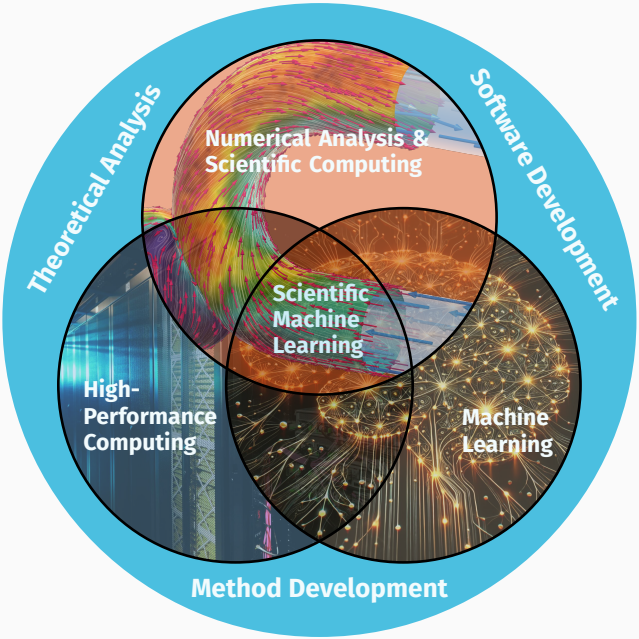
Towards Physics-Informed Machine Learning-Based Surrogate Models for Challenging Problems

Alexander Heinlein[1]

4TU.AMI Workshop Strategic Research Initiative on Model Reduction for Industrial Applications, June 16, 2025

[1]Delft University of Technology

SCaLA – Scalable Scientific Computing and Learning Algorithms



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)
Julia Pelzer, **Miriam Schulte**	(University of Stuttgart)
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

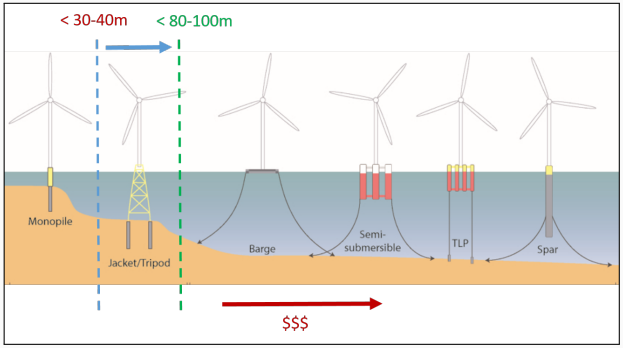
Designing of Perforated Monopiles for Offshore Wind Energy

Perforated monopiles

Monopiles are the **most used and cheapest** solution of support structures in **offshore wind energy**.

→ **Perforated monopiles reduce the wave load**.

What is the **optimal perforation shape**?

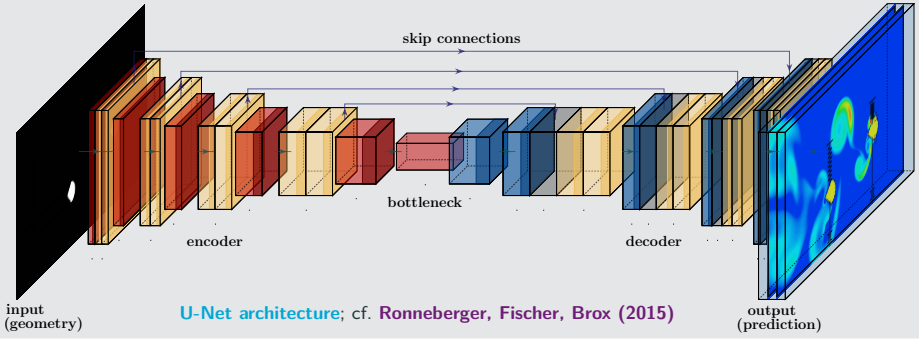


Fully resolved CFD simulations are **costly**, but **rough predictions** may be **sufficient**.

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)

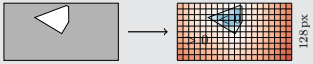
- FNOs: Li et al. (2021)
- PCA-Net: Bhattacharya et al. (2021)
- Random features: Nelsen and Stuart (2021)
- CNOs: Raonić et al. (2023)

Comparison OpenFOAM® Versus CNN (Relative Error 2 %)

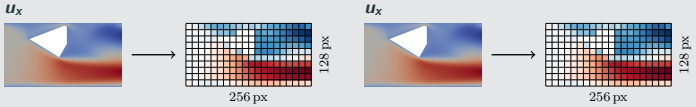
We automatically generate geometries and compute the corresponding flow fields using OpenFOAM®.

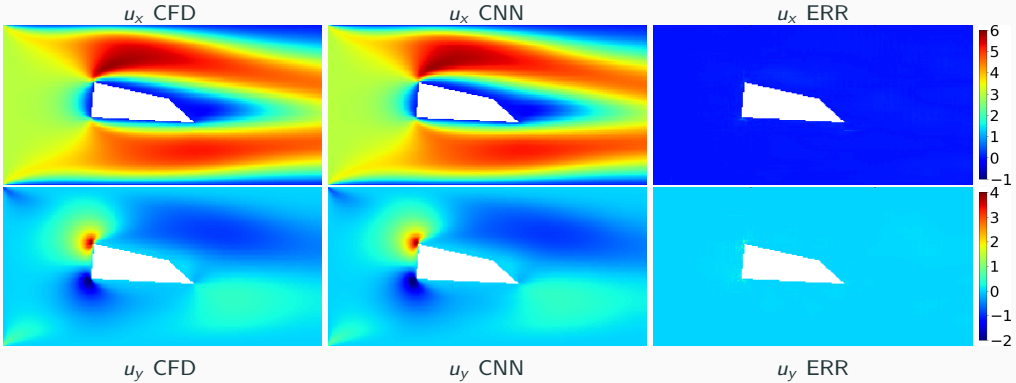
Input data

SDF (Signed Distance Function)



Output data





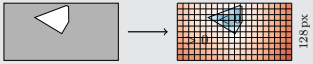
Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**.

Comparison OpenFOAM® Versus CNN (Relative Error 14 %)

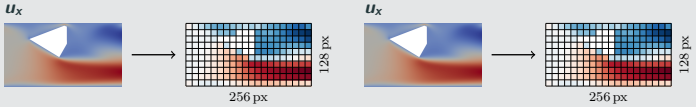
We automatically generate geometries and compute the corresponding flow fields using OpenFOAM®.

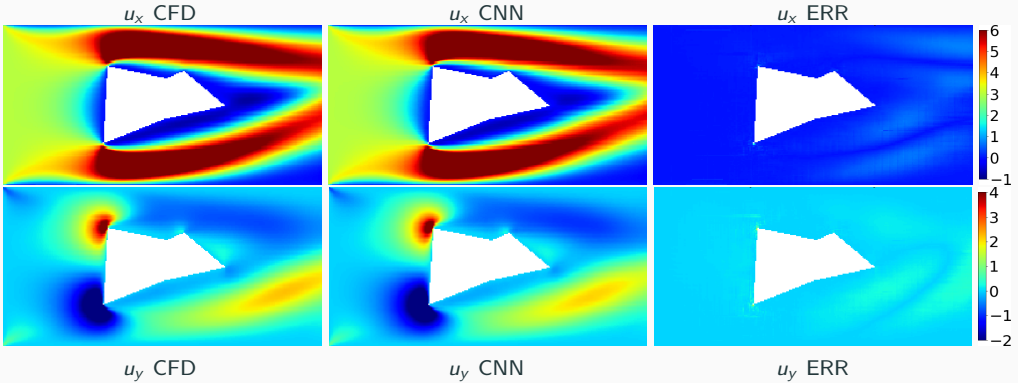
Input data

SDF (Signed Distance Function)



Output data





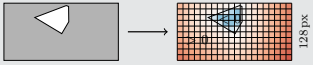
Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**.

Comparison OpenFOAM® Versus CNN (Relative Error 31 %)

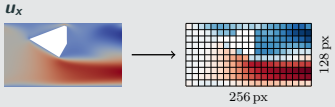
We automatically generate geometries and compute the corresponding flow fields using OpenFOAM®.

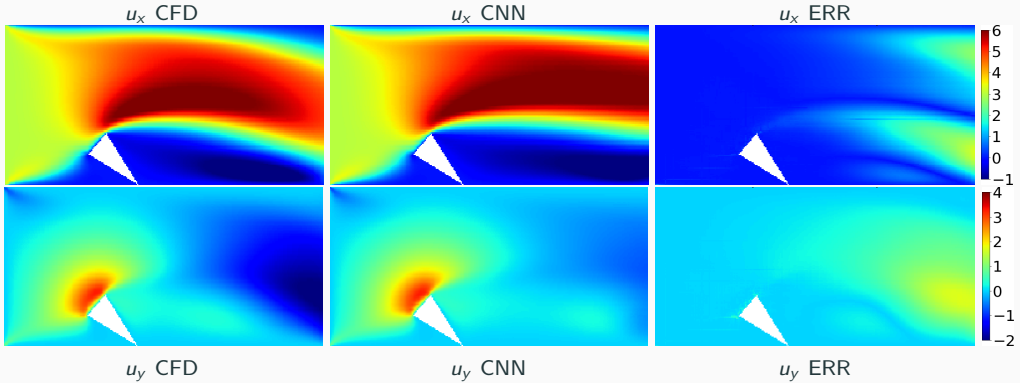
Input data

SDF (Signed Distance Function)



Output data





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

Computing Times

Data generation:

	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:

	OpenFOAM®	**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**:

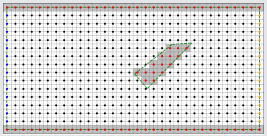
$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \|\mathcal{R}(u_{\text{CNN}}, p_{\text{CNN}})\|^2$$

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

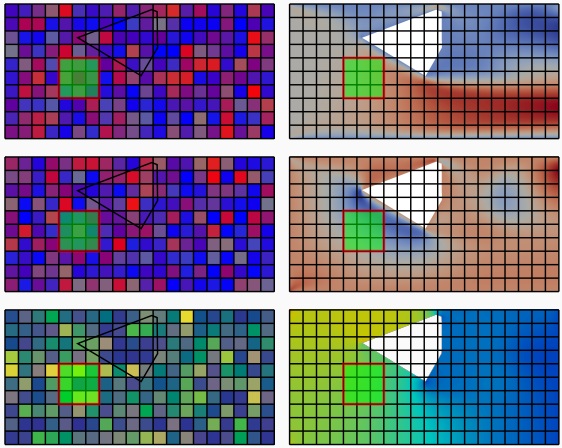
Cf. **Raissi et al. (2019), Dissanayake and Phan-Thien (1994), Lagaris et al. (1998)**.

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**



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

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

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

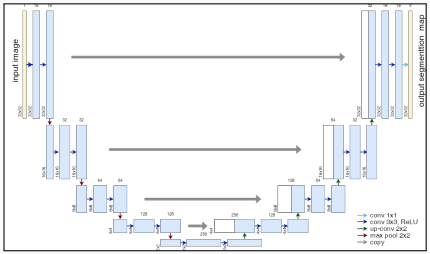
Cf. **Grimm, Heinlein, Klawonn (2025)**.

Results on $\approx$ 5 000 Geometries – Data-Based Versus Physics-Informed

	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**.

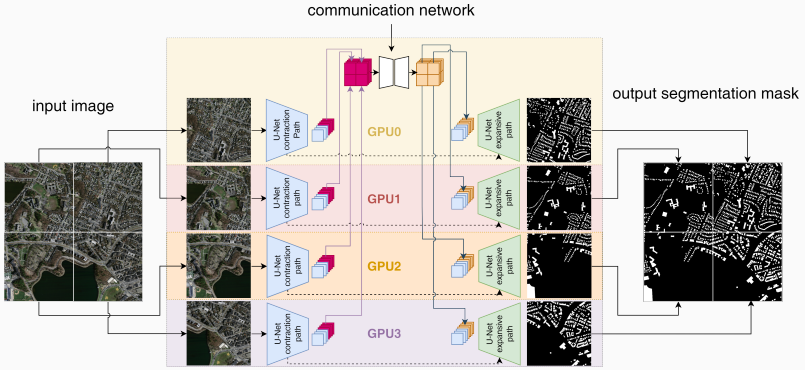
Domain Decomposition-Based U-Net Architecture



name	mem. feature maps		mem. weights	
	# of values	MB	# 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**

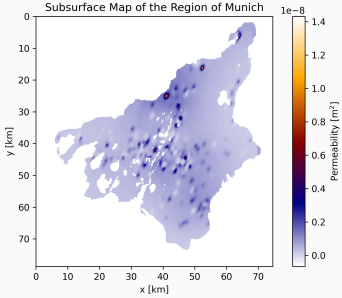
Most memory in the **U-Net** is used by **feature maps**, **not weights**
→ **Decompose feature maps** to **distribute memory consumption**.

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

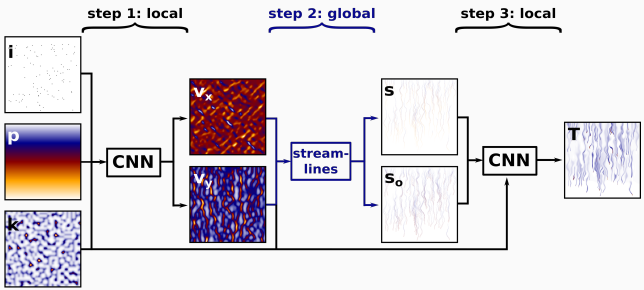


LG-CNN – Incorporating Physics Explicitly To Enable Few-Shot Learning

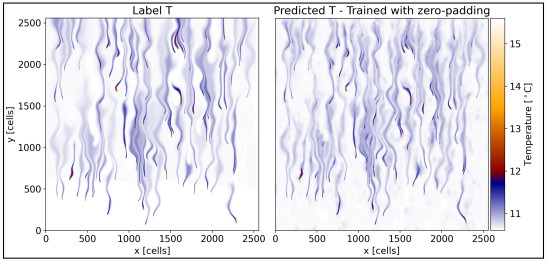
Permeability Map Munich



LGCNN - Inference Pipeline



- Training **data** is **extremely limited**: in real-world scenarios easily < 10 samples.
- **Spatial resolution** is **too high** for predicting the **full field at once** (**memory constraints**); need to **decompose**.
- **Explicit physics** (step 2) integration enables **generalization** from a **single sample**.



Pelzer, Verburg, Heinlein, Schulte (subm. 2025)

Domain decomposition-based neural neutworks and operators

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

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

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

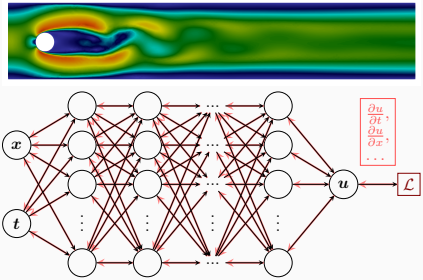
See also **Dissanayake and Phan-Thien (1994); Lagaris et al. (1998)**.

Advantages

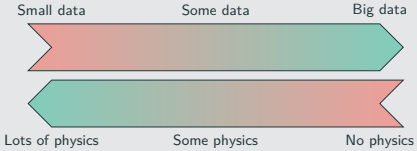
- "Meshfree"
- Small data
- Generalization properties
- High-dimensional problems
- Inverse and parameterized problems

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

Error Estimate & Spectral Bias

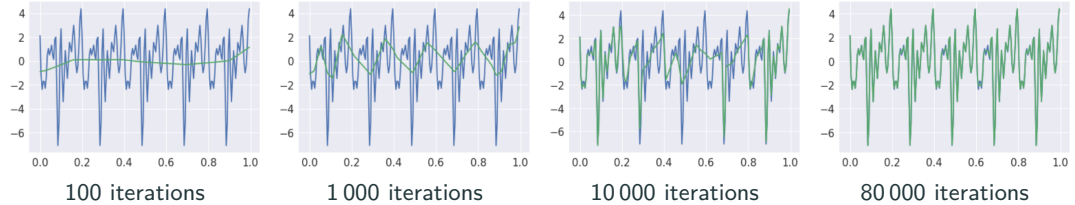
Estimate of the generalization error (Mishra and Molinaro (2022))

The generalization error (or total error) satisfies

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

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



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

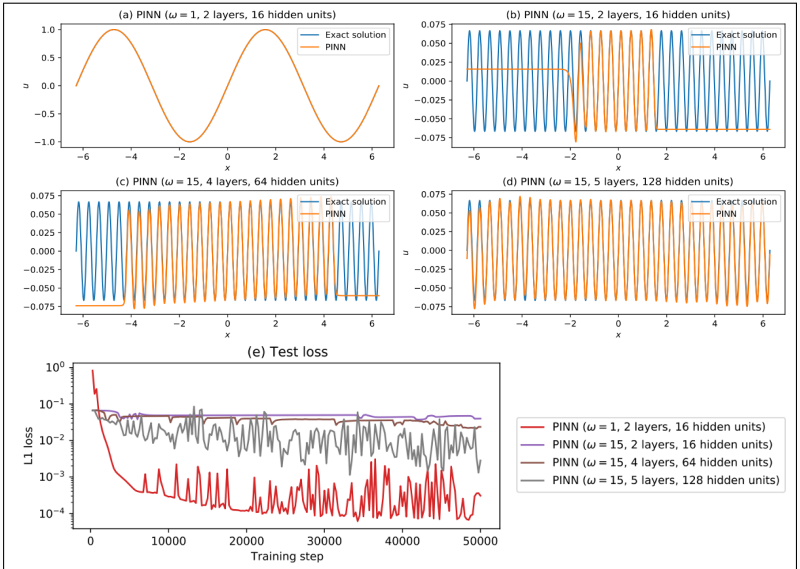
$$\begin{aligned} u' &= \cos(\omega x), \\ u(0) &= 0, \end{aligned}$$

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)**



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

Scaling of PINNs for a Simple ODE Problem

Solve

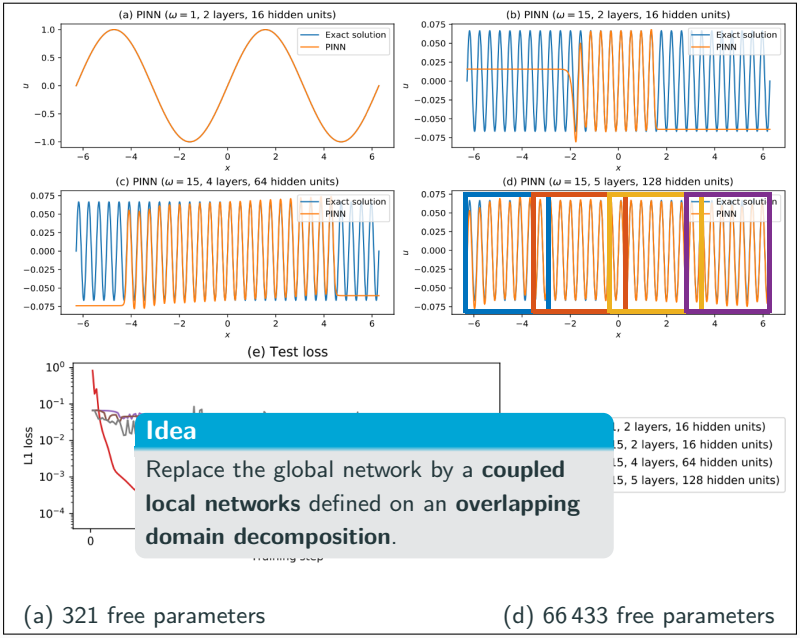
$$\begin{aligned} u' &= \cos(\omega x), \\ u(0) &= 0, \end{aligned}$$

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)

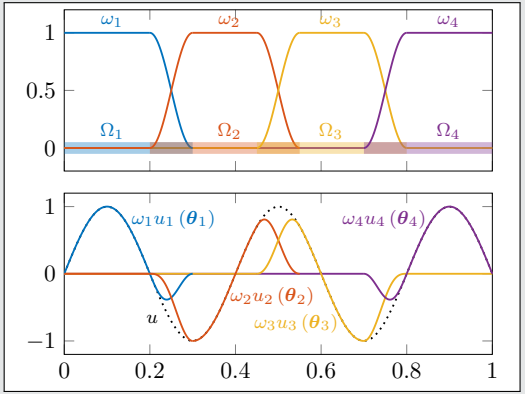
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

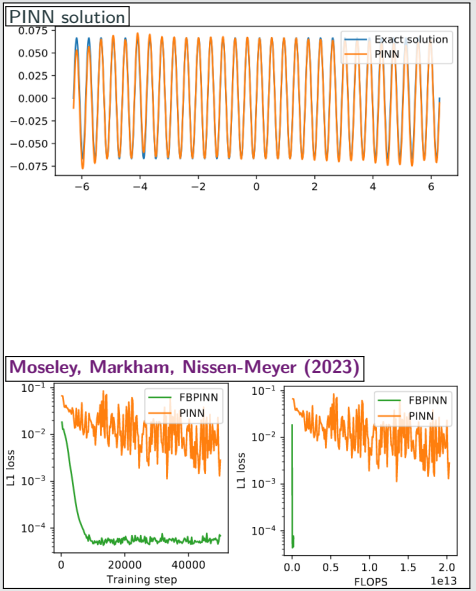
$$u(\theta_1, \ldots, \theta_J) = \sum_{j=1}^{J} \omega_j u_j(\theta_j)$$

and the **loss function**

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



1D single-frequency problem



Finite Basis Physics-Informed Neural Networks (FBPINNs)

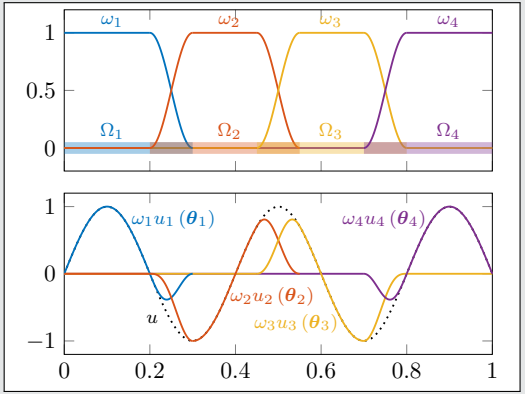
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

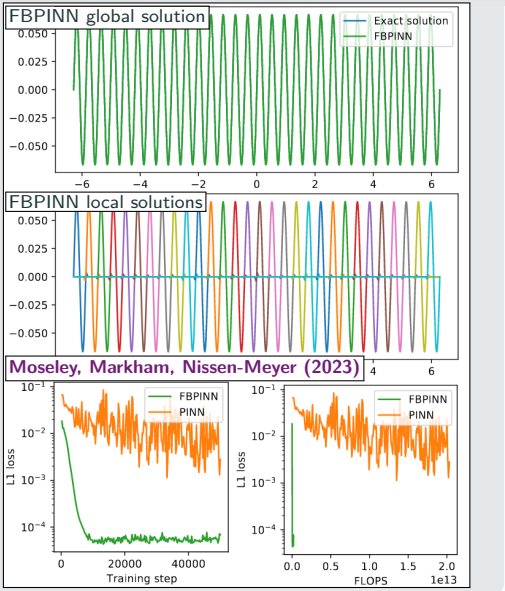
$$u(\theta_1, \dots, \theta_J) = \sum_{j=1}^{J} \omega_j u_j(\theta_j)$$

and the **loss function**

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



1D single-frequency problem



Finite Basis Physics-Informed Neural Networks (FBPINNs)

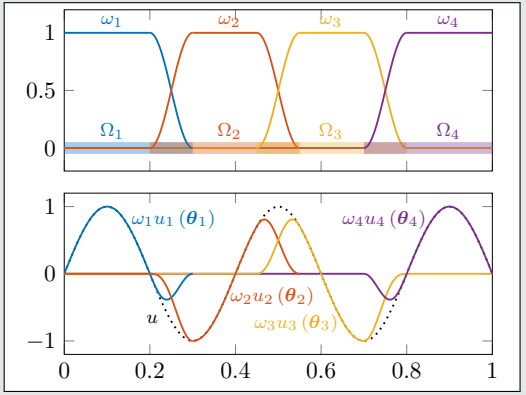
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

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

and the **loss function**

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



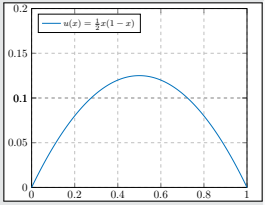
Scalability of FBPINNs

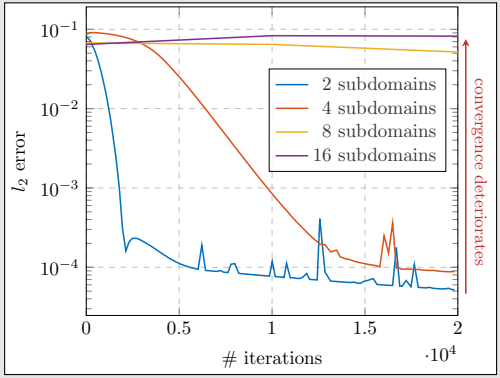
Consider the **simple boundary value problem**

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

which has the **solution**

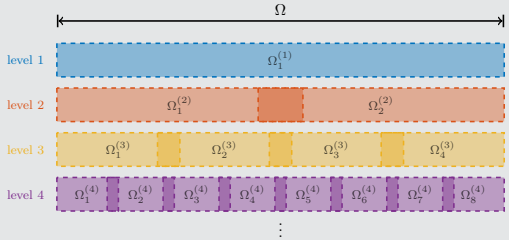
$$u(x) = 1/2 x(1-x).$$





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**

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

and the **loss function**

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

Multi-Frequency Problem

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

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

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

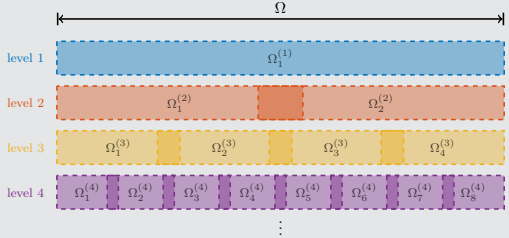
with $\omega_i = 2^i$.

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**

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

and the **loss function**

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

Multi-Frequency Problem

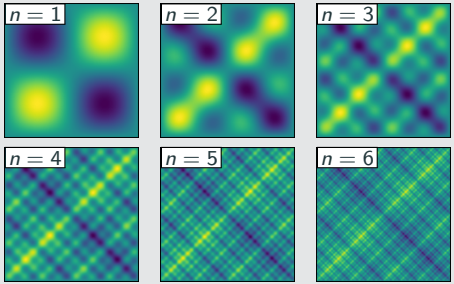
Let us now consider the **two-dimensional multi-frequency Laplace boundary value problem**

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

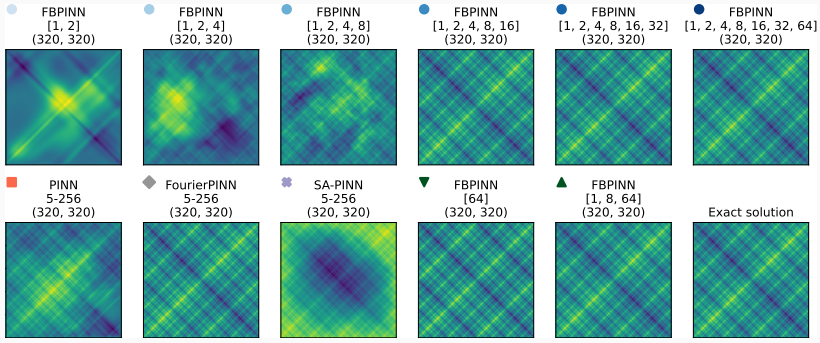
$$u = 0 \quad \text{on } \partial\Omega,$$

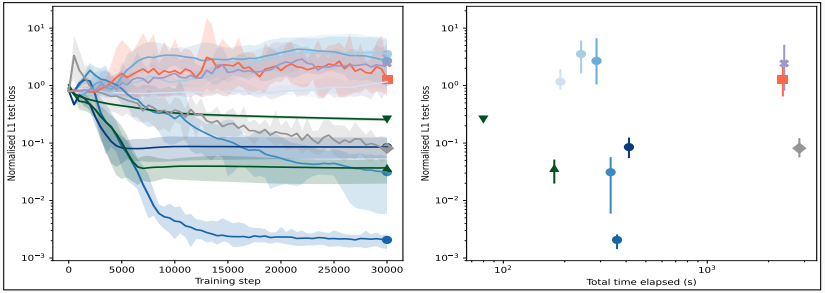
with $\omega_i = 2^i$.

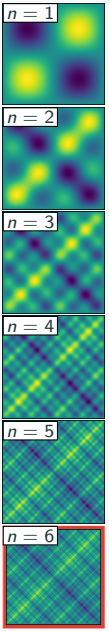
For increasing values of n, we obtain the **analytical solutions**:



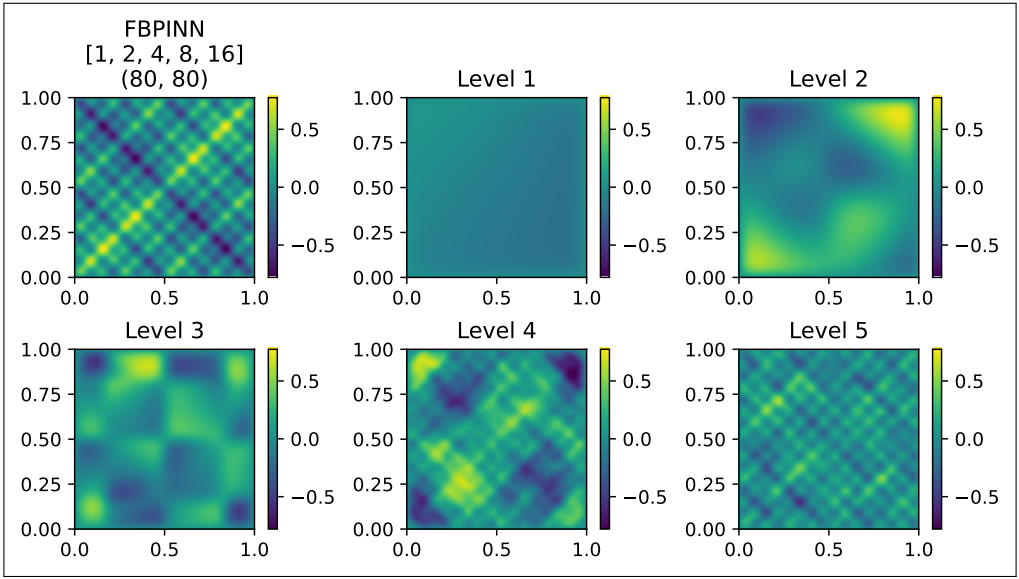
Multi-Level FBPINNs for a Multi-Frequency Problem – Strong Scaling







Multi-Frequency Problem – What the FBPINN Learns

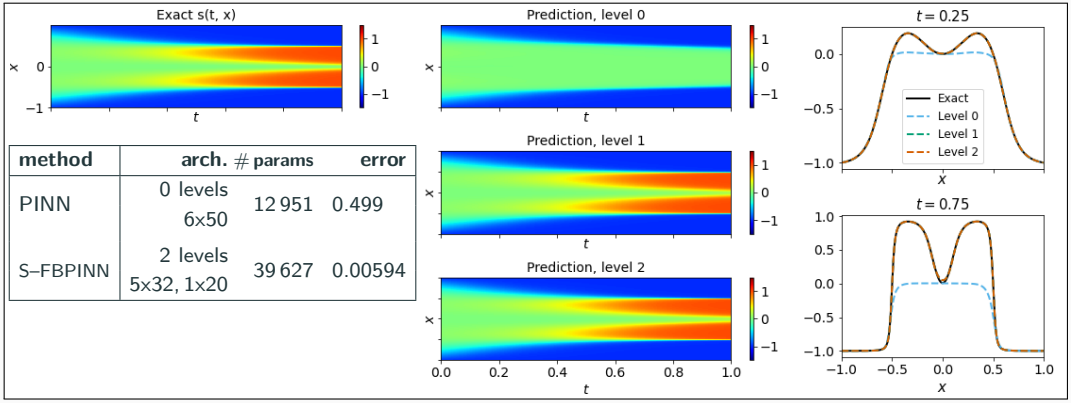


Cf. **Dolean, Heinlein, Mishra, Moseley (2024)**.

Multifidelity Stacking FBPINNs – Allen–Cahn Equation

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

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

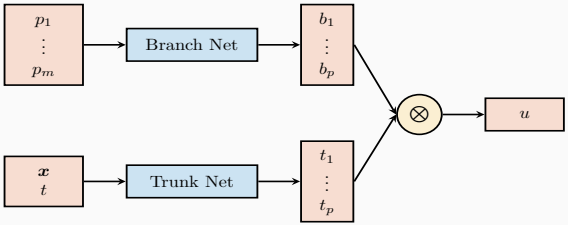


method	arch.	# params	error
PINN	0 levels 6x50	12 951	0.499
S–FBPINN	2 levels 5x32, 1x20	39 627	0.00594

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 $p_1, \dots, p_m$ 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

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

Physics-informed DeepONets

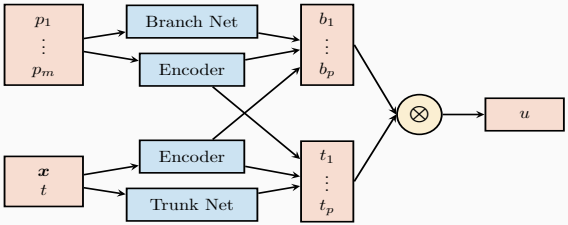
DeepONets are **compatible with the PINN approach** but **physics-informed DeepONets (PI-DeepONets)** are challenging to train.

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)**

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 $p_1, \dots, p_m$ 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

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

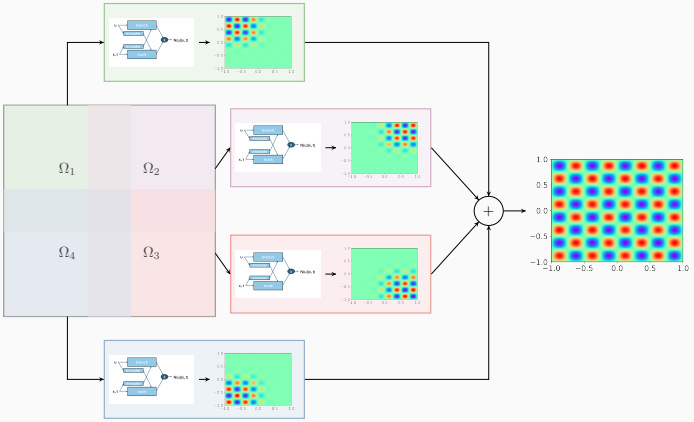
Physics-informed DeepONets

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)

FBDONs – Wave Equation

Wave equation

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

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

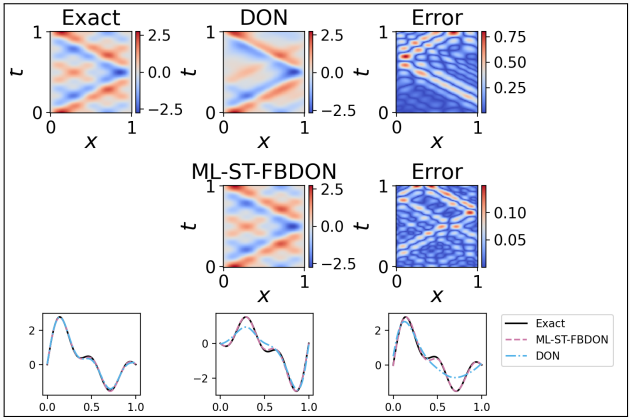
Solution: $s(x,t) = \sum_{n=1}^{5} b_n \sin(n\pi x)\cos(n\pi\sqrt{2}t)$

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 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.)**

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