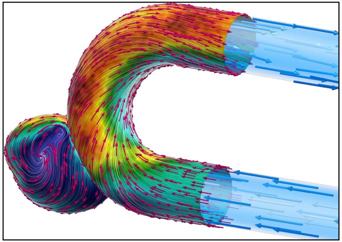
Domain decomposition for physics-informed neural networks

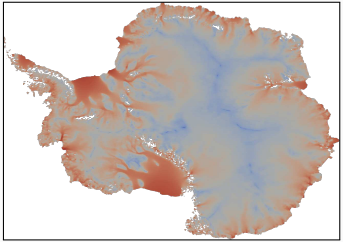
Alexander Heinlein[1]

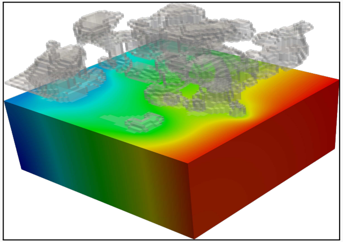
Seminar on Current Problems in Numerical Analysis, Czech Academy of Sciences, Prague, Czech Republic, May 24, 2024

[1]Delft University of Technology

Scientific Machine Learning in Computational Science and Engineering







Numerical methods

Based on physical models

+ Robust and generalizable
− Require availability of mathematical models

Machine learning models

Driven by data

+ Do not require mathematical models
− Sensitive to data, limited extrapolation capabilities

Scientific machine learning (SciML)

Combining the strengths and **compensating the weaknesses** of the individual approaches:

numerical methods **improve** machine learning techniques
machine learning techniques **assist** numerical methods

Outline

1. Physics-informed machine learning & motivation

2. Deep learning-based domain decomposition method

   Based on joint work with

   Victorita Dolean (TU Eindhoven)

   Serge Gratton and **Valentin Mercier** (IRIT Computer Science Research Institute of Toulouse)

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

   Based on joint work with

   Victorita Dolean (University of Strathclyde, University Côte d'Azur)

   Ben Moseley and **Siddhartha Mishra** (ETH Zürich)

4. Multifidelity domain decomposition-based physics-informed neural networks for time-dependent problems

   Based on joint work with

   Damien Beecroft (University of Washington)

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

Physics-informed machine learning & motivation

Artificial Neural Networks for Solving Ordinary and Partial Differential Equations

Isaac Elias Lagaris, Aristidis Likas, *Member, IEEE,* and Dimitrios I. Fotiadis

Published in IEEE Transactions on Neural Networks, Vol. 9, No. 5, 1998.

Approach

Solve a general differential equation subject to boundary conditions

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

by solving an **optimization problem**

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

where $\Psi_t\left(\boldsymbol{x}, \theta\right)$ is a **trial function**, $\boldsymbol{x}_i$ **sampling points inside the domain** Ω and θ are **adjustable parameters**.

Construction of the trial functions

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

$$\Psi_t(\boldsymbol{x}, \theta) = A(\boldsymbol{x}) + F(\boldsymbol{x}, \mathrm{NN}(\boldsymbol{x}, \theta))$$

- NN is a **feedforward neural network** with **trainable parameters** θ and input $x \in \mathbb{R}^n$
- A and F are **fixed functions**, chosen s.t.:
  - A **satisfies the boundary conditions**
  - F **does not contribute to the boundary conditions**

Earlier related work: **Dissanayake & Phan-Thien (1994)**

Approach

Solve a general differential equation subject to boundary conditions

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

by solving an **optimization problem**

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

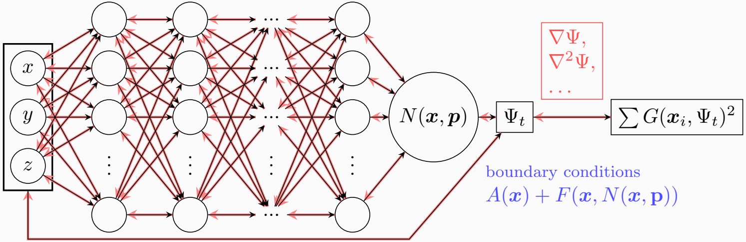
where $\Psi_t\left(\boldsymbol{x}, \boldsymbol{\theta}\right)$ is a **trial function**, $\boldsymbol{x}_i$ **sampling points inside the domain** Ω and $\boldsymbol{\theta}$ are **adjustable parameters**.

Construction of the trial functions

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

$$\Psi_t(\boldsymbol{x}, \boldsymbol{\theta}) = A(\boldsymbol{x}) + F(\boldsymbol{x}, \mathrm{NN}(\boldsymbol{x}, \boldsymbol{\theta}))$$

- NN is a **feedforward neural network** with **trainable parameters** $\boldsymbol{\theta}$ and input $x \in \mathbb{R}^n$
- A and F are **fixed functions**, chosen s.t.:
  - A **satisfies the boundary conditions**
  - F **does not contribute to the boundary conditions**



Lagaris et. al's Method – Motivation

Solve the **boundary value problem**

$$\Delta \Psi_t(\boldsymbol{x}, \boldsymbol{\theta}) + 1 = 0 \text{ on } [0,1],$$
$$\Psi_t(0, \boldsymbol{\theta}) = \Psi_t(1, \boldsymbol{\theta}) = 0,$$

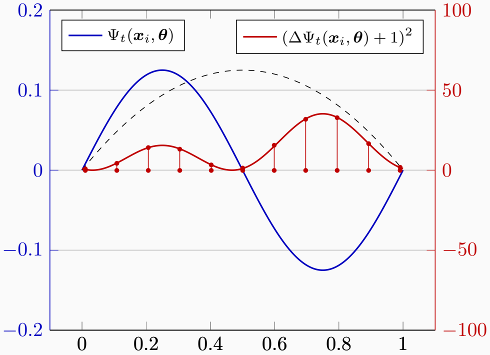
via a **collocation approach**:

$$\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}_i} \left(1 - \Delta \Psi_t(\boldsymbol{x}_i, \boldsymbol{\theta})\right)^2$$

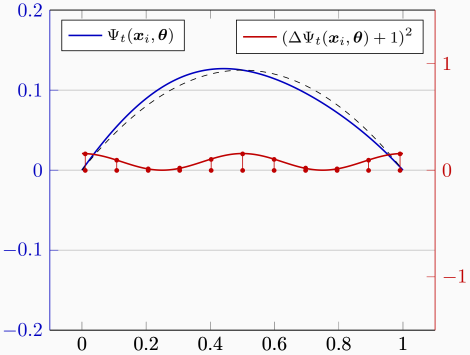
Boundary conditions

The boundary conditions can be **enforced explicitly**, for instance, via the ansatz:

$$\Psi_t(\boldsymbol{x}, \boldsymbol{\theta}) = \sin(\pi \boldsymbol{x}) \cdot F(\boldsymbol{x}, \mathrm{NN}(\boldsymbol{x}, \boldsymbol{\theta}))$$



$(\Delta \Psi_t(\boldsymbol{x}_i, \boldsymbol{\theta}) + 1)^2 >> 0$



$(\Delta \Psi_t(\boldsymbol{x}_i, \boldsymbol{\theta}) + 1)^2 \approx 0$

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**

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

It is based on the approach by **Lagaris et al. (1998)**. The main novelty of PINNs is the use of 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

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

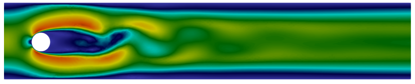
$$\mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}) = \frac{1}{N_{\text{PDE}}} \sum\nolimits_{i=1}^{N_{\text{PDE}}} \left(\mathcal{N}[u](\mathbf{x}_i, \boldsymbol{\theta}) - f(\mathbf{x}_i) \right)^2.$$

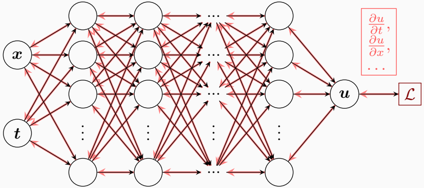
Advantages

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

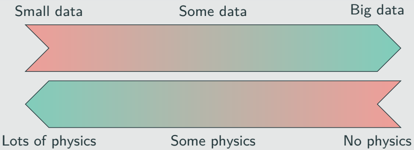
Drawbacks

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

Available Theoretical Results for PINNs – An Example

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

Estimate of the generalization error

The generalization error (or total error) satisfies

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

where

- $\mathcal{E}_G = \mathcal{E}_G(\boldsymbol{X}, \boldsymbol{\theta}) := \|\mathbf{u} - \mathbf{u}^*\|_V$ **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** respectively the **quadrature** as well as on the **neural network**

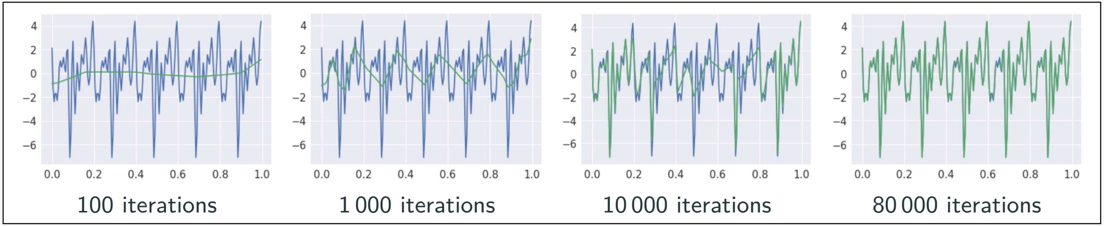
Rule of thumb:

"As long as the PINN is trained well, it also generalizes well"

Scaling Issues in Neural Network Training

Spectral bias

Neural networks **prioritize learning lower frequency functions** first irrespective of their amplitude.



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

- Solving solutions on **large domains and/or with multiscale features** potentially requires **very large neural networks**.
- Training may **not sufficiently reduce the loss** or take **large numbers of iterations**.
- Significant **increase on the computational work**

Dependence on the choice of **activation functions**: **Hong et al. (arXiv 2022)**

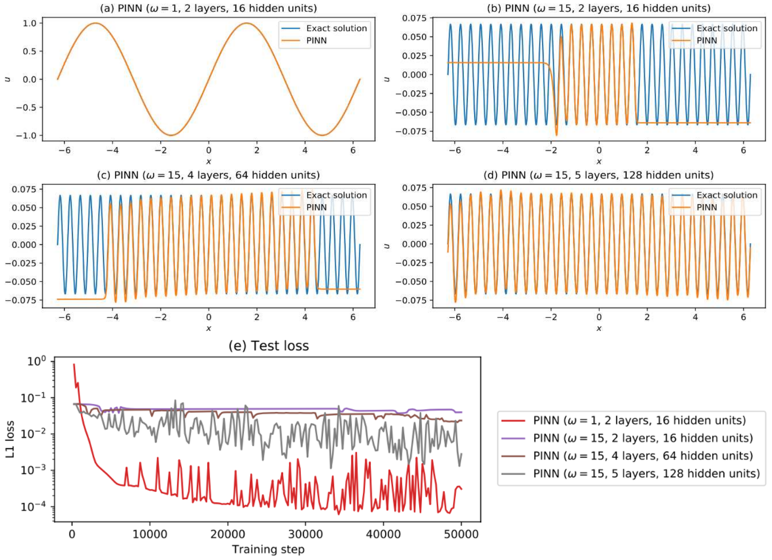
Convergence analysis of PINNs via the **neural tangent kernel**: **Wang, Yu, Perdikaris,** *When and why PINNs fail to train: A neural tangent kernel perspective*, **JCP (2022)**

Motivation – Some Observations on the Performance of PINNs

Solve

$$u' = \cos(\omega \boldsymbol{x}),$$
$$u(0) = 0,$$

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

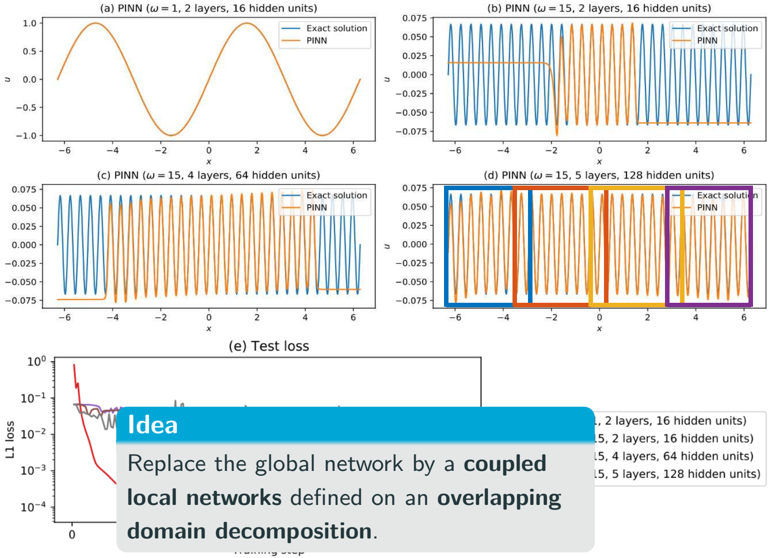
Cf. **Moseley, Markham, and Nissen-Meyer (2023)**

Motivation – Some Observations on the Performance of PINNs

Solve

$$u' = \cos(\omega \boldsymbol{x}),$$
$$u(0) = 0,$$

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



(a) 321 free parameters

(d) 66 433 free parameters

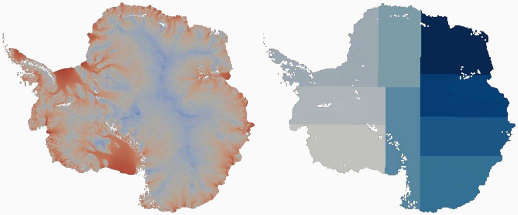
Scaling issues

- Large computational domains
- Small frequencies

Idea

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

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



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

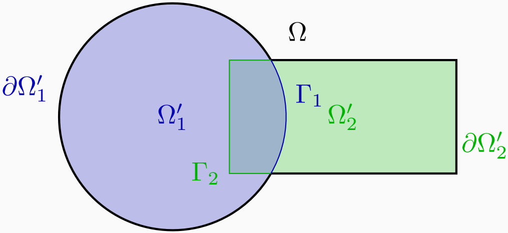
Historical remarks: The **alternating Schwarz method** is the earliest **domain decomposition method (DDM)**, which has been invented by **H. A. Schwarz** and published in **1870**:

- 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**



DDM-Based Approaches for Neural Network-Based Discretizations – Literature

A **non-exhaustive literature overview**:

- **cPINNs**: **Jagtap, Kharazmi, Karniadakis (2020)**
- **XPINNs**: **Jagtap, Karniadakis (2020)**
- **D3M**: **Li, Tang, Wu, and Liao (2019)**
- **DeepDDM**: **Li, Xiang, Xu (2020); Mercier, Gratton, Boudier (arXiv 2021); Li, Wang, Cui, Xiang, Xu (2023); Sun, Xu, Yi (arXiv 2022, arXiv 2023)**
- **Schwarz Domain Decomposition Algorithm for PINNs**: **Kim, Yang (2022, arXiv 2023)**
- **FBPINNs**: **Moseley, Markham, and Nissen-Meyer (2023); Dolean, Heinlein, Mishra, Moseley (2024, subm. 2023 / arXiv:2306.05486); Heinlein, Howard, Beecroft, Stinis (subm. 2024 / arXiv:2401.07888)**

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 the end of 2023:

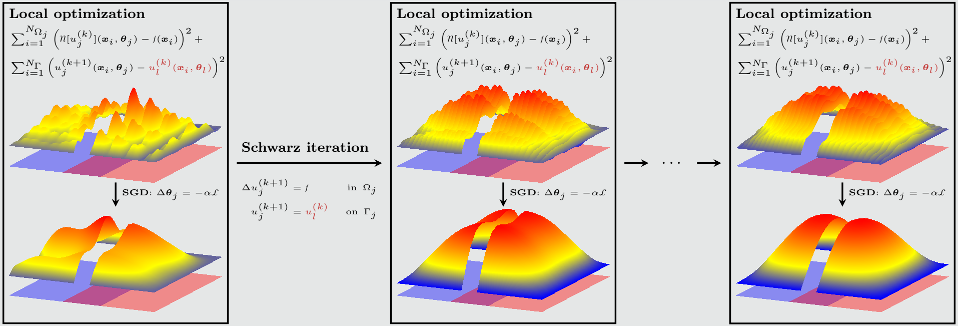
A. Klawonn, M. Lanser, J. Weber

Machine learning and domain decomposition methods – a survey

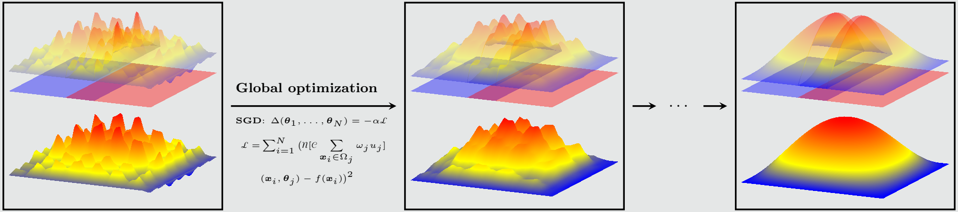
arXiv:2312.14050. 2023

Combining Schwarz Methods with Neural Network-Based Discretizations

Approach 1 – Via a classical Schwarz iteration



Approach 2 – Integration via the neural network architecture



Approach 1
Deep learning-based domain decomposition method

Deep Learning-Based Domain Decomposition Method (DeepDDM)

Li, Xiang, Xu. *Deep domain decomposition method: Elliptic problems.* **PMLR (2020)**

DeepDDM for Overlapping Schwarz

In the **DeepDDM method**, we train **local networks** u_j using a **local loss function** on each subdomain Ω_j

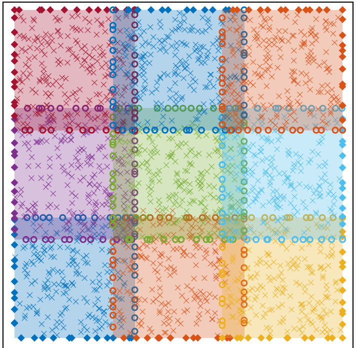
$$\mathcal{L}_j(\boldsymbol{\theta}_j) := \mathcal{L}_{\Omega_j}(\boldsymbol{\theta}_j) + \mathcal{L}_{\partial\Omega_j \setminus \Gamma_j}(\boldsymbol{\theta}_j) + \mathcal{L}_{\Gamma_j}(\boldsymbol{\theta}_j),$$

with **volume, boundary, and interface jump terms**:

$$\mathcal{L}_{\Omega_j}(\boldsymbol{\theta}_j) := \frac{1}{N_{f_j}} \sum_{i=1}^{N_{f_j}} \left(\mathcal{N}(u_j(\boldsymbol{x}_i, \boldsymbol{\theta}_j)) - f(\boldsymbol{x}_i) \right)^2$$

$$\mathcal{L}_{\partial\Omega_j \setminus \Gamma_j}(\boldsymbol{\theta}_j) := \frac{1}{N_{g_j}} \sum_{i=1}^{N_{g_j}} \left(\mathcal{B}(u_j(\hat{\boldsymbol{x}}_i, \boldsymbol{\theta}_j)) - g(\hat{\boldsymbol{x}}_i) \right)^2$$

$$\mathcal{L}_{\Gamma_j}(\boldsymbol{\theta}_j) := \frac{1}{N_{\Gamma_j}} \sum_{i=1}^{N_{\Gamma_j}} \left(\mathcal{D}(u_j(\tilde{\boldsymbol{x}}_i, \boldsymbol{\theta}_j)) - \mathcal{D}(u_l(\tilde{\boldsymbol{x}}_i, \boldsymbol{\theta}_j)) \right)^2$$



Algorithm 1: DeepDDM for Ω_j

Data: Sampling points X_j, initial network parameters $\boldsymbol{\theta}_j^0$

while *convergence (local network & interface values) not reached* **do**

  Train local network u_j;

  Communicate & **update** interface values $\mathcal{D}\left(u_l(\tilde{\boldsymbol{x}}_i; \boldsymbol{\theta}_j)\right)$ from other subdomains Ω_l;

end

Numerical Experiments

Strong scaling

Fix the **problem complexity** & **increase** the **model capacity**.

Optimal scaling: improving the convergence rate and/or accuracy at the same rate as the increase of model capacity.

Weak scaling

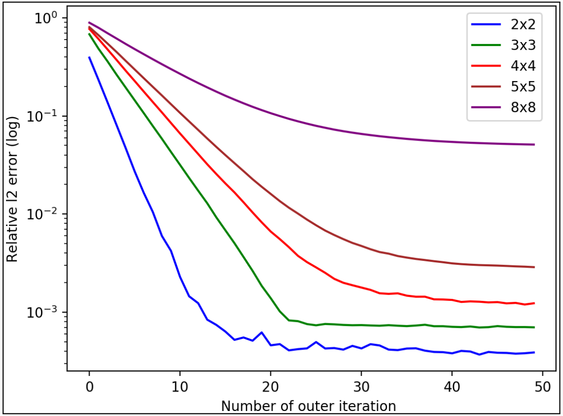
Increase the **problem complexity** & the **model capacity** at the same rate.

Optimal scaling: constant convergence rate and/or accuracy to stay approximately constant.

Let first consider a **strong scaling study** for a **two-dimensional Laplacian model problem**:

$$-\Delta u = 1 \quad \text{in } \Omega,$$
$$u = 0 \quad \text{on } \partial\Omega.$$

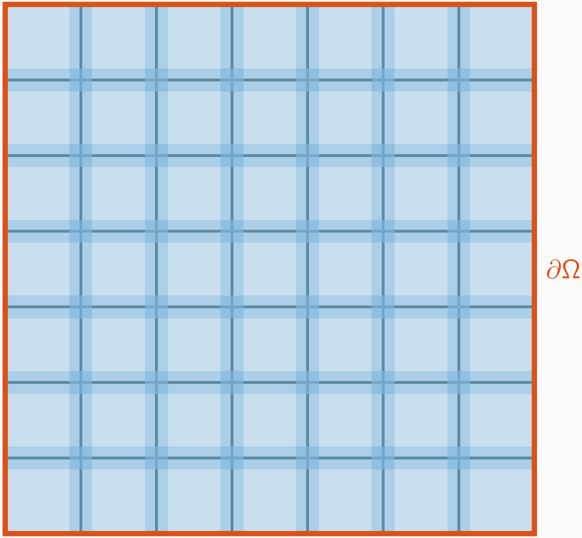
We increase the model capacity by **increasing the number of subdomains**.



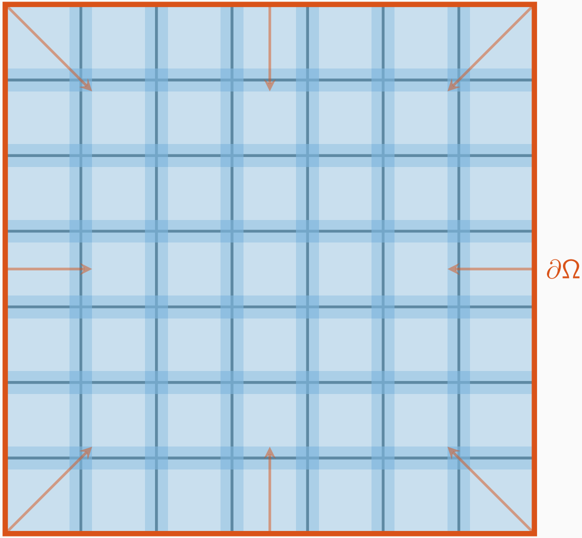
Scaling issue

We observe that the performance of the DeepDDM method **deteriorates**.

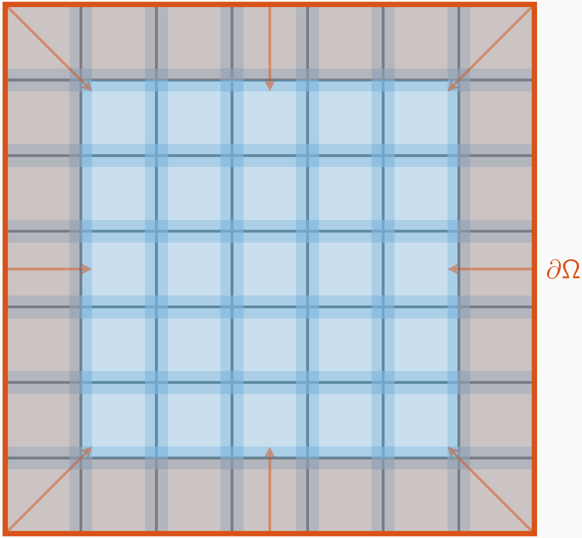
Transport of Information One-Level Overlapping Schwarz Methods



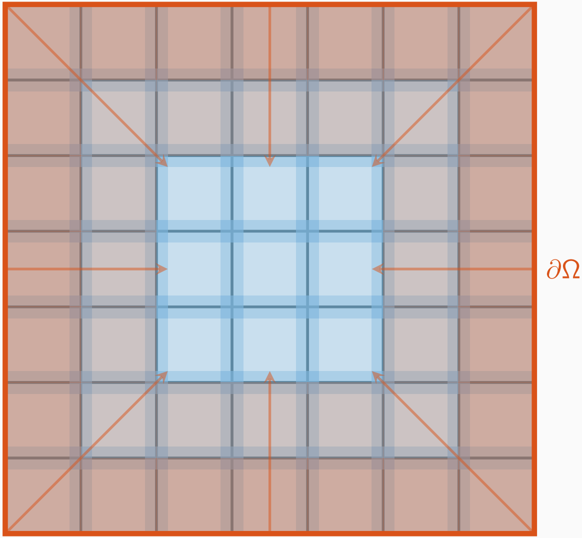
Transport of Information One-Level Overlapping Schwarz Methods



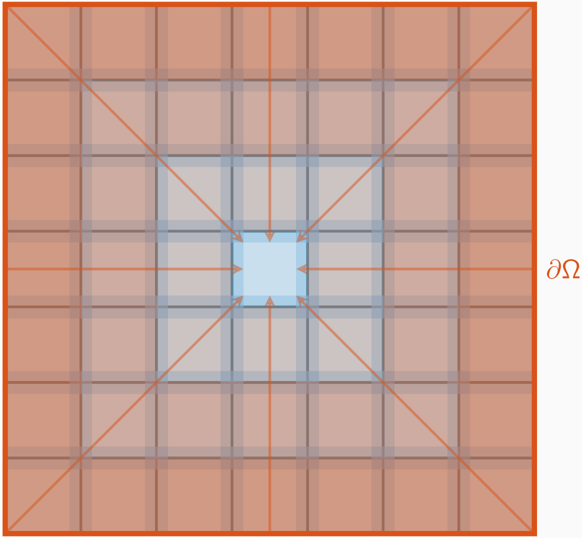
Transport of Information One-Level Overlapping Schwarz Methods



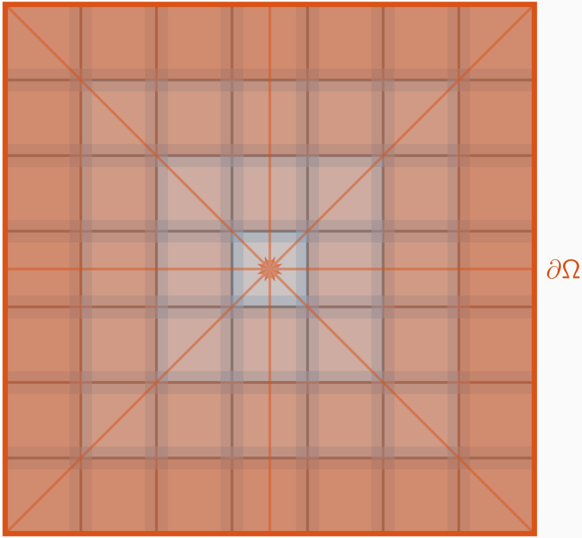
Transport of Information One-Level Overlapping Schwarz Methods



Transport of Information One-Level Overlapping Schwarz Methods



Transport of Information One-Level Overlapping Schwarz Methods



Information (in particular, boundary data) is **only exchanged via the overlapping regions**, leading to **slow convergence** $\rightarrow$ establish a **faster / global transport of information**.

Fast Transport of Information via a Coarse Level

Coarse space for the DeepDDM method

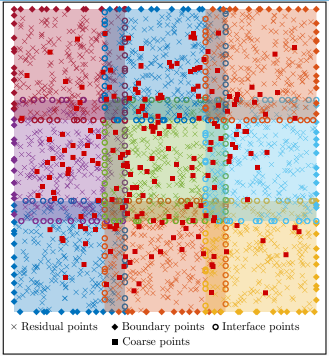
- Sparse sampling $\boldsymbol{X}_0 = \left\{\boldsymbol{x}_i^0\right\}_i$ over the whole domain Ω
- Train a coarse network (global PINN) u_0 with **additional loss term**

$$\lambda_f \frac{1}{N_0} \sum_{\boldsymbol{x}_i^0 \in \boldsymbol{X}_0} \left(u_0(\boldsymbol{x}_i^0) - \sum_{j=1}^{J} E_j(\chi_j u_j(\boldsymbol{x}_i^0))\right)^2$$

  for **incorporating information from the first level**. Here,
  - E_j **extension by zero** outside Ω_j
  - χ_j local **partition of unity function**
- **Incorporate coarse information** into the loss for the local subdomain Ω_j:

$$\frac{1}{N_{\Gamma_j}} \sum_{i=1}^{N_{\Gamma_j}} \left(\mathscr{D}\left(u_j\left(\tilde{\boldsymbol{x}}_i, \boldsymbol{\theta}_j\right)\right) - W_j^i\right)^2$$

  with $W_f^i = \mathscr{D}(\lambda_c u_l(\tilde{\boldsymbol{x}}_i) + (1-\lambda_c) u_0(\tilde{\boldsymbol{x}}_i))$.



Algorithm 2: Two-level DeepDDM

Data: X_j, X_0, θ_j^0, λ_f, and λ_c

while *conv. (local & interface) not reached* **do**

  Train local network u_j;

  Comm. & comp. $\sum_{j=1}^{J} E_j(\chi_j u_j(\boldsymbol{x}_i^0))\ \forall \boldsymbol{x}_i^0 \in \boldsymbol{X}_0$;

  Train coarse network u_0;

  Comm. & update $\mathscr{D}\left(u_l(\tilde{\boldsymbol{x}}_i; \boldsymbol{\theta}_j)\right)\ \forall \Omega_l \cap \Omega_j \neq \emptyset$;

end

2D Poisson Equation – Problem Setup

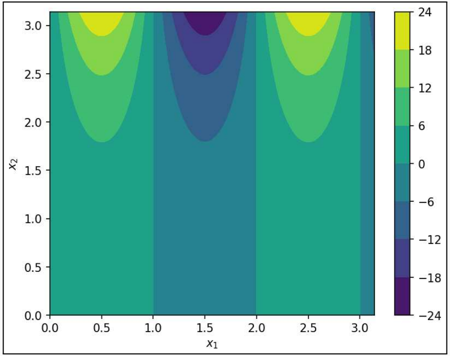
Model problem:

$$\Delta u = f \quad \text{in } \Omega = [0,\pi] \times [0,1],$$
$$u = g \quad \text{on } \partial\Omega.$$

We choose f and g such that the exact solution is

$$u(\boldsymbol{x}) = \sin(\alpha \pi x_1) e^{x_2},$$

where α is an integer.



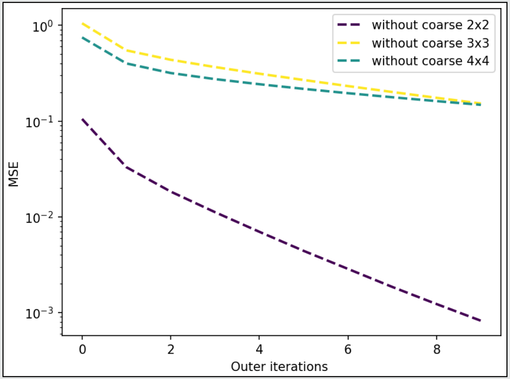
Training setup – Strong scaling

- Latin hypercube sampling for training points with $N_\Omega = 30\,000$ and $N_{\partial\Omega} = N_\Gamma = 16\,000$.
- Each network is composed of two hidden layers with 30 neurons
- Optimization of local/coarse networks: 2500 epochs using the Adam optimizer with initial learning rate $2 \cdot 10^{-4}$ and exp. decay of 0.999 every 100 epochs.
- Codes implemented in TensorFlow2 (v2.2.0) run on a single NVIDIA GeForce GTX 1080 Ti.
- The overlap is set to 30% of the subdomain diameter

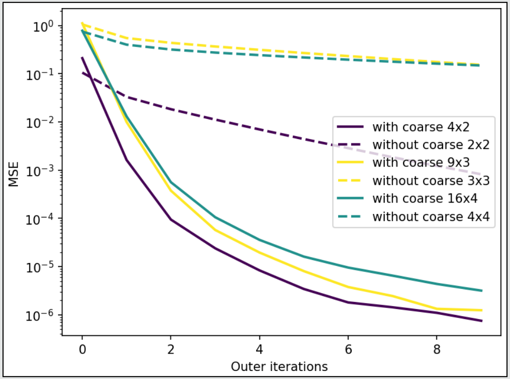
2D Poisson Equation – Weak Scaling

Increasing the frequency while **increasing the number of subdomains**.

One-level DeepDDM



Two-level DeepDDM



→ **Adding a coarse level fixes the scaling issue**.

Approach 2

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

Finite Basis Physics-Informed Neural Networks (FBPINNs)

In the **finite basis physics informed neural network (FBPINNs) method** introduced in **Moseley, Markham, and Nissen-Meyer (2023)**, we solve the **partial differential equation**

$$\mathcal{N}[u](\mathbf{x}) = f(\mathbf{x}) \quad \text{in } \Omega$$

using the **PINN** approach and **hard enforcement of the boundary conditions**, similar to **Lagaris et al. (1998)**.

FBPINNs use the **network architecture**

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

and the **loss function**

$$\mathcal{L}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \frac{1}{N} \sum_{i=1}^{N} \Big(\mathcal{N}[\mathcal{C} \sum_{\mathbf{x}_i \in \Omega_j} \omega_j u_j](\mathbf{x}_i, \boldsymbol{\theta}_j) - f(\mathbf{x}_i) \Big)^2.$$

- **Overlapping DD**: $\Omega = \bigcup_{l=1}^{J} \Omega_j$
- **Window functions** ω_j with $\text{supp}(\omega_j) \subset \Omega_j$ and $\sum_{j=1}^{J} \omega_j \equiv 1$ on Ω

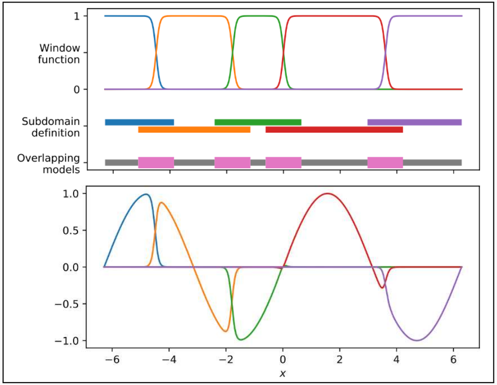
Hard enf. of boundary conditions

Loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum\nolimits_{i=1}^{N} (\mathcal{N}[\mathcal{C} u](\mathbf{x}_i, \boldsymbol{\theta}) - f(\mathbf{x}_i))^2,$$

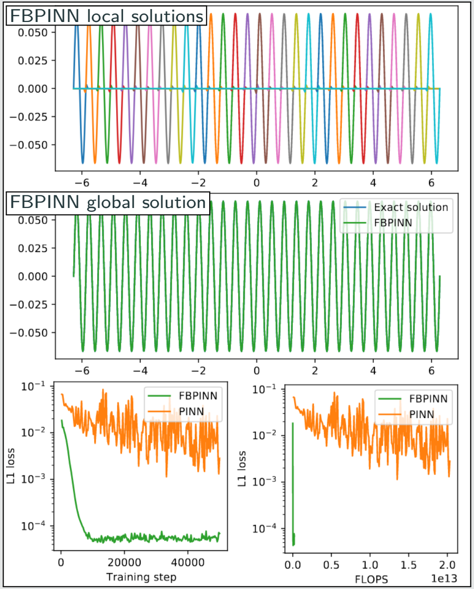
with constraining operator $\mathcal{C}$, which **explicitly enforces the boundary conditions**.

→ Often **improves training performance**



Numerical Results for FBPINNs

PINN vs FBPINN (Moseley et al. (2023))



Scalability of FBPINNs

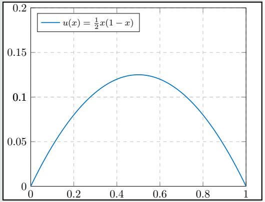
Consider the **simple boundary value problem**

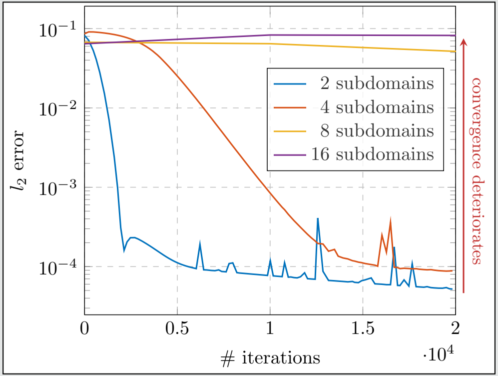
$$-u'' = 1 \text{ in } [0,1],$$

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

which has the **solution**

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





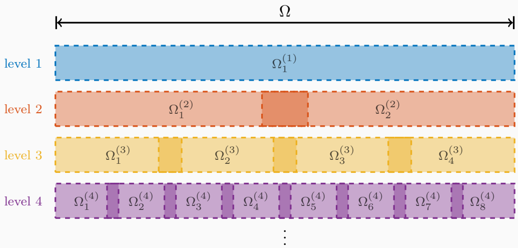
We introduce a **hierarchy of L overlapping domain decompositions**

$$\Omega = \bigcup_{j=1}^{J^{(l)}} \Omega_j^{(l)}$$

and corresponding window functions $\omega_j^{(l)}$ with

$$\text{supp}(\omega_j^{(l)}) \subset \Omega_j^{(l)} \text{ and } \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \equiv 1 \text{ on } \Omega.$$

This yields the **L-level FBPINN algorithm**:

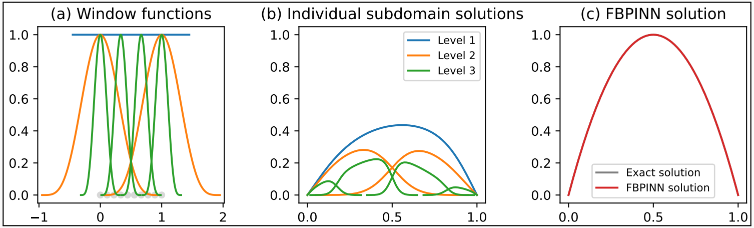


L-level network architecture

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

Loss function

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

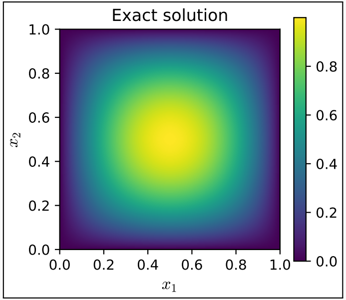


Multilevel FBPINNs – 2D Laplace

Let us consider the **simple two-dimensional boundary value problem**

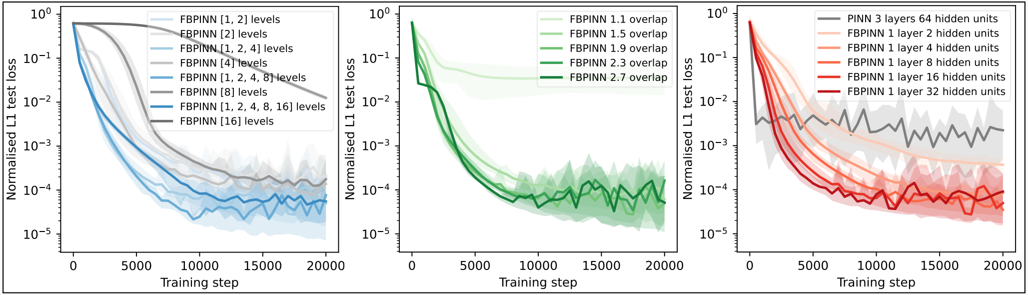
$$-\Delta u = 32(x(1-x) + y(1-y)) \quad \text{in } \Omega = [0,1]^2,$$
$$u = 0 \quad \text{on } \partial\Omega,$$

which has the **solution** $u(x,y) = 16(x(1-x)y(1-y))$.



Baseline model:

# levels	# hidden units	overlap δ
3	16	1.9



Cf. **Dolean, Heinlein, Mishra, Moseley (submitted 2023 / arXiv:2306.05486)**. Implementation using JAX

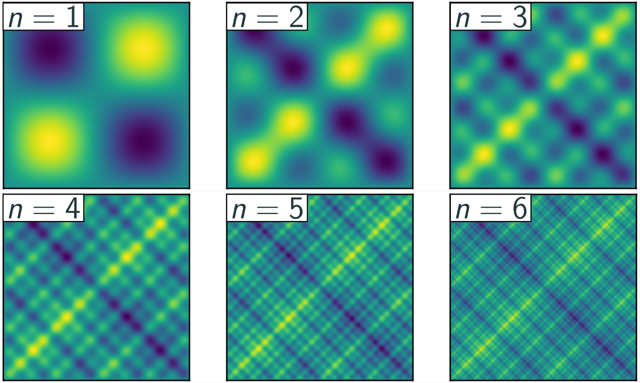
Multi-Frequency Problem

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

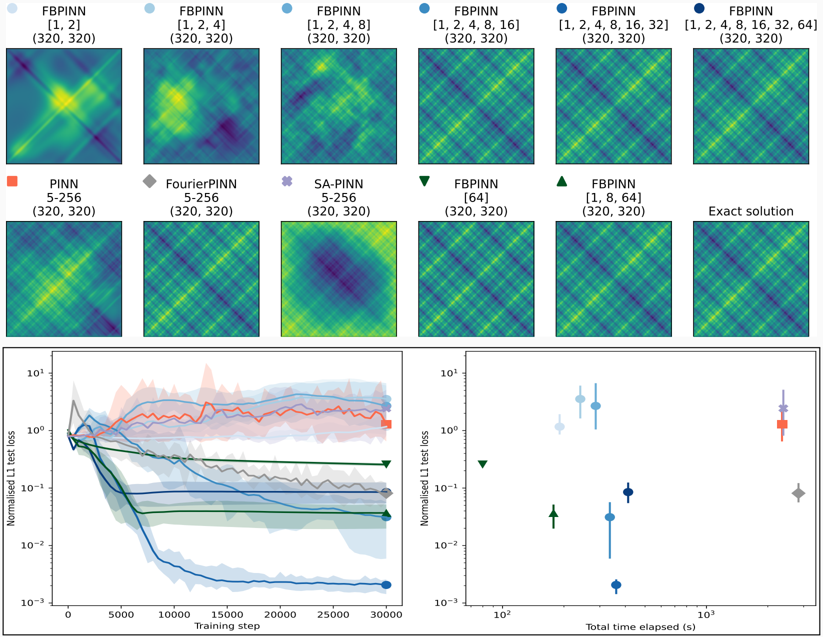
$$\begin{aligned} -\Delta u &= 2\sum_{i=1}^{n} (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y) \quad && \text{in } \Omega = [0,1]^2, \\ u &= 0 && \text{on } \partial\Omega, \end{aligned}$$

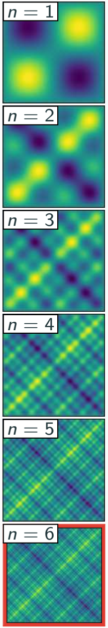
with $\omega_i = 2^i$.

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

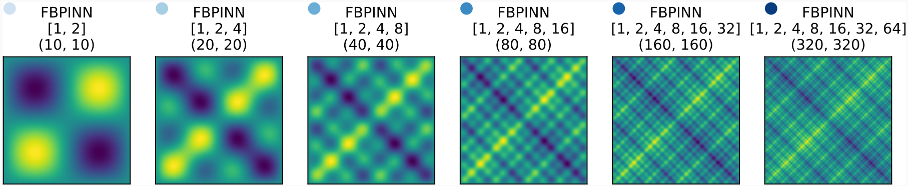


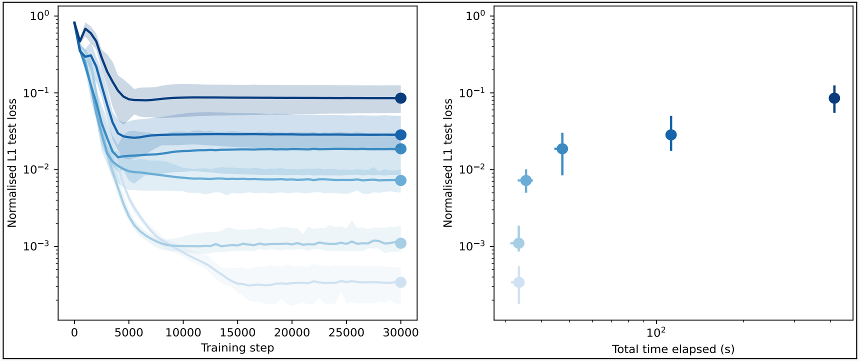
Multi-Level FBPINNs for a Multi-Frequency Problem – Strong Scaling

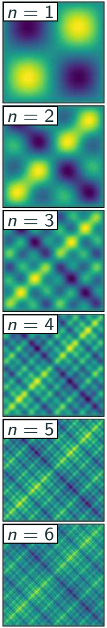




Multi-Level FBPINNs for a Multi-Frequency Problem – Weak Scaling







- Ongoing: **analysis and improvement of the convergence**

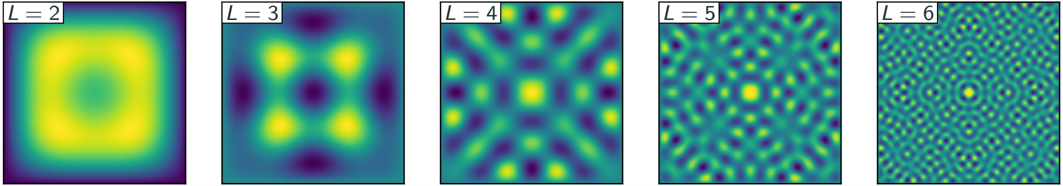
Cf. **Dolean, Heinlein, Mishra, Moseley (submitted 2023 / arXiv:2306.05486)**.

Helmholtz Problem

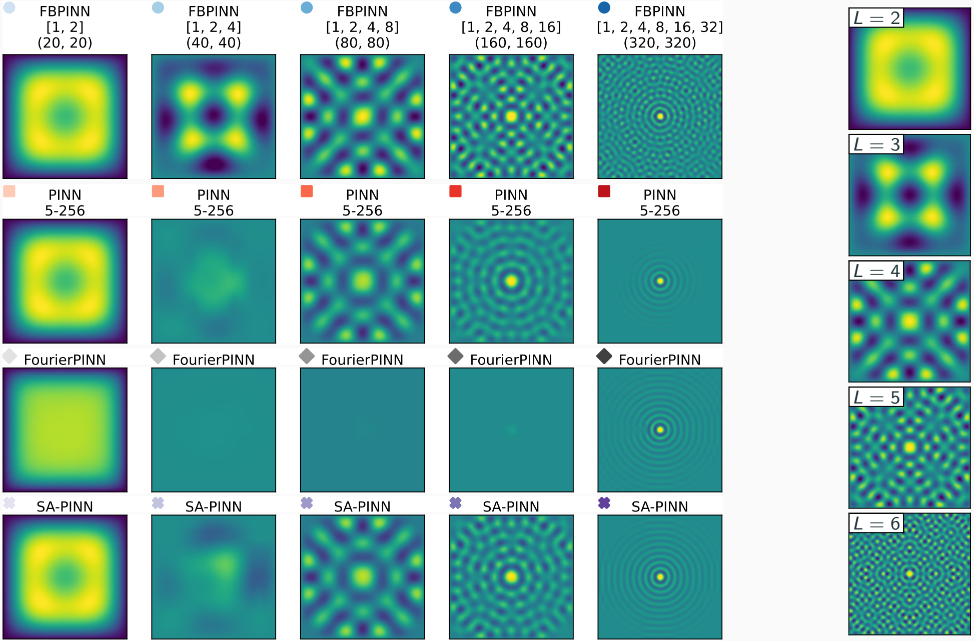
Finally, let us consider the **two-dimensional Helmholtz boundary value problem**

$$\Delta u - k^2 u = f \quad \text{in } \Omega = [0,1]^2,$$
$$u = 0 \quad \text{on } \partial\Omega,$$
$$f(\mathbf{x}) = e^{-\frac{1}{2}(\|\mathbf{x}-0.5\|/\sigma)^2}.$$

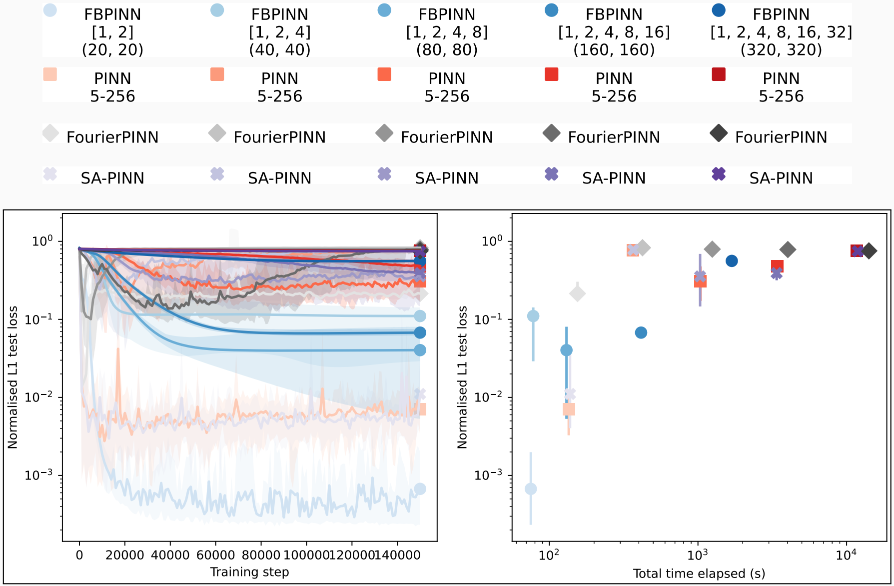
With $k = 2^L\pi/1.6$ and $\sigma = 0.8/2^L$, we obtain the **solutions**:

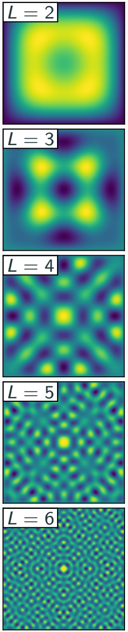


Multi-Level FBPINNs for the Helmholtz Problem – Weak Scaling



Multi-Level FBPINNs for the Helmholtz Problem – Weak Scaling





Cf. **Dolean, Heinlein, Mishra, Moseley (submitted 2023 / arXiv:2306.05486)**.

Multifidelity domain decomposition-based physics-informed neural networks for time-dependent problems

PINNs for Time-Dependent Problems

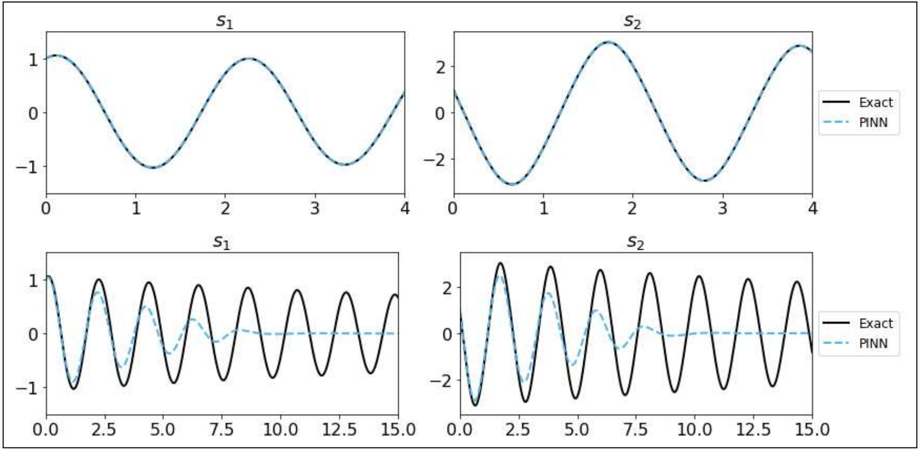
We investigate the performance of PINNs for **time-dependent problems**. Therefore, consider the simple **pedulum problem**:

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

Problem parameters

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

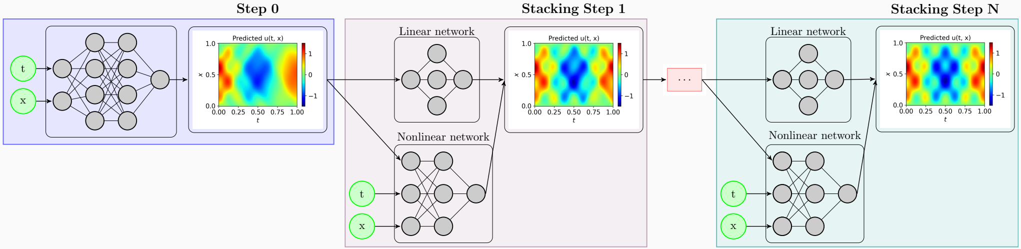
- **Top:** $T = 4$
- **Bottom:** $T = 20$



Stacking Multifidelity PINNs

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

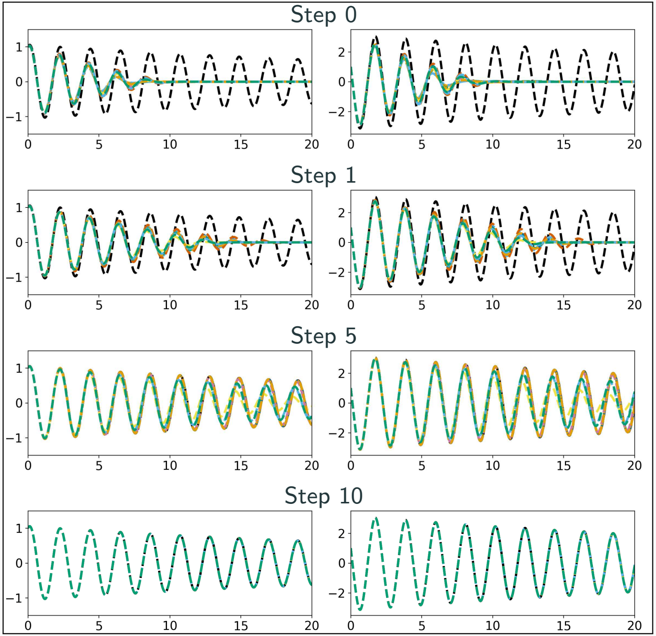
$$u^{MF}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}) = (1 - |\alpha|)\, u^{MF}_{\mathrm{linear}}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}, u^{SF}) + |\alpha|\, u^{MF}_{\mathrm{nonlinear}}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}, u^{SF})$$



Related works (non-exhaustive list)

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

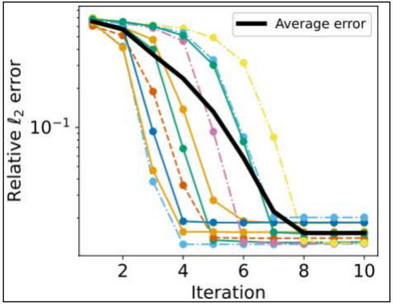
Stacking Multifidelity PINNs for the Pendulum Problem



Pendulum problem:

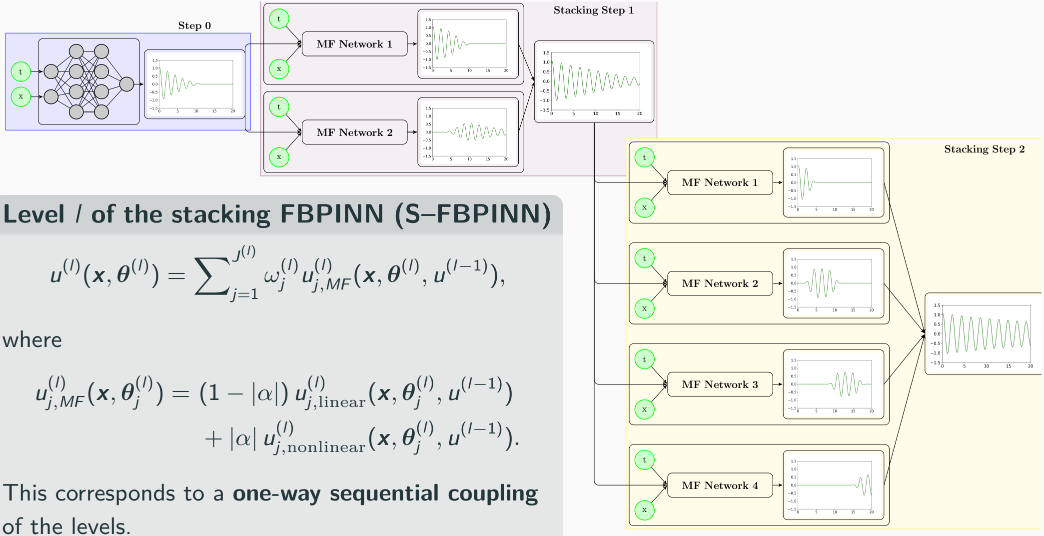
$$\frac{ds_1}{dt} = s_2,$$
$$\frac{ds_2}{dt} = -\frac{b}{m}s_2 - \frac{g}{L}\sin(s_1).$$

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



Stacking Multifidelity FBPINNs

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



Level l of the stacking FBPINN (S–FBPINN)

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

where

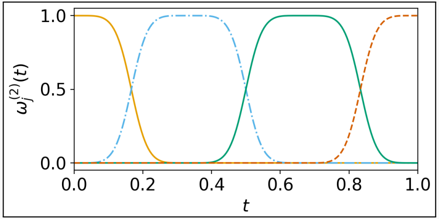
$$u_{j,MF}^{(l)}(\boldsymbol{x}, \boldsymbol{\theta}_j^{(l)}) = (1 - |\alpha|)\, u_{j,\mathrm{linear}}^{(l)}(\boldsymbol{x}, \boldsymbol{\theta}_j^{(l)}, u^{(l-1)}) + |\alpha|\, u_{j,\mathrm{nonlinear}}^{(l)}(\boldsymbol{x}, \boldsymbol{\theta}_j^{(l)}, u^{(l-1)}).$$

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

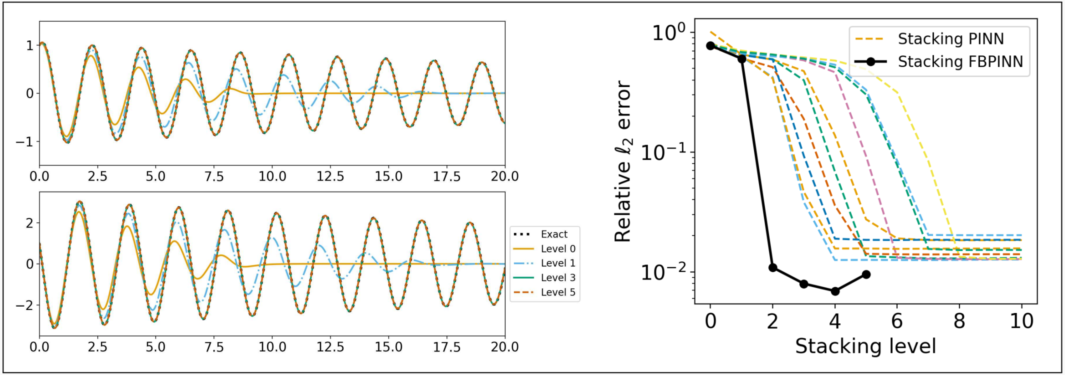
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

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

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



Exemplary partition of unity in time



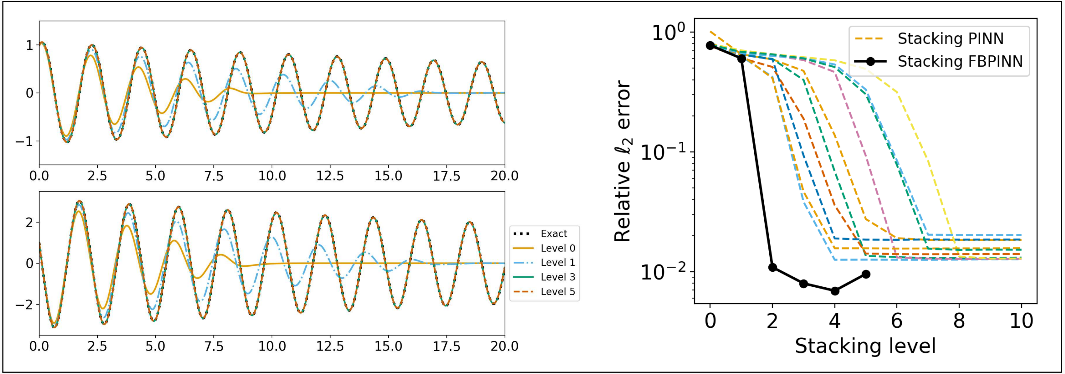
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

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

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

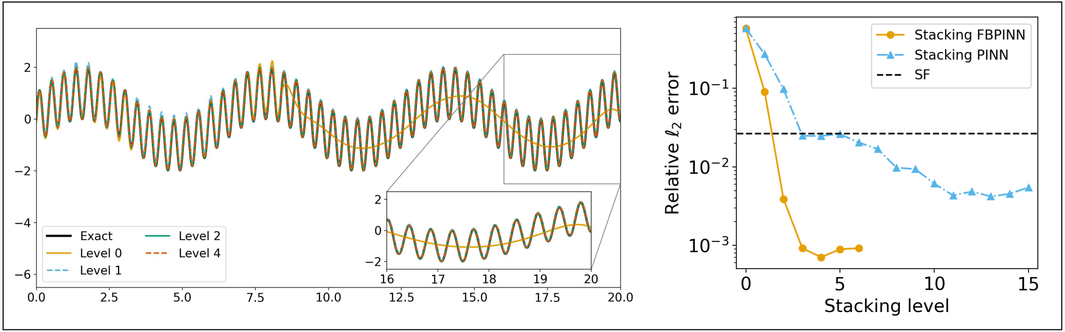


Second, we consider a **two-frequency problem**:

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

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

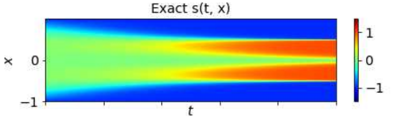


→ Due to the **multiscale structure of the problem**, the **improvements** due to the **multifidelity FBPINN approach** are **even stronger**.

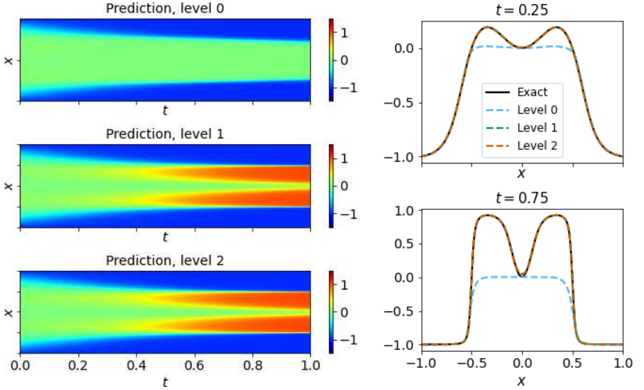
Numerical Results – Allen–Cahn Equation

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. (arXiv 2023)**.

PINNs

- **Training of PINNs can be challenging** when:
  - **scaling to large domains / high frequency solutions**
  - **multiple loss terms** have to be balanced
- **Convergence of PINNs has yet to be understood better**

DeepDDM for PINNs

- The **DeepDDM method** is a **classical Schwarz iteration** with local PINN solver.
- **Scalability** is enabled by **adding a coarse level**.

Multilevel 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**.

Multifidelity stacking FBPINNs

- The **combination of multifidelity stacking PINNs with FBPINNs** yields **significant improvements in the accuracy and efficiency** for **time-dependent problems**.

Thank you for your attention!