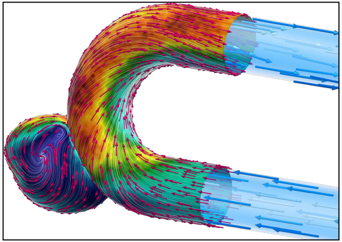
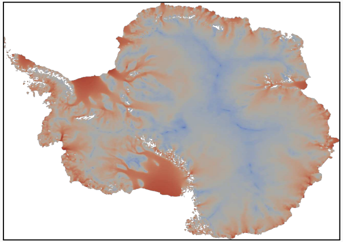
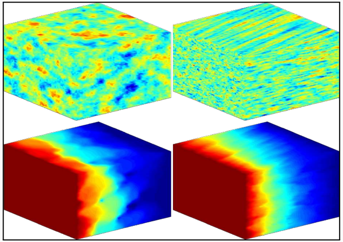
Towards Physics-Informed Machine Learning-Based Surrogate Models for Challenging Problems

Alexander Heinlein[1]

Siemens Simulation and Digital Twin Talks (Virtual), April 28, 2025

[1]Delft University of Technology

Scientific Computing 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

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)
Matthias Eichinger, **Viktor Grimm**, **Axel Klawonn**	(University of Cologne)
Corné Verburg	(Delft University of Technology)

2 Domain decomposition-based neural networks and operators

Based on joint work with

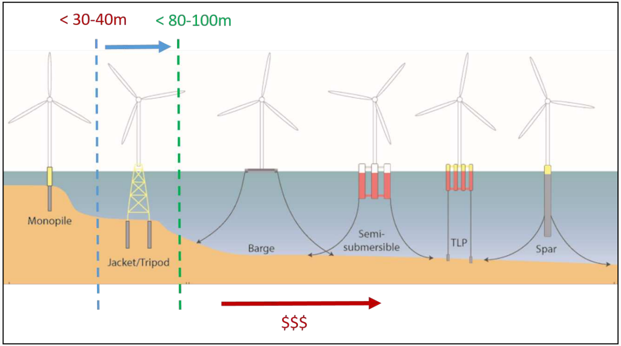
Damien Beecroft	(University of Washington)
Victorita Dolean	(Eindhoven University of Technology)
Bianca Giovanardi, **Coen Visser**	(Delft University of Technology)
Amanda A. Howard and **Panos Stinis**	(Pacific Northwest National Laboratory)
Siddhartha Mishra	(ETH Zürich)
Ben Moseley	(Imperial College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

Surrogate models for varying computational domains

Perforated monopiles

Monopiles are the **most used and cheapest** solution of support structures in **offshore wind energy**.
$\rightarrow$ **Perforated monopiles reduce the wave load**.

What is the **optimal perforation shape**?

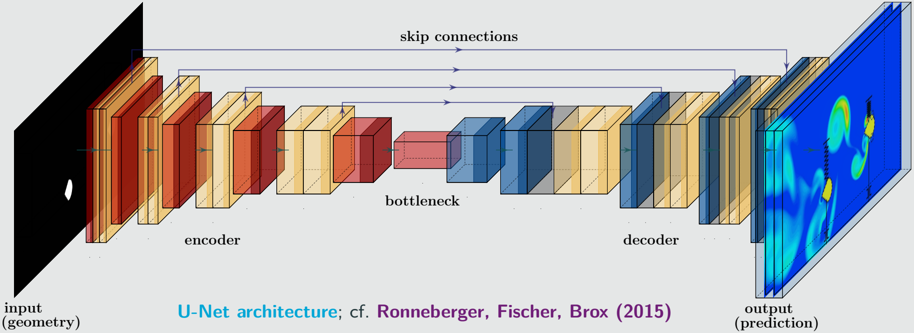


Fully resolved CFD simulations are **costly**, but **rough predictions** may be **sufficient**.

Convolutional Neural Network-Based Surrogate Model

CNN-based approach

We employ a **convolutional neural network (CNN)** (**LeCun (1998)**) to predict the stationary flow field, given an **image of the geometry as input**.



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

Related works (non-exhaustive)

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

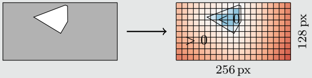
Operator learning (non-exhaustive)

- FNOs: **Li et al. (2021)**
- PCA-Net: **Bhattacharya et al. (2021)**
- Random features: **Nelsen and Stuart (2021)**
- CNOs: **Raonić et al. (2023)**

We automatically generate geometries and compute the corresponding flow fields using OpenFOAM®.

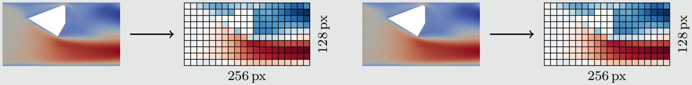
Input data

SDF (Signed Distance Function)

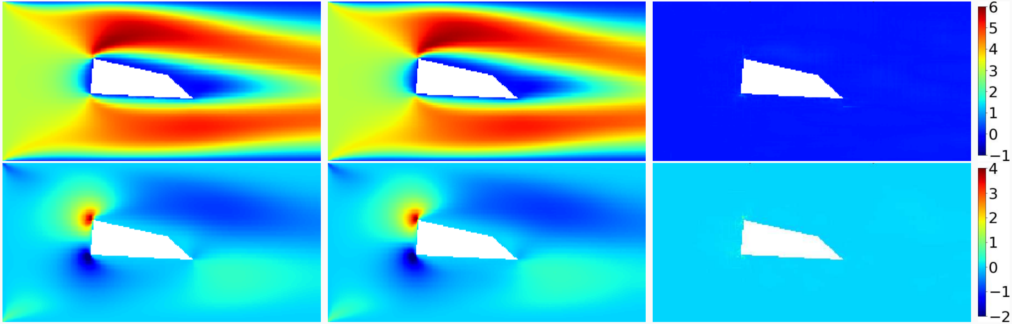


Output data

u_x u_x



u_x CFD u_x CNN u_x ERR



u_y CFD u_y CNN u_y ERR

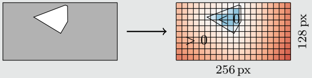
Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**.

Comparison OpenFOAM® Versus CNN (Relative Error 14 %)

We automatically generate geometries and compute the corresponding flow fields using OpenFOAM®.

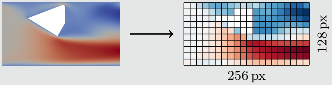
Input data

SDF (Signed Distance Function)

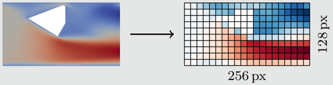


Output data

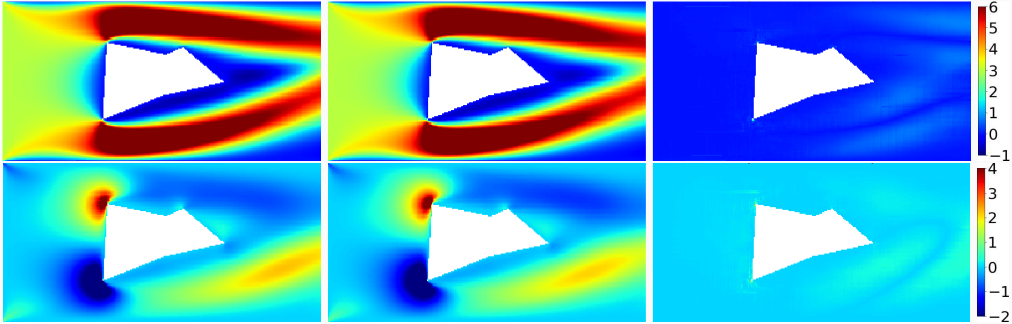
u_x



u_x



u_x CFD | u_x CNN | u_x ERR



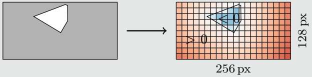
u_y CFD | u_y CNN | u_y ERR

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

We automatically generate geometries and compute the corresponding flow fields using OpenFOAM®.

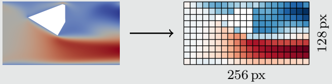
Input data

SDF (Signed Distance Function)

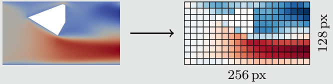


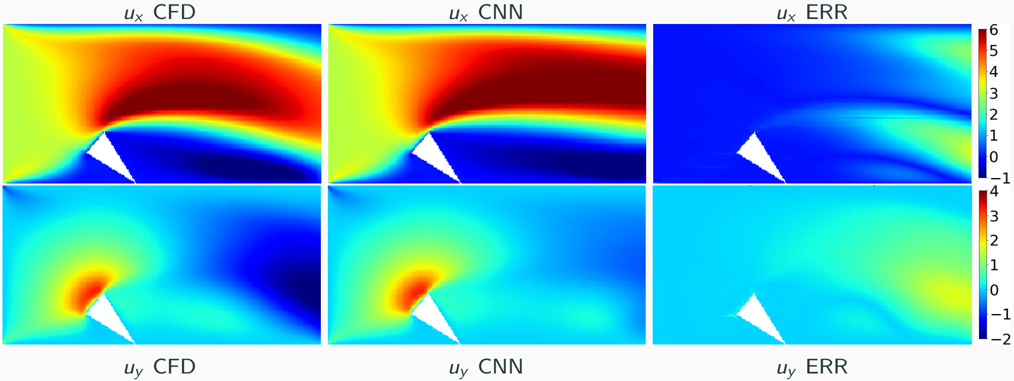
Output data

u_x



u_x





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:

	U-Net	
# decoders	1	2
# parameters	$\approx$ 34 m	$\approx$ 53.5 m
time/epoch	**195 s**	**270 s**

Comparison CFD Vs NN:

	CFD	**U-Net**	
	CPU	CPU	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**.

Unsupervised Learning Approach – PDE Loss Using Finite Differences

Physics-informed loss function

We train the CNN by incorporating the **squared PDE residuals** into the **loss function**:

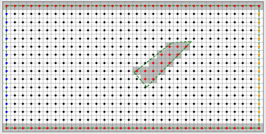
$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \|\mathcal{R}(u_{\text{CNN}}, p_{\text{CNN}})\|^2$$

Here, N_{PDE} is the number of training configs.

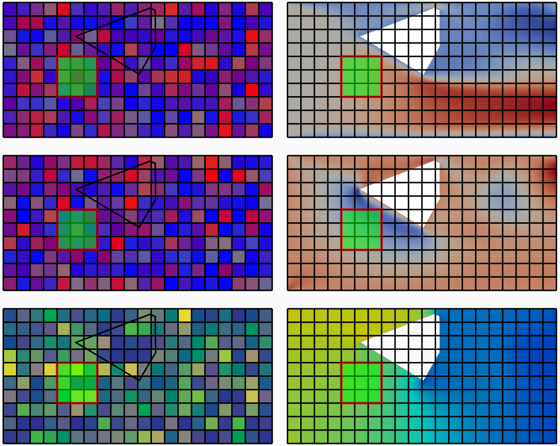
Cf. **Raissi et al. (2019)**, **Dissanayake and Phan-Thien (1994)**, **Lagaris et al. (1998)**.

We discretize the differential operators using **finite differences on the output pixel image**.

Boundary conditions



We explcitly enforce boundary conditions on the output image → **hard constraints**



$\|\mathcal{R}(u_{\text{CNN}}, p_{\text{CNN}})\|^2 >> 0$ $\qquad$ $\|\mathcal{R}(u_{\text{CNN}}, p_{\text{CNN}})\|^2 \approx 0$

Here, we consider ther **Navier–Stokes equations**:

$$\mathcal{R}(u_{\text{CNN}}, p_{\text{CNN}}) = \begin{bmatrix} -\nu \Delta \vec{u} + (u \cdot \nabla)\, \vec{u} + \nabla p \\ \nabla \cdot u \end{bmatrix}$$

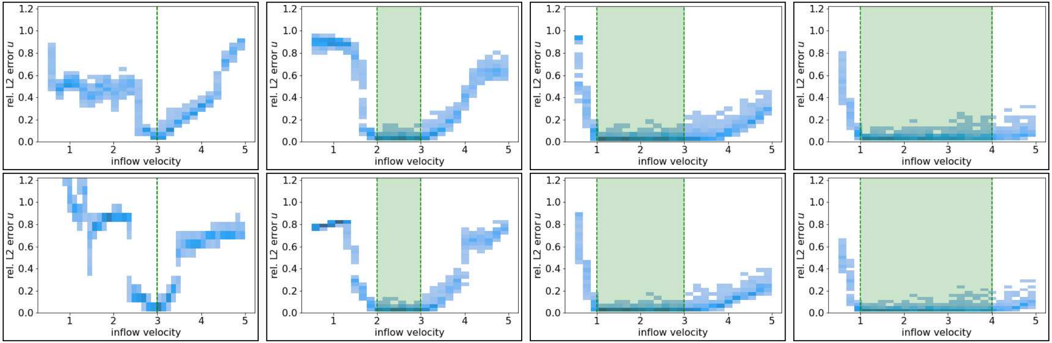
Cf. **Grimm, Heinlein, Klawonn (2025)**.

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 %**	$4.2 \cdot 10^{-2}$	$1.1 \cdot 10^{-1}$	

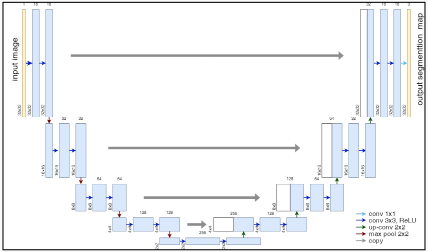
$\rightarrow$ The results for the **physics-informed approach** are **comparable to the data-based approach**; the **errors are slightly higher**. However, no **reference data at all is needed for the training**.

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 %

Domain Decomposition-Based U-Net Architecture

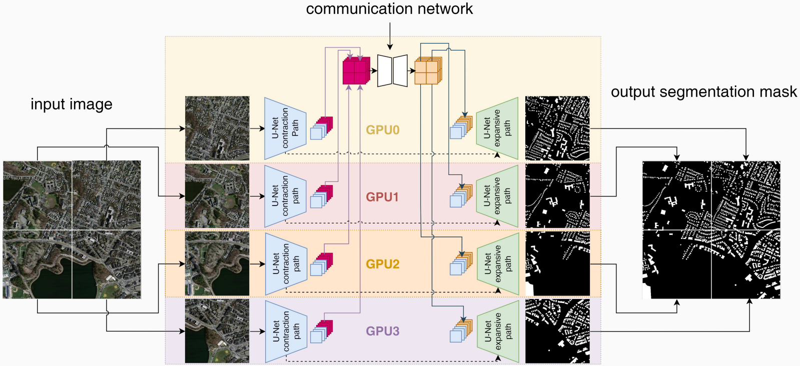


name	mem. feature maps # of values	MB	mem. weights # of values	MB
input block	268 M	1 024.0	38 848	**0.148**
encoder blocks	314 M	1 320	18 M	**72**
decoder blocks	754 M	3880	12 M	**47**
output block	3.1 M	12.0	195	**0.001**

Most memory in the U-Net is used by feature maps, **not weights**

→ **Decompose feature maps** to **distribute memory consumption**.

Cf. **Verburg, Heinlein, Cyr (2025).**



Domain decomposition-based neural neutworks and operators

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, Nissen-Meyer (2023); Dolean, H., Mishra, Moseley (2024, 2024); H., Howard, Beecroft, Stinis (2025); Howard, Jacob, Murphy, H., Stinis (arXiv 2024)
- **DD for RaNNs, ELMS, Random Feature Method**: Dong, Li (2021); Dang, Wang (2024); Sun, Dong, Wang (2024); Sun, Wang (2024); Chen, Chi, E, Yang (2022); Shang, H., Mishra, Wang (2025)
- **DDMs for CNNs**: Gu, Zhang, Liu, Cai (2022); Lee, Park, Lee (2022); Klawonn, Lanser, Weber (2024); Verburg, Heinlein, Cyr (2025)

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

A. Klawonn, M. Lanser, J. Weber
Machine learning, domain decomposition methods – a survey
Computational Science and Engineering. 2024

Physics-Informed Neural Networks (PINNs)

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

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

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

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

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

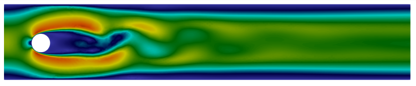
See also **Dissanayake and Phan-Thien (1994); Lagaris et al. (1998)**.

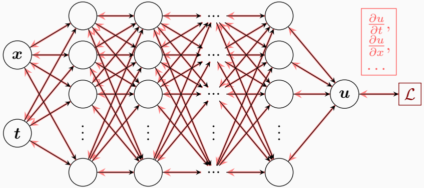
Advantages

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

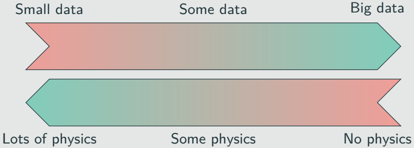
Drawbacks

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

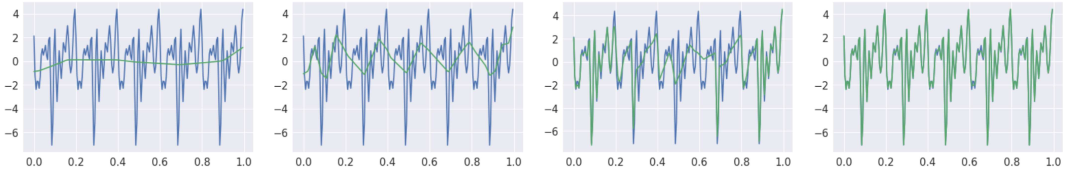
Estimate of the generalization error (Mishra and Molinaro (2022))

The generalization error (or total error) satisfies

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

- $\mathcal{E}_G = \mathcal{E}_G(\boldsymbol{X}, \theta) := \|\mathbf{u} - \mathbf{u}^*\|_V$ **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)**

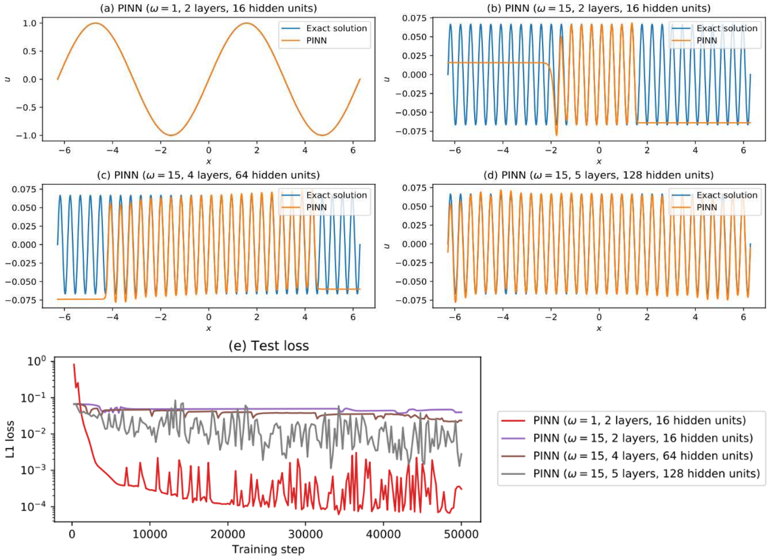
Related works: Cao et al. (2021), Wang, et al. (2022), Hong et al. (arXiv 2022), Xu et al (2024), ...

Scaling of PINNs for a Simple ODE Problem

Solve

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

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



(a) 321 free parameters

(d) 66 433 free parameters

Scaling issues

- Large computational domains
- Small frequencies

Cf. **Moseley, Markham, and Nissen-Meyer (2023)**

Scaling of PINNs for a Simple ODE Problem

Solve

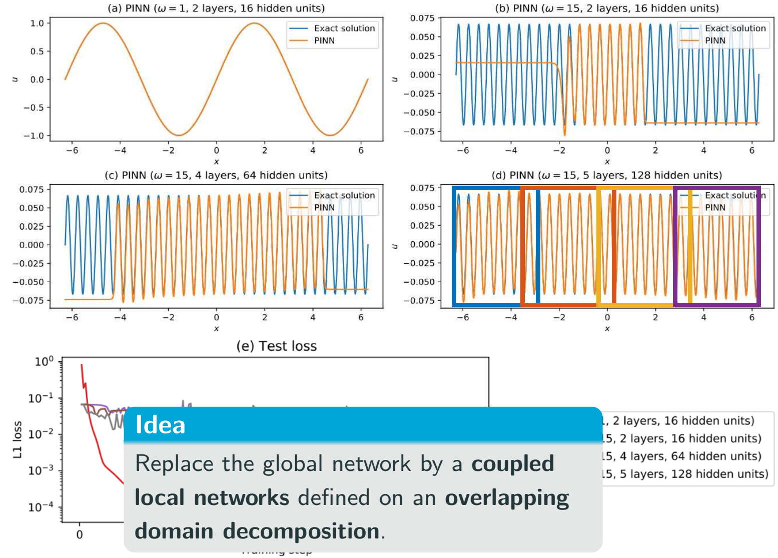
$$
\begin{aligned}
u' &= \cos(\omega \boldsymbol{x}), \\
u(0) &= 0,
\end{aligned}
$$

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

Scaling issues

- Large computational domains
- Small frequencies

Cf. **Moseley, Markham, and Nissen-Meyer (2023)**



Idea

Replace the global network by a **coupled local networks** defined on an **overlapping domain decomposition**.

(a) 321 free parameters

(d) 66 433 free parameters

Finite Basis Physics-Informed Neural Networks (FBPINNs)

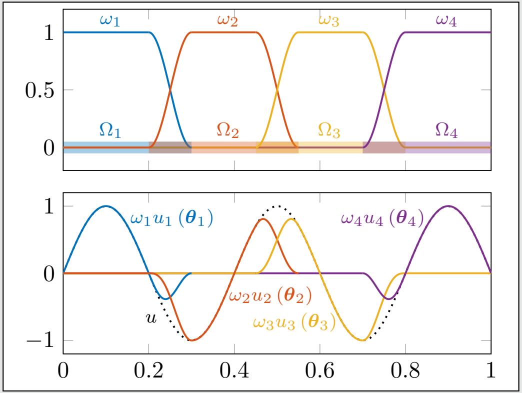
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

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

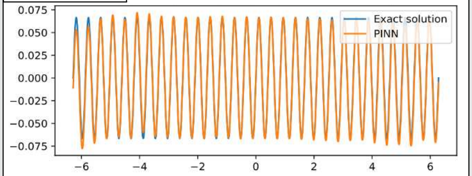
and the **loss function**

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

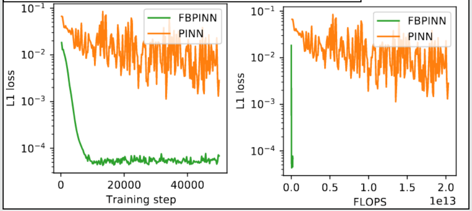


1D single-frequency problem

PINN solution



Moseley, Markham, Nissen-Meyer (2023)



Finite Basis Physics-Informed Neural Networks (FBPINNs)

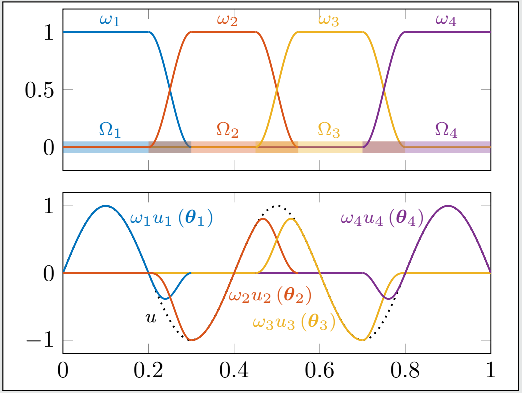
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

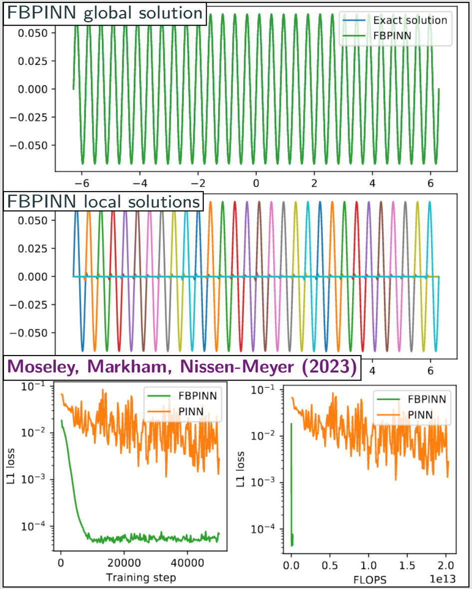
$$u(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \sum\nolimits_{j=1}^{J} \omega_j u_j(\boldsymbol{\theta}_j)$$

and the **loss function**

$$\mathcal{L} = \frac{1}{N} \sum\nolimits_{i=1}^{N} \Big(n[\sum\nolimits_{\mathbf{x}_i \in \Omega_j} \omega_j u_j](\mathbf{x}_i, \boldsymbol{\theta}_j) - f(\mathbf{x}_i) \Big)^2.$$

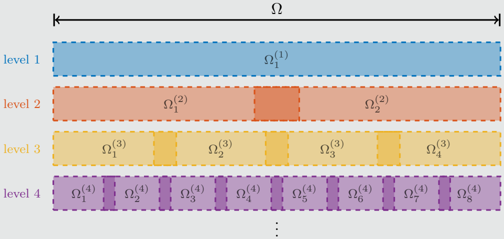


1D single-frequency problem



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**

$$u\left(\boldsymbol{\theta}_1^{(1)}, \ldots, \boldsymbol{\theta}_{J(L)}^{(L)}\right) = \sum\nolimits_{l=1}^{L} \sum\nolimits_{i=1}^{N^{(l)}} \omega_j^{(l)} u_j^{(l)} \left(\boldsymbol{\theta}_j^{(l)}\right)$$

and the **loss function**

$$\mathcal{L} = \frac{1}{N} \sum\nolimits_{i=1}^{N} \left(n[\sum\nolimits_{\mathbf{x}_i \in \Omega_j^{(l)}} \omega_j^{(l)} u_j^{(l)}](\mathbf{x}_i, \boldsymbol{\theta}_j^{(l)}) - f(\mathbf{x}_i)\right)^2.$$

Multi-Frequency Problem

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

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

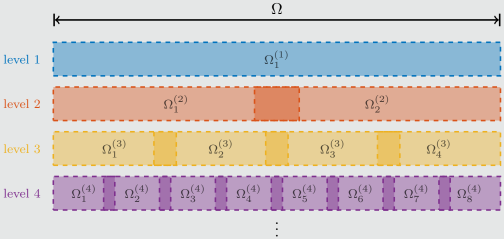
with $\omega_i = 2^i$.

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

Multi-Level FBPINNs

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**

$$u\left(\boldsymbol{\theta}_1^{(1)}, \ldots, \boldsymbol{\theta}_{J^{(L)}}^{(L)}\right) = \sum\nolimits_{l=1}^{L} \sum\nolimits_{i=1}^{N^{(l)}} \omega_j^{(l)} u_j^{(l)}\left(\boldsymbol{\theta}_j^{(l)}\right)$$

and the **loss function**

$$\mathcal{L} = \frac{1}{N} \sum\nolimits_{i=1}^{N} \left(n[\sum\nolimits_{\boldsymbol{x}_i \in \Omega_j^{(l)}} \omega_j^{(l)} u_j^{(l)}](\boldsymbol{x}_i, \boldsymbol{\theta}_j^{(l)}) - f(\boldsymbol{x}_i)\right)^2.$$

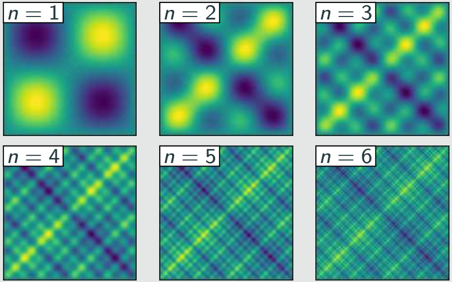
Multi-Frequency Problem

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

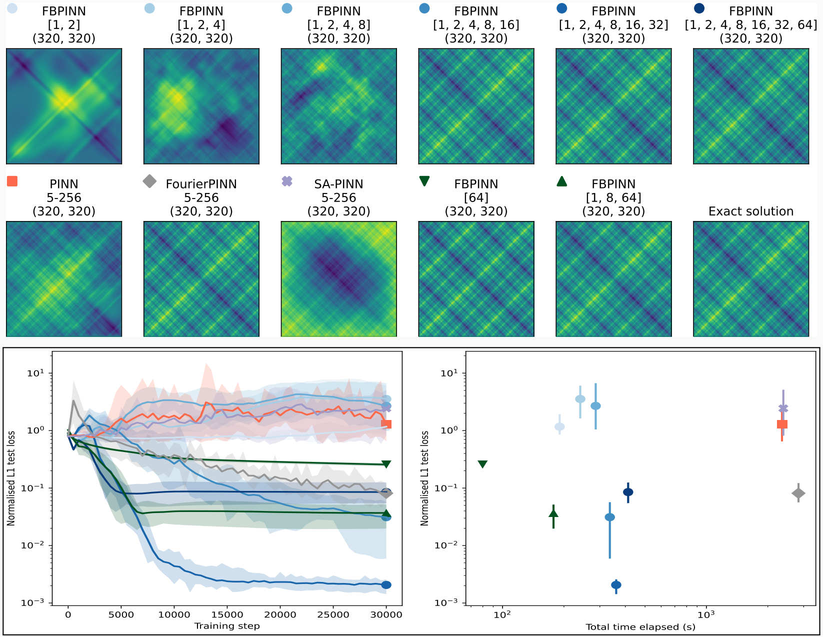
$$\begin{aligned} -\Delta u &= 2 \sum_{i=1}^{n} (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y) && \text{in } \Omega, \\ u &= 0 && \text{on } \partial\Omega, \end{aligned}$$

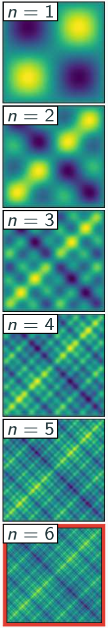
with $\omega_i = 2^i$.

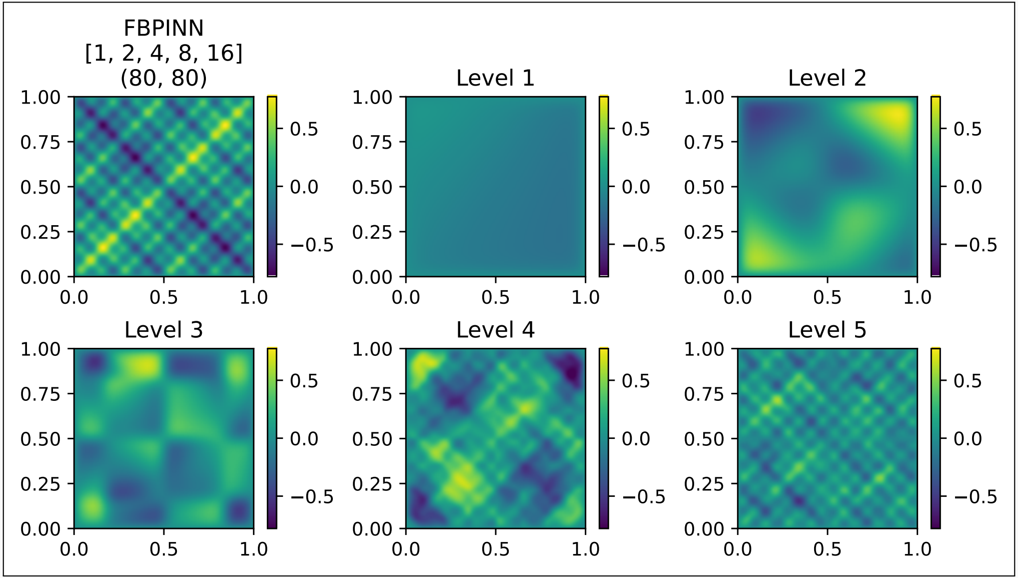
For increasing values of n, we obtain the **analytical solutions**:



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



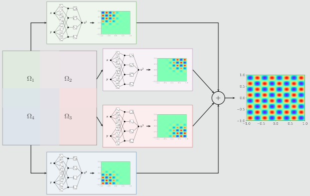


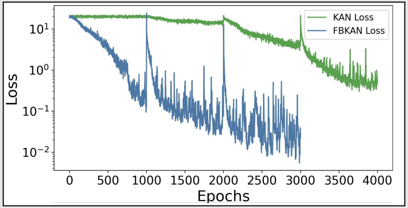


Cf. **Dolean, Heinlein, Mishra, Moseley (2024)**.

Further Directions for Domain Decomposition Approaches

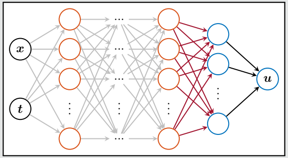
Kolmogorov–Arnold networks (KANs)





- **Finite basis Kolmogorov–Arnold networks (FBKANs)**: **Howard, Jacob, Murphy, H., Stinis (arXiv 2024)**
- **Uncertainty quantification** w/ **conformalized (FB)KANs**: **Mollaali, Moya, Howard, H., Stinis, Lin (arXiv 2025)**

Physics-informed randomized neural networks



trained, fixed, and **SVD** weights;
nonlinear and **linear** neurons

	no prec.		Schwarz prec.	
	iter	e_{L^2}	iter	e_{L^2}
CG	$>$ 2000	$1.95 \cdot 10^{-2}$	8	$5.03 \cdot 10^{-3}$
CGS	$>$ 2000	$2.63 \cdot 10^{-2}$	4	$5.04 \cdot 10^{-3}$
BICG	$>$ 2000	$1.03 \cdot 10^{-2}$	8	$5.08 \cdot 10^{-3}$
GMRES	$>$ 2000	$8.68 \cdot 10^{-2}$	13	$5.07 \cdot 10^{-3}$

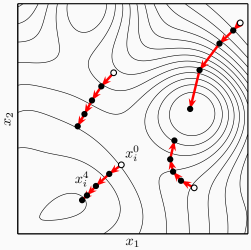
- **Schwarz domain decomposition preconditioning** for **localized physics-informed randomized neural networks (PIRaNNs)**: **Shang, H., Mishra, Wang (2025)**

PACMANN – Point Adaptive Collocation Method for Artificial Neural Networks

In **Visser, Heinlein, and Giovanardi (arXiv 2024)**, the collocation points are updated by solving the **min-max problem**

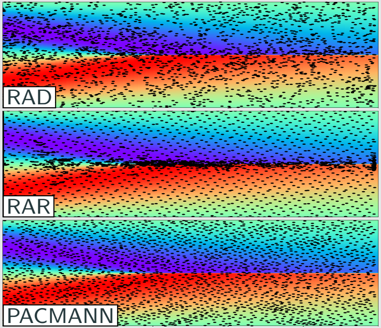
$$\min_{\boldsymbol{\theta}} \left[\omega_{\text{data}} \mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \max_{\boldsymbol{X} \subset \Omega} \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}}(\boldsymbol{X}, \boldsymbol{\theta}) \right].$$

Different from the other residual-based adaptive sampling methods, the **existing collocation points are moved** using a gradient-based optimizers, such as gradient ascent, RMSprop (**Hinton (2018)**), Adam (**Kingma, Ba (2017)**), etc.



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

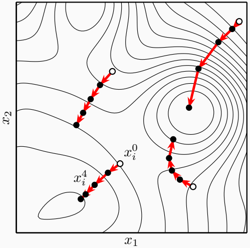


PACMANN – Point Adaptive Collocation Method for Artificial Neural Networks

In **Visser, Heinlein, and Giovanardi (arXiv 2024)**, the collocation points are updated by solving the **min-max problem**

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

Different from the other residual-based adaptive sampling methods, the **existing collocation points are moved** using a gradient-based optimizers, such as **gradient ascent**, **RMSprop** (**Hinton (2018)**), **Adam** (**Kingma, Ba (2017)**), etc.



5D Poisson equation

sampling method	L_2 relative error		mean
	mean	1 SD	runtime [s]
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
PACMANN–Adam	**5.93%**	**0.46%**	778

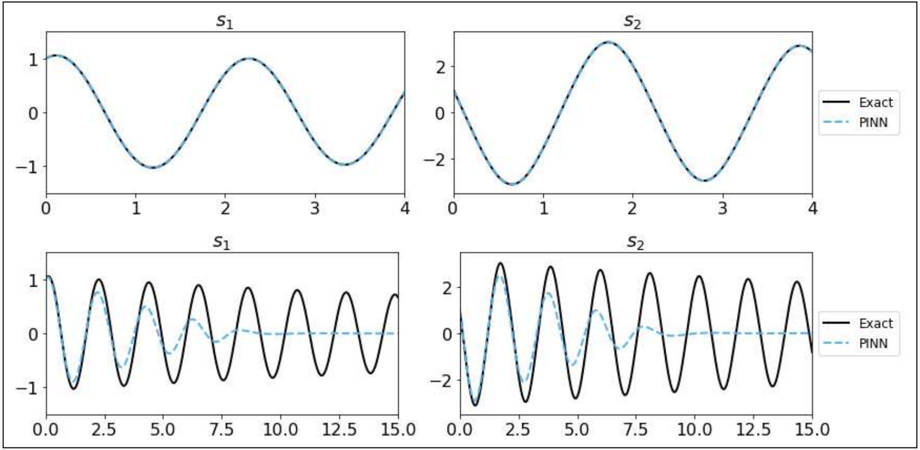
We investigate the performance of PINNs for **time-dependent problems**. Therefore, consider the simple **pedulum problem**:

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

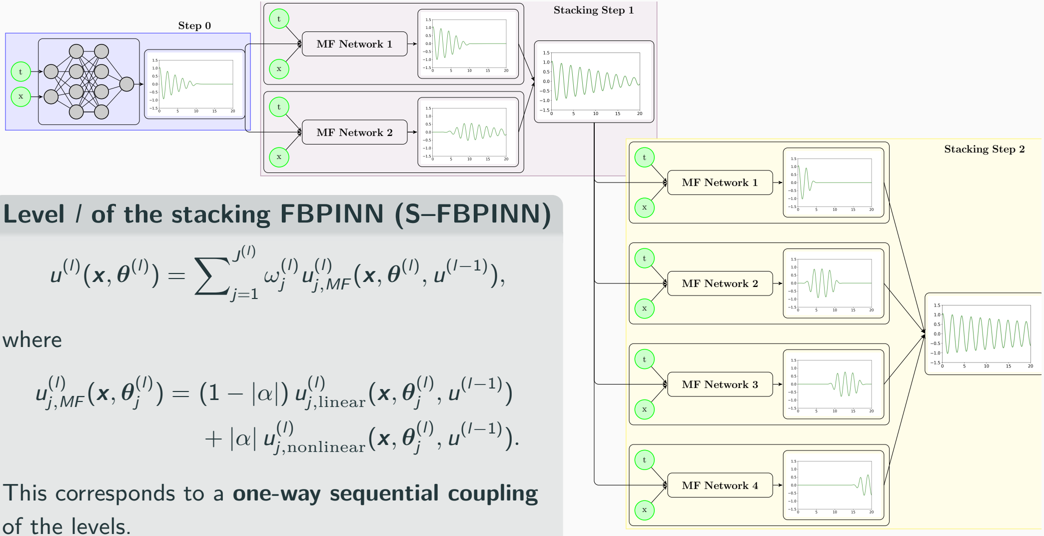
Problem parameters

$m = L = 1,\ b = 0.05,$
$g = 9.81$

- **Top:** $T = 4$
- **Bottom:** $T = 20$



In **Heinlein, Howard, Beecroft, and Stinis (2025)**, 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

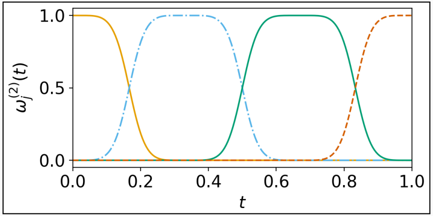
$$u_{j,MF}^{(l)}(\boldsymbol{x}, \boldsymbol{\theta}_j^{(l)}) = (1 - |\alpha|)\, u_{j,\mathrm{linear}}^{(l)}(\boldsymbol{x}, \boldsymbol{\theta}_j^{(l)}, u^{(l-1)}) + |\alpha|\, u_{j,\mathrm{nonlinear}}^{(l)}(\boldsymbol{x}, \boldsymbol{\theta}_j^{(l)}, u^{(l-1)}).$$

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

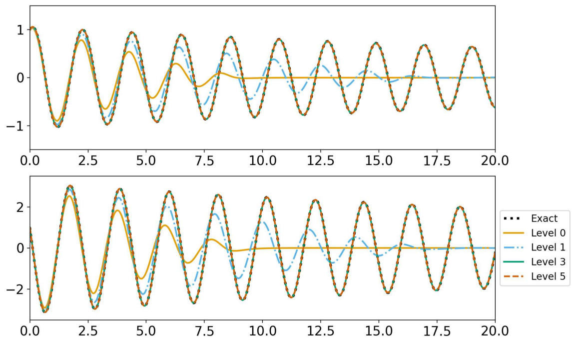
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

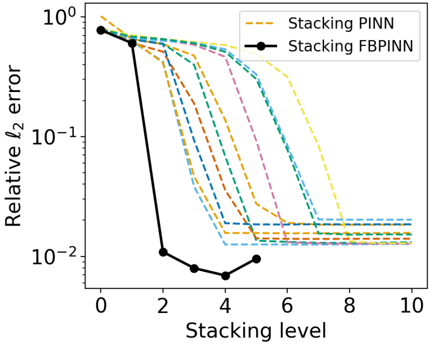
$$\frac{ds_1}{dt} = s_2,$$
$$\frac{ds_2}{dt} = -\frac{b}{m}s_2 - \frac{g}{L}\sin(s_1)$$

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



Exemplary partition of unity in time





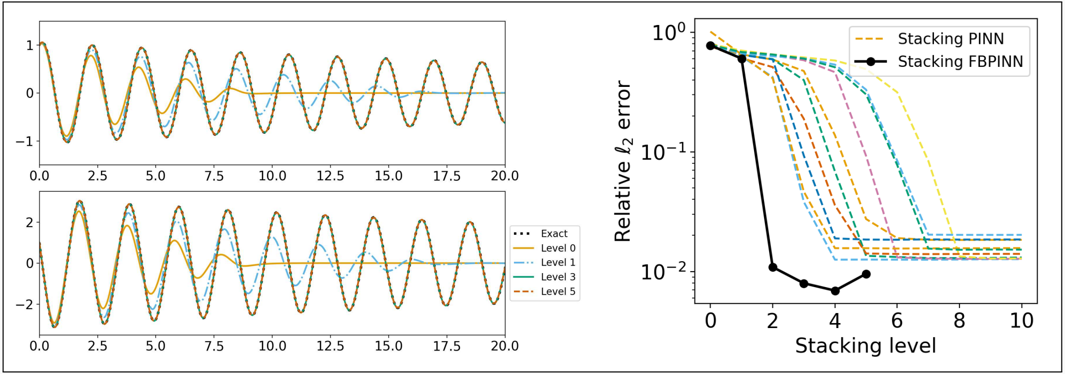
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

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

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

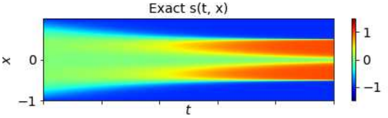
Model details:

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

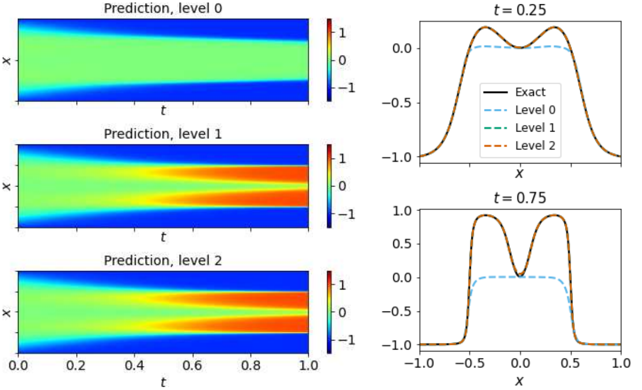


Finally, we consider the **Allen–Cahn equation**:

$$\begin{aligned} s_t - 0.0001 s_{xx} + 5s^3 - 5s &= 0, && t \in (0,1], x \in [-1,1], \\ s(x,0) &= x^2 \cos(\pi x), && x \in [-1,1], \\ s(x,t) &= s(-x,t), && t \in [0,1], x=-1, x=1, \\ s_x(x,t) &= s_x(-x,t), && t \in [0,1], x=-1, x=1. \end{aligned}$$



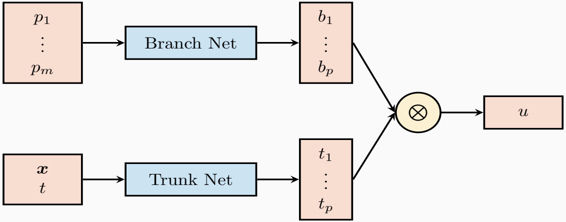
method	arch.	# params	error
PINN	0 levels 6x50	12 951	0.499
S–FBPINN	2 levels 5x32, 1x20	39 627	0.00594



PINN **gets stuck** at **fixed point of the of dynamical system**; cf. **Rohrhofer et al. (2023)**.

Deep Operator Networks (DeepONets / DONs)

Neural operators learn operators between function spaces using neural networks. Here, we learn the **solution operator** of a initial-boundary value problem parametrized with $p_1, \ldots, p_m$ using **DeepONets** as introduced in **Lu et al. (2021)**.



Single-layer case

The DeepONet architecture is based on the **single-layer case** analyzed in **Chen and Chen (1995)**. In particular, the authors show **universal approximation properties for continuous operators**.

The architecture is based on the following ansatz for presenting the parametrized solution

$$u_{(p_1,\ldots,p_m)}(\boldsymbol{x}, t) = \sum_{i=1}^{p} \underbrace{b_i(p_1, \ldots, p_m)}_{\text{branch}} \cdot \underbrace{t_i(\boldsymbol{x}, t)}_{\text{trunk}}$$

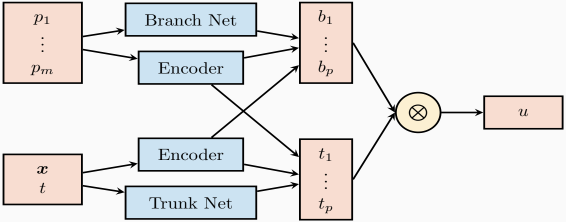
Physics-informed DeepONets

DeepONets are **compatible with the PINN approach** but **physics-informed DeepONets (PI-DeepONets)** are challenging to train.

Other operator learning approaches

- **FNOs**: **Li et al. (2021)**
- **PCA-Net**: **Bhattacharya et al. (2021)**
- **Random features**: **Nelsen and Stuart (2021)**
- **CNOs**: **Raonić et al. (2023)**

Neural operators learn operators between function spaces using neural networks. Here, we learn the **solution operator** of a initial-boundary value problem parametrized with $p_1, \dots, p_m$ using **DeepONets** as introduced in **Lu et al. (2021)**.



Modified architecture

In our numerical experiments, we employ the **modified DeepONet architecture** introduced in **Wang, Wang, and Perdikaris (2022)**.

The architecture is based on the following ansatz for presenting the parametrized solution

$$u_{(p_1,\dots,p_m)}(\boldsymbol{x}, t) = \sum_{i=1}^{p} \underbrace{b_i(p_1, \dots, p_m)}_{\text{branch}} \cdot \underbrace{t_i(\boldsymbol{x}, t)}_{\text{trunk}}$$

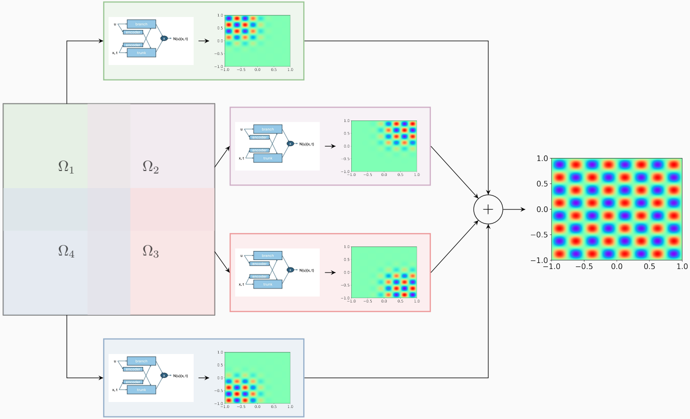
Physics-informed DeepONets

DeepONets are **compatible with the PINN approach** but **physics-informed DeepONets (PI-DeepONets)** are challenging to train.

Other operator learning approaches

- **FNOs**: **Li et al. (2021)**
- **PCA-Net**: **Bhattacharya et al. (2021)**
- **Random features**: **Nelsen and Stuart (2021)**
- **CNOs**: **Raonić et al. (2023)**

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 (2025)

Wave equation

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

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

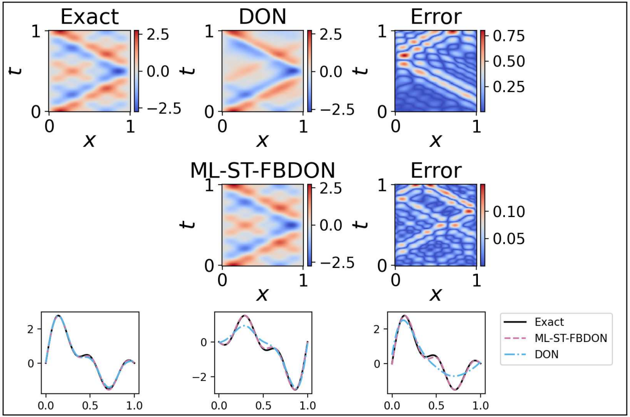
Solution: $s(x,t) = \sum_{n=1}^{5} b_n \sin(n\pi x)\cos(n\pi\sqrt{2}t)$

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 1 000 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.)**

Scientific Machine Learning in Academia and Beyond:
From Theory to Real-World Impact (in Industry)

- **Dates**: June 17, 2025, 12.30–17.30
- **Location**: Crowne Plaza Hotel, Utrecht
- **Lunch & networking**: 12.30–13.30; closing discussion & drinks to follow.
- An afternoon with **talks, case studies, and lively discussions** on advancing scientific machine learning from theory to real-world deployment — tackling core challenges like *uncertainty quantification, data assimilation, graph-based modelling, and operator learning*.
- **Confirmed plenary speakers**:
  - Max Welling (UvA, CUSP AI)
  - Stefan Kurz (ETH Zürich & Bosch)
  - Koen Strien (Ignition Computing)
  - Maxim Pisarenco (ASML)
  - Jan Willem van de Meent (UvA)

Computational Science NL

CWI Research Semester Programme:

Bridging Numerical Analysis and Scientific Machine Learning: Advances and Applications

Co-organizers: Victorita Dolean (TU/e), Alexander Heinlein (TU Delft), Benjamin Sanderse (CWI), Jemima Tabbeart (TU/e), Tristan van Leeuwen (CWI)

- **Autumn School** (October 27–31, 2025):
  - Chris Budd (University of Bath)
  - Ben Moseley (Imperial College London)
  - Gabriele Steidl (Technische Universität Berlin)
  - Andrew Stuart (California Institute of Technology)
  - Andrea Walther (Humboldt-Universität zu Berlin)
  - Ricardo Baptista (University of Toronto)
- **Workshop** (December 1–3, 2025):
  - 3 days with plenary talks (academia & industry) and an industry panel
  - Confirmed plenary speakers:
    - Marta d'Elia (Atomic Machines)
    - Benjamin Peherstorfer (New York University)
    - Andreas Roskopf (Fraunhofer Institute)

Join us for inspiring talks, hands-on sessions, and industry collaboration!

Summary

Surrogate models for varying computational domains

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

Multilevel FBPINNs and Extensions

- **Multilevel FBPINNs scale PINNs** to **large domains** and **high frequencies** via **domain decomposition and multilevel hierarchies**.
- **Extensions: Multifidelity stacking improves performance**, e.g., for **time-dependent problems**; **DeepONets** predict parametrized problems with **enhanced scalability**.

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!

Topical Activity Group
Scientific Machine Learning