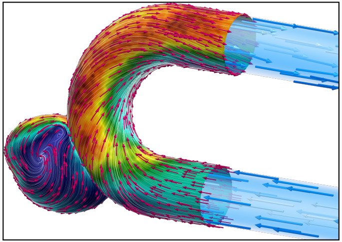
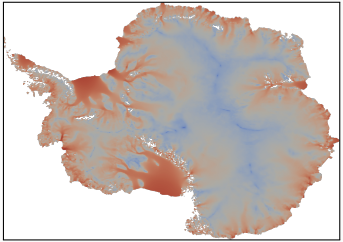
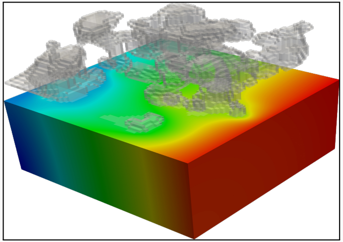
Geometric Challenges in Machine Learning-Based Surrogate Models

Alexander Heinlein[1]

International Conference on Applied AI and Scientific Machine Learning (CASML 2024), Indian Institute of Science (IISc), Bangalore, India, December 14-18

[1]Delft University of Technology

Numerical Analysis and Machine Learning

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

Outline

1 Surrogate models for varying computational domains

Based on joint work with

Eric Cyr	(Sandia National Laboratories)
Mattias Eichinger, **Viktor Grimm**, and **Axel Klawonn**	(University of Cologne)
Corné Verburg	(Delft University of Technology)

2 Domain decomposition-based deep operator networks

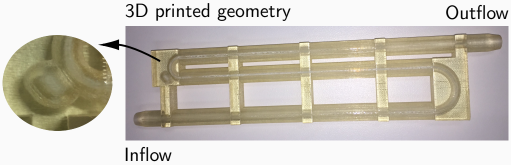
Based on joint work with

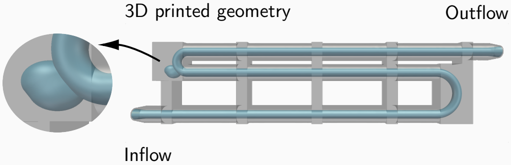
Damien Beecroft	(University of Washington)
Eric Cyr	(Sandia National Laboratories)
Victorita Dolean	(Eindhoven University of Technology)
Bianca Giovanardi, **Corné Verburg**, **Coen Visser**	(Delft University of Technology)
Amanda A. Howard and **Panos Stinis**	(Pacific Northwest National Laboratory)
Siddhartha Mishra	(ETH Zürich)
Ben Moseley	(Imperial College London)

Surrogate models for varying computational domains

Computational Fluid Dynamics (CFD) Simulations are Time Consuming

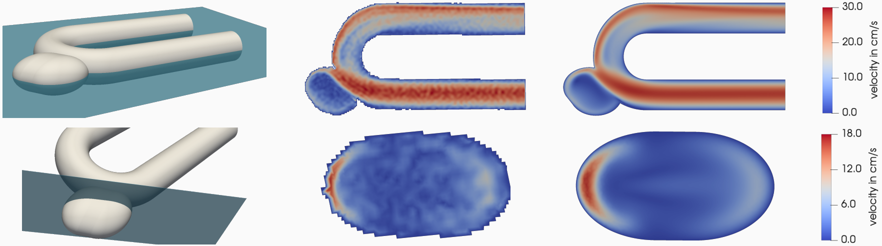
In **Giese, Heinlein, Klawonn, Knepper, Sonnabend (2019)**, a benchmark for **comparing MRI measurements and CFD simulations** of hemodynamics in **intracranial aneurysms** was proposed.





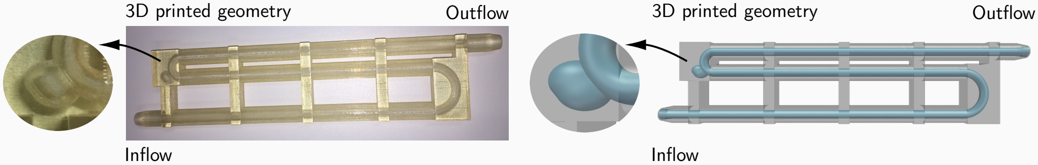
To obtain accurate simulation results, a simulation with $\approx 10\,\mathrm{m}$ d.o.f.s has been carried out. On $O(100)$ MPI ranks, the computation of a steady state took $O(1)\,\mathrm{h}$ on CHEOPS supercomputer at UoC.

Slice position | MRI | CFD

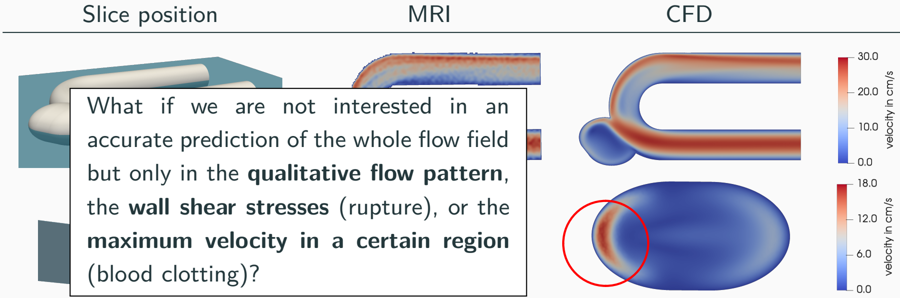


Computational Fluid Dynamics (CFD) Simulations are Time Consuming

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



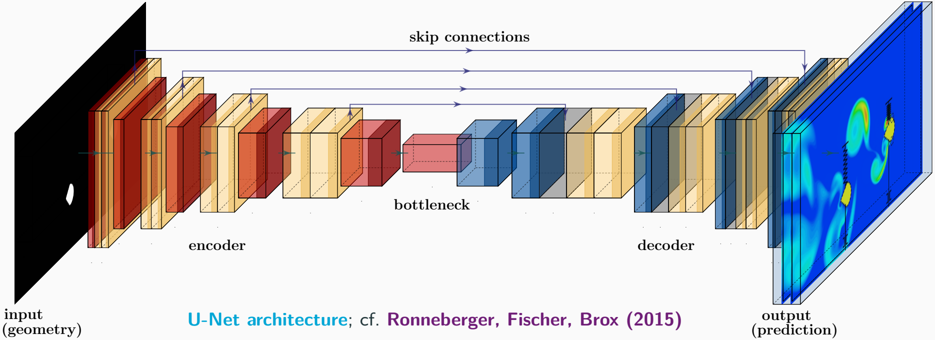
To obtain accurate simulation results, a simulation with $\approx 10\,$m d.o.f.s has been carried out. On $O(100)$ MPI ranks, the computation of a steady state took $O(1)\,$h on CHEOPS supercomputer at UoC.



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

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

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

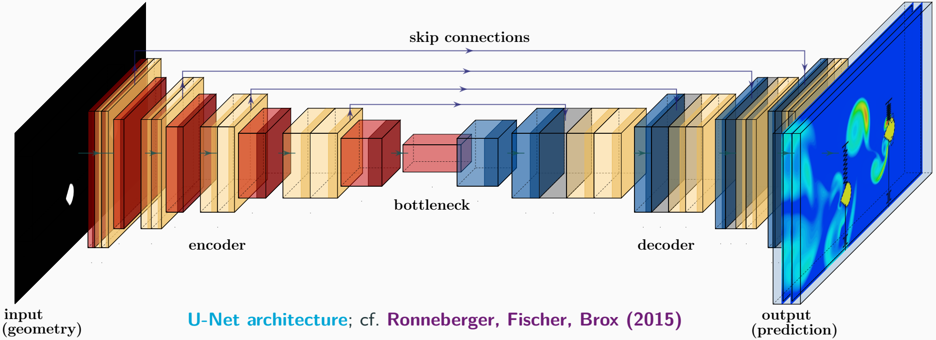


U-Net architecture; cf. **Ronneberger, Fischer, Brox (2015)**

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



U-Net architecture; cf. **Ronneberger, Fischer, Brox (2015)**

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. (2021).**
- **Neural operators: Kovachki, Li, Liu, Azizzadenesheli, Bhattacharya, Stuart, and Anandkumar (arXiv preprints 2020, 2021).**

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

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

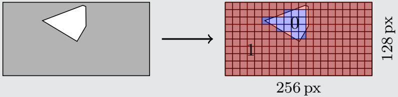
where $\vec{u}$ and p are the velocity and pressure fields and ν is the viscosity. Furthermore, we prescribe the previously described boundary conditions.

Software pipeline

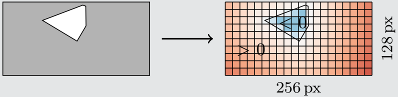
1. Define the boundary of the polygonal obstacle and **create the corresponding STL (standard triangulation language) file**.
2. **Generate a hexahedral compute grid** (`snappyHexMesh`).
3. Run the **CFD simulation** (`simpleFoam`).
4. **Interpolate geometry information and flow field** onto a pixel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

Input data

Binary

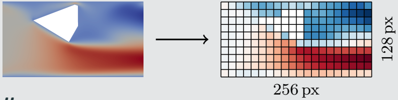


SDF (Signed Distance Function)

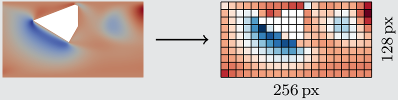


Output data

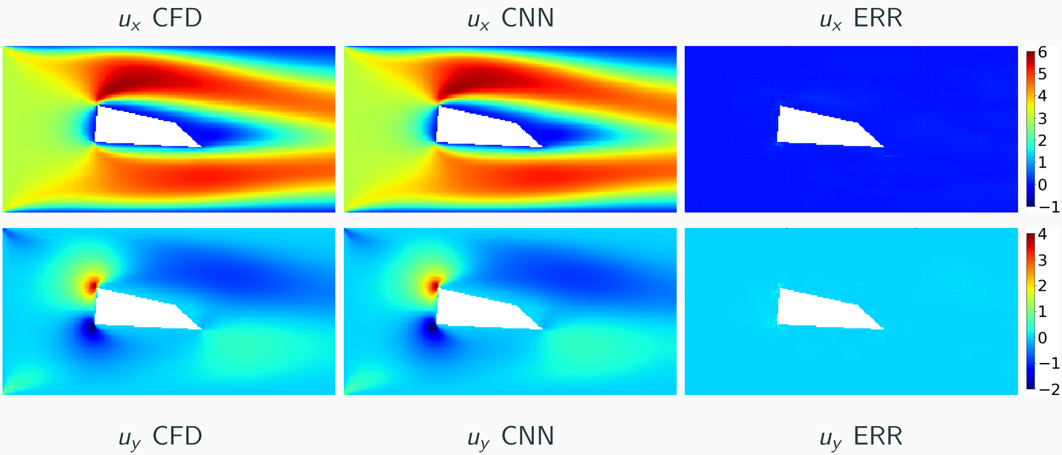
u_x



u_y

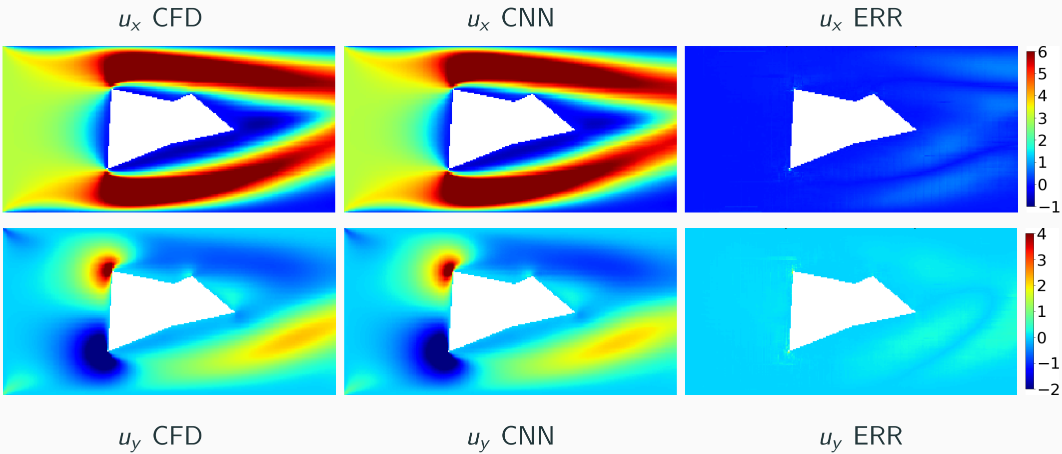


Comparison CFD Vs NN (Relative Error 2 %)



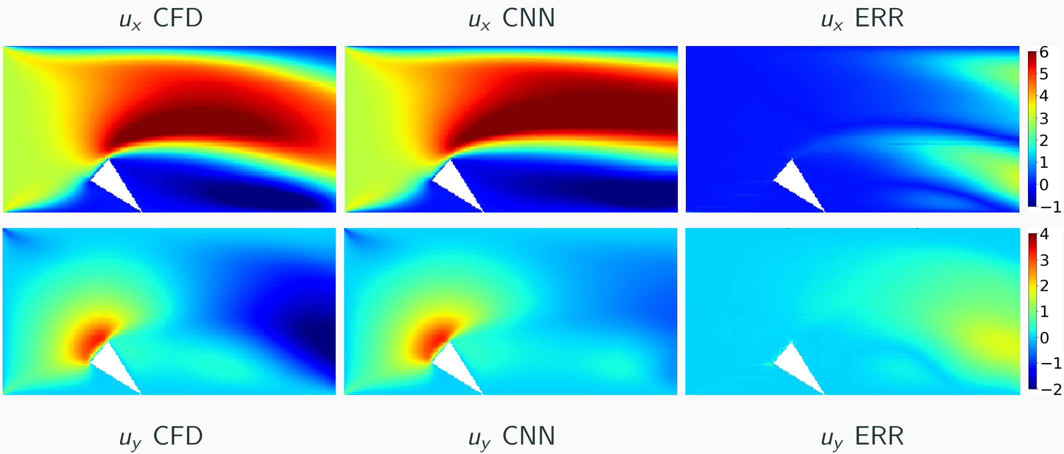
Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**

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)**

Computing Times

Data:

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

Training:

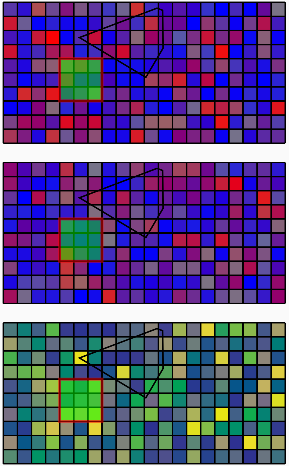
	Bottleneck CNN		U-Net	
# decoders	1	2	1	2
# parameters	$\approx$ 47 m	$\approx$ 85 m	$\approx$ 34 m	$\approx$ 53.5 m
time/epoch	**180 s**	**245 s**	**195 s**	**270 s**

Comparison CFD Vs NN:

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

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

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



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

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

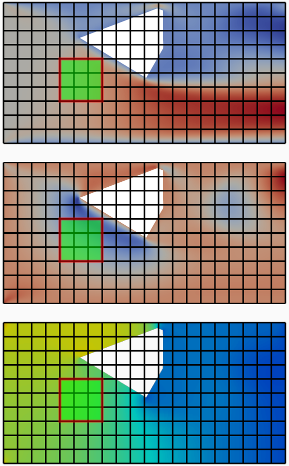
where u_{NN} and p_{NN} are the output images of our CNN and

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

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:

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

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



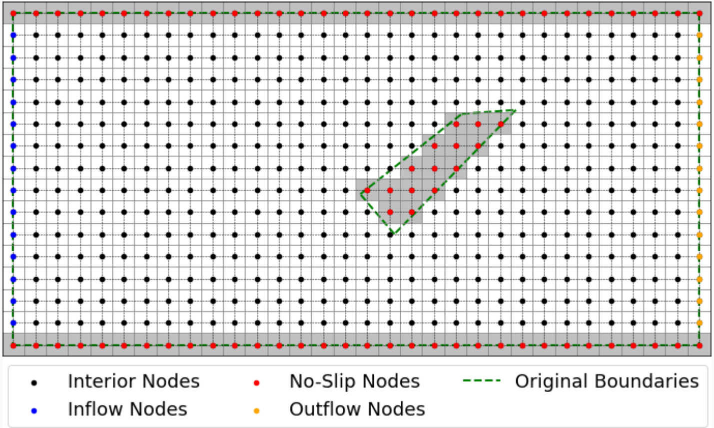
$$\left\| \begin{matrix} F_{\text{mom}}(u_{NN}, p_{NN}) \\ F_{\text{mass}}(u_{NN}, p_{NN}) \end{matrix} \right\|^2 >> 0 \qquad \left\| \begin{matrix} F_{\text{mom}}(u_{NN}, p_{NN}) \\ F_{\text{mass}}(u_{NN}, p_{NN}) \end{matrix} \right\|^2 \approx 0$$

Cf. **Grimm, Heinlein, Klawonn**

Physics-Informed Approach & Boundary Conditions

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

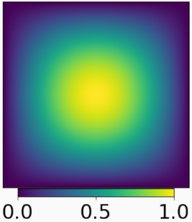
→ On a single geometry, this training of the neural network just corresponds to an **unconventional way of solving the Navier-Stokes equations using finite differences**.



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**.

Convergence Comparison – CNN Versus FDM



Solve

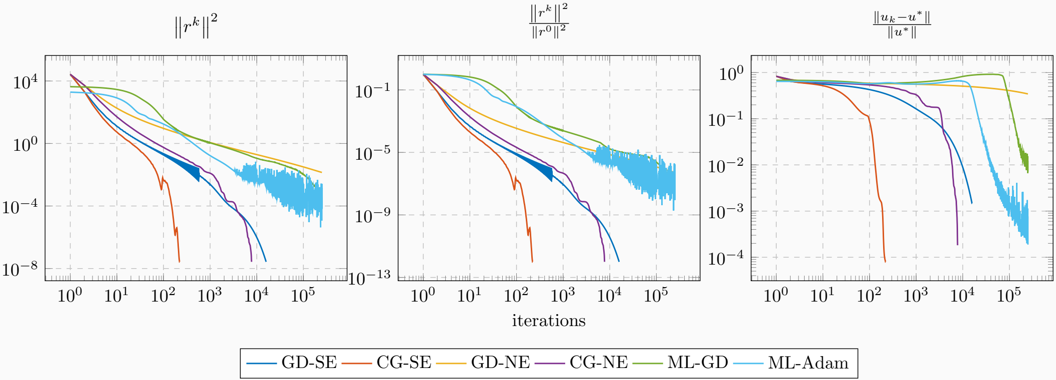
$$-\Delta u = f$$

using

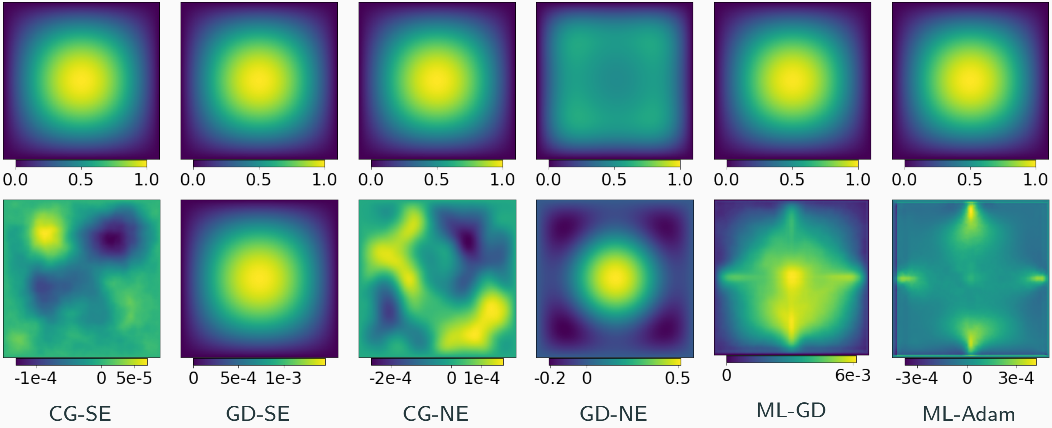
- classical finite differences
- **ML**: CNN

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

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



Convergence Comparison – CNN Versus FDM



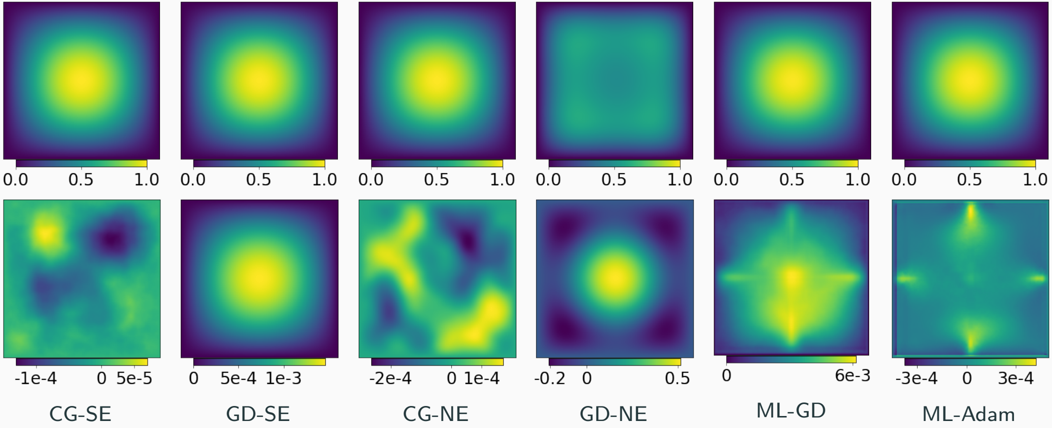
The results are in alignment with the **spectral bias of neural networks**. The neural network approximations yield a low error norm compared with the residual (MSE loss).

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

Cf. **Grimm, Heinlein, Klawonn (2024)**.

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

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 (2024)**.

$\rightarrow$ **Next:** surrogate model for multiple geometries

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

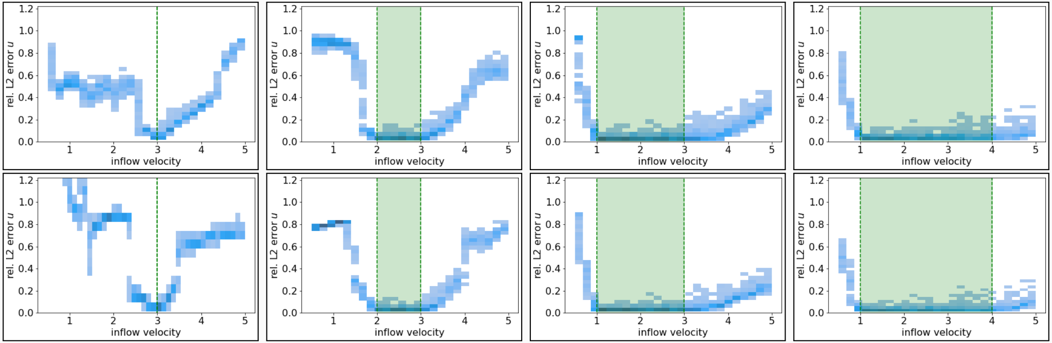
	training data	error	$\frac{\lVert u_{NN}-u \rVert_2}{\lVert u \rVert_2}$	$\frac{\lVert p_{NN}-p \rVert_2}{\lVert p \rVert_2}$	mean residual momentum	mean residual mass	# epochs trained
data-based	10%	train.	2.07%	10.98%	$1.1 \cdot 10^{-1}$	$1.4 \cdot 10^{0}$	500
		val.	4.48 %	15.20 %	$1.6 \cdot 10^{-1}$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

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

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

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

Generalization With Respect to the Inflow Velocity

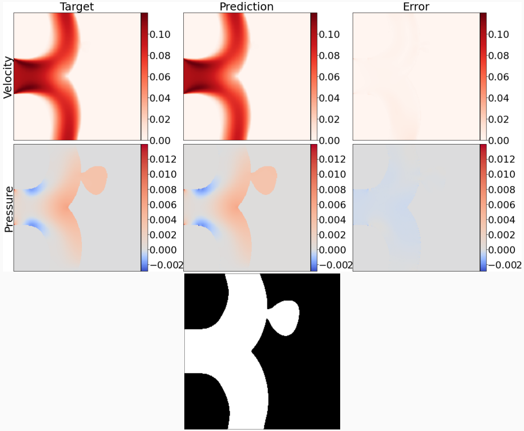


order	# data	range inflow vel.	[0.5, 1.0]	[1.0, 2.0]	[2.0, 3.0]	[3.0, 4.0]	[4.0, 5.0]
2	1 000	[3.0, 3.0]	55.5 %	48.1 %	31.1 %	17.4 %	61.5 %
		[2.0, 3.0]	89.3 %	57.4 %	**4.0 %**	15.5 %	59.1 %
		[1.0, 3.0]	40.2 %	**3.8 %**	**4.3 %**	7.1 %	20.4 %
		[1.0, 4.0]	31.3 %	**4.0 %**	**4.3 %**	**5.8 %**	7.7 %
2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

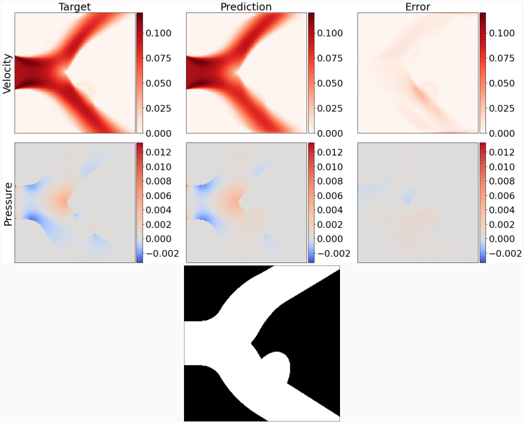
Training: 500 geometries **Validation:** $\approx 1\,200$ geometries

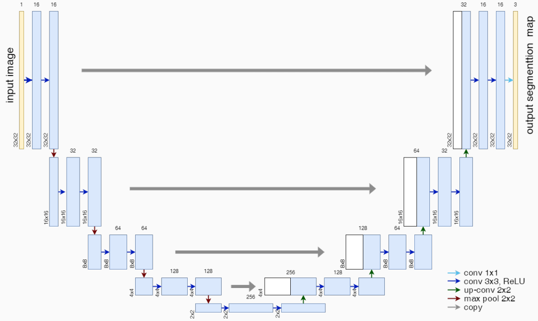
Relative $L2$**-error** on the **validation data set** in u: **4.9** %, in p: **9.5** %.

u: 3.3%, p: 7.4%



u: 13.2%, p: 35.1%



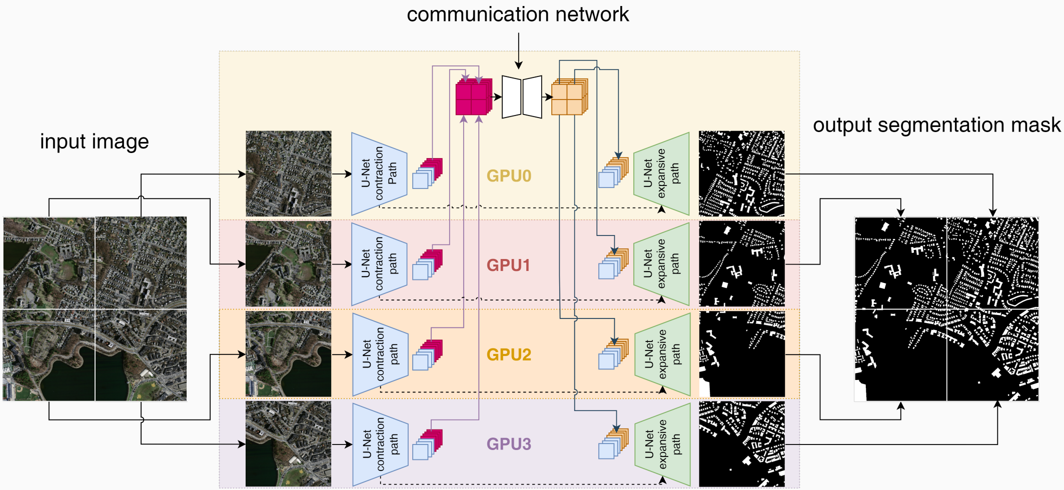


- As an example for a **convolutional neural network (CNN)**, we employ the **U-Net architecture** introduced in **Ronneberger, Fischer, and Brox (2015)**.
- The U-Net yields **state-of-the-art accuracy in semantic image segmentation** and other **image-to-image tasks**.

Below: *memory consumption for training on a single* 1024×1024 *image.*

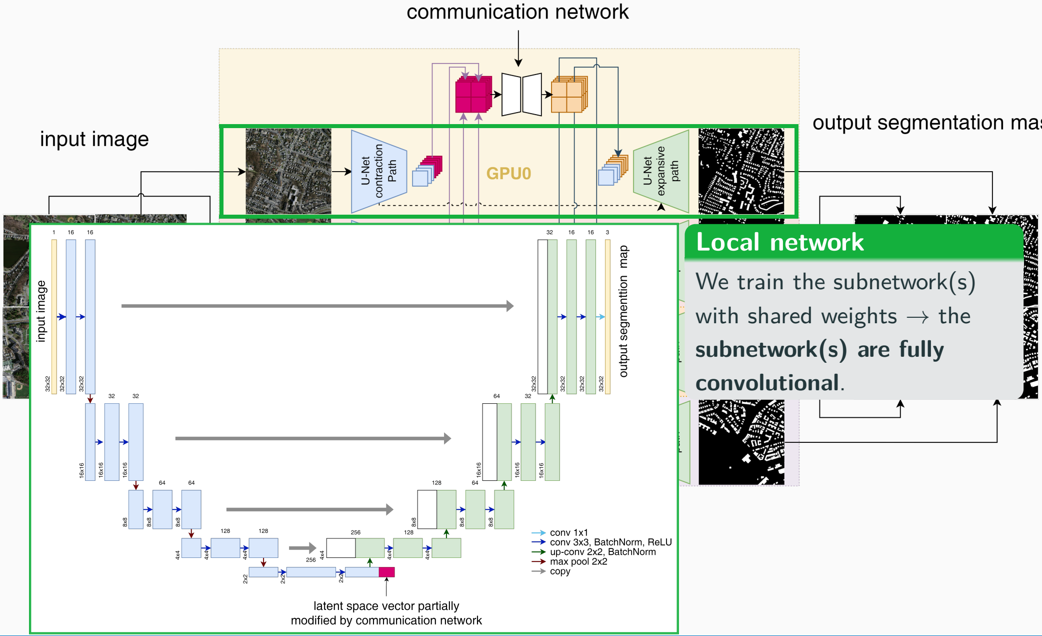
name	size	# channels		mem. feature maps		mem. weights	
		input	output	# of values	MB	# of values	MB
input block	1 024	3	64	268 M	1 024.0	38 848	0.148
encoder block 1	512	64	128	167 M	704.0	221 696	0.846
encoder block 2	256	128	256	84 M	352.0	885 760	3.379
encoder block 3	128	256	512	42 M	176.0	3 540 992	13.508
encoder block 4	64	512	1 024	21 M	88.0	14 159 872	54.016
decoder block 1	64	1,024	512	50 M	192.0	9 177 088	35.008
decoder block 2	128	512	256	101 M	384.0	2 294 784	8.754
decoder block 3	256	256	128	201 M	768.0	573 952	2.189
decoder block 4	512	128	64	402 M	1 536.0	143 616	0.548
output block	1 024	64	3	3.1 M	12.0	195	0.001

Decomposing the U-Net



Cf. **Verburg, Heinlein, Cyr (subm. 2024).**

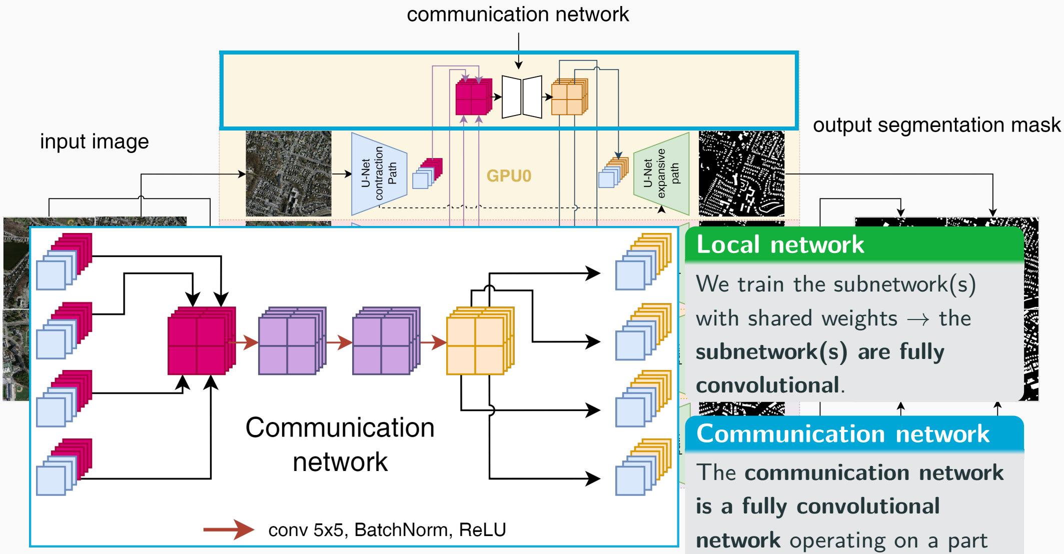
Decomposing the U-Net



Local network

We train the subnetwork(s) with shared weights $\rightarrow$ the **subnetwork(s) are fully convolutional**.

Decomposing the U-Net



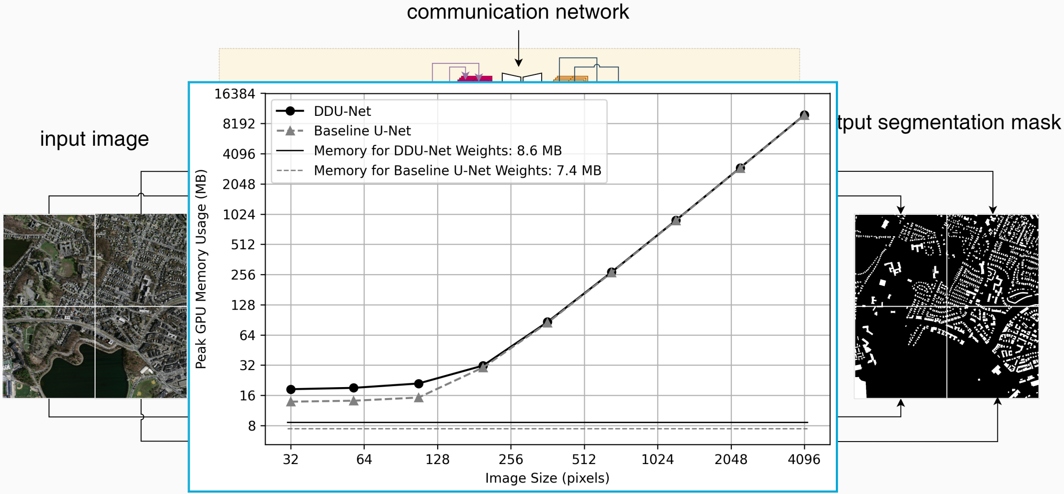
Local network

We train the subnetwork(s) with shared weights → the **subnetwork(s) are fully convolutional**.

Communication network

The **communication network is a fully convolutional network** operating on a part of the coarse feature maps.

Decomposing the U-Net



- Distribution of feature maps results in **significant reduction of memory usage on a single GPU**
- **Moderate additional memory usage** due to the **communication network**

Domain decomposition-based deep operator networks

Domain Decomposition Methods and Machine Learning – Literature

A **non-exhaustive literature overview**:

- **Machine Learning for adaptive BDDC, FETI–DP, and AGDSW**: **Heinlein, Klawonn, Lanser, Weber (2019, 2020, 2021, 2021, 2021, 2022); Klawonn, Lanser, Weber (2024)**
- **cPINNs, XPINNs**: **Jagtap, Kharazmi, Karniadakis (2020); Jagtap, Karniadakis (2020)**
- **Classical Schwarz iteration for PINNs or DeepRitz (D3M, DeepDDM, etc)**:: **Li, Tang, Wu, and Liao (2019); Li, Xiang, Xu (2020); Mercier, Gratton, Boudier (arXiv 2021); Dolean, Heinlein, Mercier, Gratton (subm. 2024 / arXiv:2408.12198); Li, Wang, Cui, Xiang, Xu (2023); Sun, Xu, Yi (arXiv 2023, 2024); Kim, Yang (2023, 2024, 2024)**
- **FBPINNs**, **FBKANs**: **Moseley, Markham, and Nissen-Meyer (2023); Dolean, Heinlein, Mishra, Moseley (2024, 2024); Heinlein, Howard, Beecroft, Stinis (acc. 2024 / arXiv:2401.07888); Howard, Jacob, Murphy, Heinlein, Stinis (arXiv:2406.19662)**
- **DDMs for CNNs**: **Gu, Zhang, Liu, Cai (2022); Lee, Park, Lee (2022); Klawonn, Lanser, Weber (2024); Verburg, Heinlein, Cyr (subm. 2024)**

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.

An overview of the state-of-the-art in mid 2024:

A. Klawonn, M. Lanser, J. Weber

Machine learning and domain decomposition methods – a survey

Computational Science and Engineering. 2024

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**

$$\mathcal{N}[u] = f, \quad \text{in } \Omega.$$

PINNs use a **hybrid loss function**:

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

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

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

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

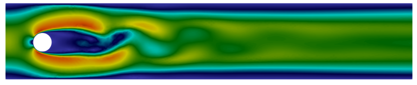
See also **Dissanayake and Phan-Thien (1994); Lagaris et al. (1998)**.

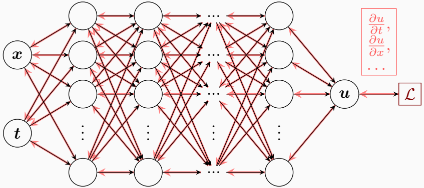
Advantages

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

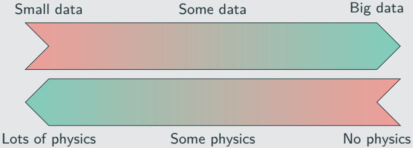
Drawbacks

- **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 $\mathcal{L}_{\text{data}}$

Theoretical Result for PINNs

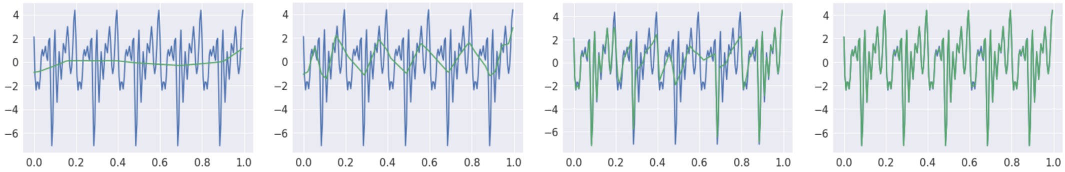
Estimate of the generalization error (Mishra and Molinaro (2022))

The generalization error (or total error) satisfies

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

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

Rule of thumb: **"As long as the PINN is trained well, it also generalizes well"**



100 iterations | 1 000 iterations | 10 000 iterations | 80 000 iterations

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

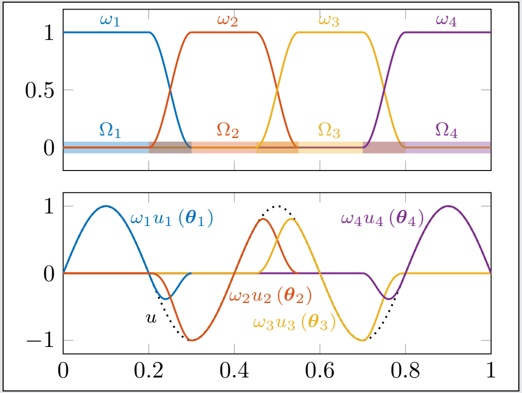
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

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

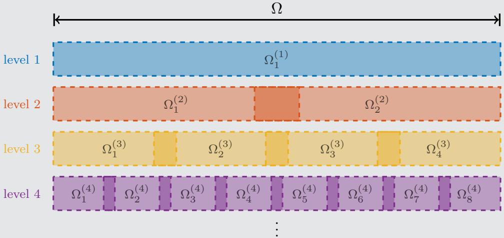
and the **loss function**

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



Multi-level FBPINNs (ML-FBPINNs)

ML-FBPINNs (**Dolean, Heinlein, Mishra, Moseley (2024)**) are based on a **hierarchy of domain decompositions:**



This yields the **network architecture**

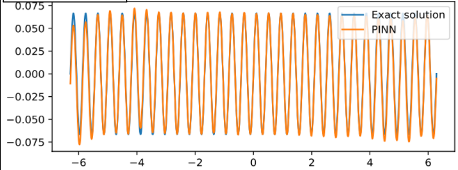
$$u\left(\theta_1^{(1)}, \ldots, \theta_{J^{(L)}}^{(L)}\right) = \sum_{l=1}^{L} \sum_{i=1}^{N^{(l)}} \omega_j^{(l)} u_j^{(l)}\left(\theta_j^{(l)}\right)$$

and the **loss function**

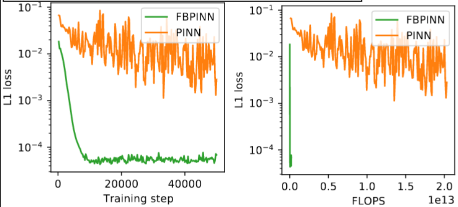
$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(n[\sum_{x_i \in \Omega_j^{(l)}} \omega_j^{(l)} u_j^{(l)}](x_i, \theta_j^{(l)}) - f(x_i) \right)^2.$$

1D single-frequency problem

PINN solution



Moseley, Markham, Nissen-Meyer (2023)



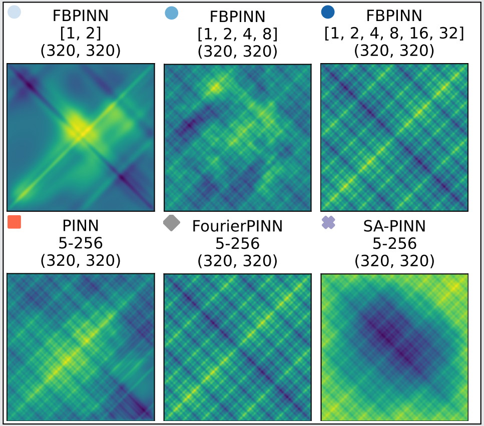
2D multi-frequency problem

Consider

$$-\Delta u = 2\sum_{i=1}^{6} (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y) \quad \text{in } \Omega,$$

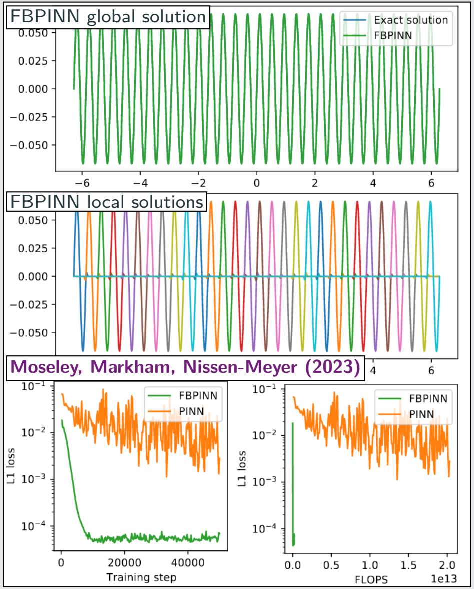
$$u = 0 \quad \text{on } \partial\Omega,$$

with $\omega_i = 2^i$.



PINN vs FBPINN

1D single-frequency problem



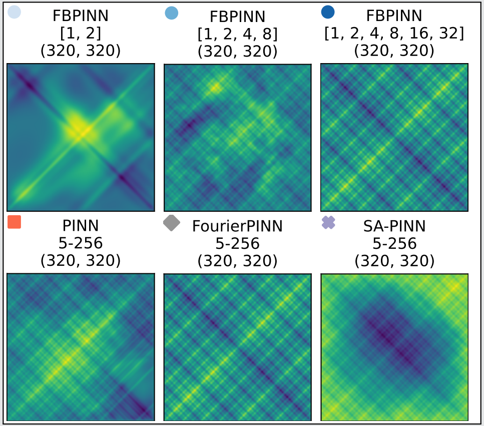
2D multi-frequency problem

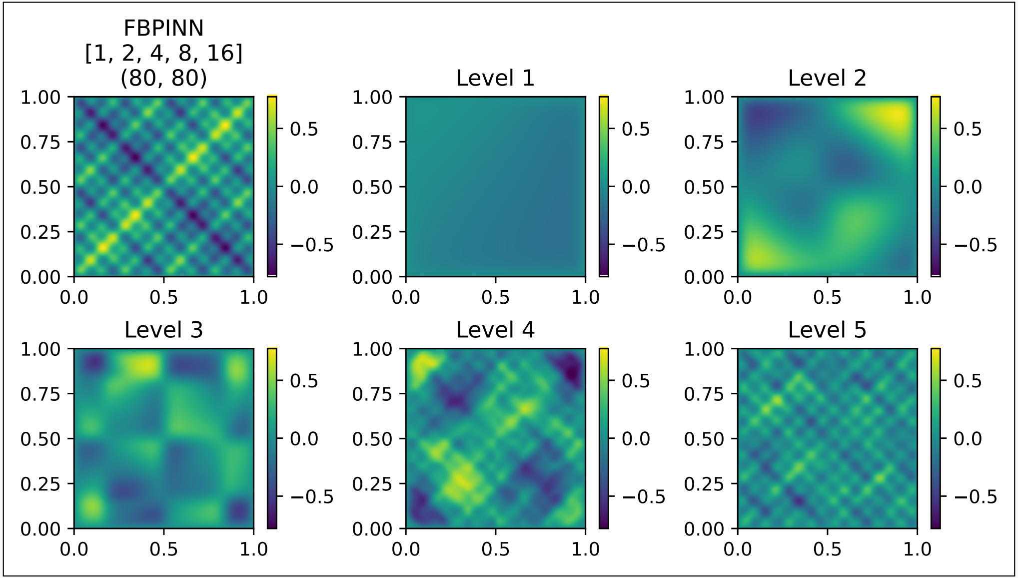
Consider

$$-\Delta u = 2\sum_{i=1}^{6} (\omega_i \pi)^2 \sin(\omega_i \pi x)\sin(\omega_i \pi y) \quad \text{in } \Omega,$$

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

with $\omega_i = 2^i$.





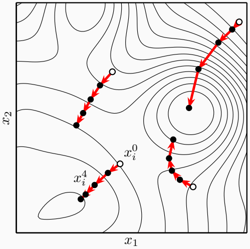
Cf. **Dolean, Heinlein, Mishra, Moseley (2024)**.

PACMANN – Point Adaptive Collocation Method for Artificial Neural Networks

In **Visser, Heinlein, and Giovanardi (arXiv:2411.19632)**, are adapted by solving the **min-max problem**

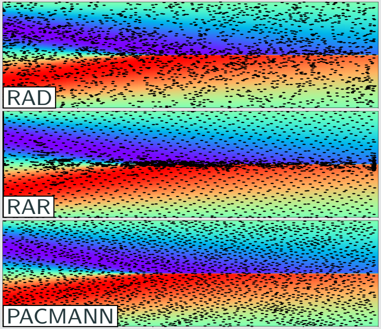
$$\min_{\boldsymbol{\theta}} \left[\omega_{\text{data}} \mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \max_{\boldsymbol{X} \subset \mathcal{D}} \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}}(\boldsymbol{X}, \boldsymbol{\theta}) \right].$$

Different from other residual-based adaptive sampling methods, such as **residual-based adaptive refinement (RAR)** and **residual-based adaptive distribution (RAD)**, in PACMANN, the **existing collocation are moved** using a gradient-based optimizer.



Burger's equation

Sampling method	L_2 relative error		Mean runtime [s]
	Mean	1 SD	
Uniform grid	25.9%	14.2%	425
Hammersley grid	0.61%	0.53%	443
Random resampling	0.40%	0.35%	**423**
RAR	0.11%	**0.05%**	450
RAD	0.16%	0.10%	463
RAR-D	0.24%	0.21%	503
PACMANN–Adam	**0.07%**	**0.05%**	461

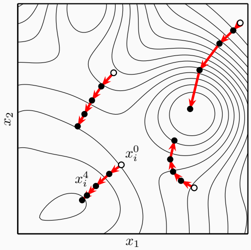


PACMANN – Point Adaptive Collocation Method for Artificial Neural Networks

In **Visser, Heinlein, and Giovanardi (arXiv:2411.19632)**, are adapted by solving the **min-max problem**

$$\min_{\boldsymbol{\theta}} \left[\omega_{\text{data}} \mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \max_{\boldsymbol{X} \subset \mathcal{D}} \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}}(\boldsymbol{X}, \boldsymbol{\theta}) \right].$$

Different from other residual-based adaptive sampling methods, such as **residual-based adaptive refinement (RAR)** and **residual-based adaptive distribution (RAD)**, in PACMANN, the **existing collocation are moved** using a gradient-based optimizer.



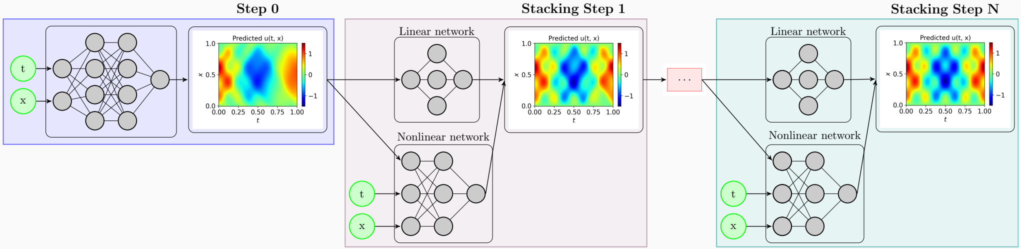
5D Poisson equation

Sampling method	L_2 relative error		Mean runtime [s]
	Mean	1 SD	
Uniform grid	17.89%	0.94%	742
Hammersley grid	82.08%	3.23%	**734**
Random resampling	11.03%	0.69%	772
RAR	56.84%	4.46%	753
RAD	10.07%	0.75%	851
RAR-D	88.30%	1.53%	774
*Adam	**5.93%**	**0.46%**	778

Stacking Multifidelity PINNs

In the **stacking multifidelity PINNs approach** introduced in **Howard, Murphy, Ahmed, Stinis (arXiv 2023)**, **multiple PINNs are trained in a recursive way.** In each step, a model u^{MF} is trained based on the previous model u^{SF}:

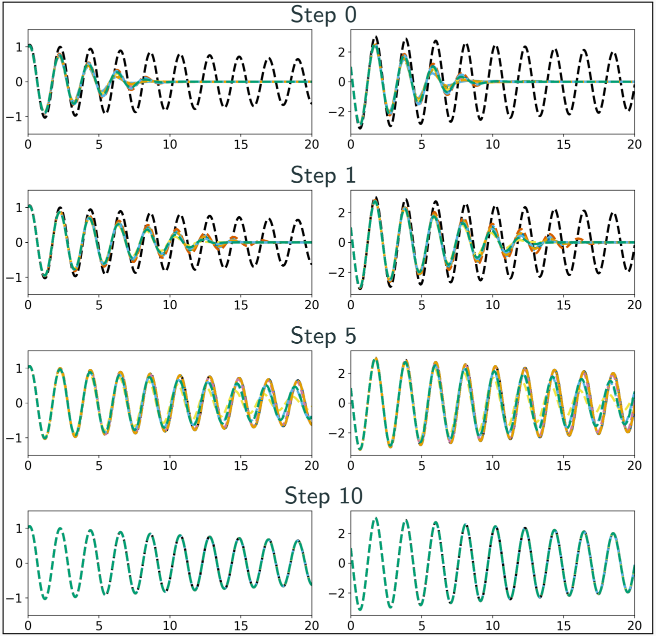
$$u^{MF}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}) = (1 - |\alpha|)\, u^{MF}_{\mathrm{linear}}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}, u^{SF}) + |\alpha|\, u^{MF}_{\mathrm{nonlinear}}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}, u^{SF})$$



Related works (non-exhaustive list)

- **Cokriging & multifidelity Gaussian process regression:** E.g., **Wackernagel (1995); Perdikaris et al. (2017); Babaee et al. (2020)**
- **Multifidelity PINNs & DeepONet: Meng and Karniadakis (2020); Howard, Fu, and Stinis (2024); Howard, Perego, Karniadakis, Stinis (2023); Howard, Murphy, Ahmed, Stinis (arXiv 2023)**
- **Galerkin, multi-level, and multi-stage neural networks: Ainsworth and Dong (2021); Ainsworth and Dong (2022); Aldirany et al. (2024); Wang and Lai (2024)**

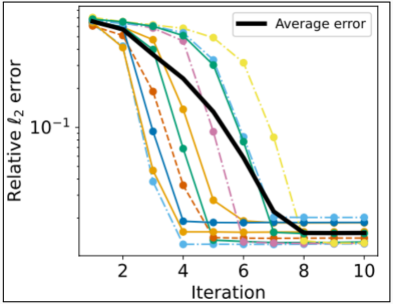
Stacking Multifidelity PINNs for the Pendulum Problem



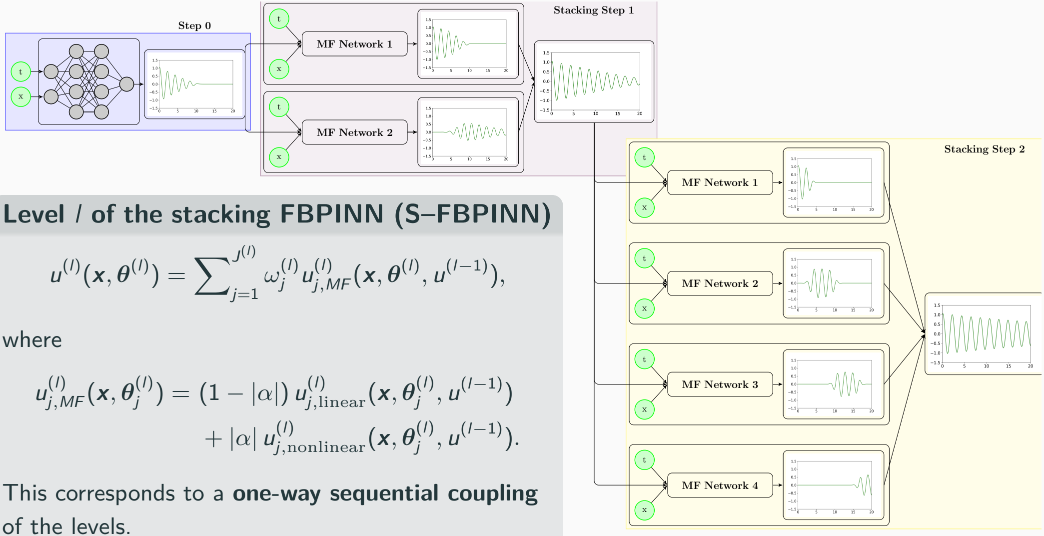
Pendulum problem:

$$\frac{ds_1}{dt} = s_2,$$
$$\frac{ds_2}{dt} = -\frac{b}{m}s_2 - \frac{g}{L}\sin(s_1).$$

with $m = L = 1$, $b = 0.05$, $g = 9.81$, and $T = 20$.



In **Heinlein, Howard, Beecroft, and Stinis (acc. 2024 / arXiv:2401.07888)**, we **combine stacking multifidelity PINNs with FBPINNs** by using an FBPINN model in each stacking step.



Level l of the stacking FBPINN (S–FBPINN)

$$u^{(l)}(\boldsymbol{x},\boldsymbol{\theta}^{(l)}) = \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} u_{j,MF}^{(l)}(\boldsymbol{x},\boldsymbol{\theta}^{(l)}, u^{(l-1)}),$$

where

$$u_{j,MF}^{(l)}(\boldsymbol{x},\boldsymbol{\theta}_j^{(l)}) = (1-|\alpha|)\, u_{j,\mathrm{linear}}^{(l)}(\boldsymbol{x},\boldsymbol{\theta}_j^{(l)}, u^{(l-1)}) + |\alpha|\, u_{j,\mathrm{nonlinear}}^{(l)}(\boldsymbol{x},\boldsymbol{\theta}_j^{(l)}, u^{(l-1)}).$$

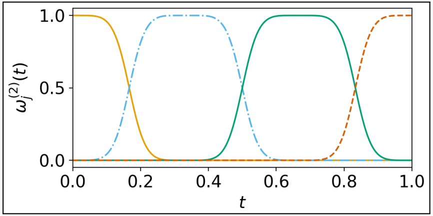
This corresponds to a **one-way sequential coupling** of the levels.

Numerical Results – Pendulum Problem

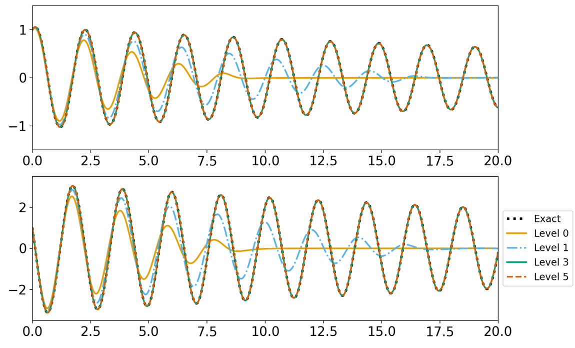
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

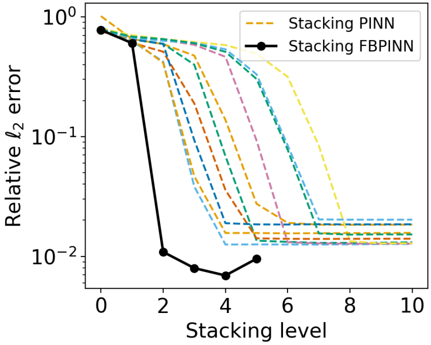
$$\frac{ds_1}{dt} = s_2,$$
$$\frac{ds_2}{dt} = -\frac{b}{m} s_2 - \frac{g}{L}\sin(s_1)$$

with $m = L = 1$, $b = 0.05$, $g = 9.81$, and $T = 20$.



Exemplary partition of unity in time





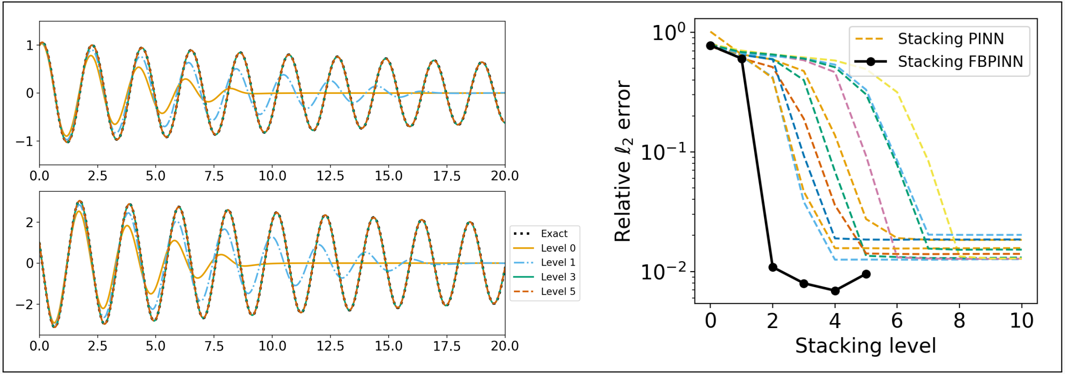
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

$$\frac{d\vartheta_1}{dt} = \vartheta_2,$$
$$\frac{d\vartheta_2}{dt} = -\frac{b}{m}\vartheta_2 - \frac{g}{L}\sin(\vartheta_1)$$

with $m = L = 1$, $b = 0.05$, $g = 9.81$, and $T = 20$.

Model details:

method	arch.	# levels	# params	error
S–PINN	5x50, 1x20	4	63 018	0.0125
S–FBPINN	3x32, 1x 4	2	34 570	0.0074



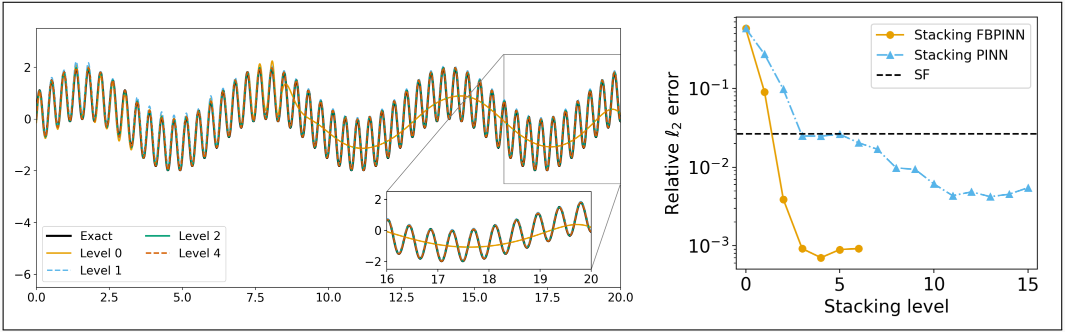
Numerical Results – Two-Frequency Problem

Second, we consider a **two-frequency problem**:

$$\frac{ds}{dx} = \omega_1 \cos(\omega_1 x) + \omega_2 \cos(\omega_2 x),$$
$$s(0) = 0,$$

on domain $\Omega = [0, 20]$ with $\omega_1 = 1$ and $\omega_2 = 15$.

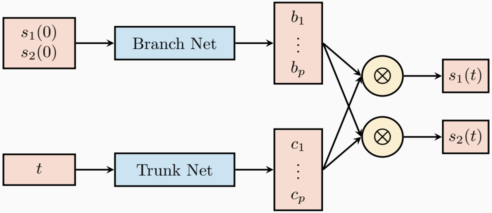
method	arch.	# levels	# params	error
PINN	4x64	0	12 673	0.6543
PINN	5x64	0	16 833	0.0265
S–PINN	4x16, 1x5	3	4900	0.0249
S–PINN	4x16, 1x5	10	11 179	0.0061
S–FBPINN	4x16, 1x5	2	7822	0.00415
S–FBPINN	4x16, 1x5	5	59 902	0.00083



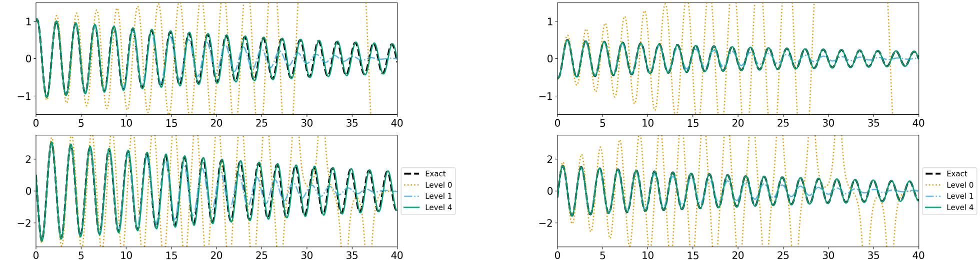
→ Due to the **multiscale structure of the problem**, the **improvements** due to the **multifidelity FBPINN approach** are **even stronger**.

DeepONets (Lu et al. (2021))

- While PINNs learn individual solutions, neural operators learn operators between function spaces, such as **solution operators**
- **Deep operator networks (DeepONets)** are **compatible with the PINN approach** but **physics-informed DeepONets (PI-DONs)** are challenging to train



Approach based on the **single-layer case** analyzed in **Chen and Chen (1995)**



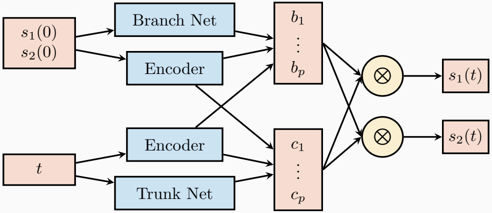
$s_1(0) = 1, s_2(0) = 1$ $\qquad$ $s_1(0) = -0.5, s_2(0) = -0.4$

Cf. **Heinlein, Howard, Beecroft, and Stinis (acc. 2024)**.

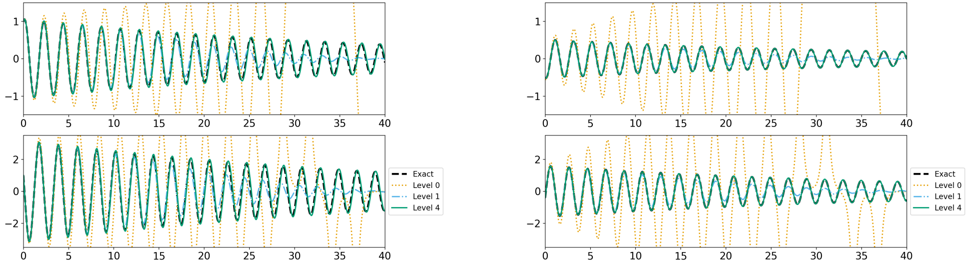
Deep Operator Networks (DeepONets / DONs)

DeepONets (Lu et al. (2021))

- While PINNs learn individual solutions, neural operators learn operators between function spaces, such as **solution operators**
- **Deep operator networks (DeepONets)** are **compatible with the PINN approach** but **physics-informed DeepONets (PI-DONs)** are challenging to train



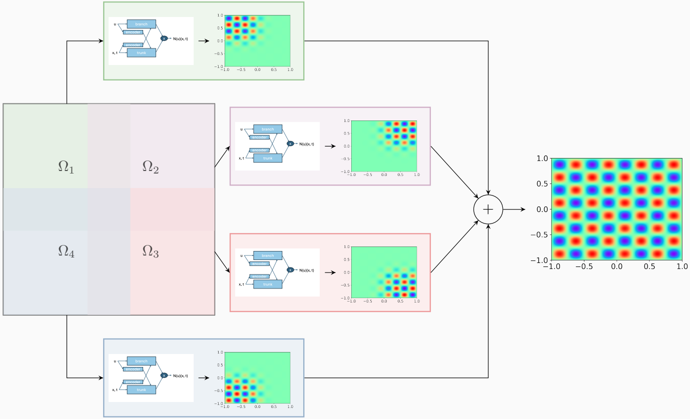
Modified DeepONet architecture; cf. **Wang, Wang, and Perdikaris (2022)**



$s_1(0) = 1, s_2(0) = 1$ $\qquad$ $s_1(0) = -0.5, s_2(0) = -0.4$

Cf. **Heinlein, Howard, Beecroft, and Stinis (acc. 2024)**.

Finite Basis DeepONets (FBDONs)



Howard, Heinlein, Stinis (in prep.)

Variants:

Shared-trunk FBDONs (ST-FBDONs)

The trunk net learns spatio-temporal basis functions. In ST-FBDONs, we use the same trunk network for all subdomains.

Stacking FBDONs

Combination of the stacking multifidelity approach with FBDONs.

Heinlein, Howard, Beecroft, Stinis (acc. 2024/arXiv:2401.07888)

DD-DONs Pendulum

Pendulum problem

$$\frac{ds_1}{dt} = s_2, \qquad t \in [0, T],$$
$$\frac{ds_2}{dt} = -\frac{b}{m}s_2 - \frac{g}{L}\sin(s_1), \qquad t \in [0, T],$$

where $m = L = 1$, $b = 0.05$, $g = 9.81$, and $T = 20$.

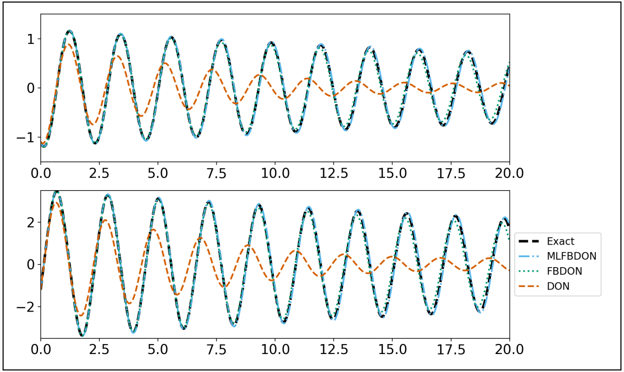
Parametrization

Initial conditions:

$$s_1(0) \in [-2, 2] \qquad s_2(0) \in [-1.2, 1.2]$$

$s_1(0)$ and $s_2(0)$ are the also inputs of the branch network.

Training on 50 k different configurations



Mean rel. l_2 error on 100 config.	
DeepONet	0.94
FBDON (32 subd.)	0.84
MLFBDON ([1, 4, 8, 16, 32] subd.)	0.27

Cf. **Howard, Heinlein, Stinis (in prep.)**

DD-DONs Wave Equation

Wave equation

$$\frac{d^2 s}{dt^2} = 2\frac{d^2 s}{dx^2}, \qquad (x,t) \in [0,1]^2$$

$$s_t(x,0) = 0, x \in [0,1], \quad s(0,t) = s(1,t) = 0,$$

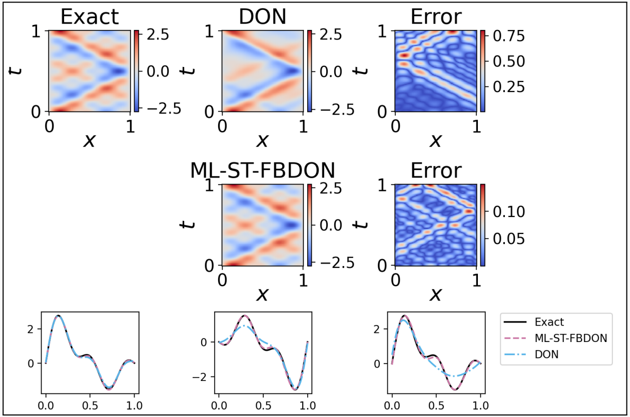
Solution: $s(x,t) = \sum_{n=1}^{5} b_n \sin(n\pi x)\cos(n\pi\sqrt{2}t)$

Parametrization

Initial conditions for s parametrized by $b = (b_1, \ldots, b_5)$ (normally distributed):

$$s(x,0) = \sum_{n=1}^{5} b_n \sin(n\pi x) \quad x \in [0,1]$$

Training on 1000 random configurations.



Mean rel. l_2 error on 100 config.	
DeepONet	0.30 ± 0.11
ML-ST-FBDON ([1, 4, 8, 16] subd.)	0.05 ± 0.03
ML-FBDON ([1, 4, 8, 16] subd.)	0.08 ± 0.04

→ Sharing the trunk network does not only save in the number of parameters but even yields better performance

Cf. **Howard, Heinlein, Stinis (in prep.)**

Summary

Surrogate models for varying computational domains

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

Domain decomposition-based deep operator networks

- **Domain decomposition methods** can help to **improve the performance of PINNs and neural operators**, especially for (but not restricted to) **large domains** and **multiscale problems**.

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!