

# Domain decomposition for physics-informed neural networks

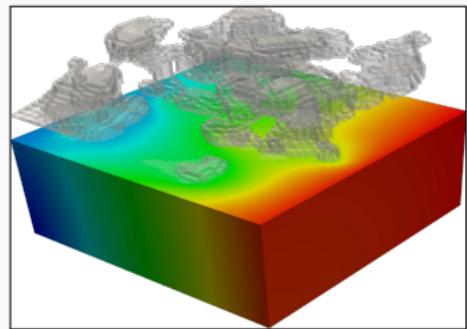
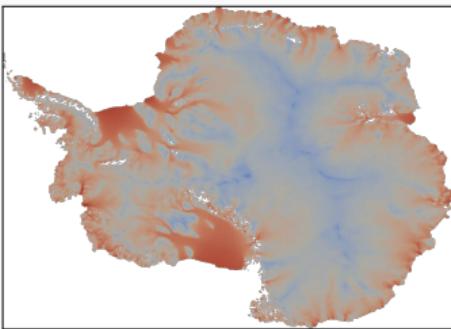
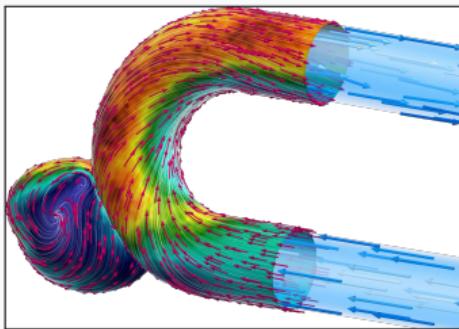
---

Alexander Heinlein<sup>1</sup>

Crunch Seminar, Division of Applied Mathematics, Brown University (Virtual), U.S., March 22, 2024

<sup>1</sup>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	<b>improve</b>	machine learning techniques
machine learning techniques	<b>assist</b>	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

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

by solving an **optimization problem**

$$\min_{\theta} \sum_{x_i} G(x_i, \Psi_t(x_i, \theta), \nabla\Psi_t(x_i, \theta), \nabla^2\Psi_t(x_i, \theta))^2$$

where  $\Psi_t(x, \theta)$  is a **trial function**,  $x_i$  sampling points inside the domain  $\Omega$  and  $\theta$  are **adjustable parameters**.

### Construction of the trial functions

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

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

- NN is a **feedforward neural network** with **trainable parameters**  $\theta$  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

# Neural Networks for Solving Differential Equations

## Approach

Solve a general differential equation subject to boundary conditions

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

by solving an **optimization problem**

$$\min_{\theta} \sum_{x_i} G(x_i, \Psi_t(x_i, \theta), \nabla\Psi_t(x_i, \theta), \nabla^2\Psi_t(x_i, \theta))^2$$

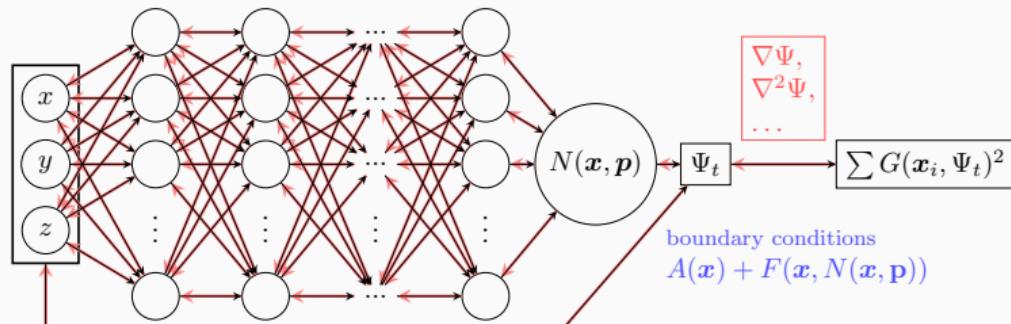
where  $\Psi_t(x, \theta)$  is a **trial function**,  $x_i$  sampling points inside the domain  $\Omega$  and  $\theta$  are adjustable parameters.

## Construction of the trial functions

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

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

- NN is a **feedforward neural network** with **trainable parameters  $\theta$**  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

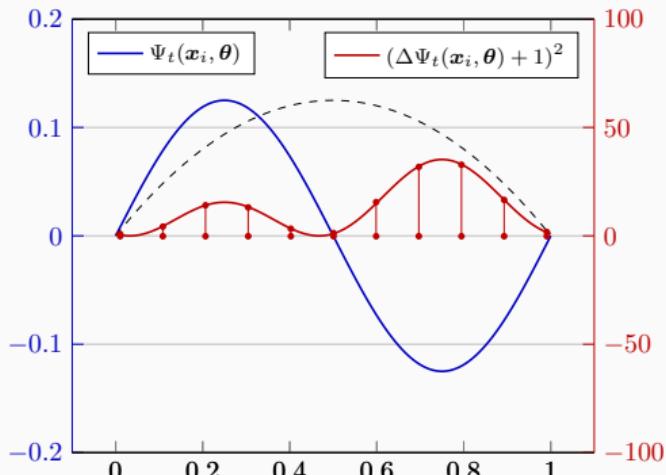
Solve the **boundary value problem**

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

$$\Psi_t(0, \theta) = \Psi_t(1, \theta) = 0,$$

via a **collocation approach**:

$$\min_{\theta} \sum_{x_i} (1 - \Delta \Psi_t(x_i, \theta))^2$$

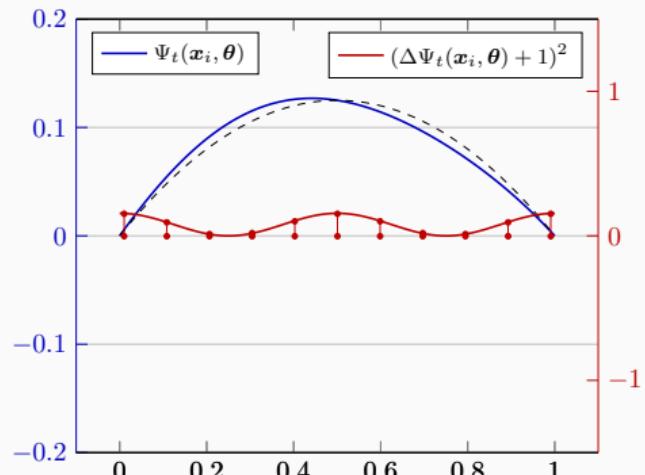


$$(\Delta \Psi_t(x_i, \theta) + 1)^2 >> 0$$

## Boundary conditions

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

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



$$(\Delta \Psi_t(x_i, \theta) + 1)^2 \approx 0$$

# 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**

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

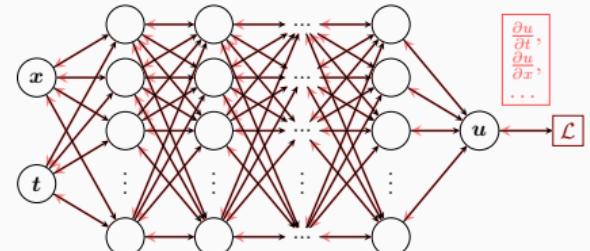
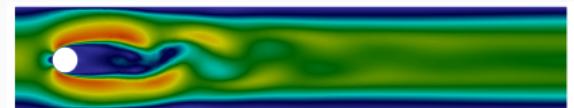
It is based on the approach by [Lagaris et al. \(1998\)](#). The main novelty of PINNs is the use of a **hybrid loss function**:

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

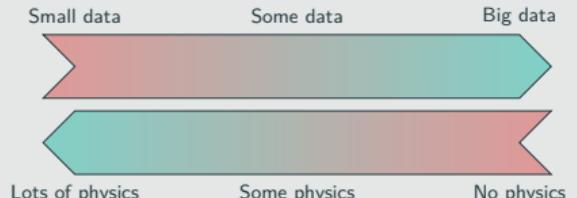
where  $\omega_{\text{data}}$  and  $\omega_{\text{PDE}}$  are **weights** and

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

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



## Hybrid loss



## 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

- 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

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

where

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

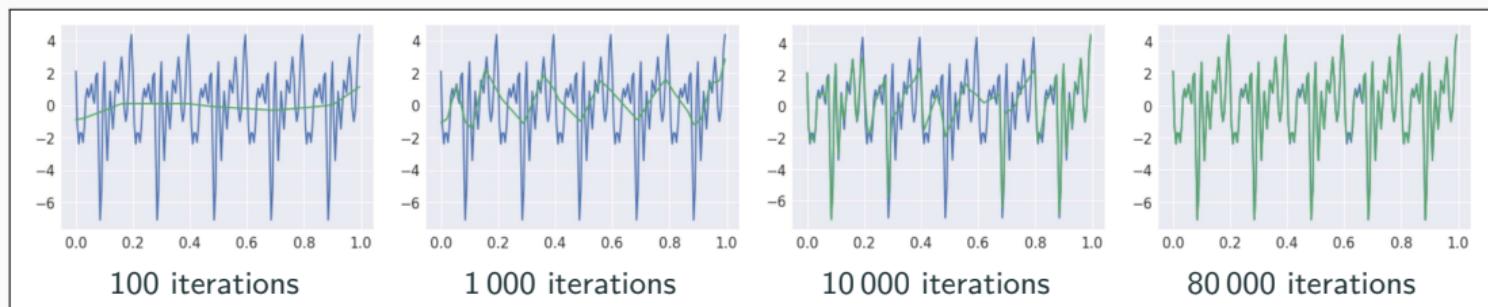
*Rule of thumb:*

“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

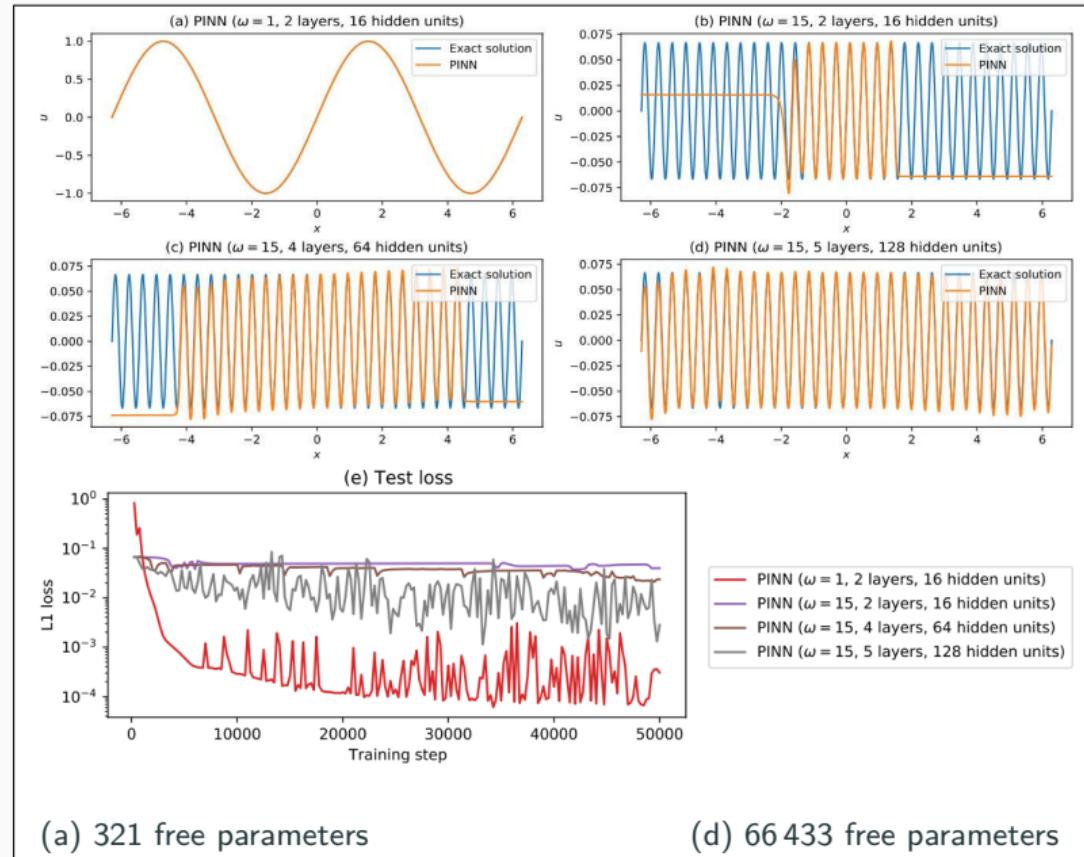
$$\begin{aligned} u' &= \cos(\omega x), \\ u(0) &= 0, \end{aligned}$$

for different values of  $\omega$   
using PINNs with  
varying network  
capacities.

## Scaling issues

- Large computational domains
- Small frequencies

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



# Motivation – Some Observations on the Performance of PINNs

Solve

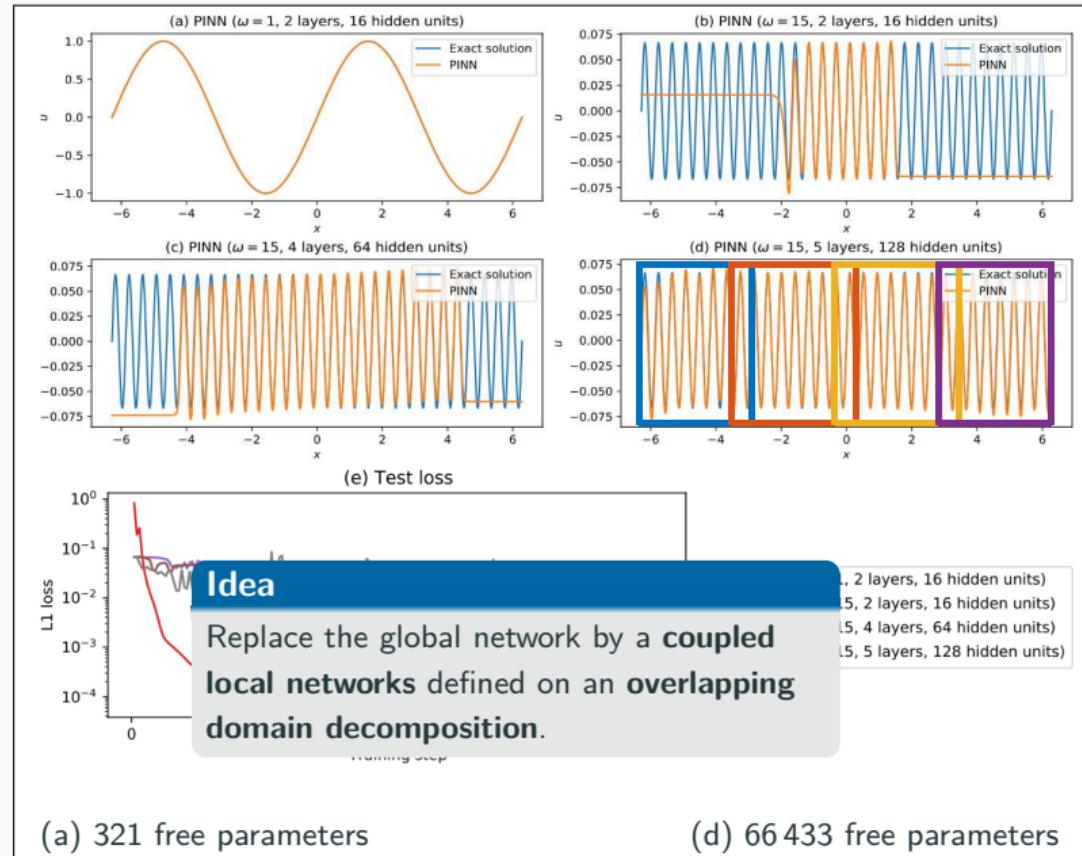
$$\begin{aligned} u' &= \cos(\omega x), \\ u(0) &= 0, \end{aligned}$$

for different values of  $\omega$   
using PINNs with  
varying network  
capacities.

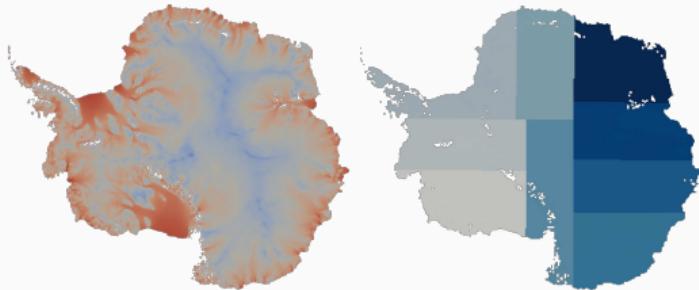
## Scaling issues

- Large computational domains
- Small frequencies

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



# Domain Decomposition Methods



Images based on Heinlein, Perego, Rajamanickam (2022)

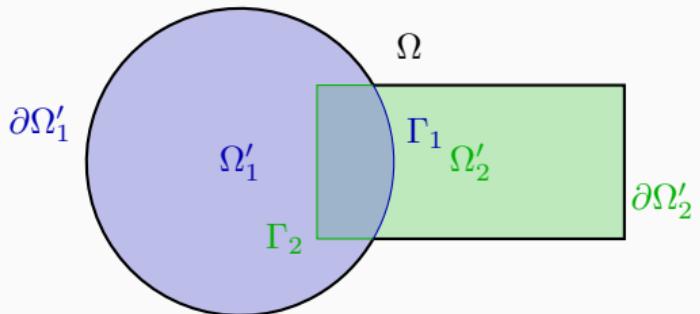
**Historical remarks:** The **alternating Schwarz method** is the earliest **domain decomposition method (DDM)**, which has been invented by **H. A. Schwarz** and published in **1870**:

- 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**

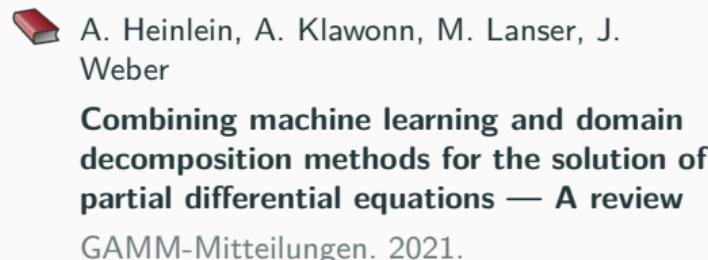


# DDM-Based Approaches for Neural Network-Based Discretizations – Literature

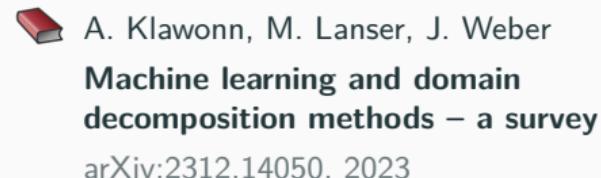
A non-exhaustive literature overview:

- 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:

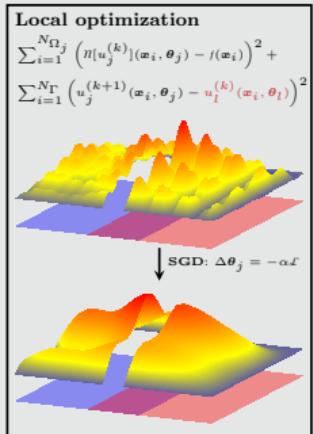


An overview of the state-of-the-art in the end of 2023:



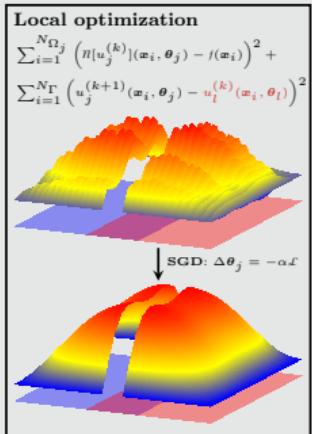
# Combining Schwarz Methods with Neural Network-Based Discretizations

## Approach 1 – Via a classical Schwarz iteration

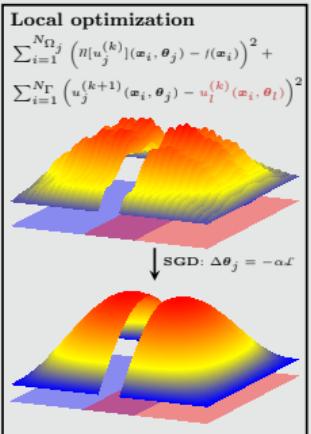


Schwarz iteration

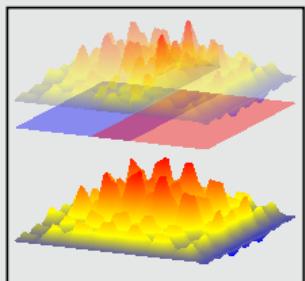
$$\begin{aligned} \Delta u_j^{(k+1)} &= f && \text{in } \Omega_j \\ u_j^{(k+1)} &= u_l^{(k)} && \text{on } \Gamma_j \end{aligned}$$



→ ⋯ →

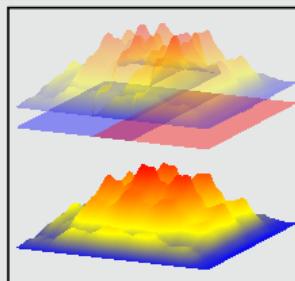


## Approach 2 – Integration via the neural network architecture

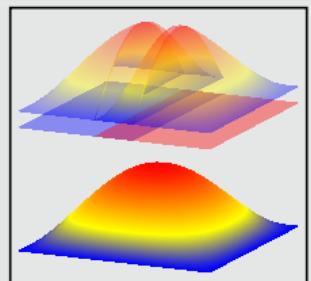


SGD:  $\Delta(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_N) = -\alpha \mathcal{L}$

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



→ ⋯ →



## **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  $\Omega_j$

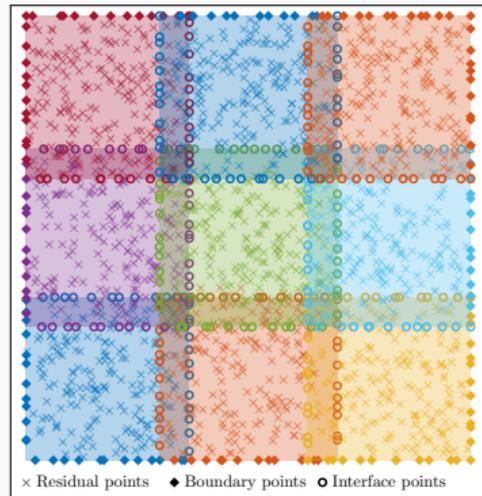
$$\mathcal{L}_j(\theta_j) := \mathcal{L}_{\Omega_j}(\theta_j) + \mathcal{L}_{\partial\Omega_j \setminus \Gamma_j}(\theta_j) + \mathcal{L}_{\Gamma_j}(\theta_j),$$

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

$$\mathcal{L}_{\Omega_j}(\theta_j) := \frac{1}{N_{f_j}} \sum_{i=1}^{N_{f_j}} (n(u_j(x_i, \theta_j)) - f(x_i))^2$$

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

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



## Algorithm 1: DeepDDM for $\Omega_j$

**Data:** Sampling points  $X_j$ , initial network parameters  $\theta_j^0$

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

**Train** local network  $u_j$ ;

**Communicate & update** interface values  $\mathcal{D}(u_l(\tilde{x}_i; \theta_j))$  from other subdomains  $\Omega_l$ ;

**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.

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

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

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

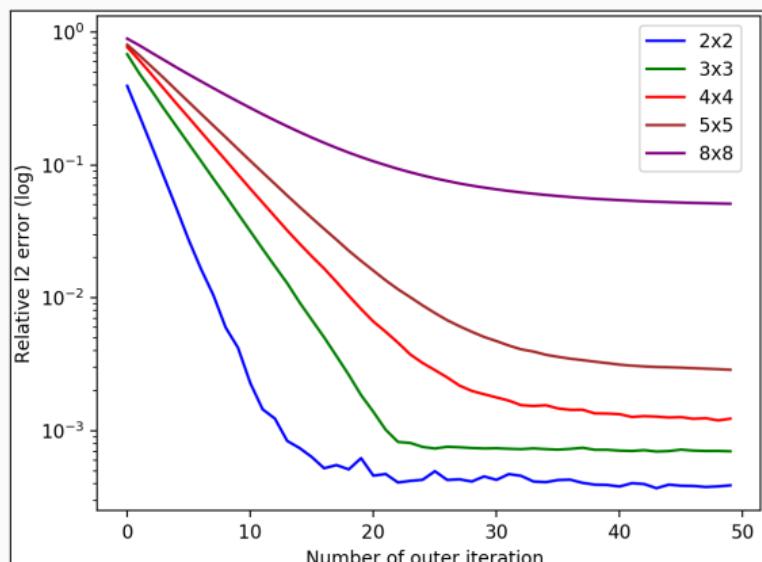
## Scaling issue

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

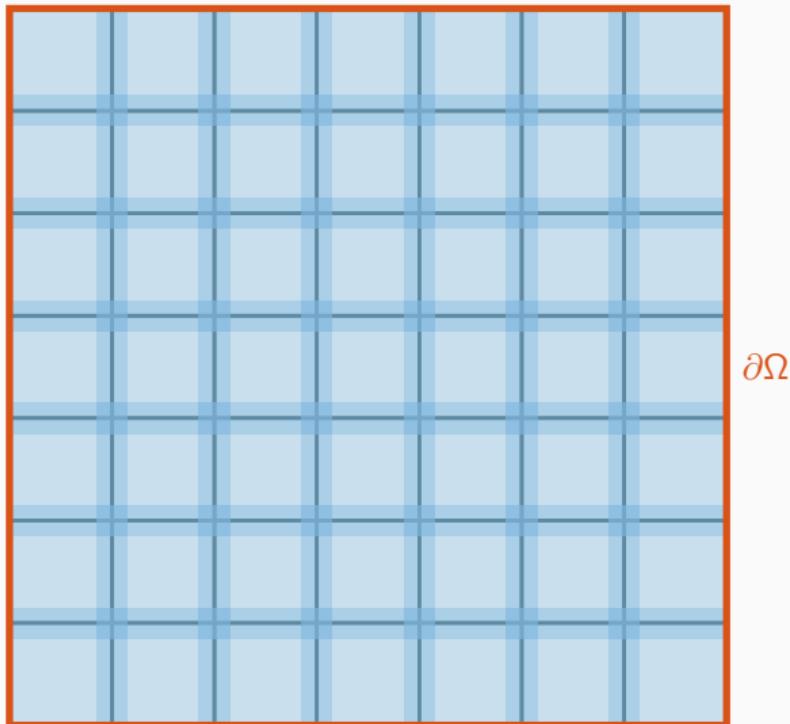
## Weak scaling

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

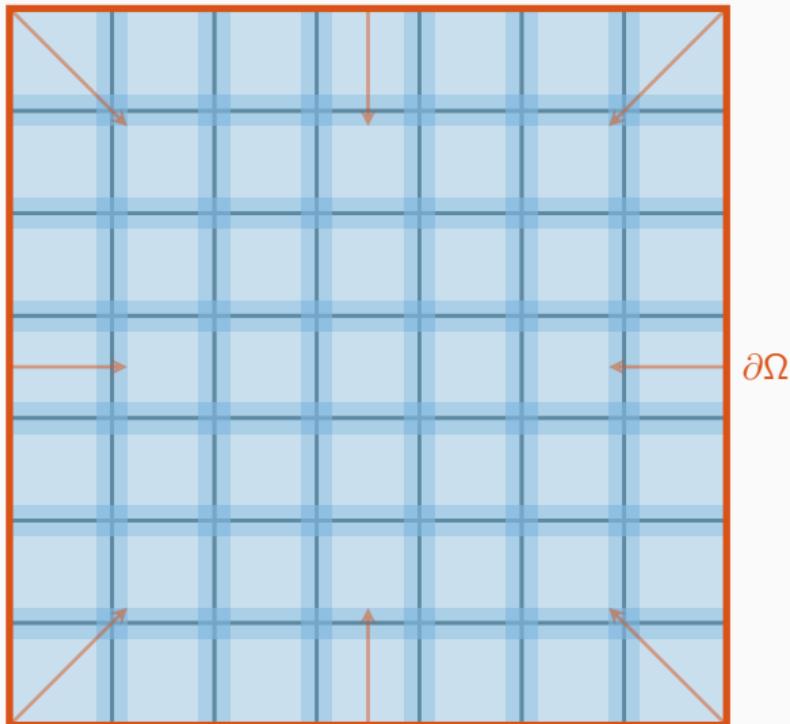
*Optimal scaling:* constant convergence rate and/or accuracy to stay approximately constant.



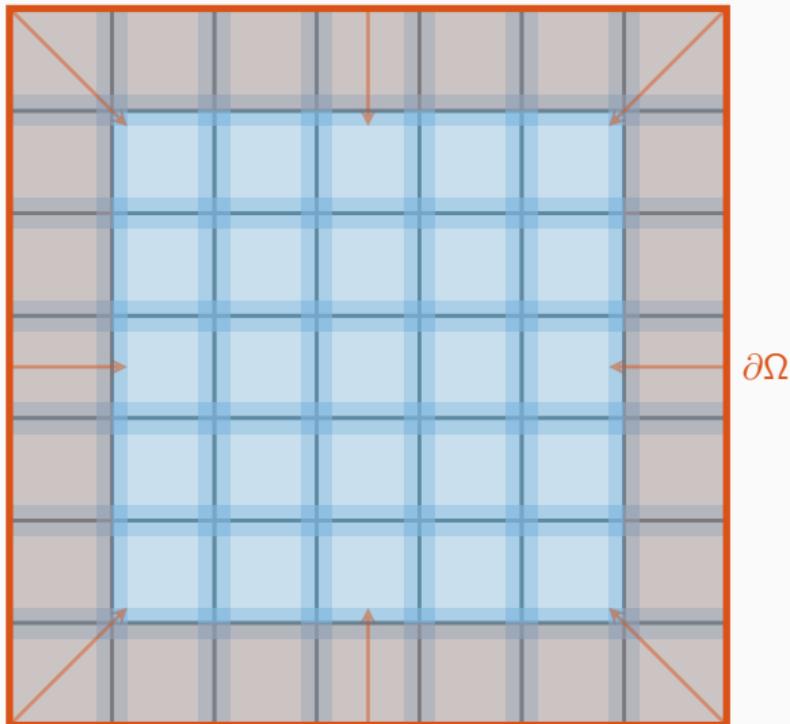
# Transport of Information One-Level Overlapping Schwarz Methods



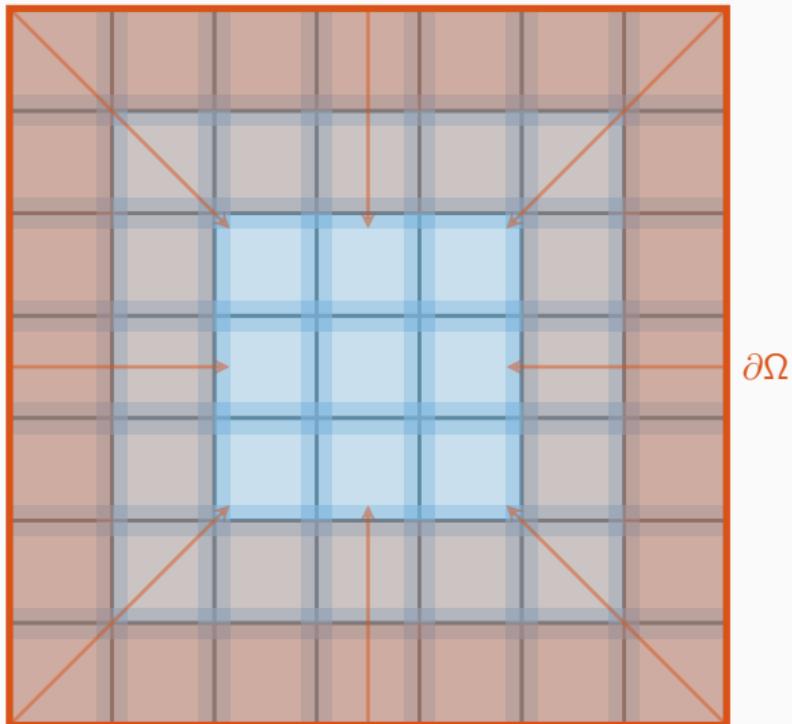
# Transport of Information One-Level Overlapping Schwarz Methods



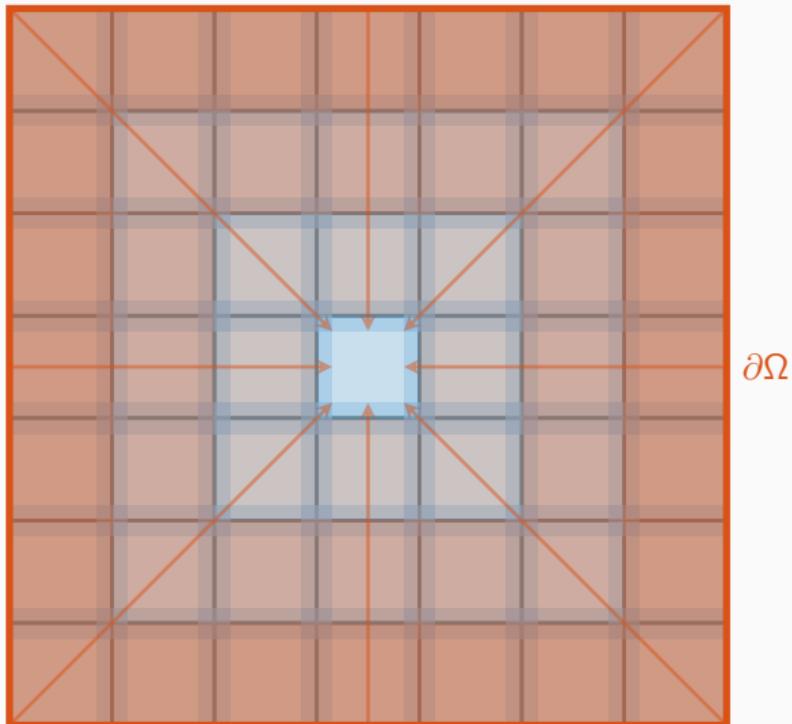
# Transport of Information One-Level Overlapping Schwarz Methods



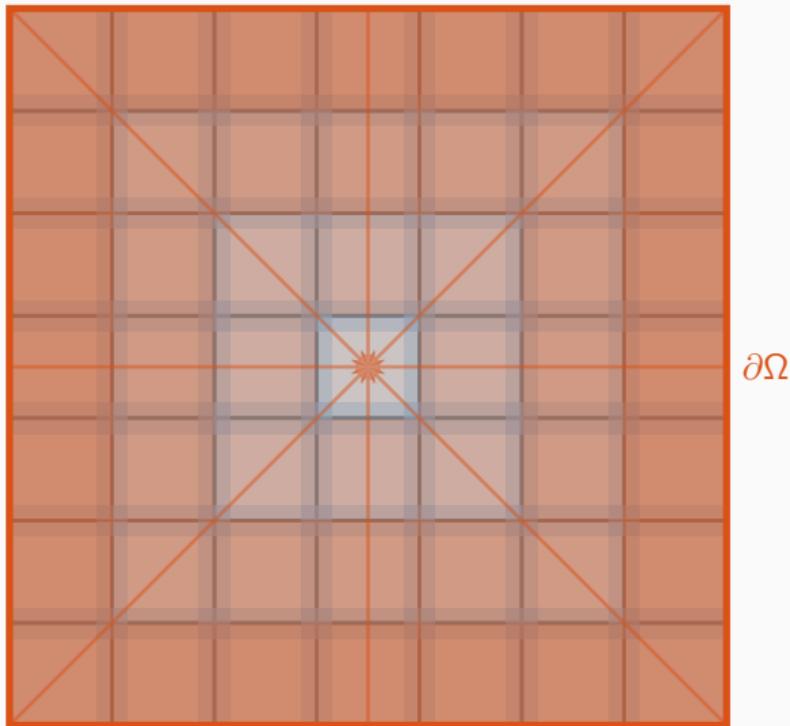
# Transport of Information One-Level Overlapping Schwarz Methods



# 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** → establish a faster / global transport of information.

# Fast Transport of Information via a Coarse Level

## Coarse space for the DeepDDM method

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

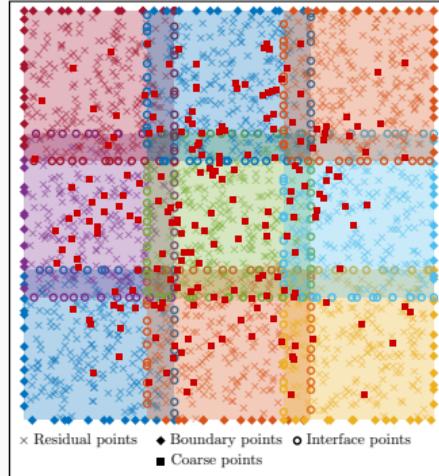
$$\lambda_f \frac{1}{N_0} \sum_{\mathbf{x}_i^0 \in \mathbf{X}_0} \left( u_0(\mathbf{x}_i^0) - \sum_{j=1}^J E_j(\chi_j u_j(\mathbf{x}_i^0)) \right)^2$$

for **incorporating information from the first level**. Here,

- $E_j$  extension by zero outside  $\Omega_j$
- $\chi_j$  local **partition of unity function**
- Incorporate coarse information** into the loss for the local subdomain  $\Omega_j$ :

$$\frac{1}{N_{\Gamma_j}} \sum_{i=1}^{N_{\Gamma_j}} \left( \mathcal{D}(u_j(\tilde{\mathbf{x}}_i, \theta_j)) - W_j^i \right)^2$$

with  $W_j^i = \mathcal{D}(\lambda_c u_l(\tilde{\mathbf{x}}_i) + (1 - \lambda_c) u_0(\tilde{\mathbf{x}}_i))$ .



## Algorithm 2: Two-level DeepDDM

Data:  $X_j$ ,  $X_0$ ,  $\theta_j^0$ ,  $\lambda_f$ , and  $\lambda_c$

while conv. (local & interface) not reached do

```
    Train local network  $u_j$ ;  
    Comm. & comp.  $\sum_{j=1}^J E_j(\chi_j u_j(\mathbf{x}_i^0)) \forall \mathbf{x}_i^0 \in \mathbf{X}_0$ ;  
    Train coarse network  $u_0$ ;  
    Comm. & update  $\mathcal{D}(u_l(\tilde{\mathbf{x}}_i; \theta_j)) \forall \Omega_l \cap \Omega_j \neq \emptyset$  ;  
end
```

# 2D Poisson Equation – Problem Setup

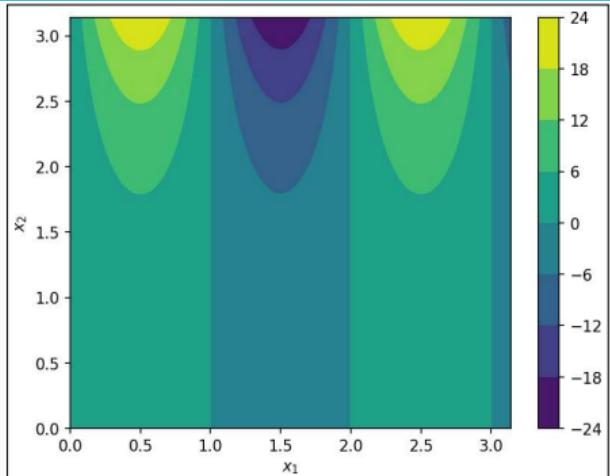
**Model problem:**

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

We choose  $f$  and  $g$  such that the exact solution is

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

where  $\alpha$  is an integer.



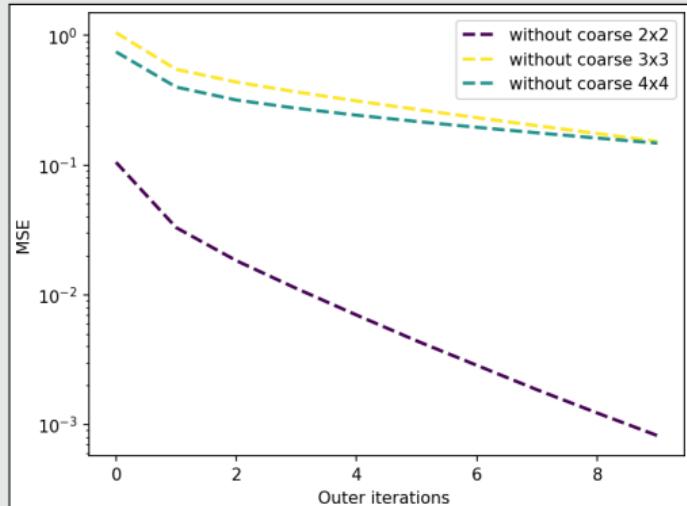
## Training setup – Strong scaling

- 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

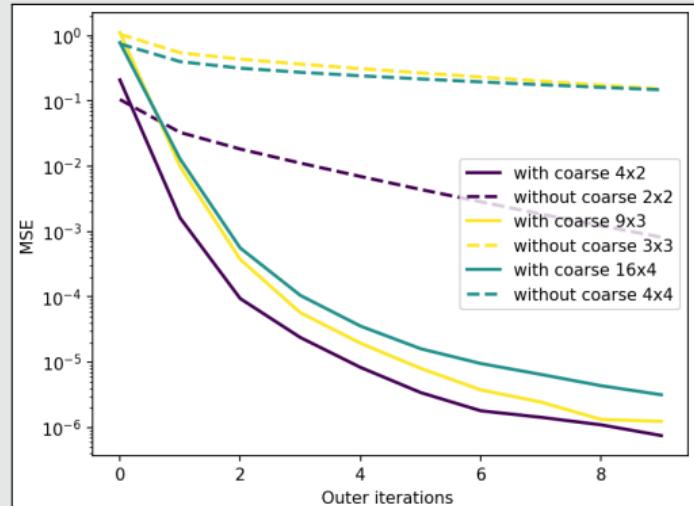
# 2D Poisson Equation – Weak Scaling

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**

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

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

FBPINNs use the **network architecture**

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

and the **loss function**

$$\mathcal{L}(\theta_1, \dots, \theta_J) = \frac{1}{N} \sum_{i=1}^N \left( n[\mathcal{C} \sum_{x_i \in \Omega_j} \omega_j u_j](\mathbf{x}_i, \theta_j) - f(\mathbf{x}_i) \right)^2.$$

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

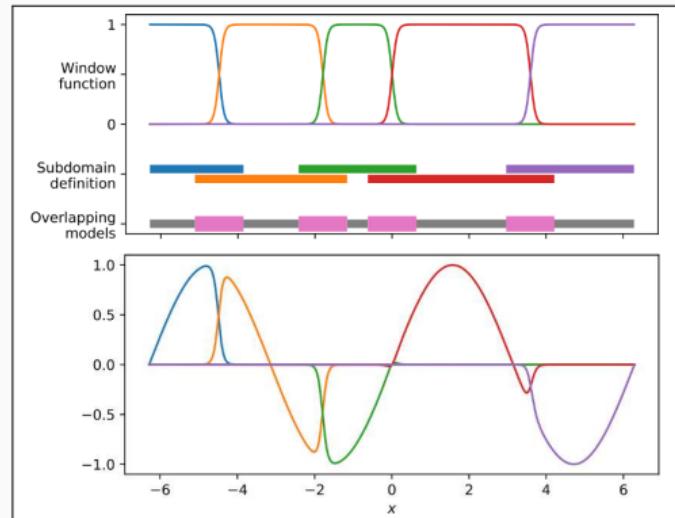
## Hard enf. of boundary conditions

Loss function

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

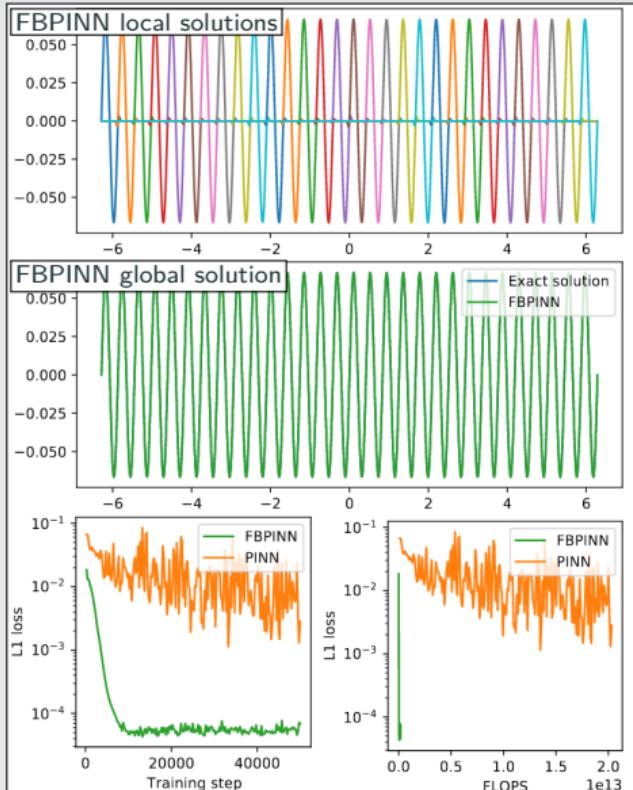
with constraining operator  $\mathcal{C}$ , which **explicitly enforces the boundary conditions**.

→ Often **improves training performance**



# Numerical Results for FBPINNs

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



## Scalability of FBPINNs

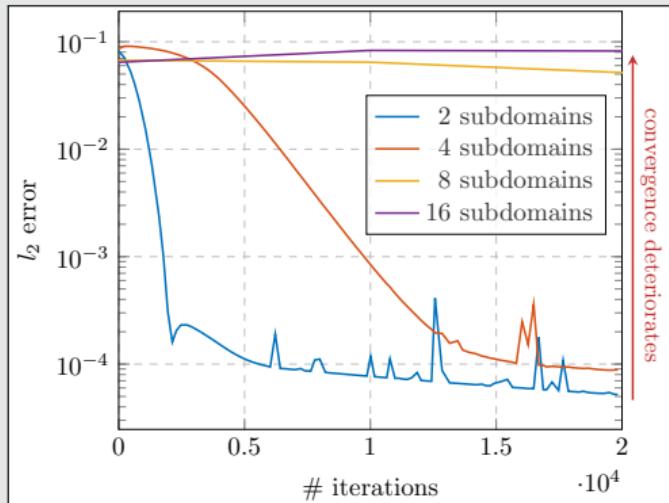
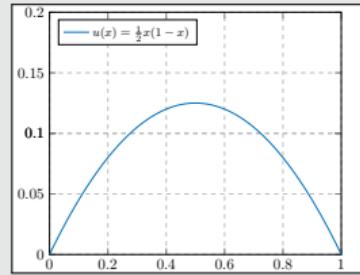
Consider the simple boundary value problem

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

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

which has the solution

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



# Multi-Level FBPINN Algorithm

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

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

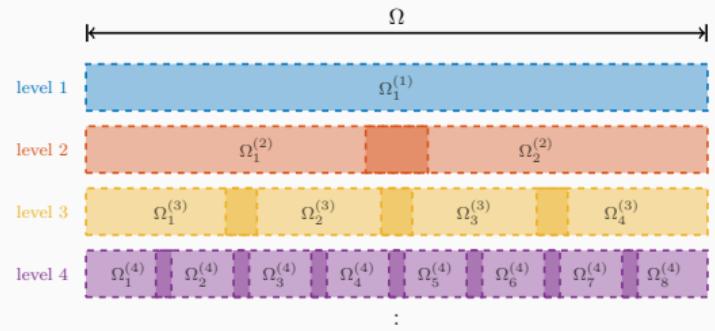
and corresponding window functions  $\omega_j^{(l)}$  with

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

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

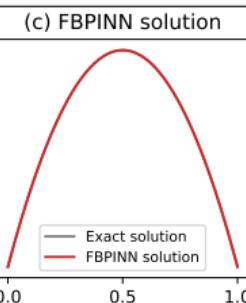
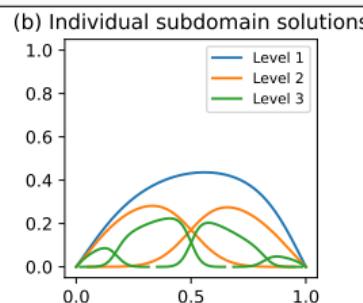
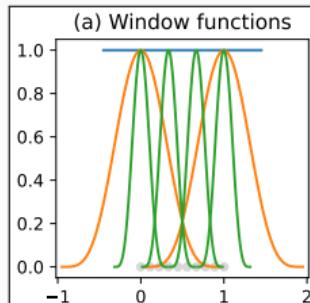
## $L$ -level network architecture

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



## Loss function

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



# Multilevel FBPINNs – 2D Laplace

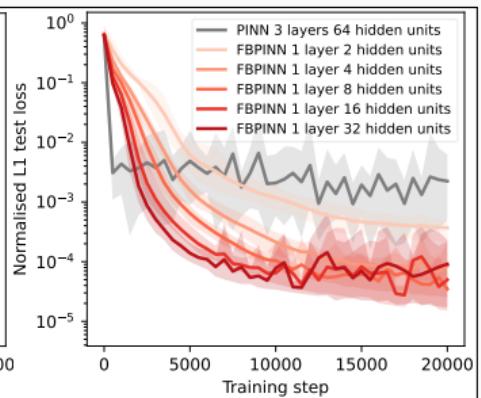
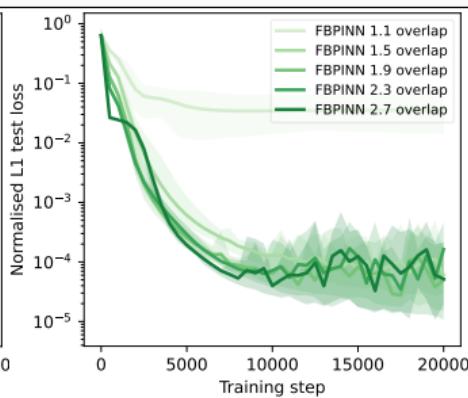
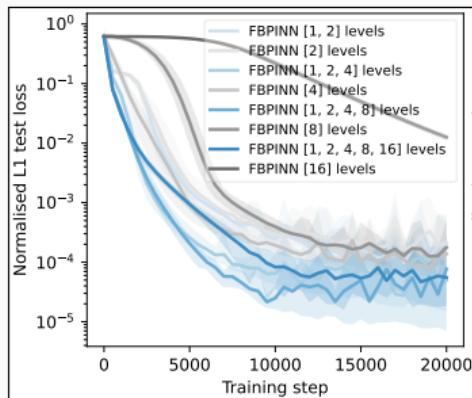
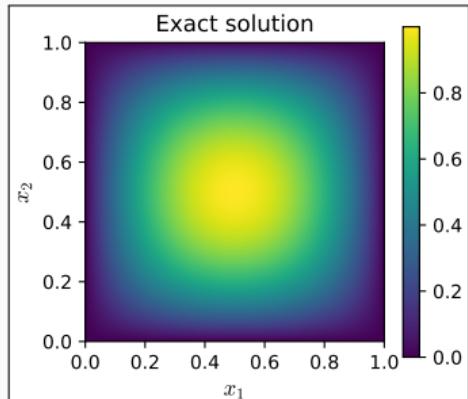
Let us consider the **simple two-dimensional boundary value problem**

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

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

**Baseline model:**

# levels	# hidden units	overlap $\delta$
3	16	1.9



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

Implementation using JAX

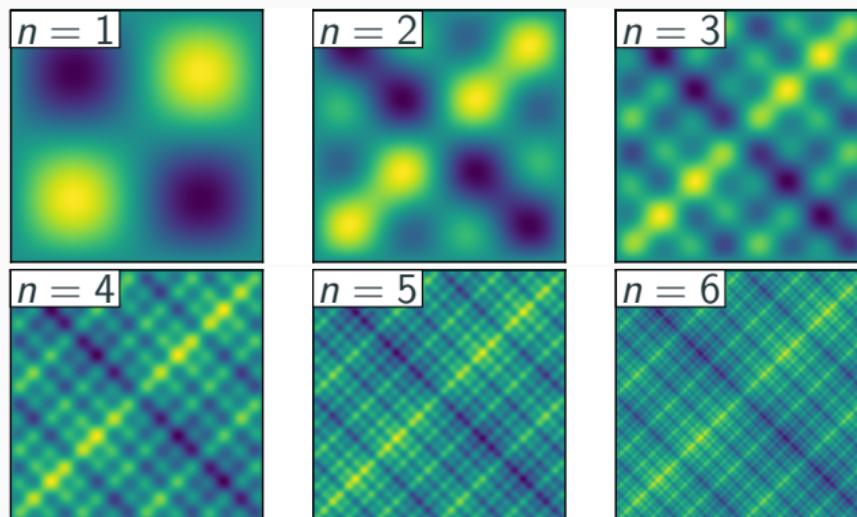
# Multi-Frequency Problem

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

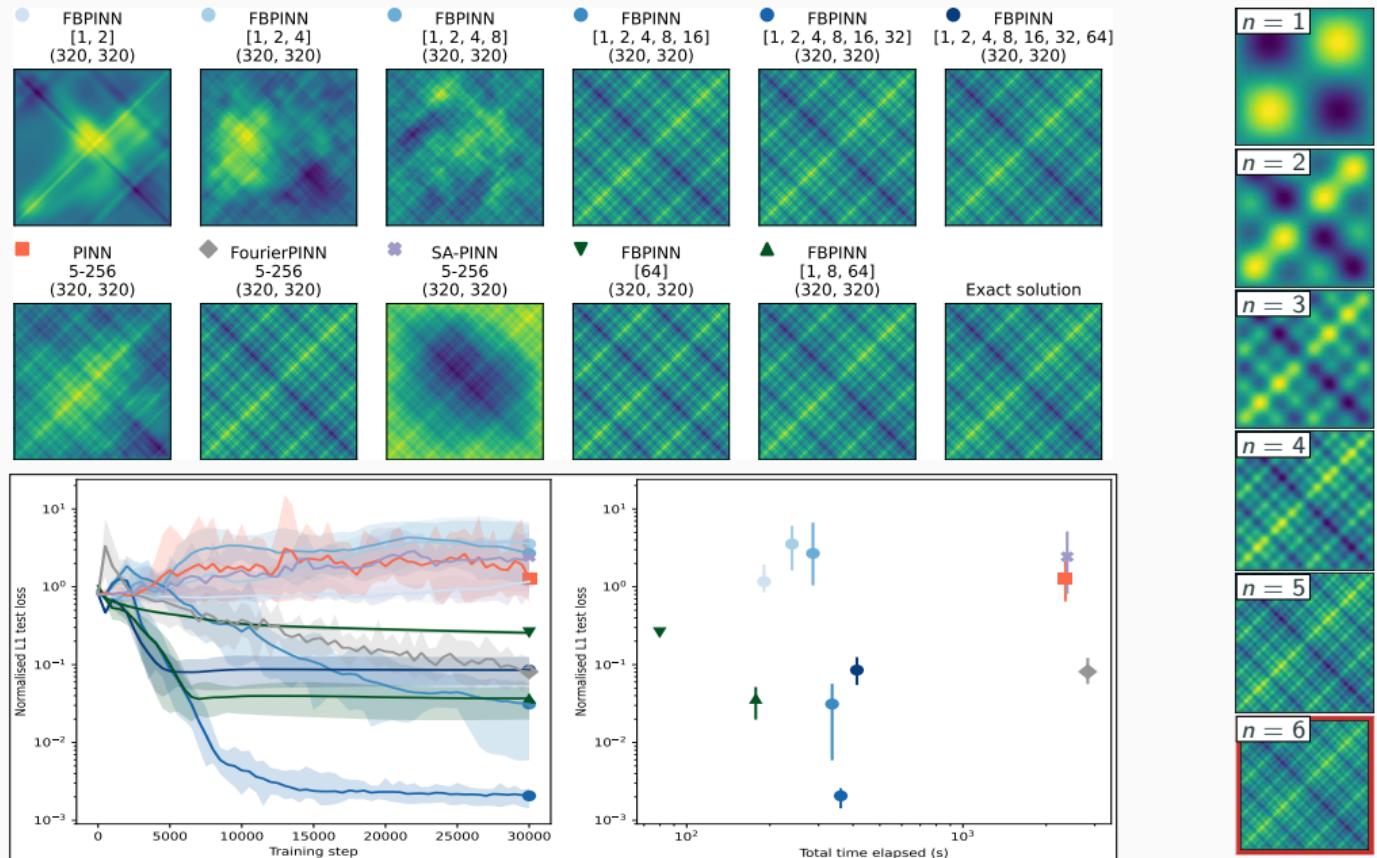
$$\begin{aligned} -\Delta u &= 2 \sum_{i=1}^n (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y) && \text{in } \Omega = [0, 1]^2, \\ u &= 0 && \text{on } \partial\Omega, \end{aligned}$$

with  $\omega_i = 2^i$ .

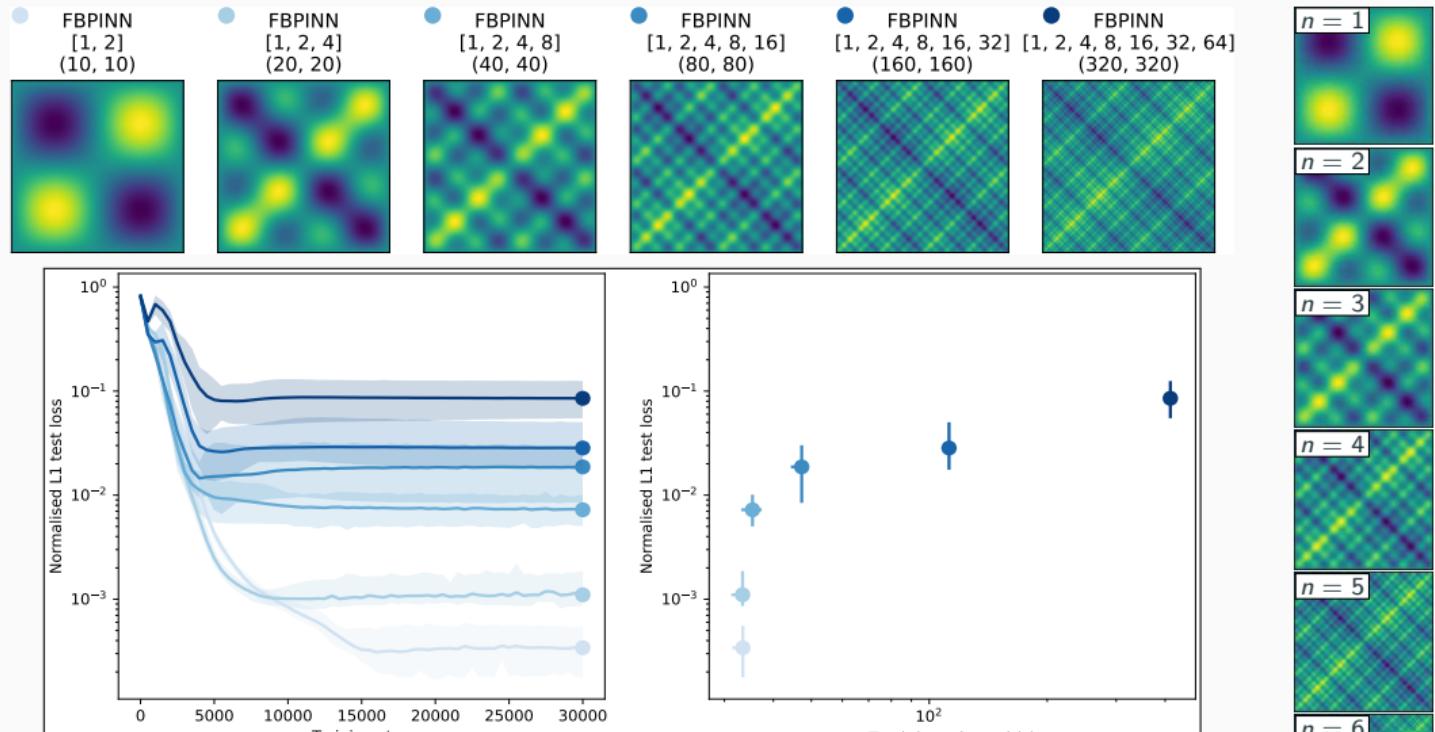
For increasing values of  $n$ , we obtain the **analytical solutions**:



# 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

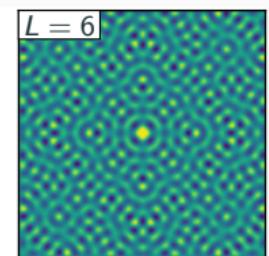
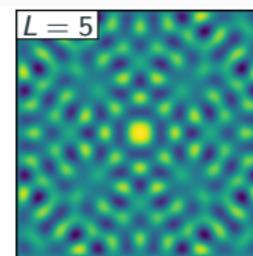
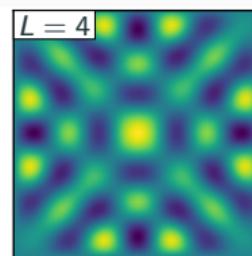
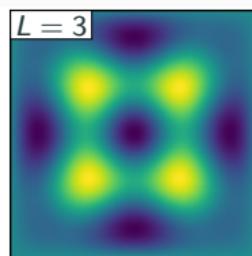
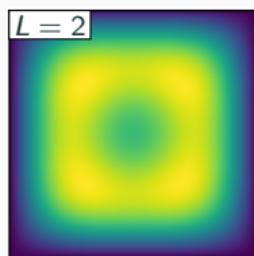
Finally, let us consider the **two-dimensional Helmholtz boundary value problem**

$$\Delta u - k^2 u = f \quad \text{in } \Omega = [0, 1]^2,$$

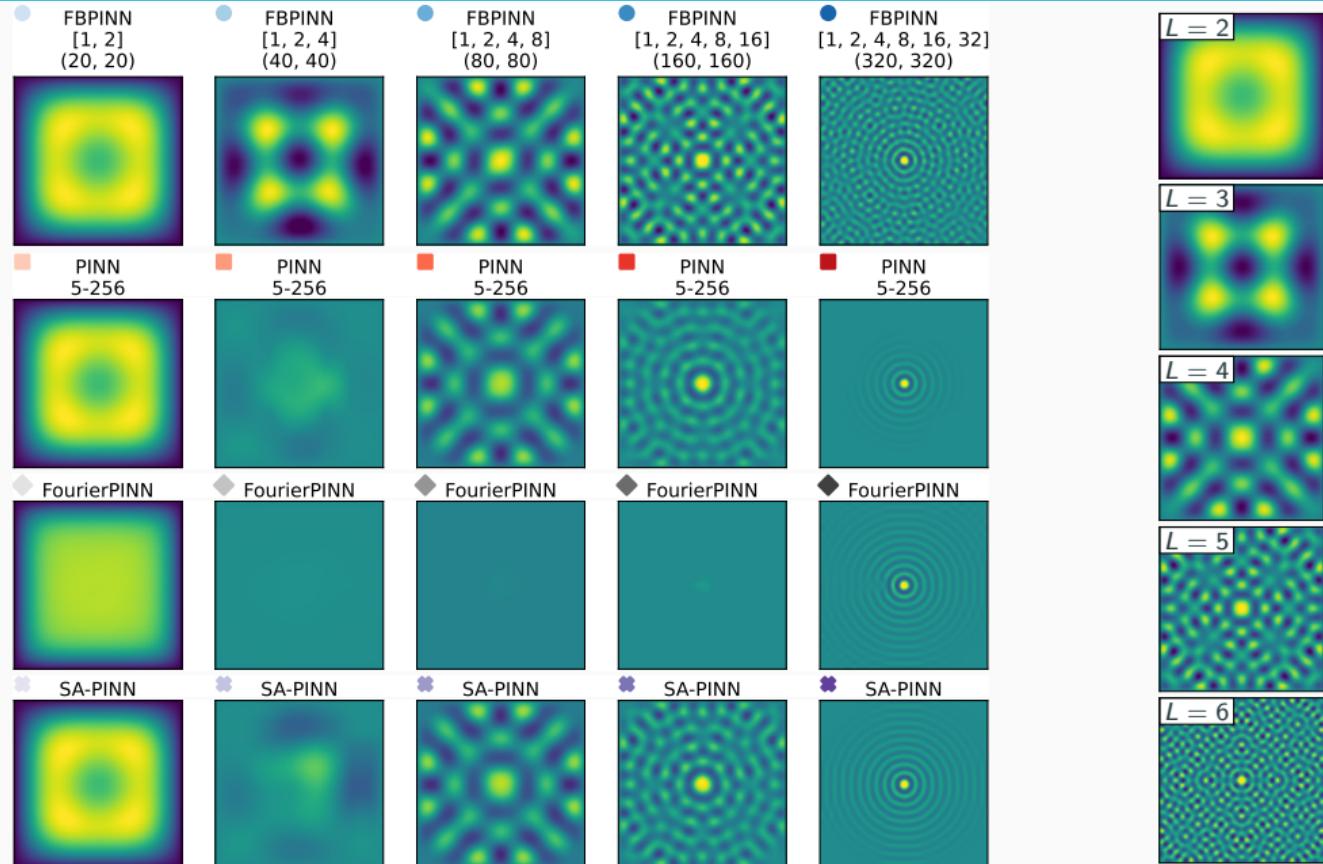
$$u = 0 \quad \text{on } \partial\Omega,$$

$$f(x) = e^{-\frac{1}{2}(\|x-0.5\|/\sigma)^2}.$$

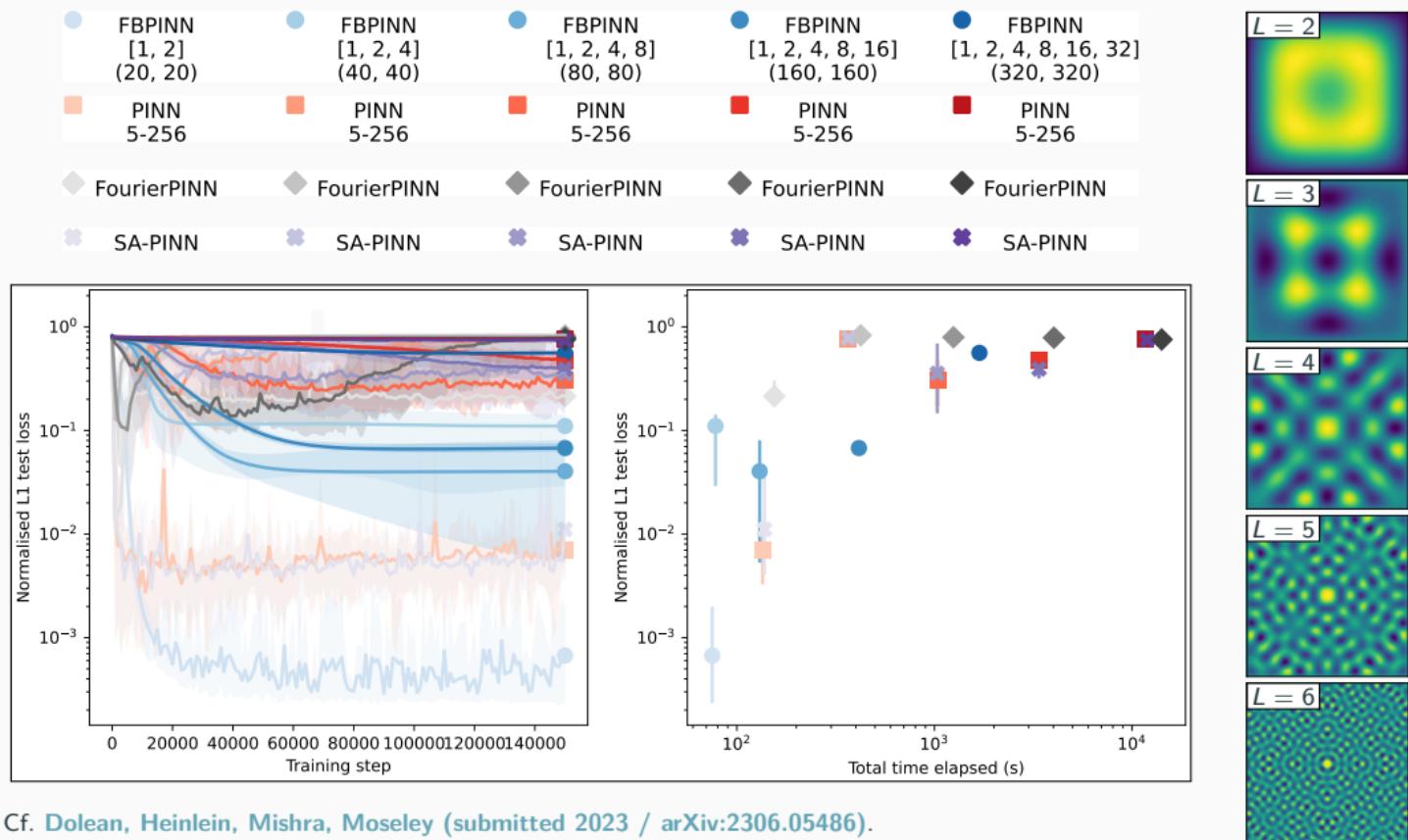
With  $k = 2^L \pi / 1.6$  and  $\sigma = 0.8 / 2^L$ , we obtain the **solutions**:



# 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 **pendulum problem**:

$$\frac{ds_1}{dt} = s_2,$$

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

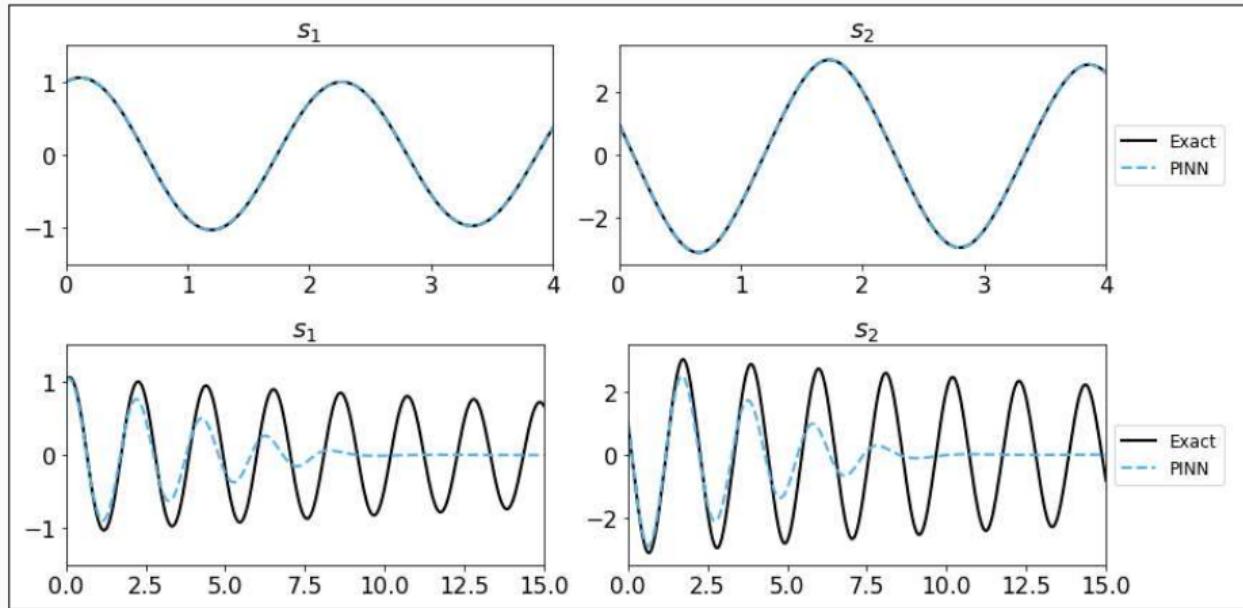
## Problem parameters

$$m = L = 1, b = 0.05,$$

$$g = 9.81$$

- Top:  $T = 4$

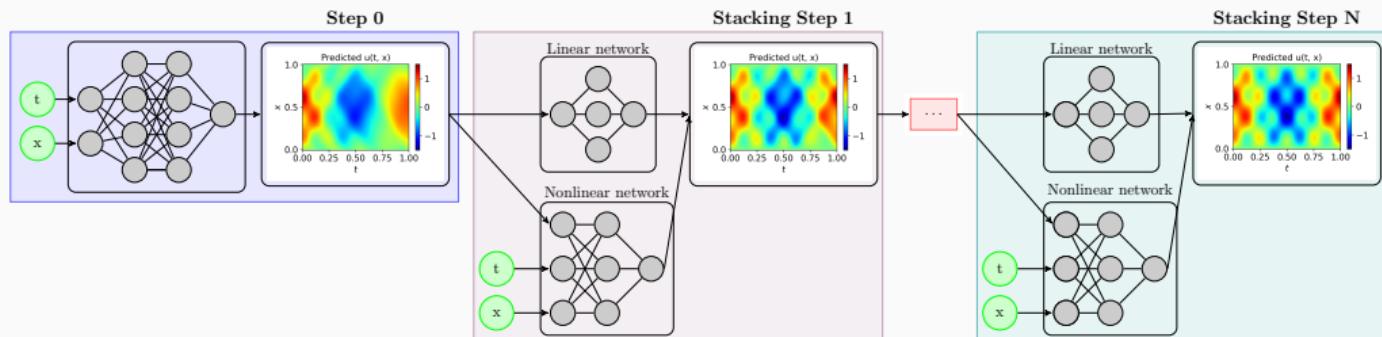
- Bottom:  $T = 20$



# 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  $u^{SF}$ :

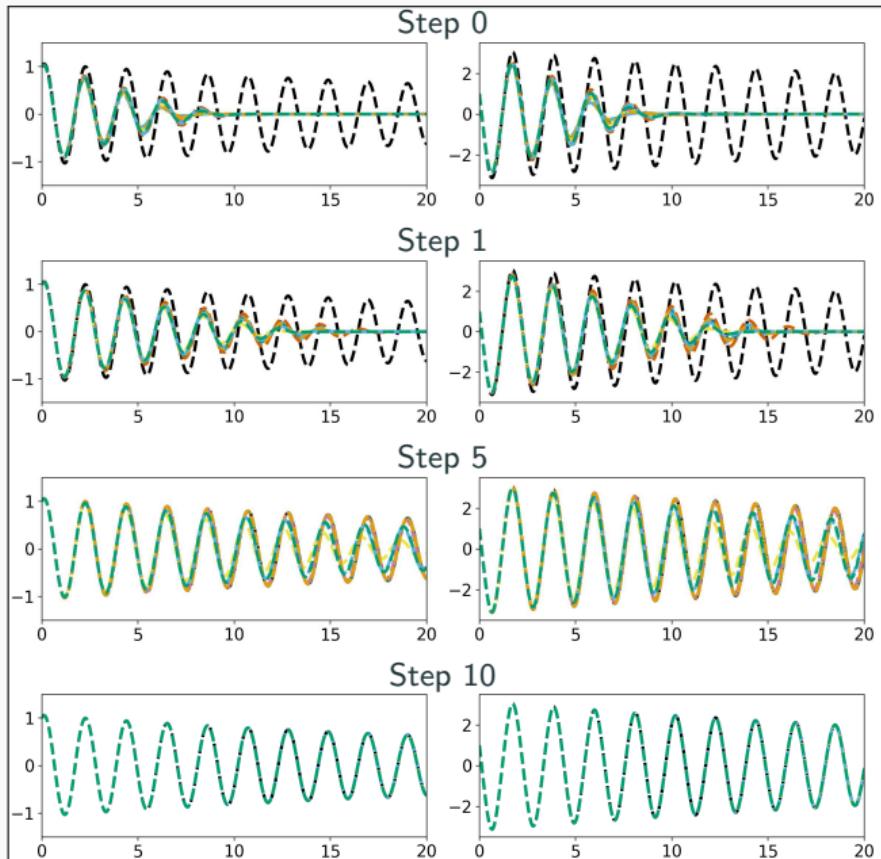
$$u^{MF}(x, \theta^{MF}) = (1 - |\alpha|) u_{\text{linear}}^{MF}(x, \theta^{MF}, u^{SF}) + |\alpha| u_{\text{nonlinear}}^{MF}(x, \theta^{MF}, u^{SF})$$



## Related works (non-exhaustive list)

- **Cokriging & multifidelity Gaussian process regression:** E.g., [Wackernagel \(1995\)](#); [Perdikaris et al. \(2017\)](#); [Babaei 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

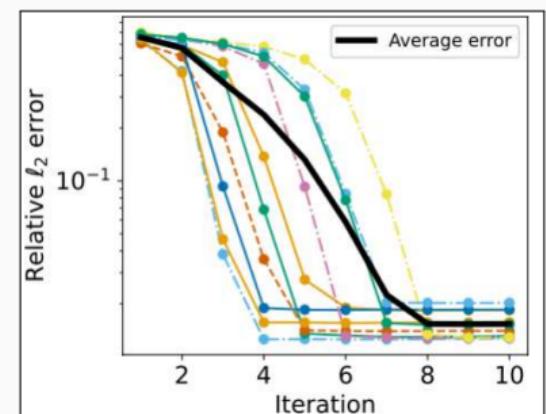


Pendulum problem:

$$\frac{d\beta_1}{dt} = \beta_2,$$

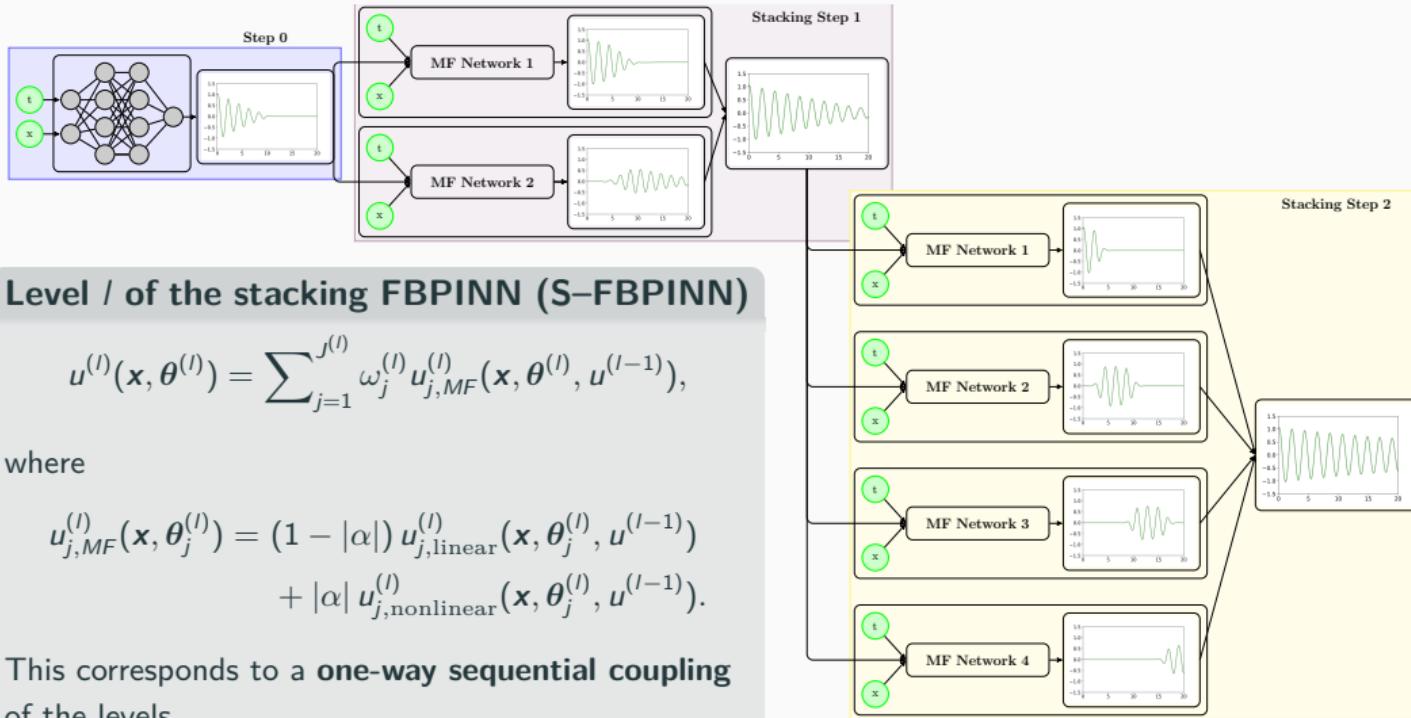
$$\frac{d\beta_2}{dt} = -\frac{b}{m}\beta_2 - \frac{g}{L} \sin(\beta_1).$$

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.



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

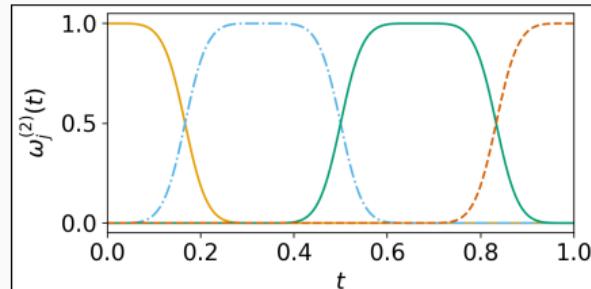
# Numerical Results – Pendulum Problem

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

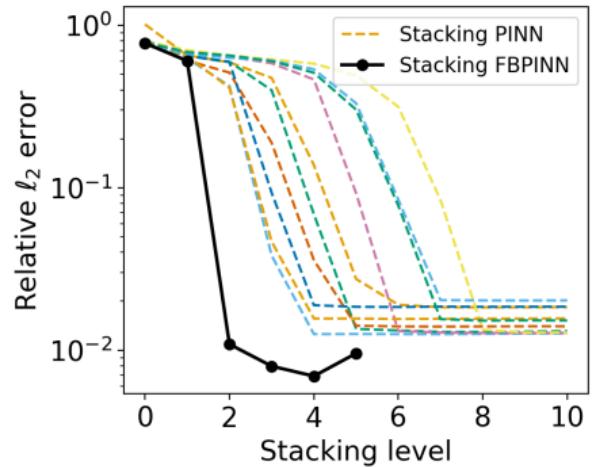
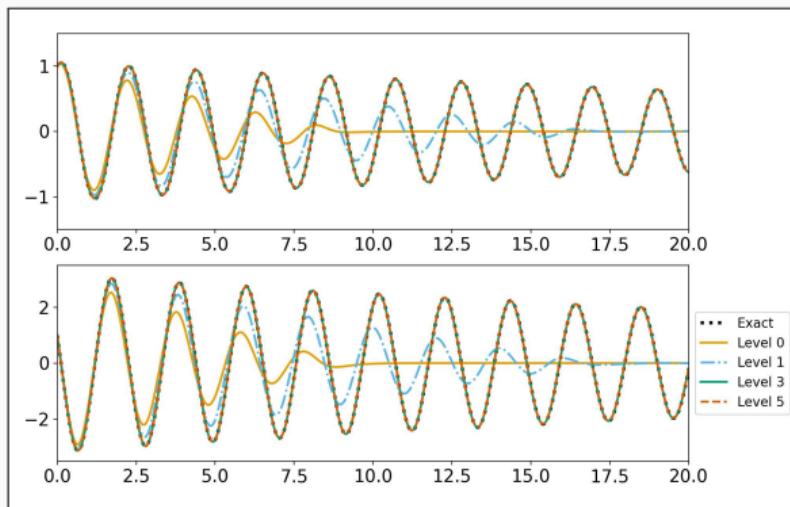
$$\frac{d\omega_1}{dt} = \omega_2,$$

$$\frac{d\omega_2}{dt} = -\frac{b}{m}\omega_2 - \frac{g}{L} \sin(\omega_1)$$

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



Exemplary partition of unity in time



# Numerical Results – Pendulum Problem

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

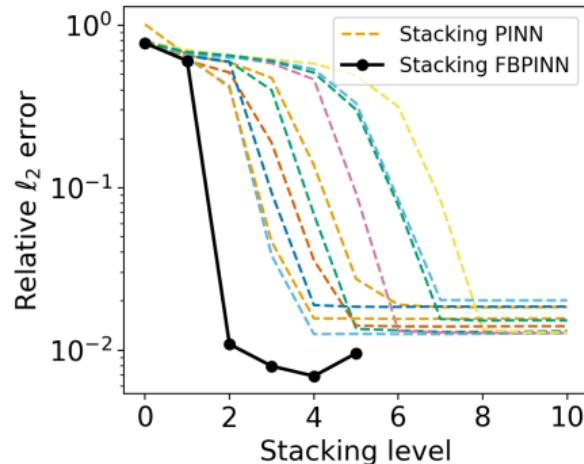
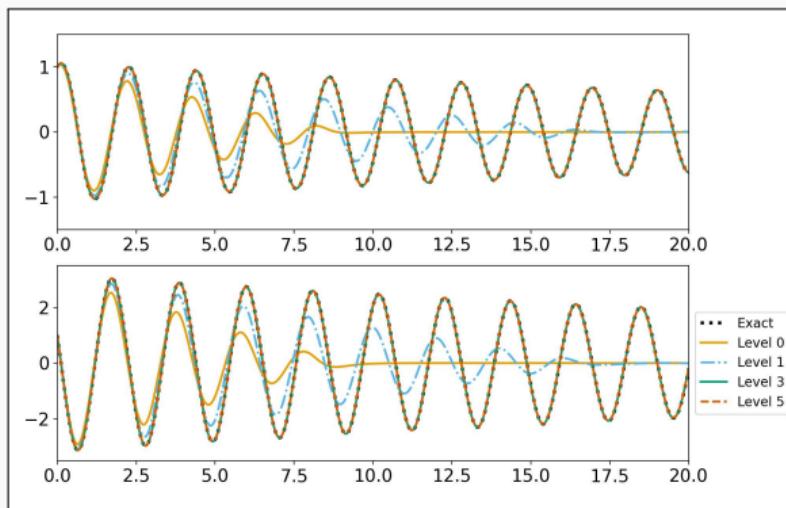
$$\frac{d\beta_1}{dt} = \beta_2,$$

$$\frac{d\beta_2}{dt} = -\frac{b}{m}\beta_2 - \frac{g}{L} \sin(\beta_1)$$

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

Model details:

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



# Numerical Results – Two-Frequency Problem

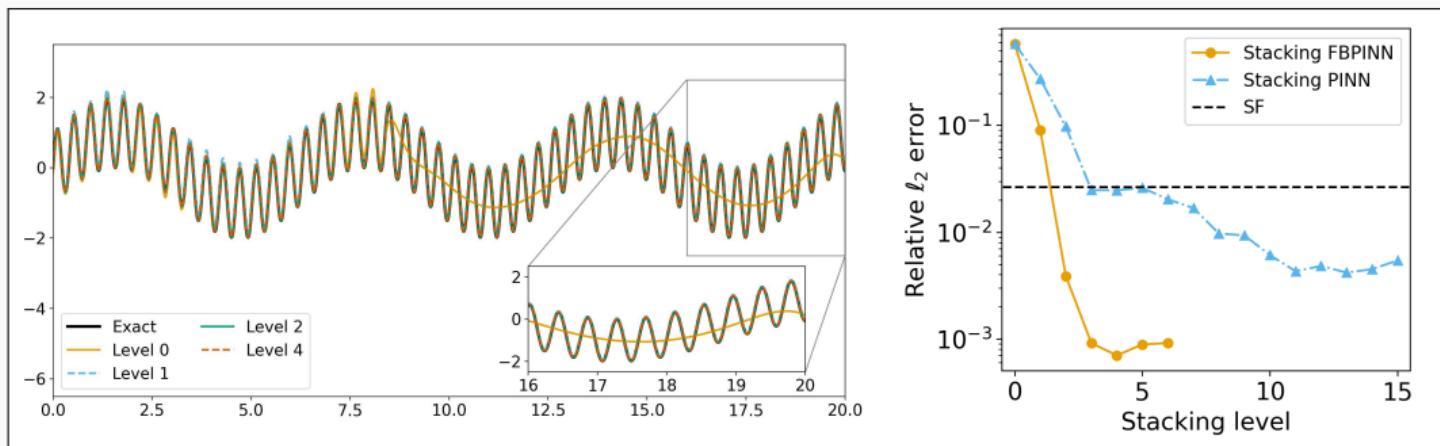
Second, we consider a **two-frequency problem**:

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

$$s(0) = 0,$$

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

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

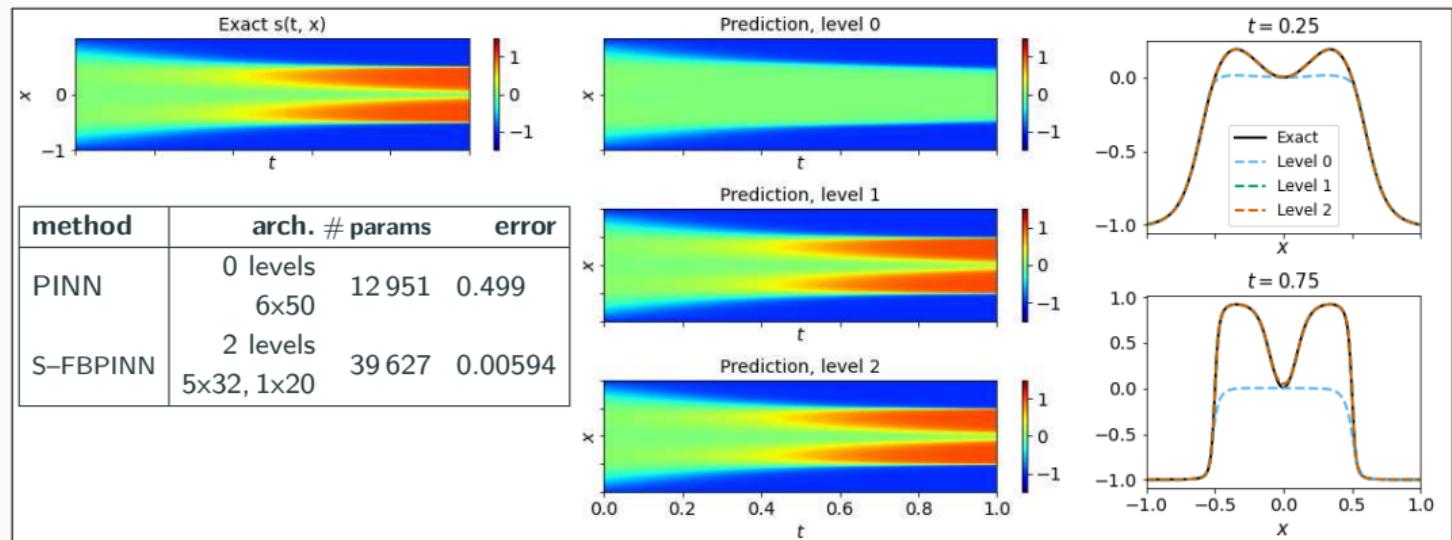


→ Due to the **multiscale structure of the problem**, the **improvements** due to the **multifidelity FBPINN approach** are **even stronger**.

# Numerical Results – Allen–Cahn Equation

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

$$\begin{aligned}\varphi_t - 0.0001\varphi_{xx} + 5\varphi^3 - 5\varphi &= 0, & t \in (0, 1], x \in [-1, 1], \\ \varphi(x, 0) &= x^2 \cos(\pi x), & x \in [-1, 1], \\ \varphi(x, t) &= \varphi(-x, t), & t \in [0, 1], x = -1, x = 1, \\ \varphi_x(x, t) &= \varphi_x(-x, t), & t \in [0, 1], x = -1, x = 1.\end{aligned}$$



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!