Domain decomposition for physics-informed neural networks

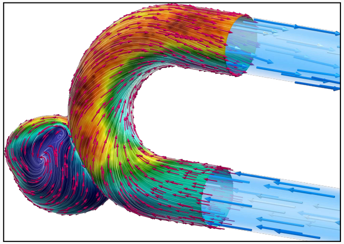
Linear and nonlinear function approximation and operator learning

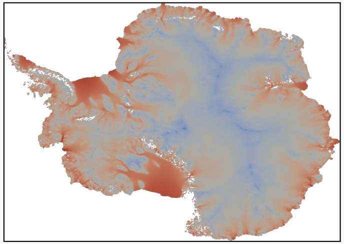
Alexander Heinlein[1]

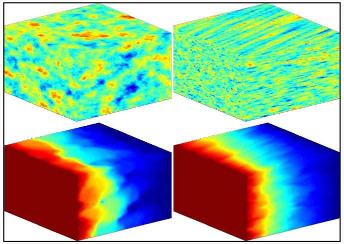
Interdisciplinary Scientific Computing Laboratory Seminar Series, Pennsylvania State University, March 14, 2025

[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. Multilevel domain decomposition-based architectures for physics-informed neural networks

   Based on joint work with

Victorita Dolean	(Eindhoven University of Technology)
Siddhartha Mishra	(ETH Zürich)
Ben Moseley	(Imperial College London)

2. Stacking multifidelity physics-informed neural networks

   Based on joint work with

Damien Beecroft	(University of Washington)
Amanda A. Howard and **Panos Stinis**	(Pacific Northwest National Laboratory)

3. Domain decomposition for randomized neural networks

   Based on joint work with

Siddhartha Mishra	(ETH Zürich)
Yong Shang and **Fei Wang**	(Xi'an Jiaotong University)

4. Domain decomposition-based physics-informed deep operator networks

   Based on joint work with

Amanda A. Howard and **Panos Stinis**	(Pacific Northwest National Laboratory)

Multilevel domain decomposition-based architectures for physics-informed neural networks

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**:

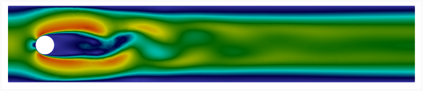
$$\mathcal{L}(\boldsymbol{\theta}) = \omega_{\text{data}}\mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \omega_{\text{PDE}}\mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}),$$

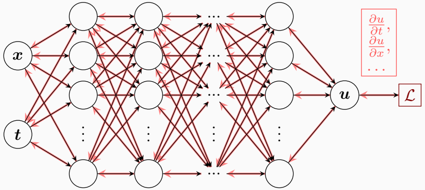
where ω_{data} and ω_{PDE} are **weights** and

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

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

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





x
t
u
$\frac{\partial u}{\partial t}, \frac{\partial u}{\partial x}, \ldots$
$\mathcal{L}$

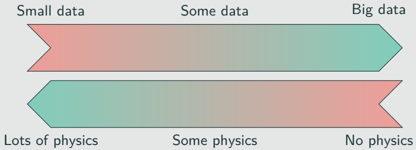
Advantages

- **"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

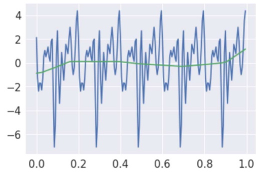
Estimate of the generalization error (Mishra and Molinaro (2022))

The generalization error (or total error) satisfies

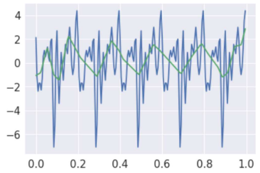
$$\mathcal{E}_G \leq C_{\text{PDE}} \mathcal{E}_T + C_{\text{PDE}} C_{\text{quad}}^{1/p} N^{-\alpha/p}$$

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

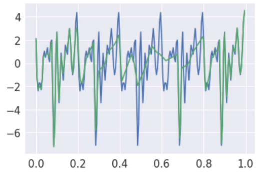
Rule of thumb: **"As long as the PINN is trained well, it also generalizes well"**



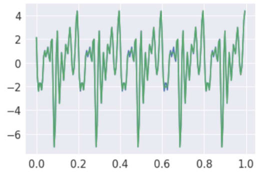
100 iterations



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

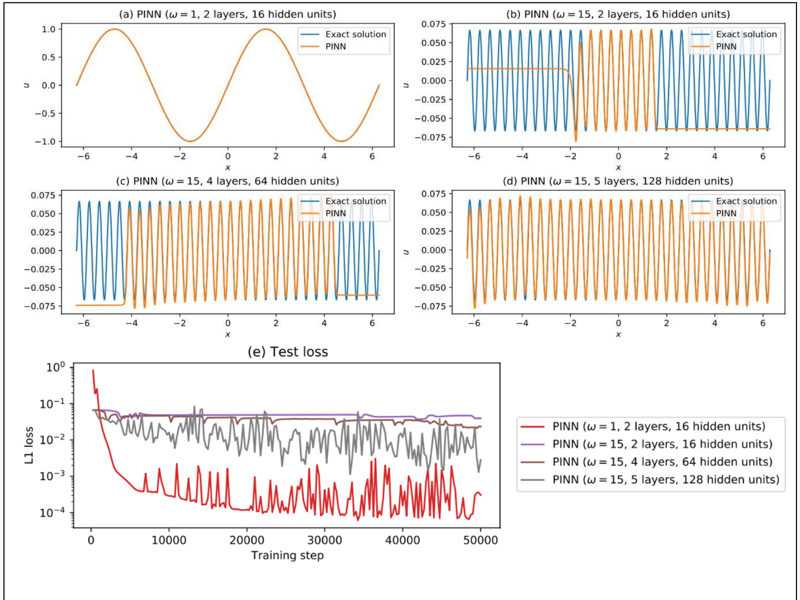
$$\begin{aligned} u' &= \cos(\omega \boldsymbol{x}), \\ u(0) &= 0, \end{aligned}$$

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

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

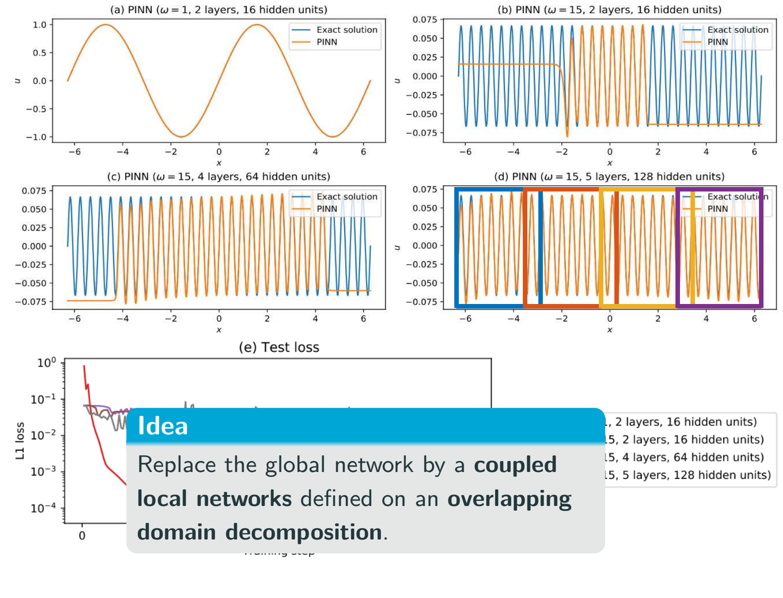
$$\begin{aligned} u' &= \cos(\omega \boldsymbol{x}), \\ u(0) &= 0, \end{aligned}$$

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)**

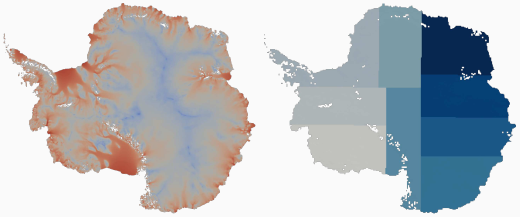


Idea

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

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

Domain Decomposition Methods


Images based on **Heinlein, Perego, Rajamanickam (2022)**

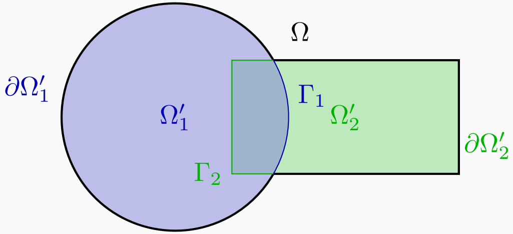
Historical remarks: The **alternating Schwarz method** is the earliest **domain decomposition method (DDM)**, which has been invented by **H. A. Schwarz** and published in **1870**:

- Schwarz used the algorithm to establish the **existence of harmonic functions** with prescribed boundary values on **regions with non-smooth boundaries**.

Idea

Decomposing a large **global problem** into smaller **local problems**:

- **Better robustness** and **scalability** of numerical solvers
- **Improved computational efficiency**
- Introduce **parallelism**



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

Finite Basis Physics-Informed Neural Networks (FBPINNs)

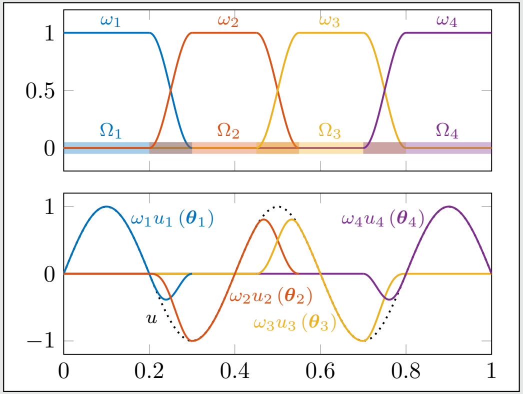
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

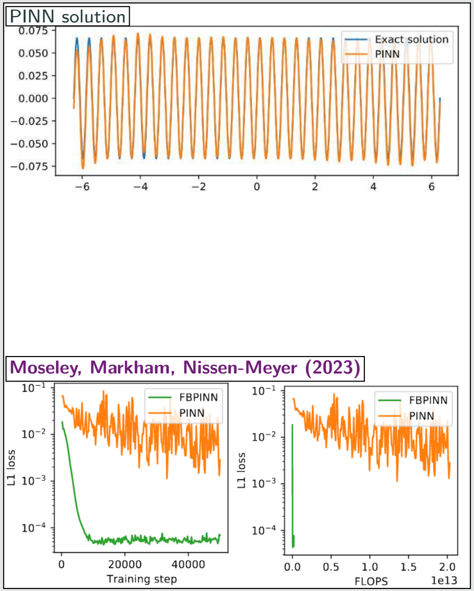
$$u(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \sum_{j=1}^{J} \omega_j u_j(\boldsymbol{\theta}_j)$$

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



Finite Basis Physics-Informed Neural Networks (FBPINNs)

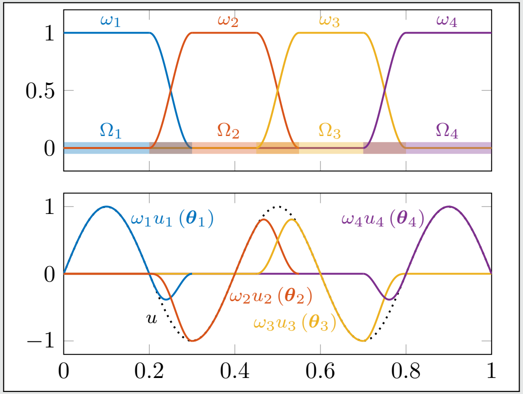
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

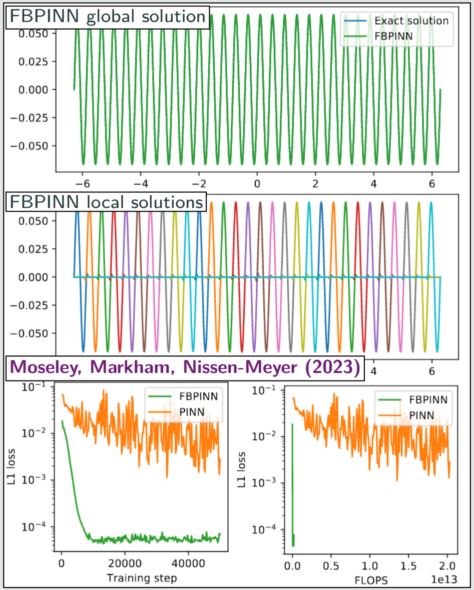
$$u(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \sum\nolimits_{j=1}^{J} \omega_j u_j(\boldsymbol{\theta}_j)$$

and the **loss function**

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



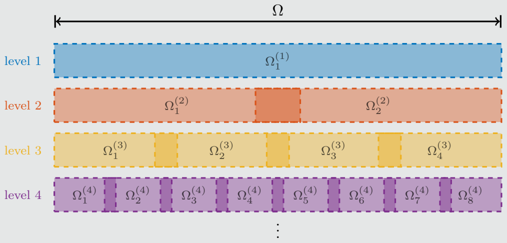
1D single-frequency problem



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**

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

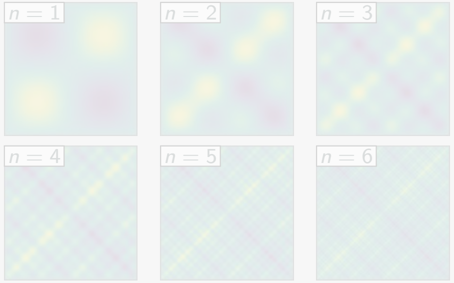
Multi-Frequency Problem

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 $\omega_i = 2^i$.

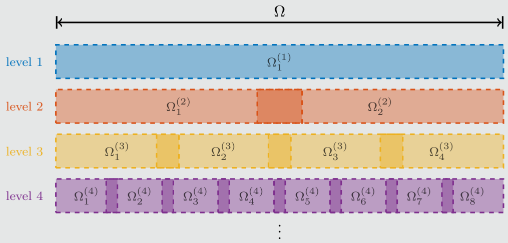
For increasing values of n, we obtain the **analytical solutions**:



Multi-Level FBPINNs

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**

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

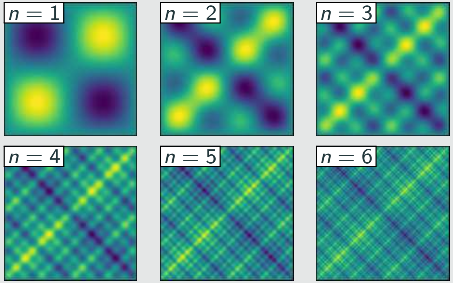
Multi-Frequency Problem

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

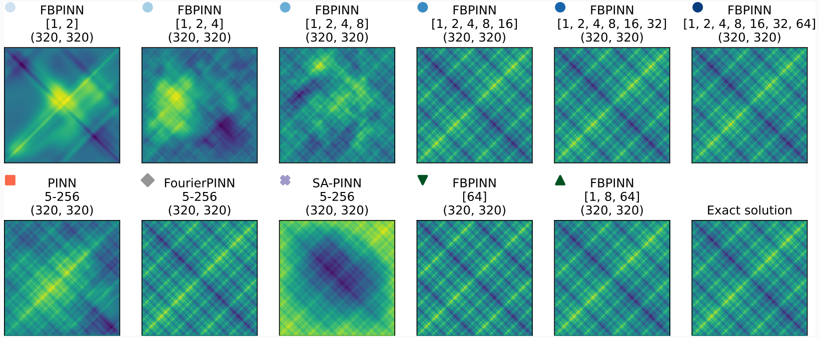
$$\begin{aligned} -\Delta u &= 2 \sum_{i=1}^{n} (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y) && \text{in } \Omega, \\ u &= 0 && \text{on } \partial\Omega, \end{aligned}$$

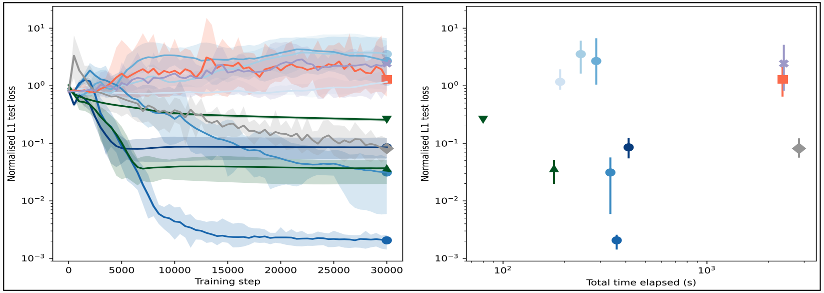
with $\omega_i = 2^i$.

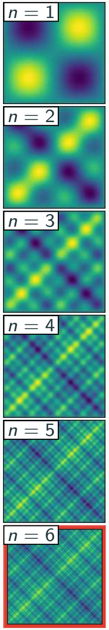
For increasing values of n, we obtain the **analytical solutions**:



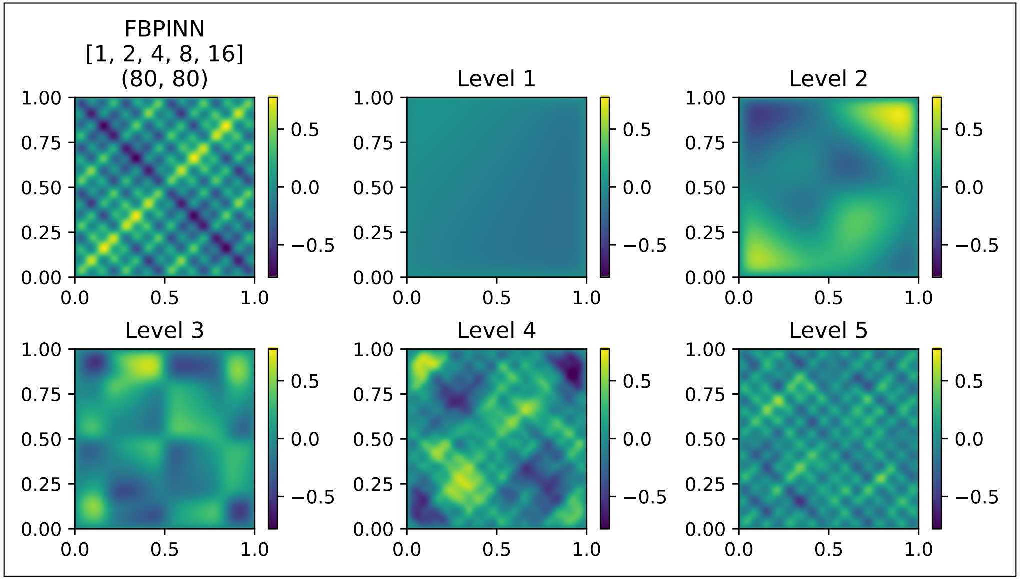
Multi-Level FBPINNs for a Multi-Frequency Problem – Strong Scaling





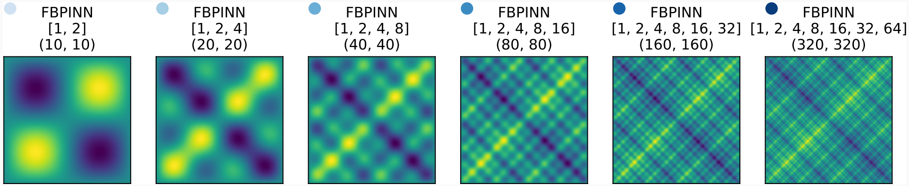


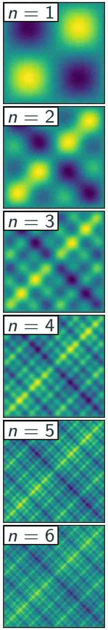
Multi-Frequency Problem – What the FBPINN Learns

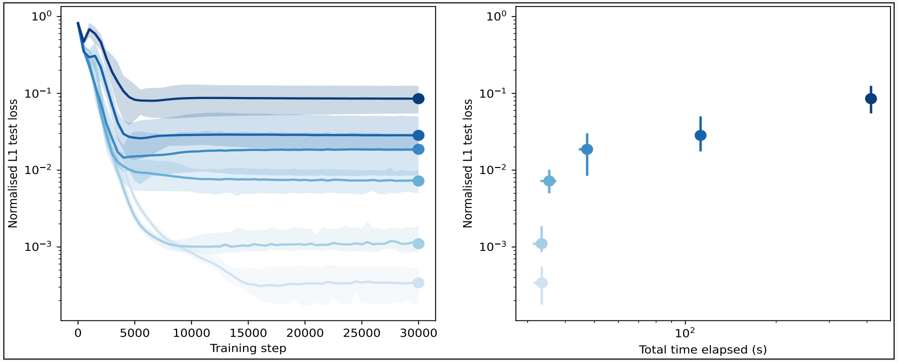


Cf. **Dolean, Heinlein, Mishra, Moseley (2024)**.

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



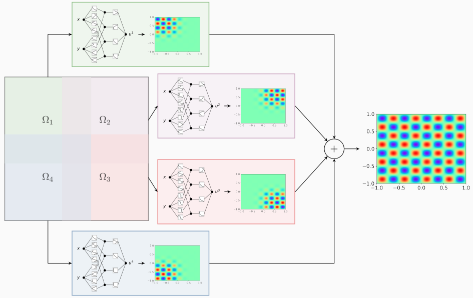


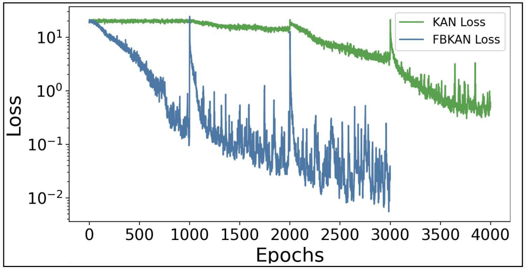


→ Details and results for the Helmholtz equation can be found in **Dolean, Heinlein, Mishra, Moseley (2024)**.

Extension to Kolmogorov–Arnold Networks (KANs)

We have also extended our approach to **Kolmogorov–Arnold networks (KANs)**, resulting in **finite basis Kolmogorov–Arnold networks (FBKANs)**. This extension achieves **similar improvements** as those observed with classical feedforward neural networks:





One-dimensional ODE problem with multiscale features

For more details, see **Howard, Jacob, Murphy, Heinlein, Stinis (arXiv 2024)**.

Stacking multifidelity physics-informed neural networks

PINNs for Time-Dependent Problems

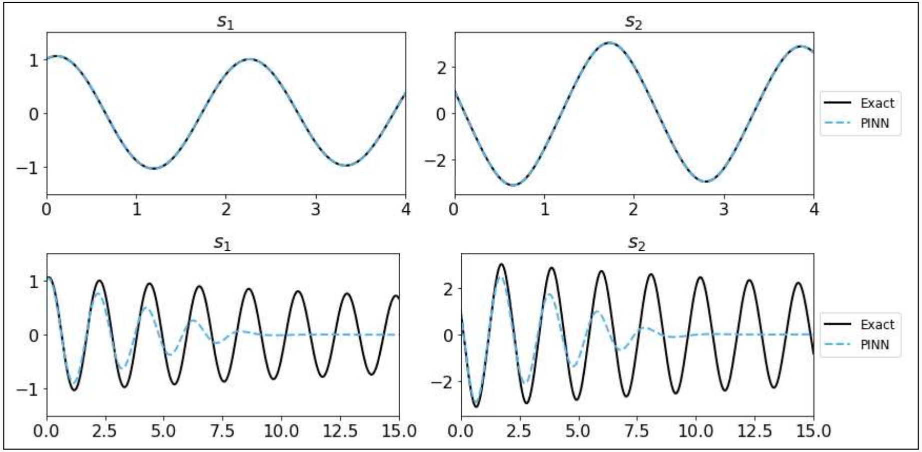
We investigate the performance of PINNs for **time-dependent problems**. Therefore, consider the simple **pedulum problem**:

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

Problem parameters

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

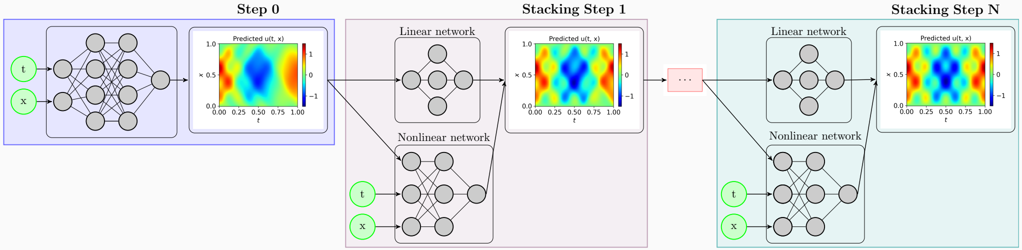
- **Top:** $T = 4$
- **Bottom:** $T = 20$



Multifidelity Stacking PINNs

In the **multifidelity stacking PINNs approach** introduced in **Howard, Murphy, Ahmed, Stinis (2025)**, **multiple PINNs are trained in a recursive way.** In each step, a model u^{MF} is trained based on the previous model u^{SF}:

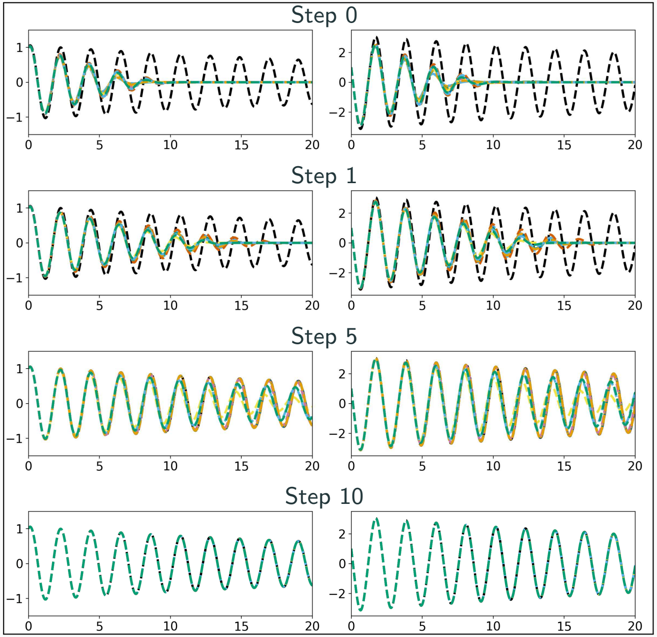
$$u^{MF}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}) = (1 - |\alpha|)\, u^{MF}_{\mathrm{linear}}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}, u^{SF}) + |\alpha|\, u^{MF}_{\mathrm{nonlinear}}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}, u^{SF})$$



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); Murphy, Ahmed, Stinis (2025)**
- **Galerkin, multi-level, and multi-stage neural networks:** **Ainsworth and Dong (2021); Ainsworth and Dong (2022); Aldirany et al. (2024); Wang and Lai (2024)**

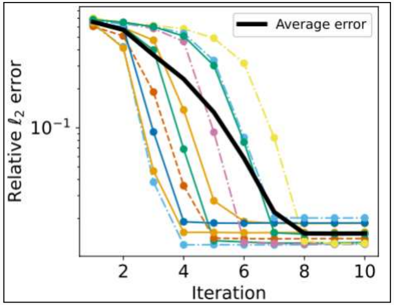
Multifidelity Stacking PINNs for the Pendulum Problem



Pendulum problem:

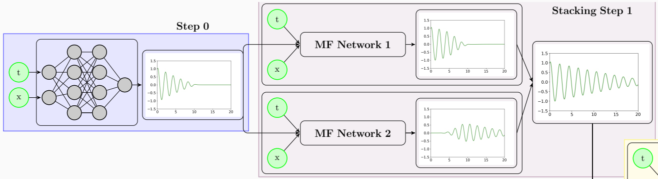
$$\frac{ds_1}{dt} = s_2,$$
$$\frac{ds_2}{dt} = -\frac{b}{m}s_2 - \frac{g}{L}\sin(s_1).$$

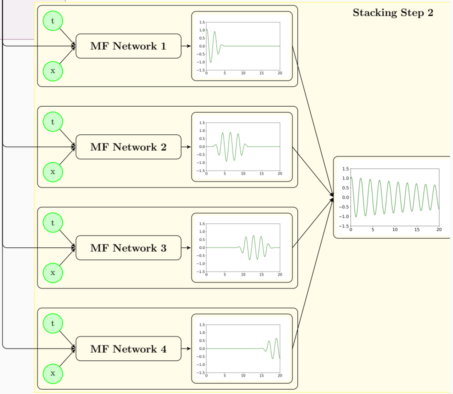
with $m = L = 1$, $b = 0.05$, $g = 9.81$, and $T = 20$.



Multifidelity Stacking FBPINNs

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

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

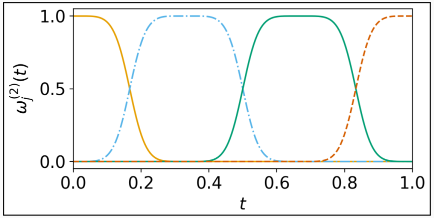
This corresponds to a **one-way sequential coupling** of the levels.

Multifidelity Stacking FBPINNs – Pendulum Problem

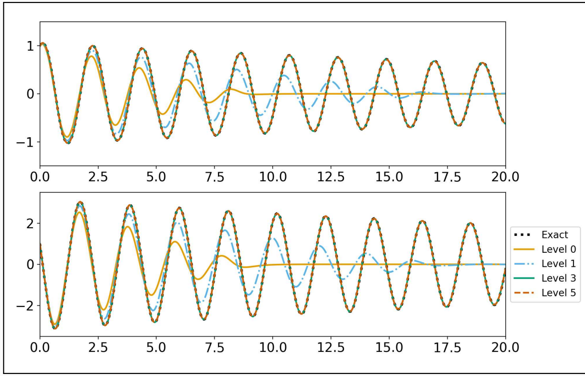
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

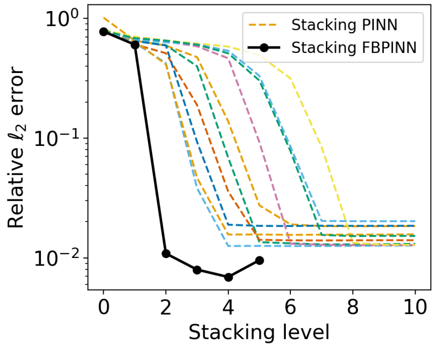
$$\frac{ds_1}{dt} = s_2,$$
$$\frac{ds_2}{dt} = -\frac{b}{m}s_2 - \frac{g}{L}\sin(s_1)$$

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



Exemplary partition of unity in time





Multifidelity Stacking FBPINNs – Pendulum Problem

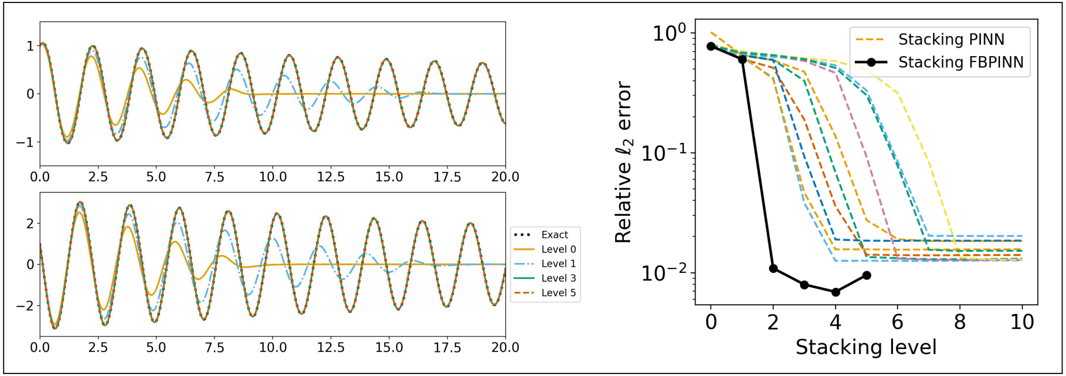
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

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

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



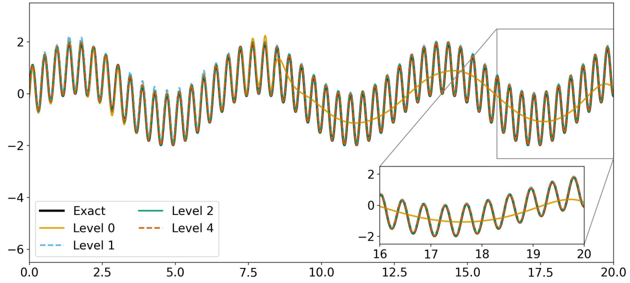
Multifidelity Stacking FBPINNs – Two-Frequency Problem

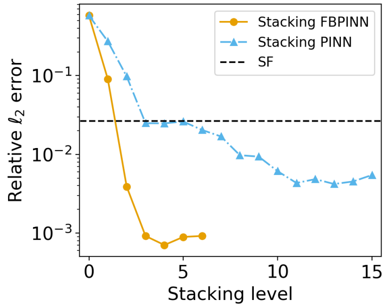
Second, we consider a **two-frequency problem**:

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

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



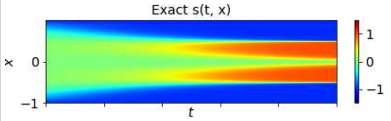


→ Due to the **multiscale structure of the problem**, the **improvements** due to the **multifidelity FBPINN approach** are **even stronger**.

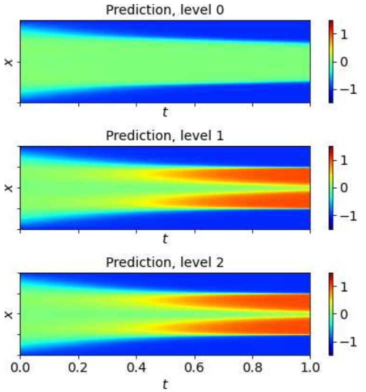
Multifidelity Stacking FBPINNs – Allen–Cahn Equation

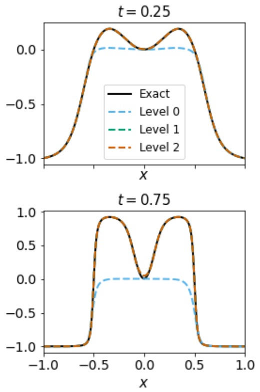
Finally, we consider the **Allen–Cahn equation**:

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



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)**.

Domain decomposition for randomized neural networks

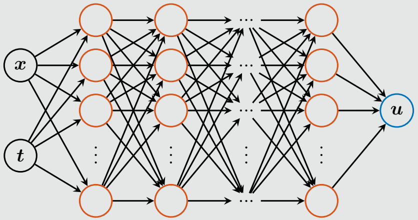
Randomized Neural Networks (RaNNs)

Neural networks

A standard **multilayer perceptron (MLP)** with L hidden layers is a **parametric** model of the form

$$u(\boldsymbol{x}, \boldsymbol{\theta}) = F_{L+1}^{\boldsymbol{A}} \cdot F_L^{\boldsymbol{W}_L, \boldsymbol{b}_L} \circ \ldots \circ F_1^{\boldsymbol{W}_1, \boldsymbol{b}_1}(\boldsymbol{x}),$$

where $\boldsymbol{A}$ is linear, and the ith hidden layer is nonlinear $F_i^{\boldsymbol{W}_i, \boldsymbol{b}_i}(\boldsymbol{x}) = \sigma(\boldsymbol{W}_i \cdot \boldsymbol{x} + \boldsymbol{b}_i)$.



In order to optimize the loss function

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}),$$

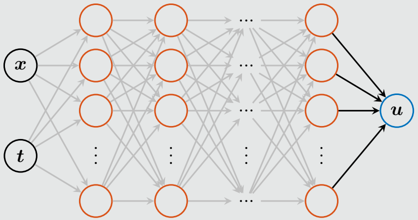
all parameters $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{W}_1, \boldsymbol{b}_1, \ldots, \boldsymbol{W}_L, \boldsymbol{b}_L)$ are **trained**.

Randomized neural networks

In **randomized neural networks (RaNNs)** as introduced by **Pao and Takefuji (1992)**,

$$u(\boldsymbol{x}, \boldsymbol{A}) = F_{L+1}^{\boldsymbol{A}} \cdot F_L^{W_L, b_L} \circ \ldots \circ F_1^{W_1, b_1}(\boldsymbol{x}),$$

the weights in the hidden layers are randomly initialized and **fixed**; only $\boldsymbol{A}$ is trainable.



The model is linear with respect to the trainable parameters $\boldsymbol{A}$, and the optimization problem reads

$$\min_{\boldsymbol{A}} \mathcal{L}(\boldsymbol{A}).$$

This can **simplify the training process**.

Physics-Informed Randomized Neural Networks (PIRaNNs)

Physics-informed randomized neural networks (PIRaNNs) make use of the aforementioned linearization of the model with respect to the trainable parameters as well as the fact that RaNNs retain **universal approximation properties**, as shown in **Igelnik and Pao (1995)**.

Consider a linear differential operator $\mathcal{A}$. Then, solving the PDE

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

using PIRaNNs yields the **linear equation system**

$$\mathcal{A}[u](\boldsymbol{x}_i) = f(\boldsymbol{x}_i), \quad i = 1, \ldots, N_{\text{PDE}},$$

where N_{PDE} is the number of **collocation points**.

Hard enforcement of boundary conditions

We construct u to explicitly satisfy bcs:

$$u(\boldsymbol{x}, \boldsymbol{A}) = G(\boldsymbol{x}) + L(\boldsymbol{x}) NN(\boldsymbol{x}, \boldsymbol{A})$$

- NN is a **feedforward neural network** with **trainable parameters $\boldsymbol{A}$**
- G and L are **fixed functions**, chosen such that u satisfies the boundary conditions

The resulting linear equation system

$$\boldsymbol{HA} = \boldsymbol{f}$$

generally does **not have a unique solution**. In fact, $\boldsymbol{H}$ is typically **rectangular**, **dense**, and **ill-conditioned**.

Solving the system using least squares corresponds to applying the **classical PINN loss function to the RaNN model** u. As we will see, this approach offers a **potentially efficient alternative**.

Randomized Neural Networks – (Non-Exhaustive) Literature Overview

Randomized neural networks

- **RaNNs**: Pao, Takefuji (1992); Pao Park, Sobajic (1994); Igelnik, Pao (1995)
- **Extreme Learning Machines (ELMs)**: Huang, Zhu, Siew (2006); Liu, Lin, Fang, Xu (2014); Gallicchio, Scardapane (2020); Calabrò, Fabiani, Siettos (2021); Ni, Dong (2023); Wang, Dong (2024)

Domain decomposition for neural networks and randomized neural networks

- **cPINNs, XPINNs**: Jagtap, Kharazmi, Karniadakis (2020); Jagtap, Karniadakis (2020)
- **Schwarz it. for PINNs or DeepRitz (D3M, DeepDDM, etc)**:: Li, Tang, Wu, Liao (2019); Li, Xiang, Xu (2020); Mercier, Gratton, Boudier (arXiv 2021); Dolean, H., Mercier, Gratton (arXiv 2024); 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 (acc. 2024); 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 (subm. 2024)

An overview of the state-of-the-art in early 2021:

A. Heinlein, A. Klawonn, M. Lanser, J. Weber

Combining machine learning, 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, domain decomposition methods – a survey

Computational Science, Engineering. 2024

Domain Decomposition-Based PIRaNNs

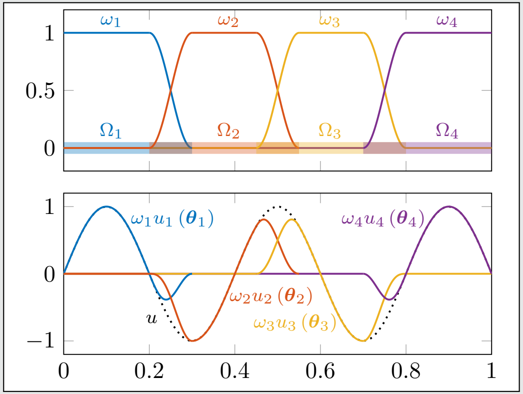
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

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**

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

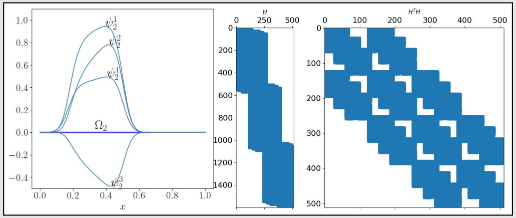


Domain decomposition for RaNNs

We employ the FBPINNs approach; cf. **Shang, Heinlein, Mishra, Wang (subm. 2024)**. This is closely related to the **random feature method (RFM)** by **Chen, Chi, E, Yang (2022)**. In particular, we solve

$$\mathcal{A}\left[\sum_{j=1}^{J} \omega_j u_j(\boldsymbol{A}_j)\right](\boldsymbol{x}_i) = f(\boldsymbol{x}_i),$$

for $i = 1, \ldots, N_{\text{PDE}}$; the boundary condtions are incorporated directly into the u_j.

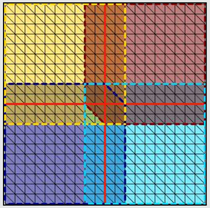


The hidden weights are randomly initialized, the resulting matrices $\boldsymbol{H}$ and $\boldsymbol{H}^\top \boldsymbol{H}$ are block-sparse.

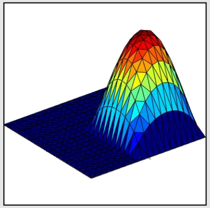
Preconditioning for Domain Decomposition-Based PIRaNNs

One-level Schwarz preconditioner

Overlap $\delta = 1h$



Solution of local problem



Based on an **overlapping domain decomposition**, we define a **one-level Schwarz operator** for $\boldsymbol{K} := \boldsymbol{H}^\top \boldsymbol{H}$

$$\boldsymbol{M}_{\text{OS-1}}^{-1}\boldsymbol{K} = \sum\nolimits_{i=1}^{N} \boldsymbol{R}_i^\top \boldsymbol{K}_i^{-1}\boldsymbol{R}_i\boldsymbol{K},$$

where $\boldsymbol{R}_i$ and $\boldsymbol{R}_i^\top$ are restriction and prolongation operators corresponding to Ω_i', and $\boldsymbol{K}_i := \boldsymbol{R}_i\boldsymbol{K}\boldsymbol{R}_i^\top$.

Here, the matrix $\boldsymbol{K}_i$ could be singular in which case we use a **pseudo inverse** $\boldsymbol{K}_i^+$ instead of $\boldsymbol{K}_i^{-1}$.

We also consider restricted and scaled additive Schwarz preconditioners; cf. **Cai, Sarkis (1999)**.

Singular Value Decomposition

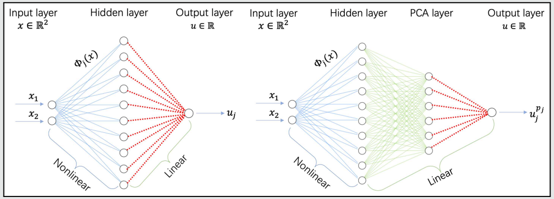
As discussed before, on each subdomain Ω_j, the RaNN is

$$\begin{aligned} u_j(\boldsymbol{x}, \boldsymbol{A}_j) &= F_{L+1}^{\boldsymbol{A}} \cdot F_L^{W_L,b_L} \circ \ldots \circ F_1^{W_1,b_1}(\boldsymbol{x}) \\ &= \boldsymbol{A}_j \begin{bmatrix} \Phi_1(\boldsymbol{x}) & \cdots & \Phi_k(\boldsymbol{x}) \end{bmatrix}^\top, \end{aligned}$$

where k is the width of the last hidden layer and the $\boldsymbol{\Phi}_k$ are the randomized basis functions.
Consider a **reduced SVD** $\boldsymbol{\Phi} = \boldsymbol{U\Sigma V}^\top$, where the entries of the matrix are $\boldsymbol{\Phi}_{i,k} = \Phi_k(\boldsymbol{x}_i)$. Then, we consider

$$\hat{u}_j(\boldsymbol{x}, \boldsymbol{A}_j) = \boldsymbol{A}_j\hat{\boldsymbol{V}}^\top \begin{bmatrix} \Phi_1(\boldsymbol{x}) & \cdots & \Phi_k(\boldsymbol{x}) \end{bmatrix}^\top,$$

where $\hat{\boldsymbol{V}}^\top$ is obtained by omitting the right singular vectors corresponding to small singular values.



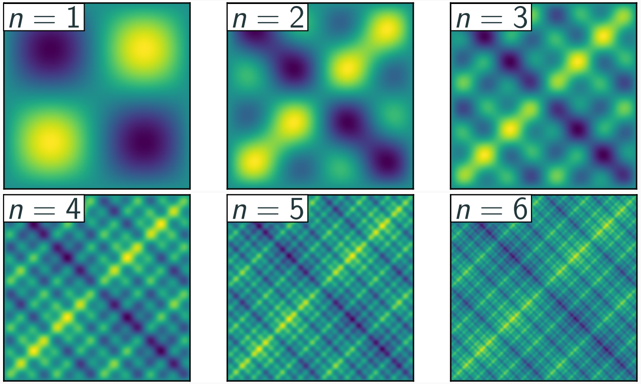
Multi-Frequency Problem

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

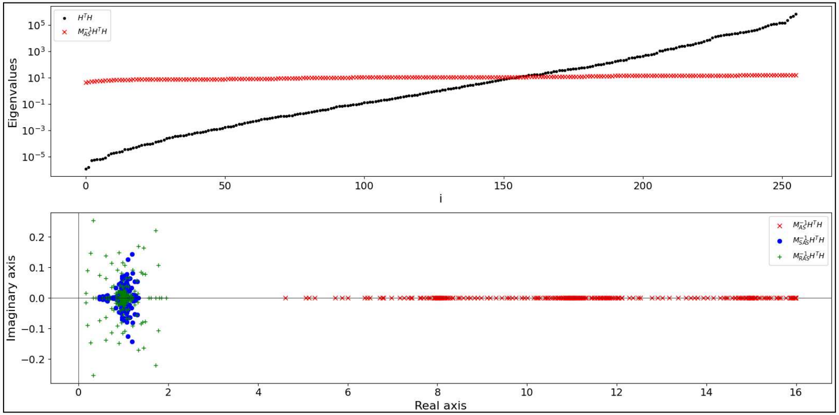
$$\begin{aligned} -\Delta u &= 2 \sum_{i=1}^{n} (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y) && \text{in } \Omega = [0,1]^2, \\ u &= 0 && \text{on } \partial\Omega, \end{aligned}$$

with $\omega_i = 2^i$.

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



Numerical Results for the Multi-Frequency Problem ($n = 2$)



	$M^{-1} = I$		$M^{-1} = M_{AS}^{-1}$		$M^{-1} = M_{RAS}^{-1}$		$M^{-1} = M_{SAS}^{-1}$	
	iter	e_{L^2}	iter	e_{L^2}	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	$5.04 \cdot 10^{-3}$	24	$5.03 \cdot 10^{-3}$	6	$5.04 \cdot 10^{-3}$
BICG	> 2000	$1.03 \cdot 10^{-2}$	8	$5.08 \cdot 10^{-3}$	32	$5.05 \cdot 10^{-3}$	11	$5.09 \cdot 10^{-3}$
GMRES	> 2000	$8.68 \cdot 10^{-2}$	13	$5.07 \cdot 10^{-3}$	31	$5.06 \cdot 10^{-3}$	11	$5.08 \cdot 10^{-3}$

4×4 subdomains; DoF $= 256$; $N = 1600$; $\boldsymbol{\theta}^0 \in \mathcal{U}(-1,1)$; stop.: $\|\boldsymbol{M}^{-1}\boldsymbol{r}^k\|_{L^2} / \|\boldsymbol{M}^{-1}\boldsymbol{r}^0\|_{L^2} \leq 10^{-5}$

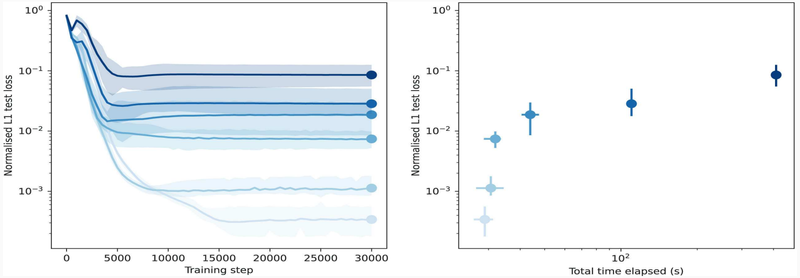
Numerical Results for the Multi-Frequency Problem ($n = 2$) – Effect of the SVD

We now investigate the effect of omitting right singular vectors associated with singular values below a varying tolerance τ.

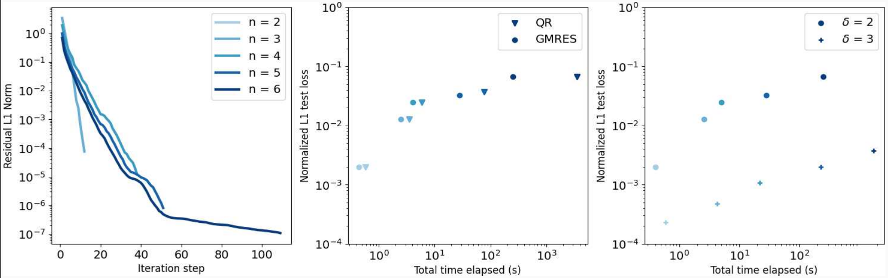
τ	DoF	M^{-1}	σ_{min}	σ_{max}	iter	e_{L^2}
10^{-4}	512	none	10^{-10}	10^{6}	> 2000	3.72e-2
		M_{AS}^{-1}	10^{-6}	10^{6}	27	5.46e-5
		M_{SAS}^{-1}	10^{-7}	10^{5}	30	5.49e-5
10^{-3}	436	none	10^{-8}	10^{5}	> 2000	3.75e-2
		M_{AS}^{-1}	10^{-5}	10^{5}	16	1.28e-4
		M_{SAS}^{-1}	10^{-6}	10^{4}	18	1.28e-4
10^{-2}	335	none	10^{-5}	10^{5}	> 2000	4.51e-2
		M_{AS}^{-1}	10^{-3}	10^{4}	14	7.14e-4
		M_{SAS}^{-1}	10^{-4}	10^{3}	13	7.11e-4
10^{-1}	212	none	10^{-3}	10^{6}	> 2000	5.01e-2
		M_{AS}^{-1}	10^{-2}	10^{3}	12	7.13e-3
		M_{SAS}^{-1}	10^{-3}	10^{2}	11	7.10e-3

4×4 subdomains; $N = 1600$; $\boldsymbol{\theta}^0 \in \mathcal{U}(-1,1)$; stop.: $\|\boldsymbol{M}^{-1}\boldsymbol{r}^k\|_{L^2}/\|\boldsymbol{M}^{-1}\boldsymbol{r}^0\|_{L^2} \leq 10^{-5}$

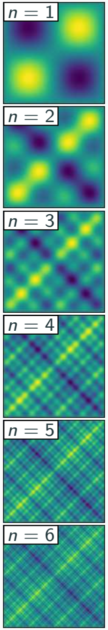
Numerical Results for the Multi-Frequency Problem



Multi-level FBPINNs; cf. **Dolean, Heinlein, Mishra, Moseley (2024)**



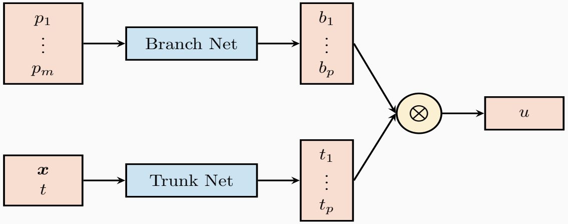
DD-PIRaNNs; cf. **Shang, Heinlein, Mishra, Wang (subm. 2024)**



Domain decomposition-based physics-informed deep operator networks

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

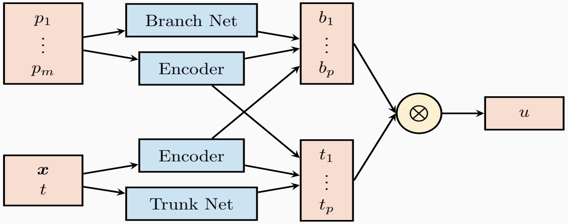
DeepONets are **compatible with the PINN approach** but **physics-informed DeepONets (PI-DeepONets)** are challenging to train.

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)**

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)**.



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

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

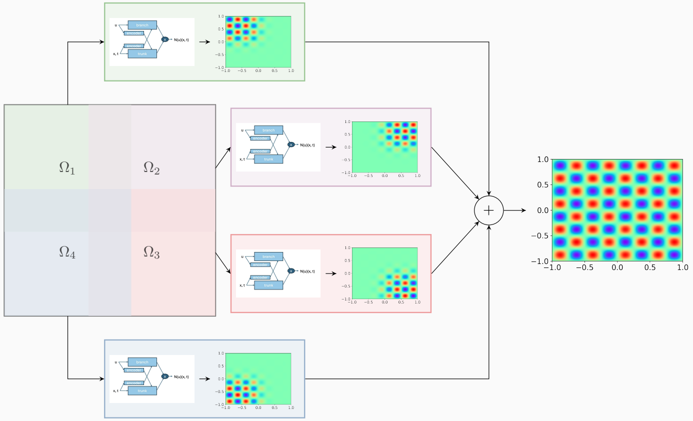
Physics-informed DeepONets

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)**

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)

FBDONs – Pendulum

Pendulum problem

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

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

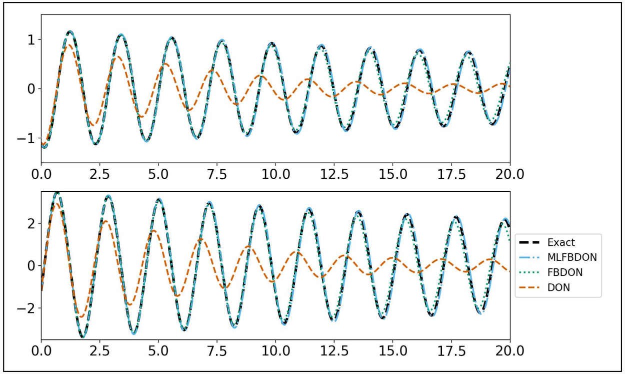
Parametrization

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.)**

FBDONs – Wave Equation

Wave equation

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

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

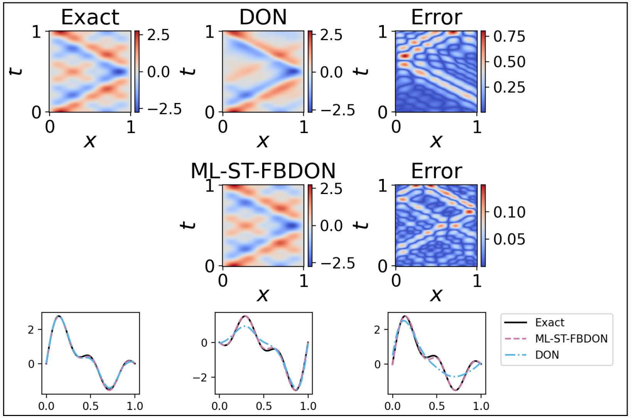
Solution: $s(x,t) = \sum_{n=1}^{5} b_n \sin(n\pi x)\cos(n\pi\sqrt{2}t)$

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.

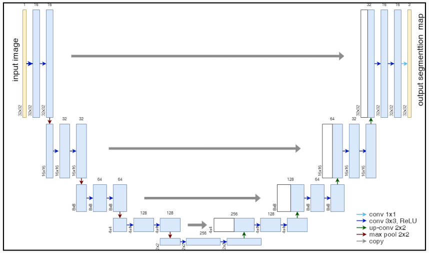


Mean rel. l_2 error on 100 config.	
DeepONet	0.30 ± 0.11
ML-ST-FBDON ([1, 4, 8, 16] subd.)	0.05 ± 0.03
ML-FBDON ([1, 4, 8, 16] subd.)	0.08 ± 0.04

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

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

Domain Decomposition-Based U-Net Architecture

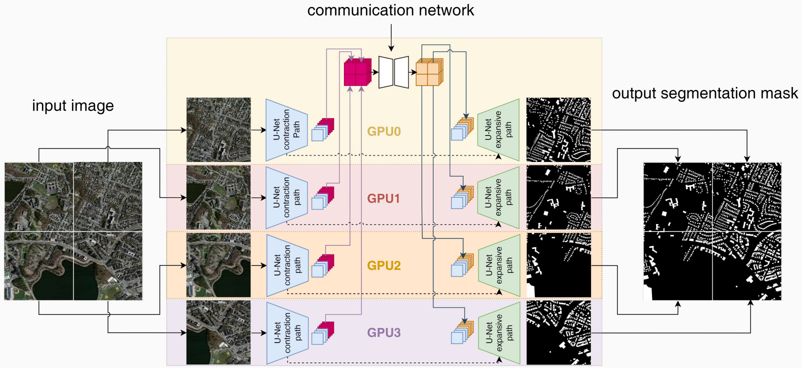


name	mem. feature maps # of values	mem. feature maps MB	mem. weights # of values	mem. weights MB
input block	268 M	1 024.0	38 848	**0.148**
encoder blocks	314 M	1 320	18 M	**72**
decoder blocks	754 M	3880	12 M	**47**
output block	3.1 M	12.0	195	**0.001**

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

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

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



CWI Research Semester: Synergies in Numerical Linear Algebra and Machine Learning

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)
- **Workshop** (December 1–3, 2025):
  - 3 days with plenary talks (academia & industry) and an industry panel
  - Confirmed plenary speakers:
    - Marta d'Elia (Meta)
    - Benjamin Peherstorfer (New York University)
    - Andreas Roskopf (Fraunhofer Institute)

CWI
Centrum Wiskunde & Informatica

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

Summary

Multilevel Finite Basis Physics Informed Neural Networks (ML-FBPINNs)

- **Schwarz domain decomposition architectures** improve the scalability of PINNs to **large domains / high frequencies**, keeping the complexity of the local networks low.
- As classical domain decomposition methods, **one-level FBPINNs** are not scalable to large numbers of subdomains; **multilevel FBPINNs** enable scalability.

Extensions to Stacking Multifidelity PINNs, RaNNs, and DeepONets

- **Multifidelity stacking PINNs with FBPINNs** improve accuracy and efficiency for **time-dependent problems**.
- **RaNNs** reduce computational cost but face ill-conditioning, mitigated by **Schwarz preconditioning** and **SVD**.
- **DeepONets** provide efficient predictions for **parametrized problems** but struggle with **multiscale problems**. **Domain decomposition** improves scalability and performance.

Thank you for your attention!

Topical Activity Group

Scientific Machine Learning