{X}_{u_s}) + \omega_{f_s} \mathcal{L}_{f_s}(\theta_s, \mathbf{X}_{f_s}) + \omega_{l_s} \underbrace{\mathcal{L}_{u_{avg}}(\theta_s, \mathbf{X}_{l_s})}_{\text{interface condition}} + \omega_{l_{fq}} \underbrace{\mathcal{L}_{l_{fq}}(\theta_s, \mathbf{X}_{l_s})}_{\text{interface condition}} + \underbrace{\text{additional interface conditions}}_{\text{optional}},$$

where

$$\mathcal{L}_{u_s}(\theta_s, \mathbf{X}_{u_s}) = \frac{1}{N_{u_s}} \sum_{i=1}^{N_{u_s}} \left| h_s(\mathbf{x}_{u_s}^i; \theta_s) - u(\mathbf{x}_{u_s}^i; \theta_s) \right|^2,$$

$$\mathcal{L}_{f_s}(\theta_s, \mathbf{X}_{f_s}) = \frac{1}{N_{f_q}} \sum_{i=1}^{N_{f_s}} \left| \mathcal{L}(h_s(\mathbf{x}_{f_s}^i; \theta_s)) - f(\mathbf{x}_{f_s}^i) \right|^2,$$

$$\mathcal{L}_{u_{avg}}(\theta_s, \mathbf{X}_{l_s}) = \sum_{\forall s^+} \left(\frac{1}{N_{l_s}} \sum_{i=1}^{N_{l_s}} \left| h_s(\mathbf{x}_{l_s}^i; \theta_s) - \{ \{ h_{s^+}(\mathbf{x}_{l_s}^i; \theta_s) \} \} \right|^2 \right),$$

$$\mathcal{L}_{l_{fq}}(\theta_s, \mathbf{X}_{l_s}) = \sum_{\forall s^+} \left(\frac{1}{N_{l_s}} \sum_{i=1}^{N_{l_s}} \left| \mathcal{L}(h_s(\mathbf{x}_{l_s}^i; \theta_s)) - \mathcal{L}(h_{s^+}(\mathbf{x}_{l_s}^i; \theta_s)) \right|^2 \right).$$

Advantages

- Separate NN for each subdomain
- Allows PINN to be trained in parallel

Drawbacks

- Multiple terms in the loss function can slow training if the weights are not well chosen
- Do not mimic completely the DD behavior

A Motivation for the FBPINN Approach

The **FBPINN** (finite basis physics-informed neural networks) approach has been proposed in [Moseley, Markham, and Nissen-Meyer \(arXiv 2021\)](#).

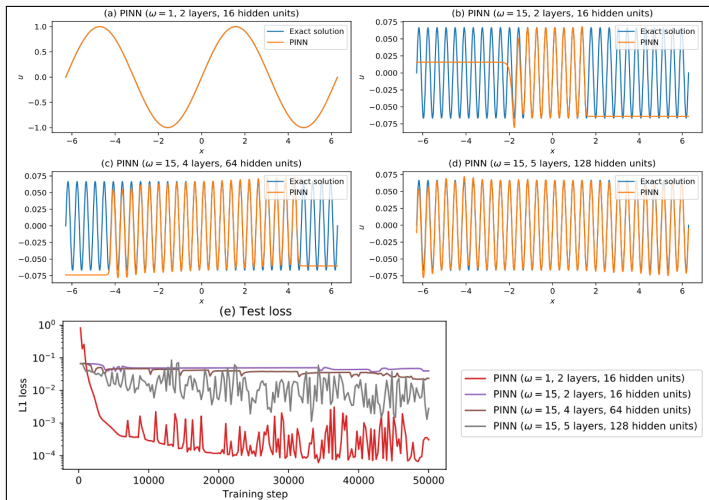
Solve

$$\frac{du}{dx} = \cos(\omega x),$$
$$u(0) = 0,$$

for different values of ω .

Scaling issues

- Size of the computational domain
- Size of frequencies



(a) 321 free parameters

(d) 66 433 free parameters

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

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

using neural networks, we employ the **PINN** approach and **enforce the boundary conditions using a constraining operator**, similar to [Lagaris et al. \(1998\)](#).

Weak enforcement of boundary conditions

Loss function

$$\mathcal{L}(\theta) = \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}} + \omega_{\text{BC}} \mathcal{L}_{\text{BC}},$$

where

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

$$\mathcal{L}_{\text{BC}}(\theta) = \frac{1}{N_{\text{BC}}} \sum_{i=1}^{N_{\text{BC}}} (\mathcal{B}_k[u](\mathbf{x}_i, \theta) - g_k(\mathbf{x}_i))^2.$$

Hard enforcement of boundary conditions

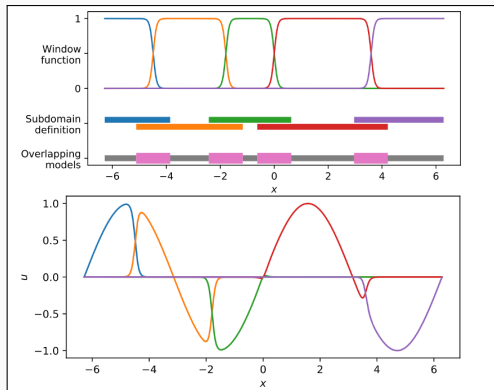
Loss function

$$\mathcal{L}(\theta) = \sum_{i=1}^{N_I} (n[\mathcal{C}u](\mathbf{x}_i, \theta) - f(\mathbf{x}_i))^2,$$

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

→ Often **improves training performance**

FBPINNs – Overlapping Domain Decomposition



- Domain decomposition: $\Omega = \cup_{j=1}^J \Omega_j$
- Collocation points (global): $\{x_i\}_{i=1}^N$
- Overlapping/interior parts Ω_j° and Ω_j^{int}
- Local solutions u_j , window functions w_j
- Global solution $\mathcal{C}u = \mathcal{C} \sum_{j, x_i \in \Omega_j} \omega_j u_j$

Global loss function

$$\begin{aligned} \mathcal{L}(\theta_1, \dots, \theta_J) = & \underbrace{\frac{1}{N} \sum_{\mathbf{x} \in X^{int}} \left(n \left[\mathcal{C} \sum_{l, \mathbf{x} \in X_l} \omega_l u_l \right] (\mathbf{x}, \theta_l) - f(\mathbf{x}) \right)^2}_{=: \mathcal{L}^\circ(\theta_1, \dots, \theta_J)} \\ & + \underbrace{\frac{1}{N} \sum_{\mathbf{x} \in X^\circ} \left(n \left[\mathcal{C} \sum_{l, \mathbf{x} \in X_l} \omega_l u_l \right] (\mathbf{x}, \theta_l) - f(\mathbf{x}) \right)^2}_{=: \mathcal{L}^{int}(\theta_1, \dots, \theta_J)}. \end{aligned}$$

Since $X_i^{int} \cap X_j^{int} = \emptyset$ for $i \neq j$,

$$\mathcal{L}^{int}(\theta_1, \dots, \theta_J) = \frac{1}{N} \sum_{j=1}^J \sum_{\mathbf{x}_i \in X_j^{int}} (n [\mathcal{C} \omega_j u_j] (\mathbf{x}_i, \theta_j) - f(\mathbf{x}_i))^2$$

- The subdomains can be split into **active** (trained in parallel) and **inactive** (fixed)
- This corresponds to classical **parallel (all active)** or **multiplicative (one active at a time)** Schwarz methods

Algorithm 1: FBPINN training step

if $j \in \mathcal{A}$ (Ω_j is an active domain) then

Perform p iterations of gradient descent on θ_j^k (θ_i^k where $i \neq j$ are kept fixed):

$$\theta_j^{k+l} = \theta_j^{k+l-1} - \lambda \nabla_{\theta_j} \mathcal{L}(\theta_1^k, \dots, \theta_{j-1}^k, \theta_j^{k+l-1}, \theta_{j+1}^k, \dots, \theta_j^k),$$

Update the solution in the overlapping regions
(communicate with neighbours):

$$\forall \mathbf{x} \in \Omega_j^o, u(\mathbf{x}, \theta_j^{k+p}) \leftarrow \sum_{l, \mathbf{x} \in \Omega_l} \omega_l u_l(\mathbf{x}, \theta_l^{k+p}).$$

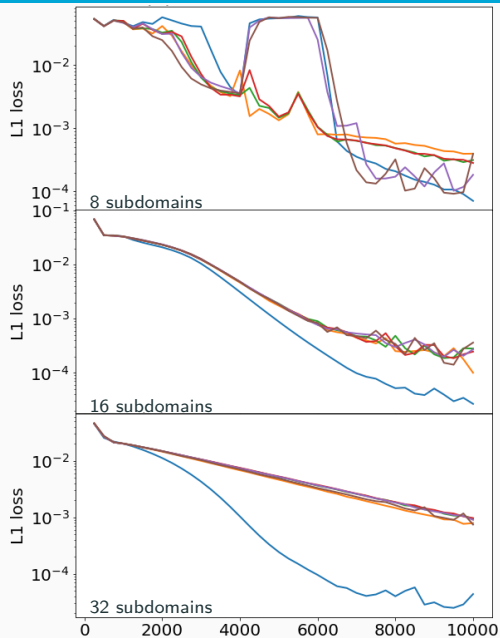
end

→ We only consider **parallel** (all active) iterations for now.

Summary

- Communication every p iterations (better for overall efficiency)
- Multiplication with a window function : a way to restrict to the local domain.
- The set of active domains can be changed after the local training is completed.

FBPINNs – Weak Scaling Study



Solve, for $\omega = 15$,

$$\frac{du}{dx} = \cos(\omega x), \quad u(0) = 0.$$

- Fixed local network size and number of local collocation points. Then, we **increase the number of subdomains**.
- We choose all subdomains as active and test the influence of



Observations

- Convergence get worse with an increasing number of subdomains
- No noticeable difference depending on how often we update, unless we update every iteration

Let us focus **updating each iteration** for now.

Laplace Problem

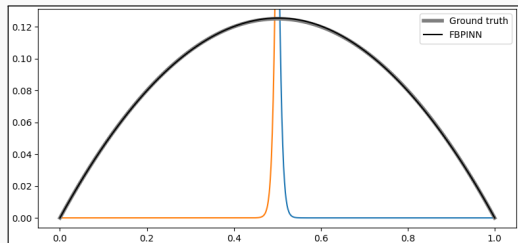
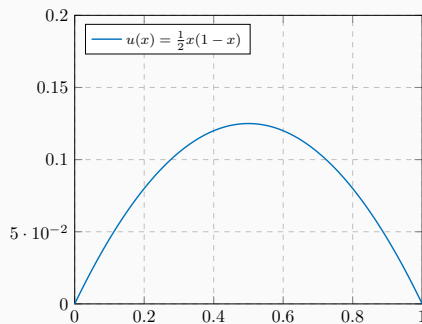
Let us now consider the simple boundary value problem

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

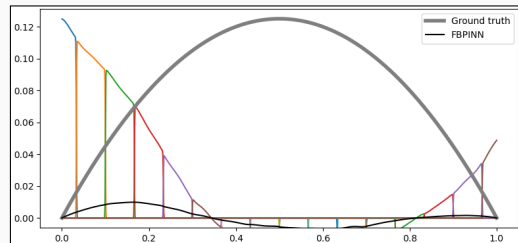
$$u(0) = u(1) = 0,$$

which has the solution

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

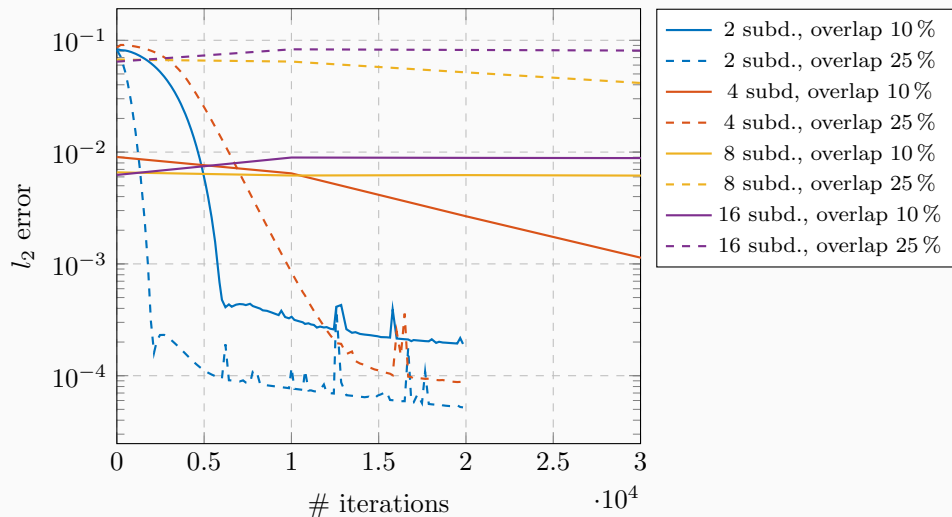


1 level, 2 subdomains

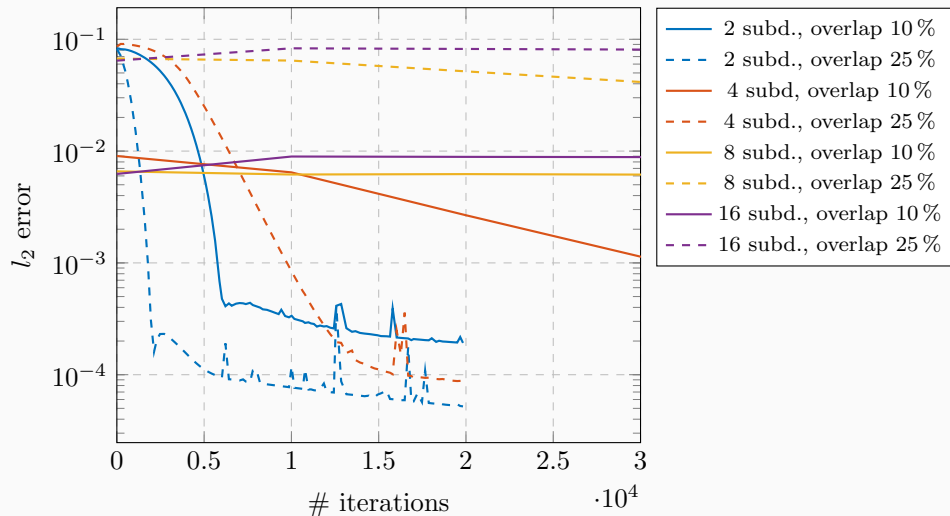


1 level, 32 subdomains

Weak Scaling – Effect of the Overlap

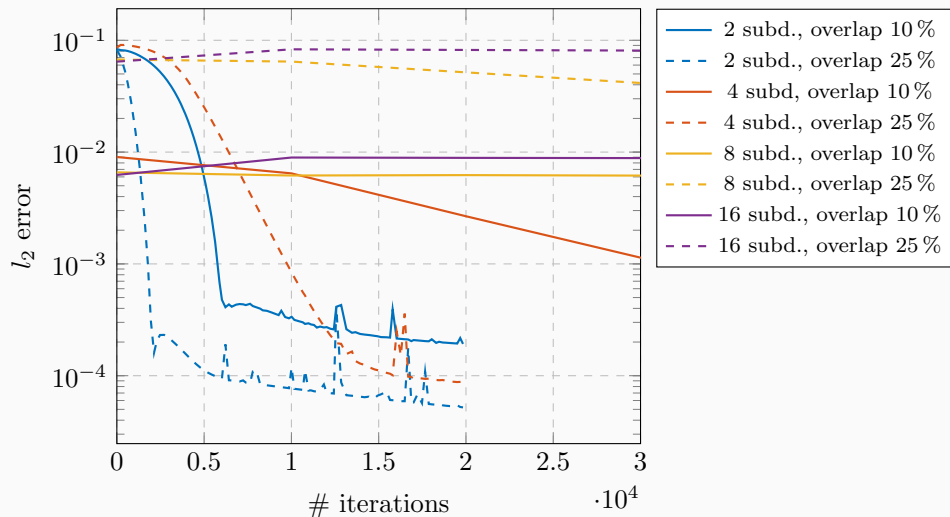


Weak Scaling – Effect of the Overlap



→ Larger overlap improves convergence.

Weak Scaling – Effect of the Overlap



→ Larger overlap **improves convergence**.

→ **No scalability** with respect to **increasing number of subdomains**.

Two-Level FBPINN Algorithm

Coarse correction and spectral bias

Questions:

- Scalability requires **global transport of information**. In domain decomposition, this is typically done using a **coarse global problem**.
- What does this mean in the **context of network training**?

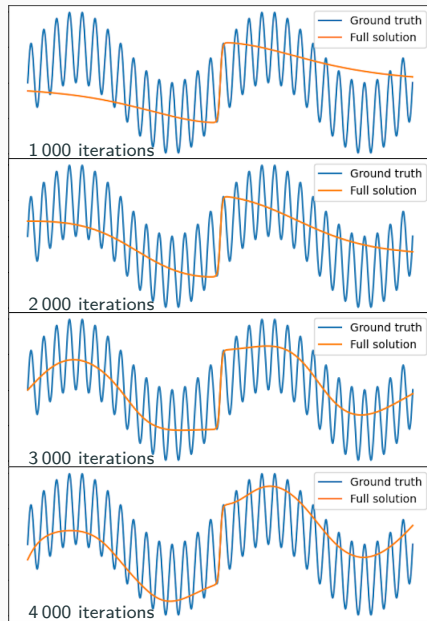
Idea:

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

Investigate this for a **simple model problem** with **two frequencies**

$$\begin{cases} \frac{du}{dx} = \omega_1 \cos(\omega_1 x) + \omega_2 \cos(\omega_2 x) \\ u(0) = 0. \end{cases}$$

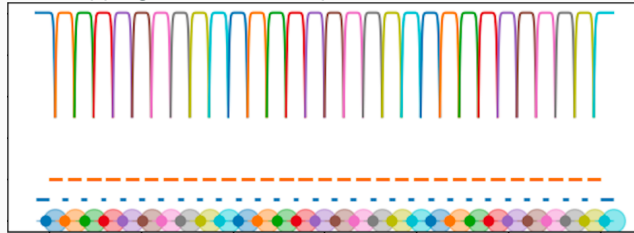
with $\omega_1 = 1$, $\omega_2 = 15$,



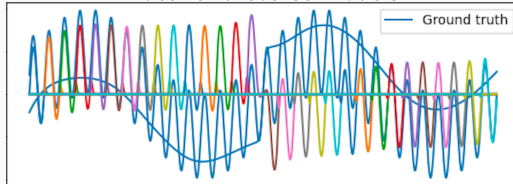
Coarse and Local Training

Now, learn the error (higher frequencies) using the **one-level FBPINN model** using **local models on 30 subdomains**.

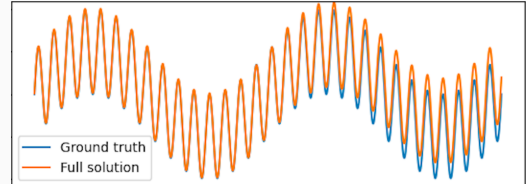
Overlapping domain decomposition into 30 subdomains



Local and coarse models



Two-level model



Laplace Problem

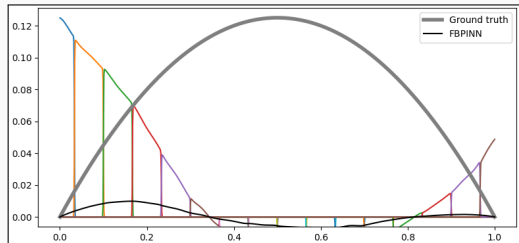
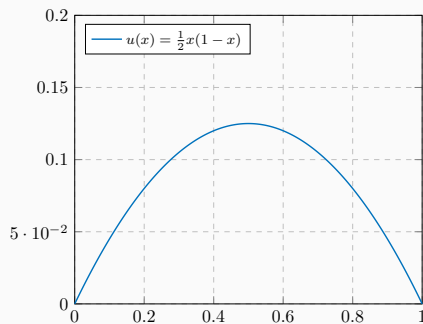
Let us, again, consider the boundary value problem

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

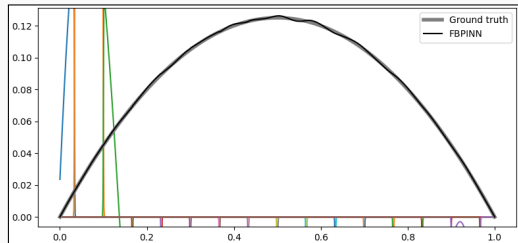
$$u(0) = u(1) = 0,$$

which has the solution

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

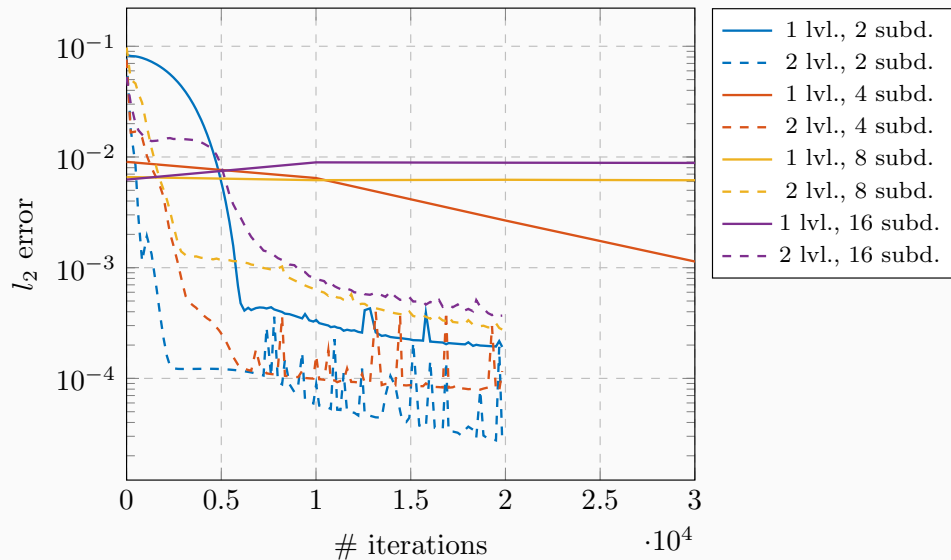


1 level, 16 subdomains



2 levels, 16 subdomains

Weak Scaling – Comparison One and Two Levels



→ Adding a second level **improves scalability**.

Multi-Frequency Laplace Problem

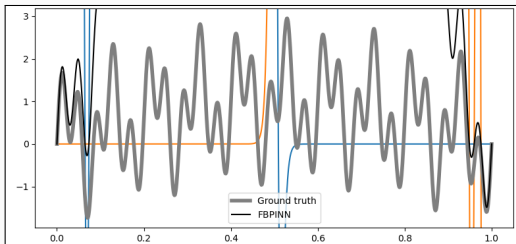
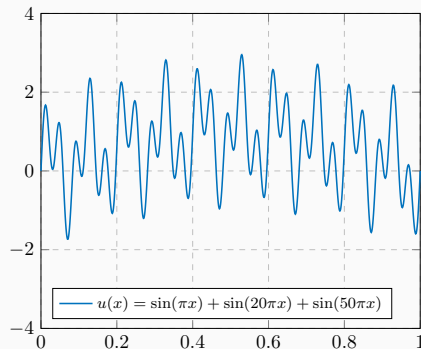
Let us now consider the boundary value problem

$$-\Delta u = \pi^2 \sin(\pi x) + (20\pi)^2 \sin(20\pi x) + (50\pi)^2 \sin(50\pi x) \text{ in } [0, 1],$$

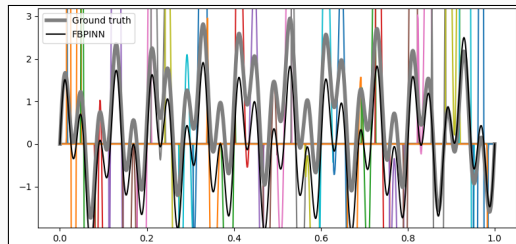
$$u(0) = u(1) = 0,$$

which has the solution

$$u(x) = \sin(\pi x) + \sin(20\pi x) + \sin(50\pi x).$$



1 level, 2 subdomains



1 level, 32 subdomains

Multi-Frequency Laplace Problem

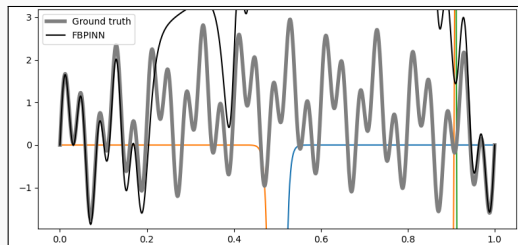
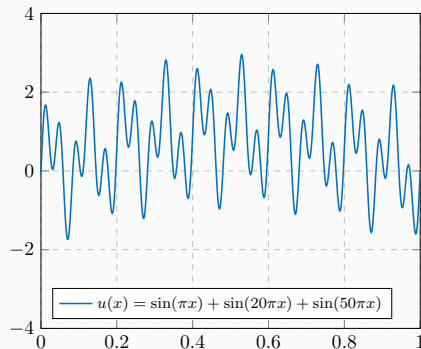
Let us now consider the boundary value problem

$$-\Delta u = \pi^2 \sin(\pi x) + (20\pi)^2 \sin(20\pi x) + (50\pi)^2 \sin(50\pi x) \text{ in } [0, 1],$$

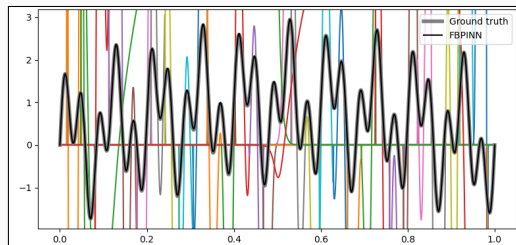
$$u(0) = u(1) = 0,$$

which has the solution

$$u(x) = \sin(\pi x) + \sin(20\pi x) + \sin(50\pi x).$$

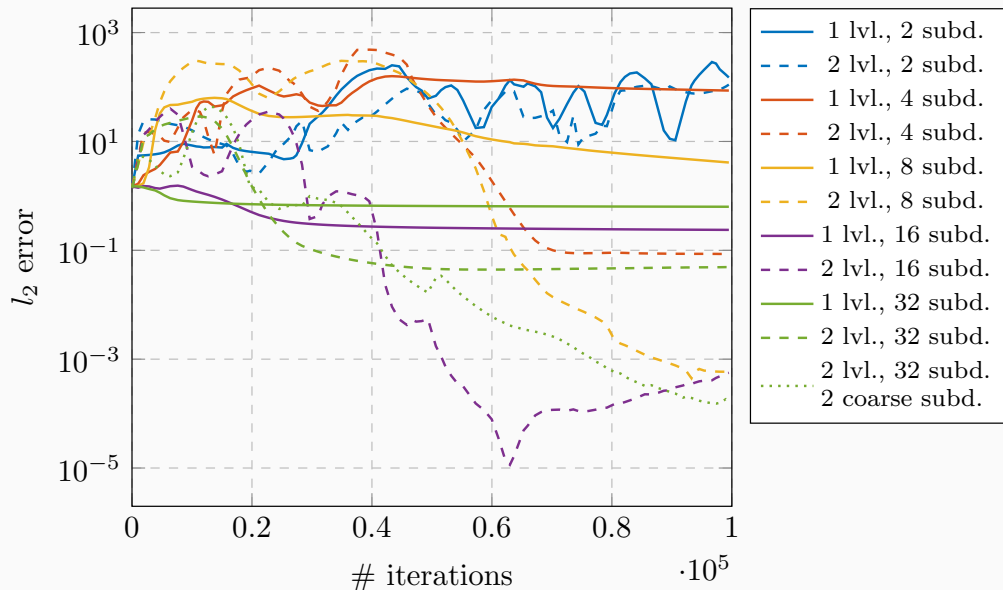


2 levels, 2 subdomains



2 levels, 32 subdomains

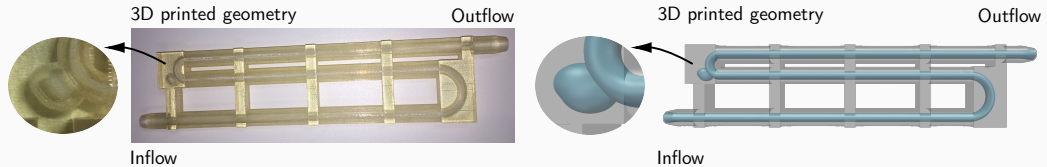
Weak Scaling – Comparison One and Two Levels



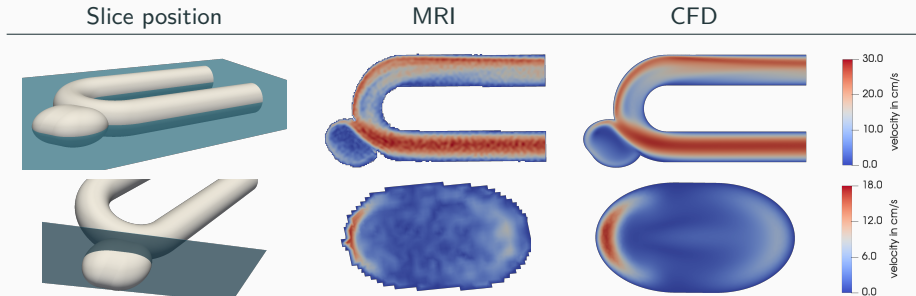
**Surrogate models for
computational fluid dynamics
simulations
Data-driven approach**

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.

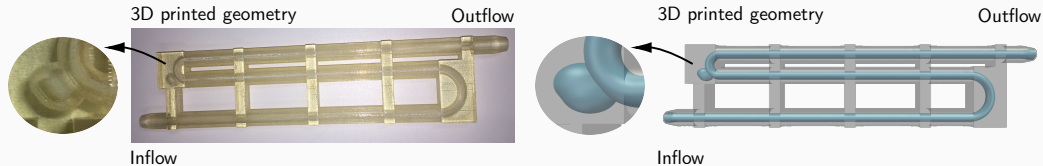


To obtain accurate simulation results, simulations with $\approx 10^6$ d.o.f.s were carried out. On $O(100)$ MPI ranks, the computation of a steady state took $O(1)$ h on CHEOPS supercomputer at UoC.

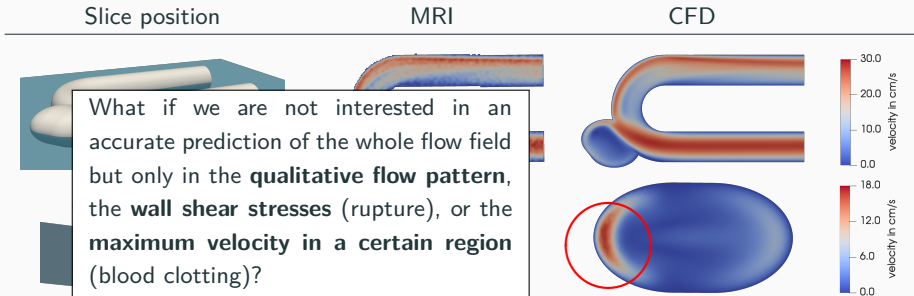


Computational Fluid Dynamics (CFD) Simulations are Time Consuming

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, simulations with $\approx 10^8$ d.o.f.s were carried out. On $O(100)$ MPI ranks, the computation of a steady state took $O(1)$ h on CHEOPS supercomputer at UoC.

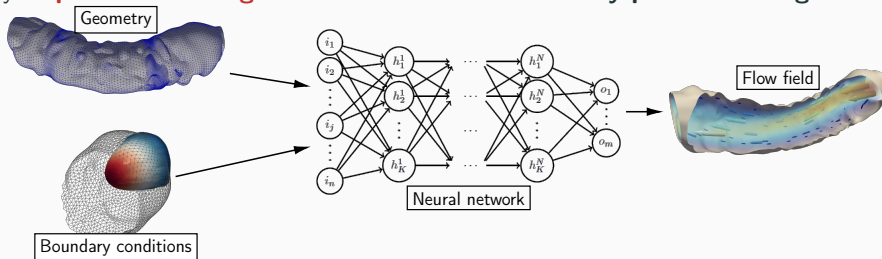


Using PINNs as Surrogate Models

Learning the solution of one specific boundary value problem (BVP), for instance, using PINNs

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

generally **requires re-training the model** once the boundary problem changes.



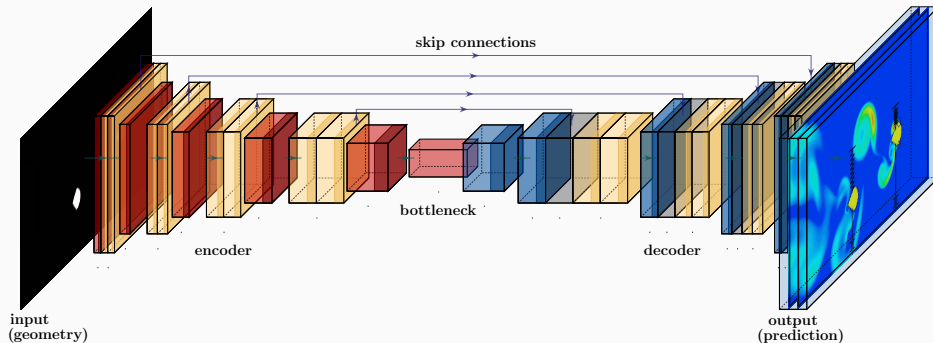
Instead, we are interested in a **single surrogate model** that can predict the solution for a variety of

- **geometries,**
- **initial and boundary conditions,** and/or
- **material parameters.**

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.

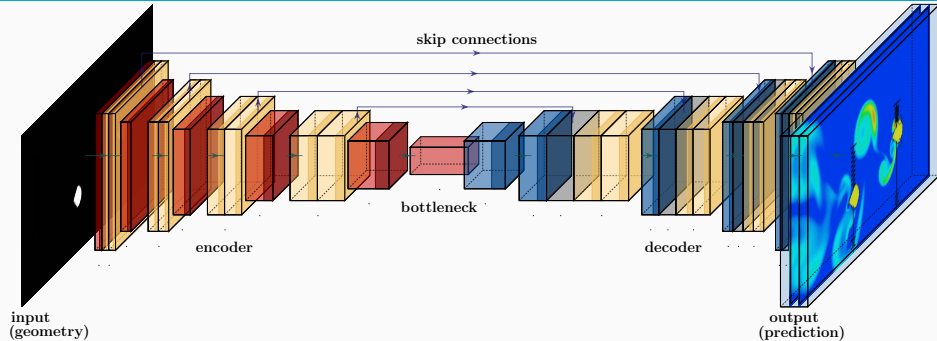
In particular, we use a pixel image of the **geometry as input** and predict an image of the resulting **stationary flow field as output**:



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.,

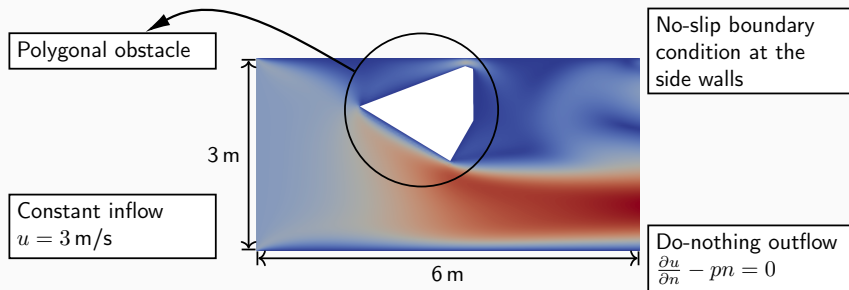
- between the right-hand side and the solution of a BVP.

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

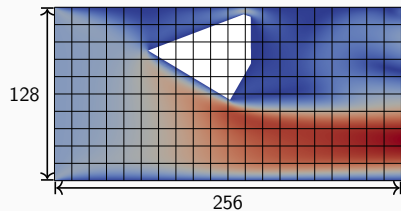
We propose a **simple model problem** to investigate predictions of a **steady flow in a channel with an obstacle**; this setup is also inspired by [Guo, Li, Iorio \(2016\)](#).



In particular, we restrict ourselves to

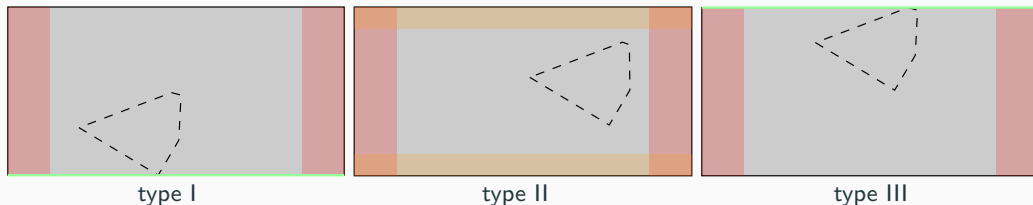
- a **simple rectangular basic geometry** and
- **fixed boundary conditions**.

However, we **vary the geometry of the polygonal obstacle**. In addition, we **interpolate the input and output data to a structured tensor product mesh** to impose a structure.



Type I – III Geometries (Eichinger, Heinlein, Klawonn (2021, 2022))

We first consider **obstacles** of the following three types; see also [Guo, Li, Iorio \(2016\)](#) for a similar approach. In particular, we **randomly generate star-shaped polygons with 3, 4, 5, 6, and 12 edges**.



First, we consider **100 000 pairs of geometry and flow data** (90 000 training; 10 000 validation) for **Type I (50 000) & Type II (50 000)**. Later, we will also consider Type III.

Computation of the Flow Data Using OpenFOAM®

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

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

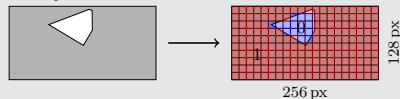
where \vec{u} and p are the velocity and pressure fields and ν is the viscosity. Furthermore, we prescribe the previously described boundary conditions.

Software pipeline

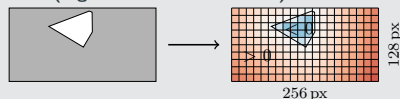
1. Define the boundary of the polygonal obstacle and **create the corresponding STL (standard triangulation language) file.**
2. **Generate a hexahedral compute grid** (snappyHexMesh).
3. Run the **CFD simulation** (simpleFoam).
4. **Interpolate geometry information and flow field** onto a pixel grid.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

Input data

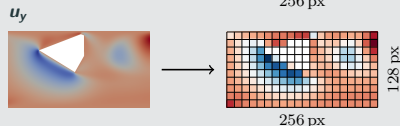
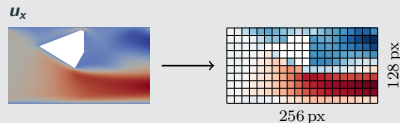
Binary



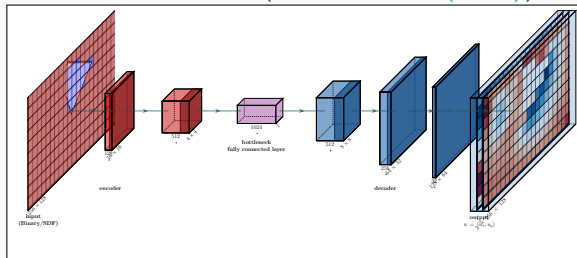
SDF (Signed Distance Function)



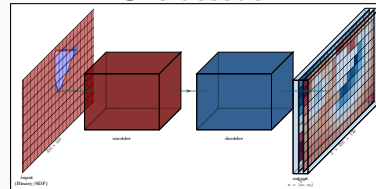
Output data



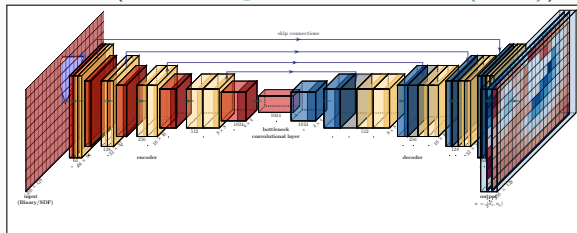
Bottleneck CNN (Guo, Li, Iorio (2016))



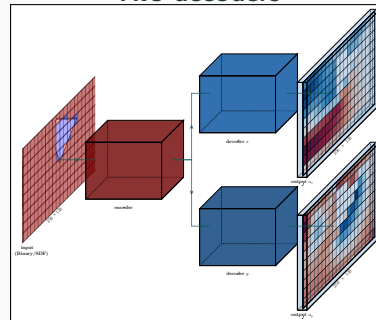
One decoder



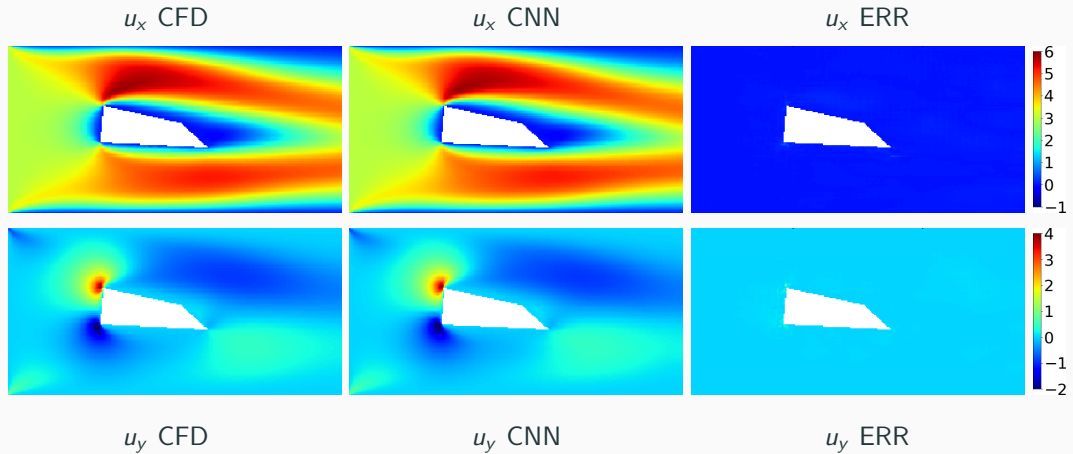
U-Net (Ronneberger, Fischer, Brox (2015))



Two decoders

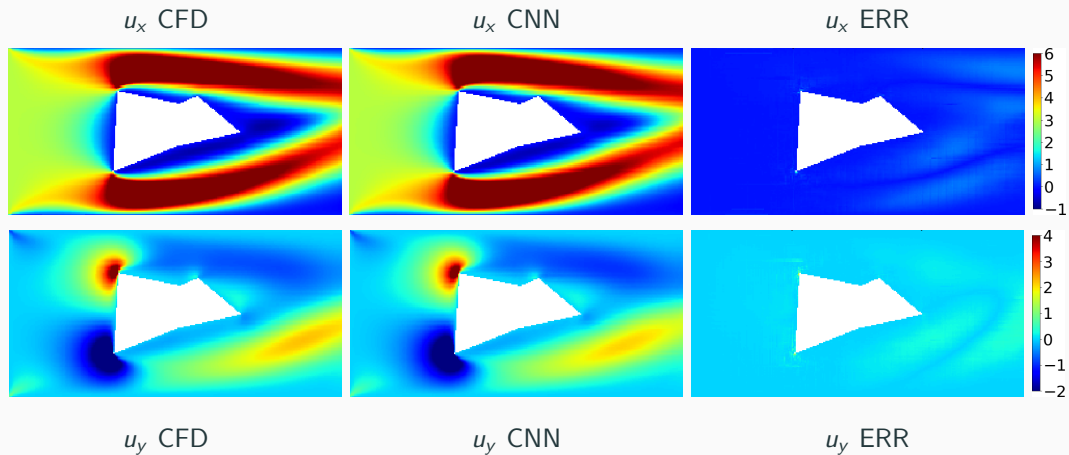


Comparison CFD Vs NN (Relative Error 2 %)



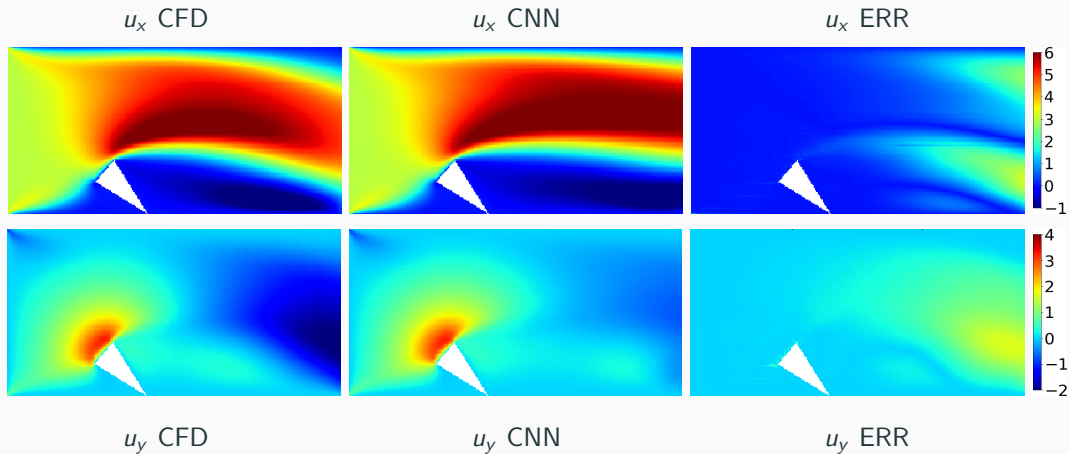
Cf. Eichinger, Heinlein, Klawonn (2021, 2022)

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))

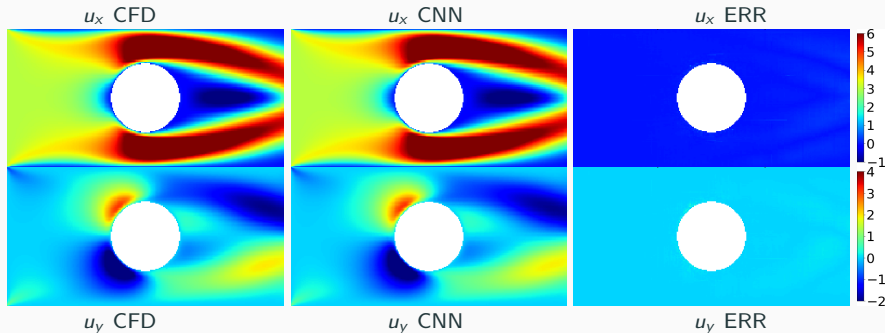
We compare the **relative error (RE)** $\frac{\|u_{i,j} - \hat{u}_{i,j}\|_2}{\|u_{i,j}\|_2 + 10^{-4}}$ averaged over all non-obstacle pixels and all validation data configurations. Furthermore: **MSE** = mean squared error; **MAE** = mean absolute error.

			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	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 %

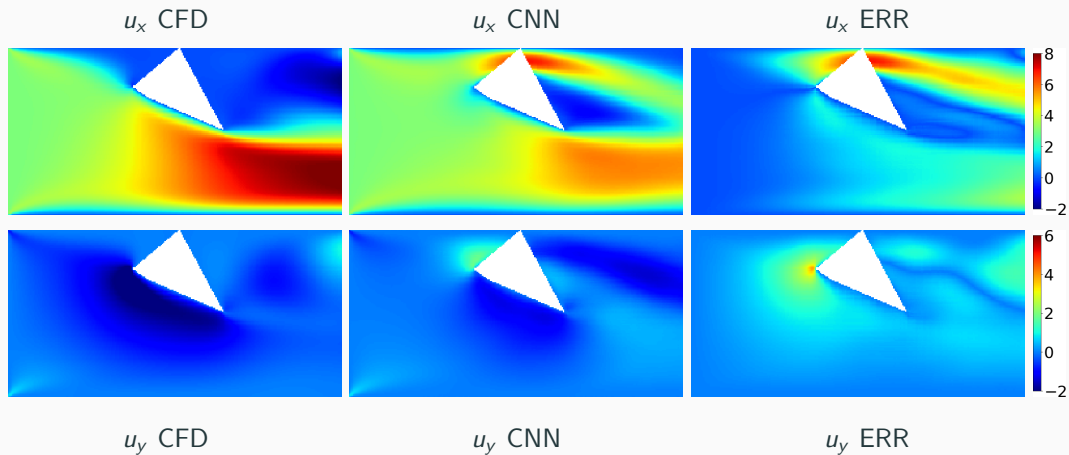
Generalization Properties (Eichinger, Heinlein, Klawonn (2021, 2022))

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 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

		type I & II		type III	
learning approach	# training epochs	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**.

Data:

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

Training:

	Bottleneck CNN		U-Net	
# decoders	1	2	1	2
# parameters	≈ 47 m	≈ 85 m	≈ 34 m	≈ 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**.

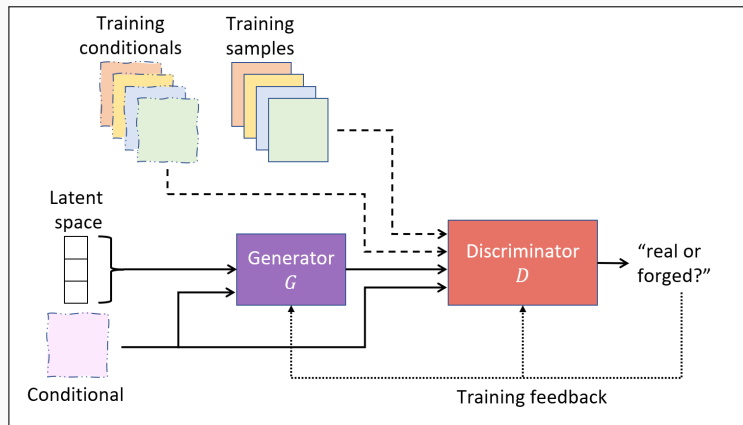
CPU: AMD Threadripper 2950X (8 \times 3.8 Ghz), 32GB RAM;

GPU: GeForce RTX 2080Ti

**Surrogate models for
computational fluid dynamics
simulations
GAN-based training**

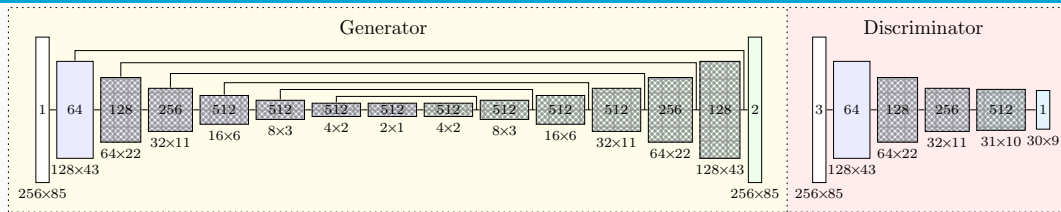
Training the Surrogate Model via GANs

Cf. Kemna, Heinlein, Vuik (accepted 2022).

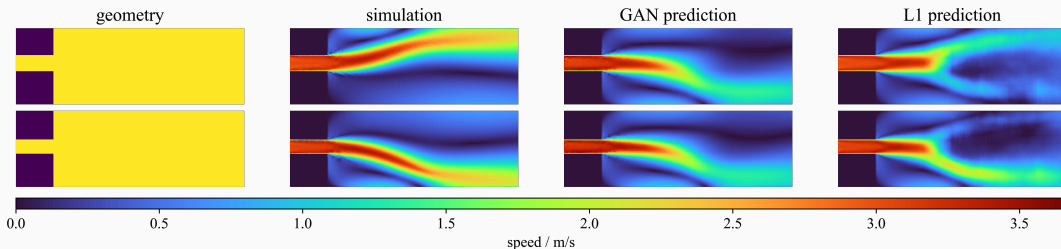


- **Generative adversarial networks (GANs)** based on [Goodfellow et al. \(2014\)](#) consist of two independent neural networks that are trained concurrently in an adversarial setting:
 - **Generator** is trained to *fool* the discriminator into classifying its outputs as training data
 - **Discriminator**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

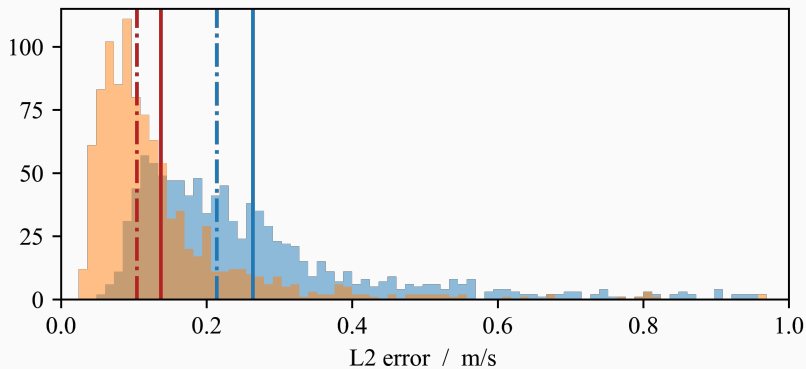
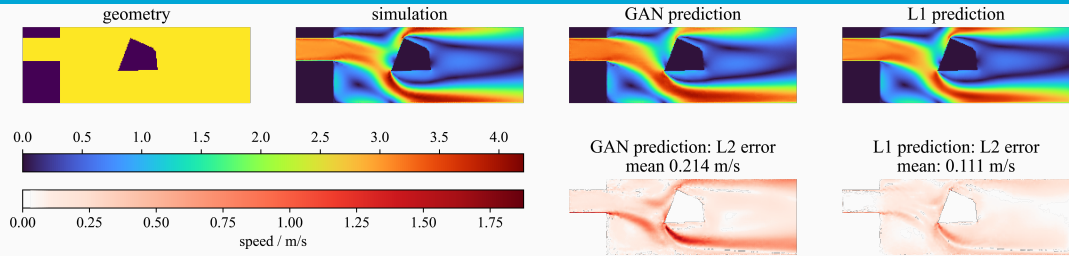
GANs for Fluid Prediction – Bifurcation Example



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

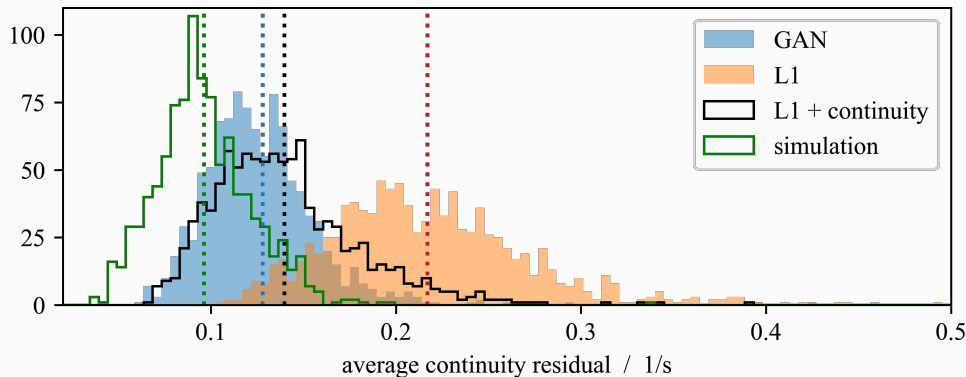


GANs for Fluid Prediction – Divergence Error

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

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

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

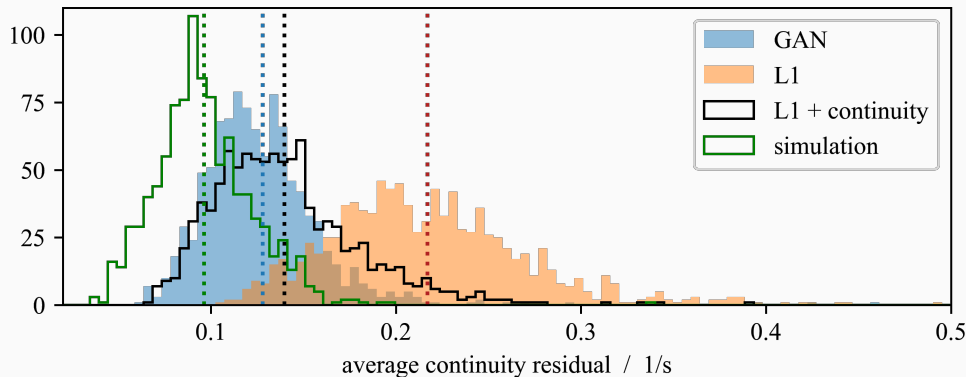


GANs for Fluid Prediction – Divergence Error

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

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

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

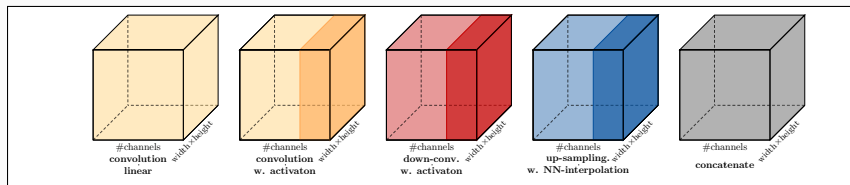
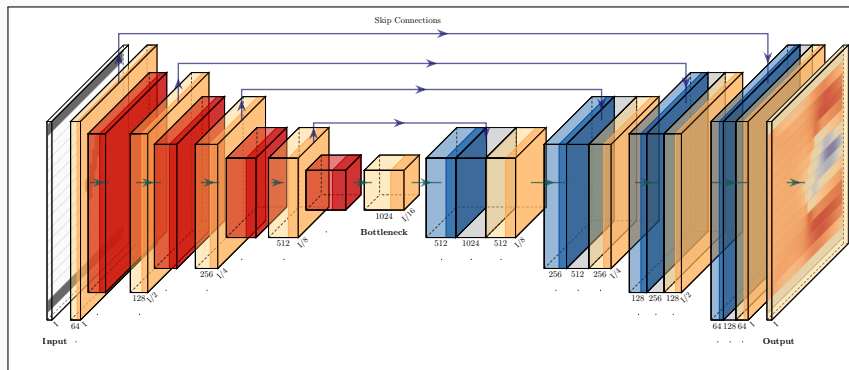


→ The GAN loss seems to **help learning the physics of the system**.

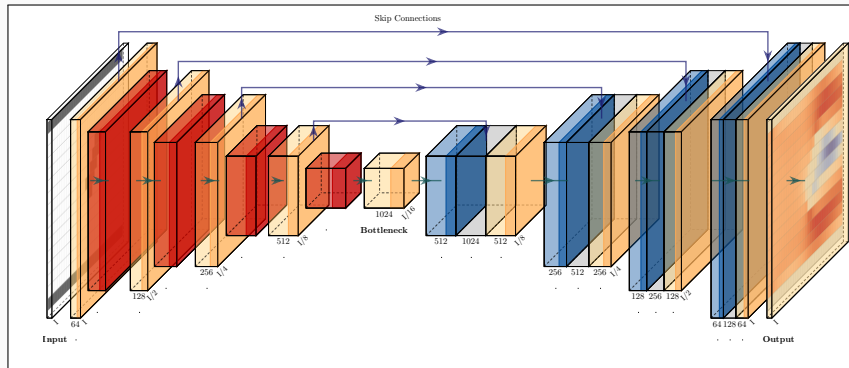
**Surrogate models for
computational fluid dynamics
simulations
Physics-aware approach**

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

→ We further **improved the U-Net architecture** for our application.



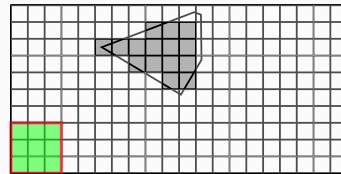
U-Net (Ronneberger, Fischer, Brox (2015)) Revisited



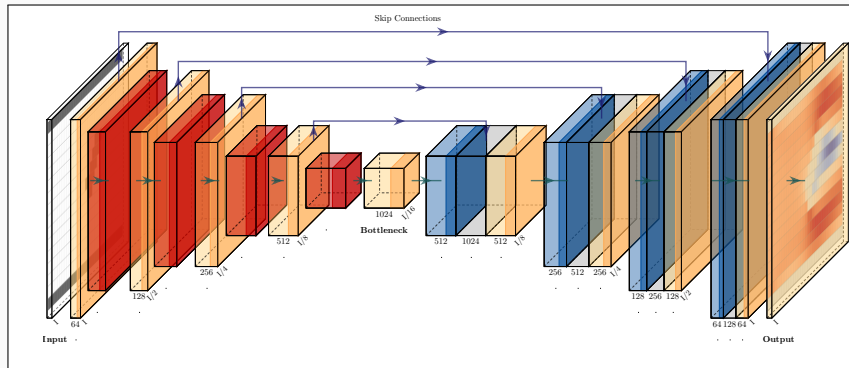
Convolution

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

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



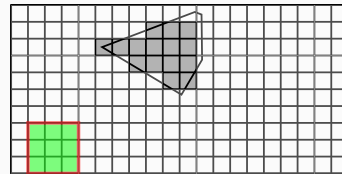
U-Net (Ronneberger, Fischer, Brox (2015)) Revisited



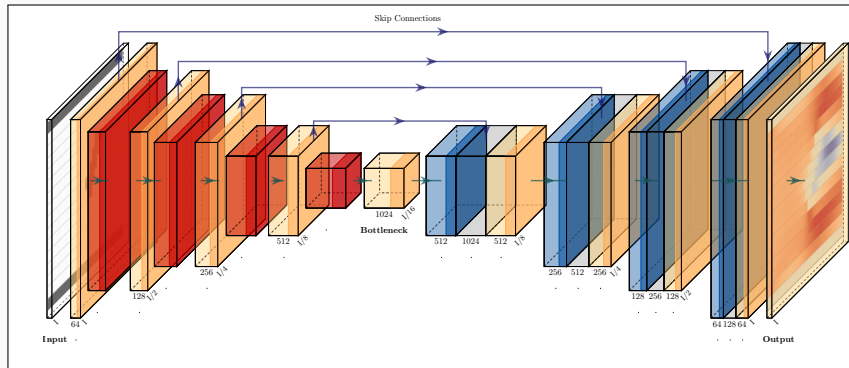
Convolution

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

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



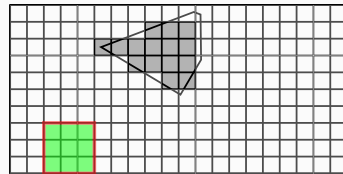
U-Net (Ronneberger, Fischer, Brox (2015)) Revisited



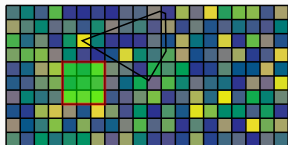
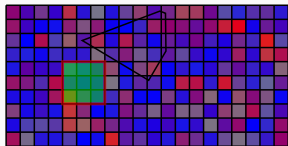
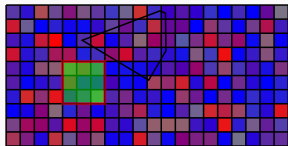
Convolution

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



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

Cf. Grimm, Heinlein, Klawonn

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

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

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

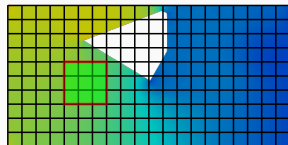
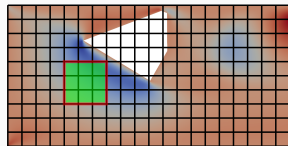
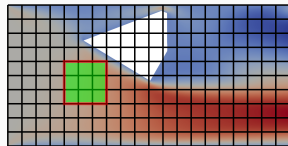
$$F_{\text{mom}}(u, p) := -\nu \Delta \bar{u} + (u \cdot \nabla) \bar{u} + \nabla p,$$

$$F_{\text{mass}}(u, p) := \nabla \cdot u.$$

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:

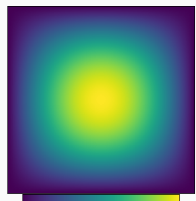
$$\begin{matrix} & \frac{\partial}{\partial x} & & & & \frac{\partial}{\partial y} \\ & \begin{matrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{matrix} & & \begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{matrix} & & \end{matrix}$$

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



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

Convergence Comparison – CNN Versus FDM



0.0 0.5 1.0

Solve

$$-\Delta u = f$$

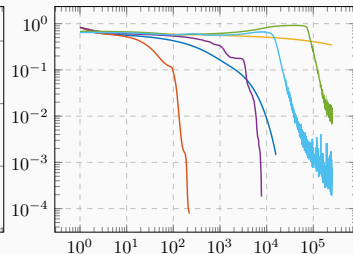
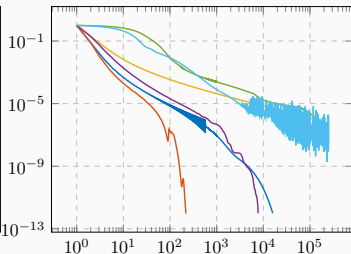
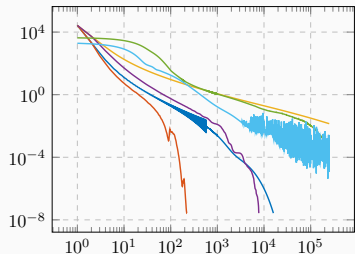
using

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

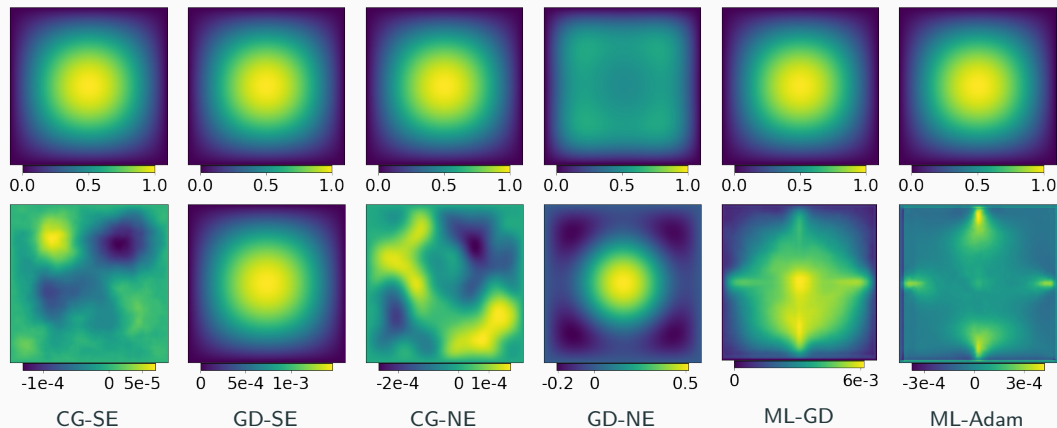
$$\|r^k\|^2$$

$$\frac{\|r^k\|^2}{\|r^0\|^2}$$

$$\frac{\|u_k - u^*\|}{\|u^*\|}$$



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 (submitted 2022).

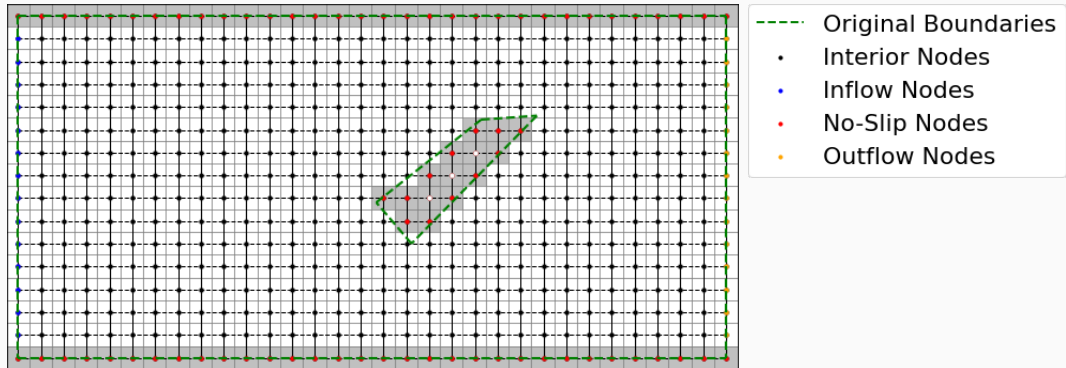
Physics-Aware Approach – Boundary Conditions

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

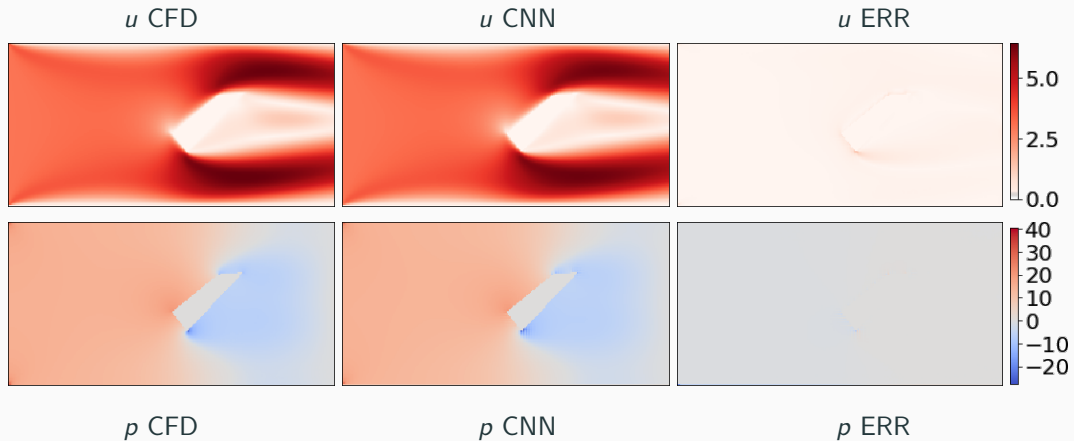
Therefore, we also call this **physics-aware** approach **unsupervised**.

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

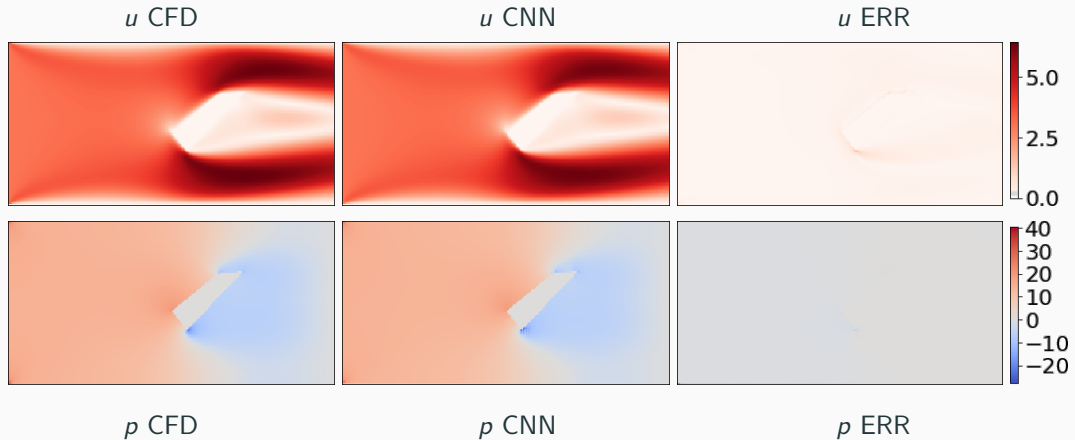
Hence, we also have to **enforce the boundary conditions of our boundary value problem**:



Physics-Aware Approach – Single Geometry

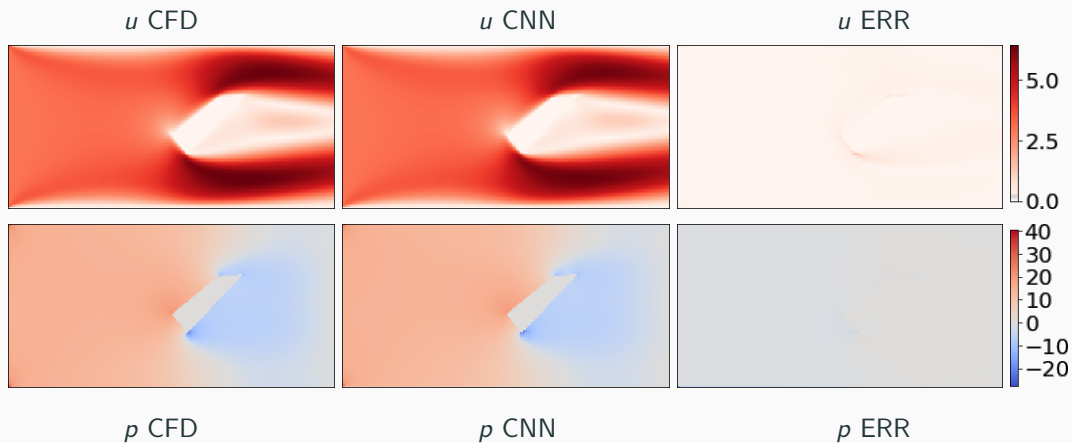


Physics-Aware Approach – Single Geometry



⇒ We can **solve the boundary value problem using a neural network.**

Physics-Aware Approach – Single Geometry



⇒ We can **solve the boundary value problem using a neural network.**

→ Now, we again build a **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.	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-aware	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		# epochs trained
					momentum	mass	
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-aware	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-aware approach** are **comparable to the data-based approach**; the **errors are slightly higher**. However, no **reference data at all is needed for the training**.

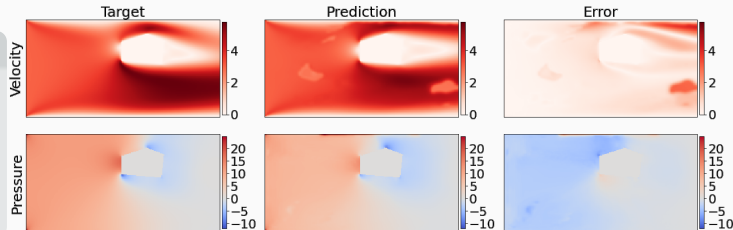
Generalization

Now, consider an obstacle which is **closer to the wall** (≈ 0.4 m) **than the training data** (≥ 0.75 m).

Supervised approach

$$\frac{\|u_{NN} - u\|_2}{\|u\|_2} = 23\%$$

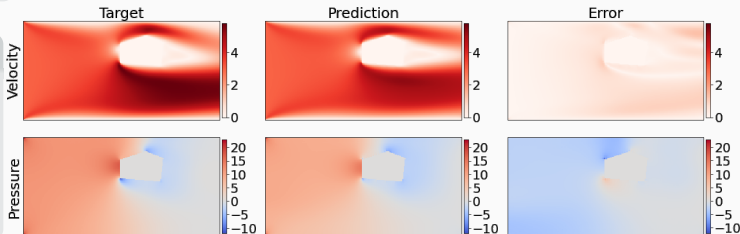
$$\frac{\|p_{NN} - p\|_2}{\|p\|_2} = 31\%$$



Unsupervised approach

$$\frac{\|u_{NN} - u\|_2}{\|u\|_2} = 14\%$$

$$\frac{\|p_{NN} - p\|_2}{\|p\|_2} = 27\%$$



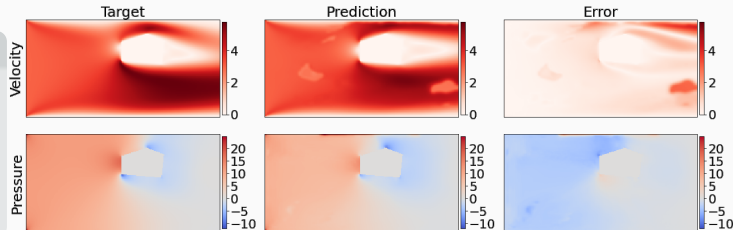
Generalization

Now, consider an obstacle which is **closer to the wall** (≈ 0.4 m) **than the training data** (≥ 0.75 m).

Supervised approach

$$\frac{\|u_{NN} - u\|_2}{\|u\|_2} = 23\%$$

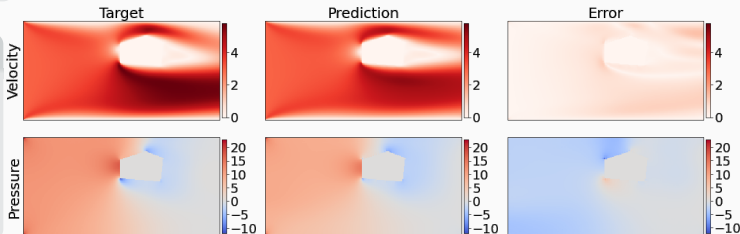
$$\frac{\|p_{NN} - p\|_2}{\|p\|_2} = 31\%$$



Unsupervised approach

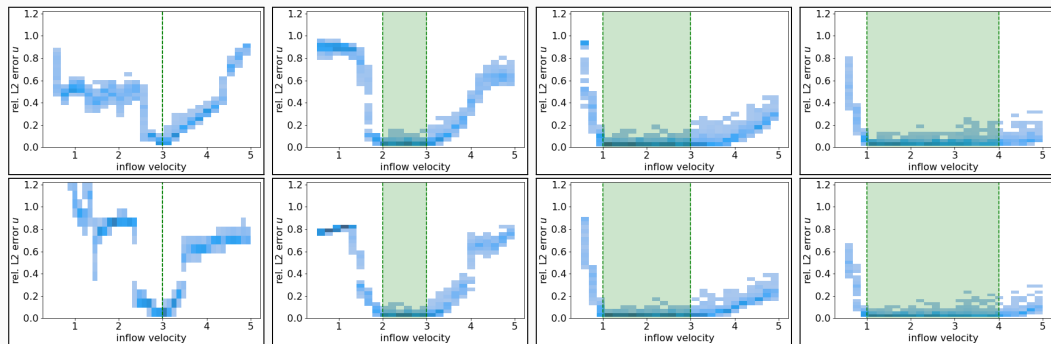
$$\frac{\|u_{NN} - u\|_2}{\|u\|_2} = 14\%$$

$$\frac{\|p_{NN} - p\|_2}{\|p\|_2} = 27\%$$



→ The unsupervised approach **generalizes slightly better**, and in particular, **the prediction is smoother and misses unphysical artifacts**.

Generalization With Respect to the Inflow Velocity

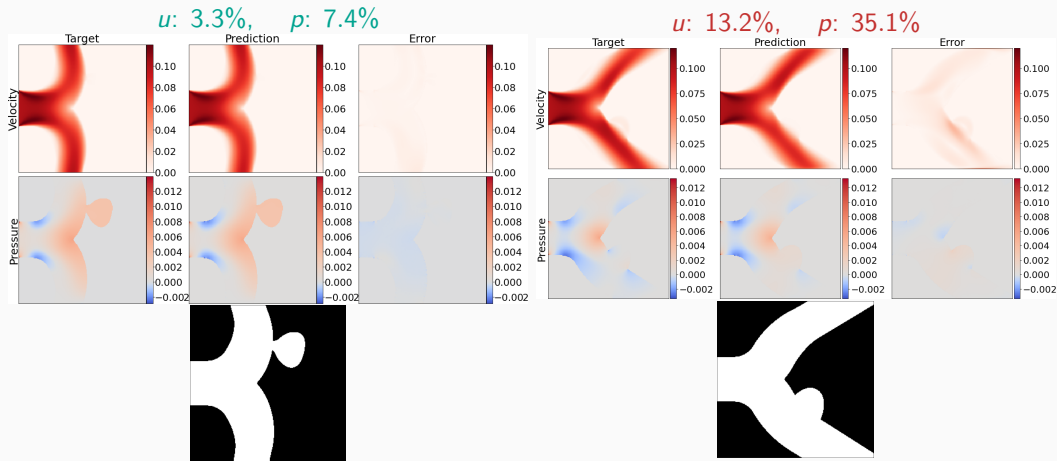


order	# data	range inflow vel.	[0.5, 1.0]	[1.0, 2.0]	[2.0, 3.0]	[3.0, 4.0]	[4.0, 5.0]
2	1 000	[3.0, 3.0]	55.5 %	48.1 %	31.1 %	17.4 %	61.5 %
		[2.0, 3.0]	89.3 %	57.4 %	4.0 %	15.5 %	59.1 %
		[1.0, 3.0]	40.2 %	3.8 %	4.3 %	7.1 %	20.4 %
		[1.0, 4.0]	31.3 %	4.0 %	4.3 %	5.8 %	7.7 %
2	4 500	[3.0, 3.0]	186.8 %	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 %

Aneurysm Geometries

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

Relative L_2 -error on the validation data set in u : **4.9%**, in p : **9.5%**.



Thank you for your attention!

Summary

- The new 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**.
- **Domain decomposition methods** can help to **improve the training process for PINNs**, especially for (but not restricted to) **large domains** and/or **multiscale problems**.
- The **FBPINN method integrates domain decomposition approaches into PINN training** in a natural way; it can also be extended to a **two-level method**.
- Using **CNNs on image data** yields an **operator learning approach for predicting fluid flow** inside **varying computational domains**; again, the **model training can be enhanced by using physics**.

Acknowledgements

- The **Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE)**