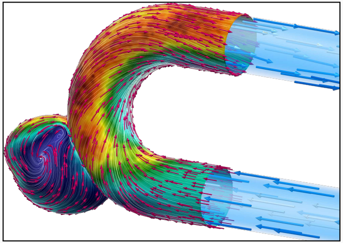
Domain decomposition for neural networks

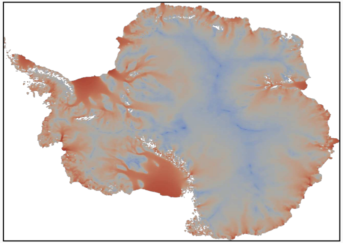
Alexander Heinlein[1]

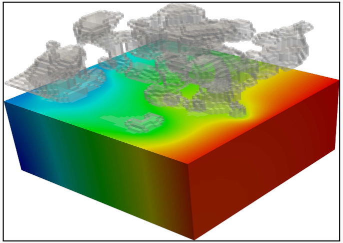
Network Platform Colloquium, University of Konstanz, Germany, October 29, 2024

[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

Scientific Machine Learning as a Standalone Field

BASIC RESEARCH NEEDS FOR

Scientific Machine Learning

Core Technologies for Artificial Intelligence

Prepared for U.S. Department of Energy Advanced Scientific Computing Research

U.S. DEPARTMENT OF ENERGY

N. Baker, A. Frank, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, K. Willcox, and S. Lee. **Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence.** USDOE Office of Science (SC), Washington, DC (United States), 2019.

Priority Research Directions

Foundational research themes:

- Domain-awareness
- Interpretability
- Robustness

Capability research themes:

- Massive scientific data analysis
- Machine learning-enhanced modeling and simulation
- Intelligent automation and decision-support for complex systems

Outline

1. Classical Domain Decomposition Methods

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

   Based on joint work with

Victorita Dolean	(Eindhoven University of Technology)
Ben Moseley and **Siddhartha Mishra**	(ETH Zürich)

3. 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)

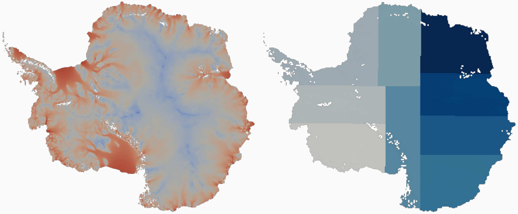
4. Domain Decomposition for Convolutional Neural Networks

   Based on joint work with

Eric Cyr	(Sandia National Laboratories)
Corné Verburg	(Delft University of Technology)

Classical Domain Decomposition Methods

Domain Decomposition Methods



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

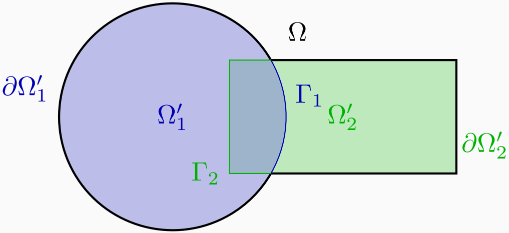
Historical remarks: The **alternating Schwarz method** is the earliest **domain decomposition method (DDM)**, which has been invented by **H. A. Schwarz** and published in **1870**:

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

Idea

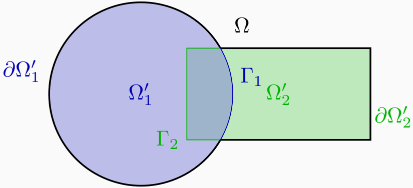
Decomposing a large **global problem** into smaller **local problems**:

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



The Alternating Schwarz Algorithm

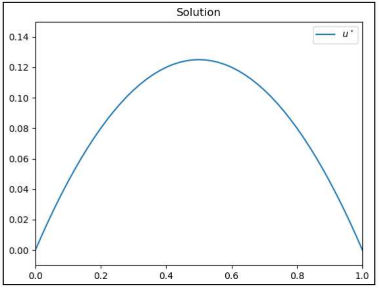
For the sake of simplicity, instead of the two-dimensional geometry,



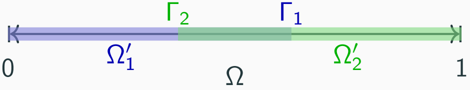
we consider the **one-dimensional Poisson equation**

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

Solution: $u(x) = -\frac{1}{2}x(x-1).$



Domain decomposition:



Let us consider the simple boundary value problem: Find u such that

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

We perform an **alternating Schwarz iteration**:

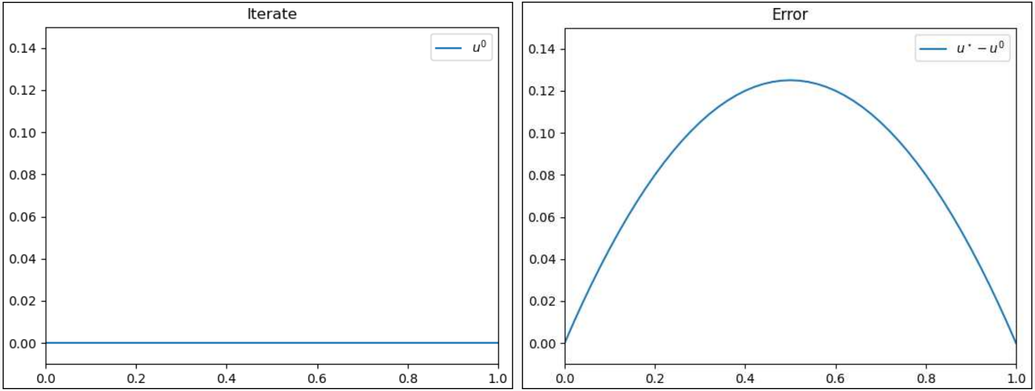


Figure 1: Iterate (left) and error (right) in iteration 0.

Let us consider the simple boundary value problem: Find u such that

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

We perform an **alternating Schwarz iteration**:

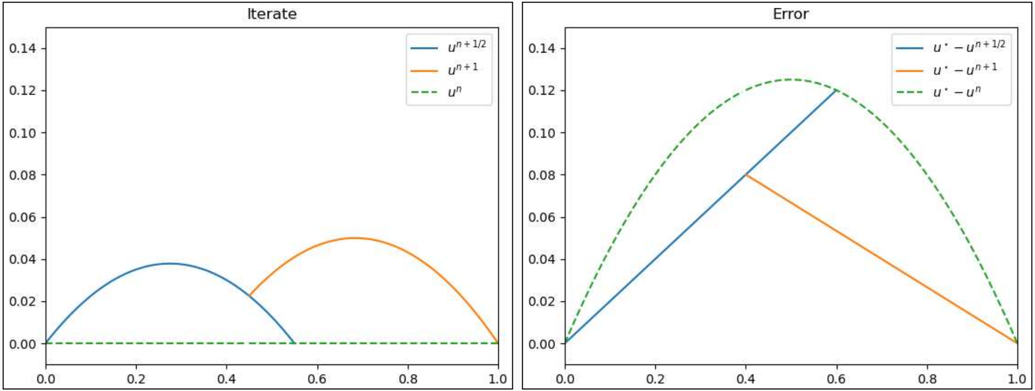


Figure 1: Iterate (left) and error (right) in iteration 1.

Let us consider the simple boundary value problem: Find u such that

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

We perform an **alternating Schwarz iteration**:

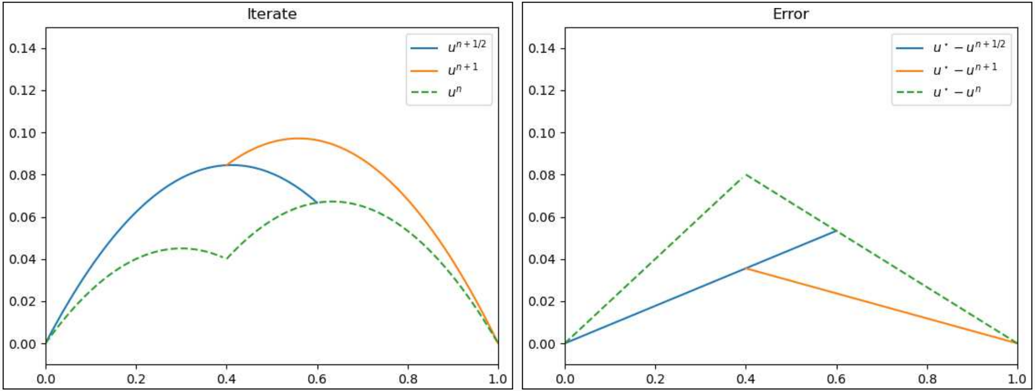


Figure 1: Iterate (left) and error (right) in iteration 2.

Let us consider the simple boundary value problem: Find u such that

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

We perform an **alternating Schwarz iteration**:

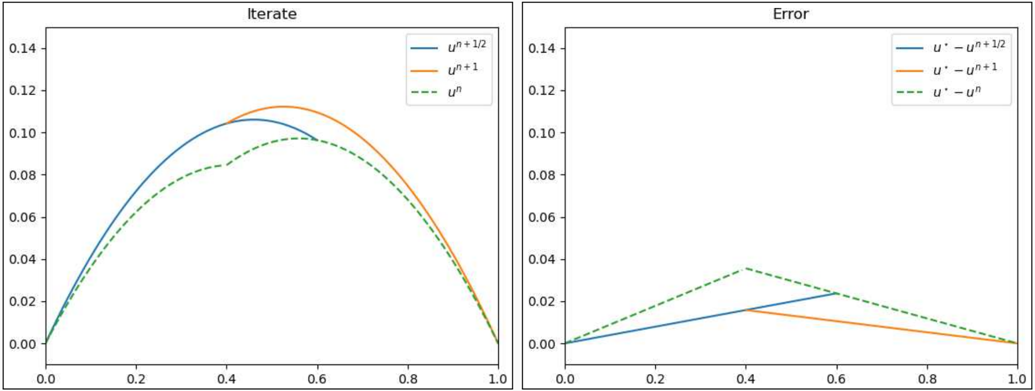


Figure 1: Iterate (left) and error (right) in iteration 3.

Let us consider the simple boundary value problem: Find u such that

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

We perform an **alternating Schwarz iteration**:

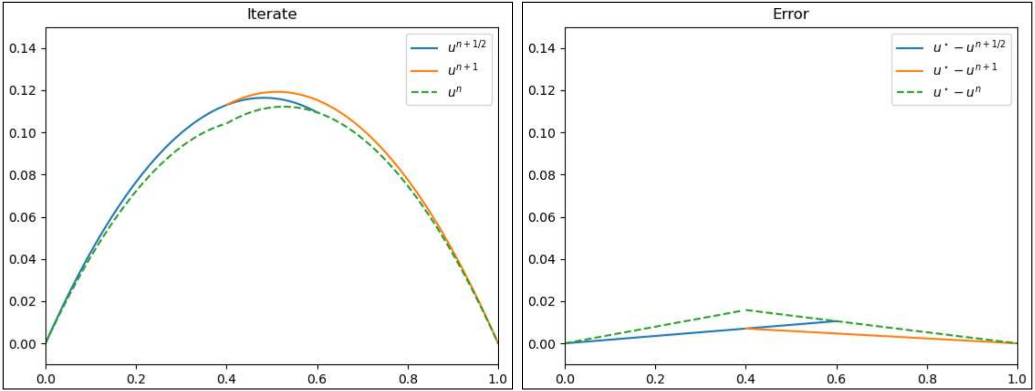


Figure 1: Iterate (left) and error (right) in iteration 4.

Let us consider the simple boundary value problem: Find u such that

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

We perform an **alternating Schwarz iteration**:

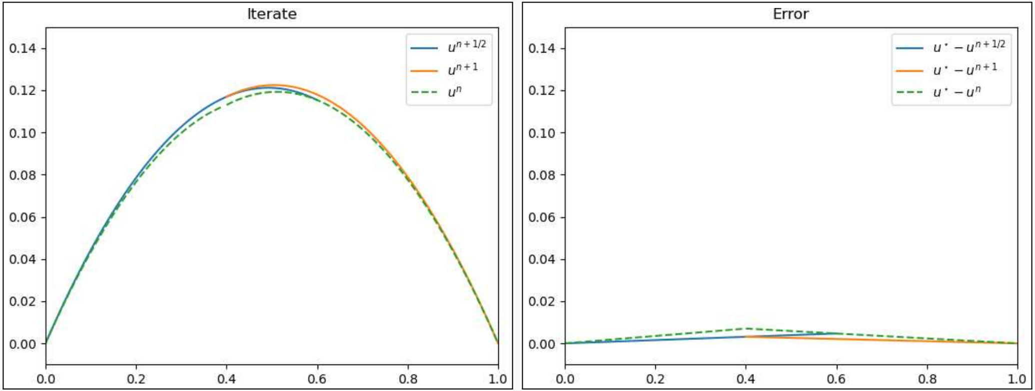


Figure 1: Iterate (left) and error (right) in iteration 5.

The alternating Schwarz algorithm is **sequential** because **each local boundary value problem** depends on the solution of the **previous Dirichlet problem**:

$$(D_1)\begin{cases} -\Delta u^{n+1/2} &= f & \text{in } \Omega_1', \\ u^{n+1/2} &= \mathbf{u^n} & \text{on } \partial\Omega_1' \\ u^{n+1/2} &= \mathbf{u^n} & \text{on } \Omega \setminus \overline{\Omega_1'} \end{cases}$$

$$(D_2)\begin{cases} -\Delta u^{n+1} &= f & \text{in } \Omega_2, \\ u^{n+1} &= \mathbf{u^{n+1/2}} & \text{on } \partial\Omega_2' \\ u^{n+1} &= \mathbf{u^{n+1/2}} & \text{on } \Omega \setminus \overline{\Omega_2'} \end{cases}$$

Idea: For all red terms, we **use the values from the previous iteration**. Then, the both Dirichlet problem **can be solved at the same time**.

The alternating Schwarz algorithm is **sequential** because **each local boundary value problem** depends on the solution of the **previous Dirichlet problem**:

$$(D_1)\begin{cases} -\Delta u^{n+1/2} = f & \text{in } \Omega_1', \\ u^{n+1/2} = \mathbf{u^{n}} & \text{on } \partial\Omega_1' \\ u^{n+1/2} = \mathbf{u^{n}} & \text{on } \Omega \setminus \overline{\Omega_1'} \end{cases}$$

$$(D_2)\begin{cases} -\Delta u^{n+1} = f & \text{in } \Omega_2, \\ u^{n+1} = \mathbf{u^{n+1/2}} & \text{on } \partial\Omega_2' \\ u^{n+1} = \mathbf{u^{n+1/2}} & \text{on } \Omega \setminus \overline{\Omega_2'} \end{cases}$$

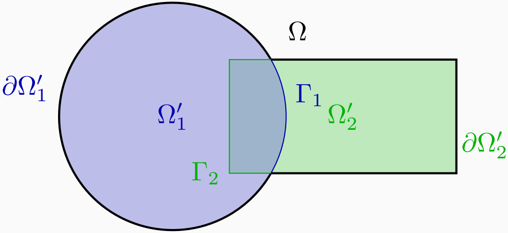
Idea: For all red terms, we **use the values from the previous iteration**. Then, the both Dirichlet problem **can be solved at the same time**.

The Parallel Schwarz Algorithm

The **parallel Schwarz algorithm** has been introduced by **Lions (1988)**. Here, we solve the local problems

$$(D_1)\begin{cases} -\Delta u_1^{n+1} &= f & \text{in } \Omega_1', \\ u_1^{n+1} &= u_2^n & \text{on } \partial\Omega_1', \end{cases}$$

$$(D_2)\begin{cases} -\Delta u_2^{n+1} &= f & \text{in } \Omega_2, \\ u_2^{n+1} &= u_1^n & \text{on } \partial\Omega_2'. \end{cases}$$



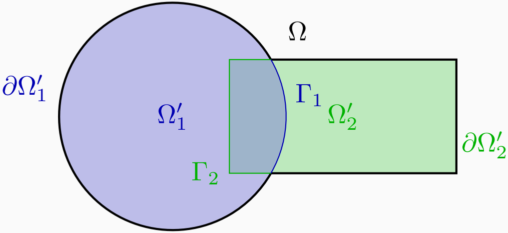
Since u_1^n and u_2^n are both computed in the previous iteration, the problems can be solved independent of each other.

The Parallel Schwarz Algorithm

The **parallel Schwarz algorithm** has been introduced by **Lions (1988)**. Here, we solve the local problems

$$(D_1)\begin{cases} -\Delta u_1^{n+1} &= f \quad \text{in } \Omega_1', \\ u_1^{n+1} &= u_2^n \quad \text{on } \partial\Omega_1', \end{cases}$$

$$(D_2)\begin{cases} -\Delta u_2^{n+1} &= f \quad \text{in } \Omega_2, \\ u_2^{n+1} &= u_1^n \quad \text{on } \partial\Omega_2'. \end{cases}$$



Since u_1^n and u_2^n are both computed in the previous iteration, the problems can be solved independent of each other.

This method is suitable for **parallel computing**!

Let us again consider the simple boundary value problem: Find u such that

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

We perform the **parallel Schwarz iteration**:

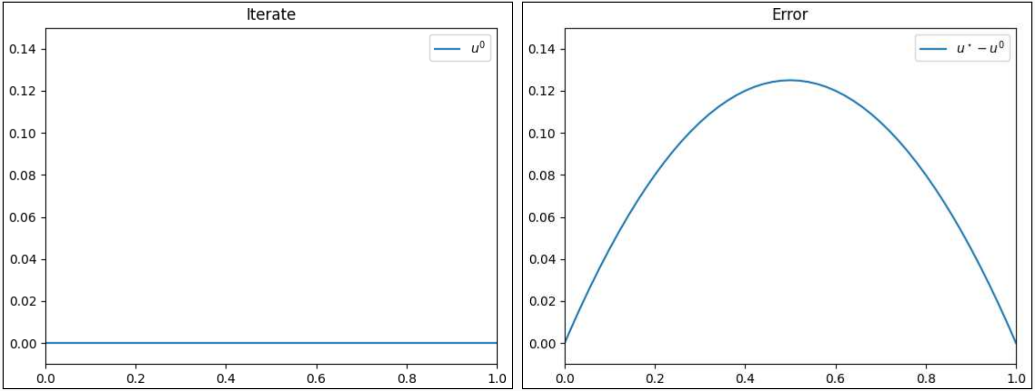


Figure 2: Iterate (left) and error (right) in iteration 0.

Let us again consider the simple boundary value problem: Find u such that

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

We perform the **parallel Schwarz iteration**:

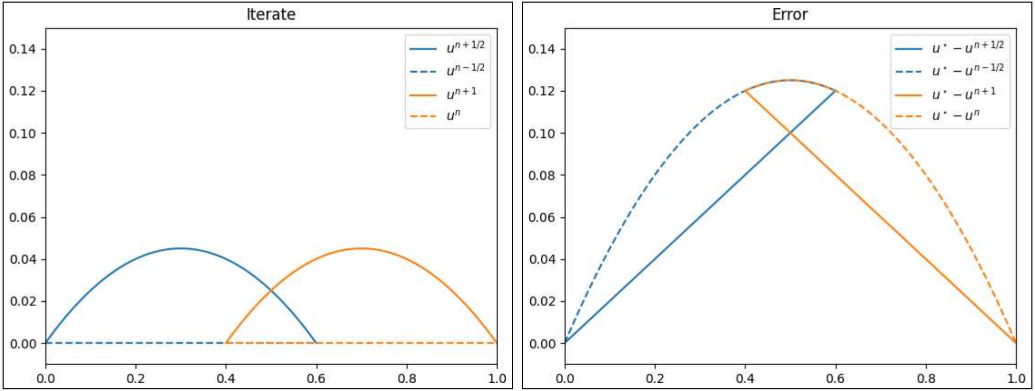


Figure 2: Iterate (left) and error (right) in iteration 1.

Let us again consider the simple boundary value problem: Find u such that

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

We perform the **parallel Schwarz iteration**:

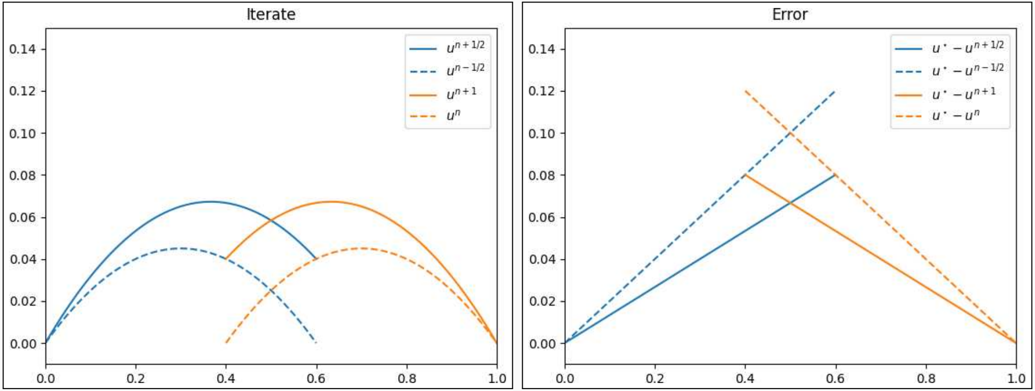


Figure 2: Iterate (left) and error (right) in iteration 2.

Let us again consider the simple boundary value problem: Find u such that

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

We perform the **parallel Schwarz iteration**:

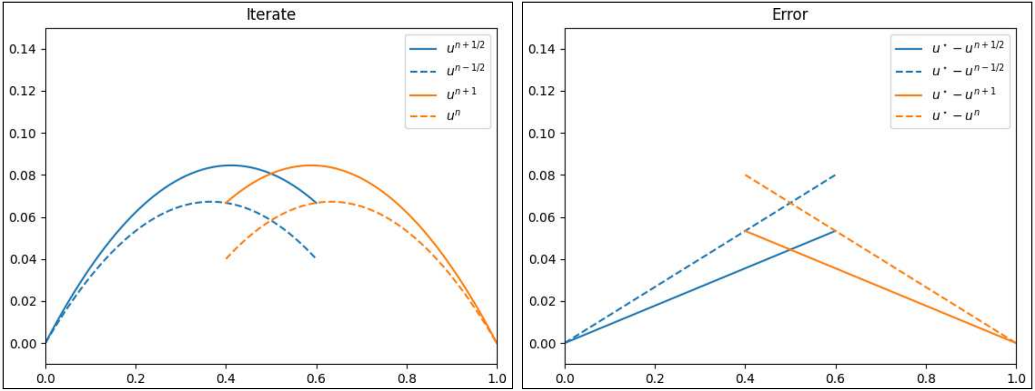


Figure 2: Iterate (left) and error (right) in iteration 3.

Let us again consider the simple boundary value problem: Find u such that

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

We perform the **parallel Schwarz iteration**:

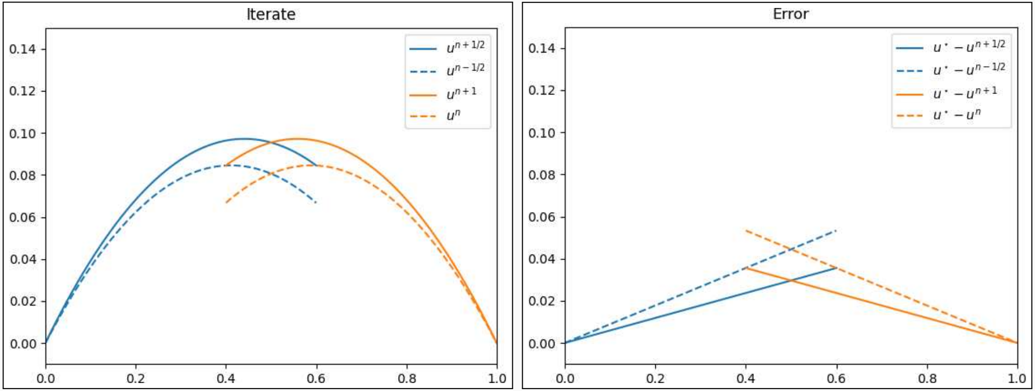


Figure 2: Iterate (left) and error (right) in iteration 4.

Let us again consider the simple boundary value problem: Find u such that

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

We perform the **parallel Schwarz iteration**:

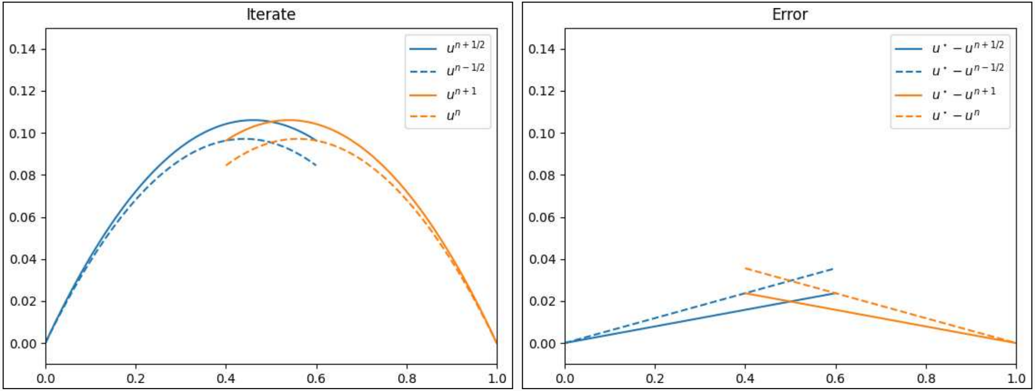


Figure 2: Iterate (left) and error (right) in iteration 5.

Solvers for Partial Different Equations

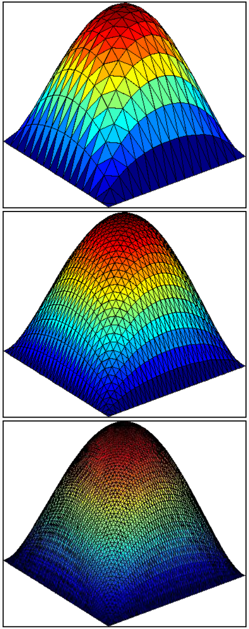
Consider a **diffusion model problem**:

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

Discretization using finite elements yields a **sparse** system of linear equations

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}.$$

The accuracy of the finite element solution depends on the refinement level of the mesh: **higher refinement** ⇒ **better accuracy**.



Direct solvers

For fine meshes, solving the system using a direct solver is not feasible due to **superlinear complexity and memory cost**.

Iterative solvers

Iterative solvers are efficient for solving sparse linear systems of equations, however, the **convergence rate generally depends on refinement level.**

Solvers for Partial Different Equations

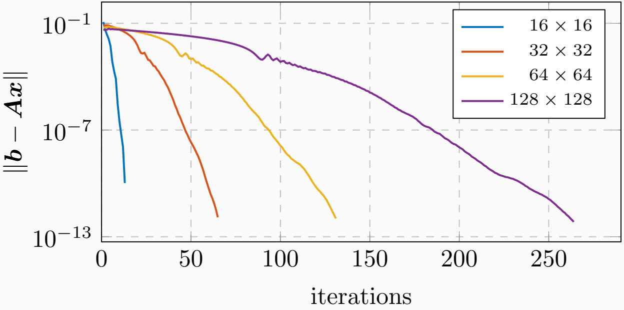
Consider a **diffusion model problem**:

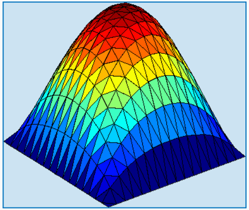
$$-\Delta u(x) = f \quad \text{in } \Omega = [0,1]^2,$$
$$u = 0 \quad \text{on } \partial\Omega.$$

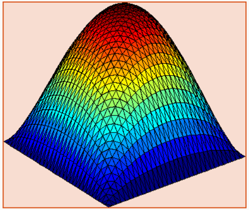
We solve the system

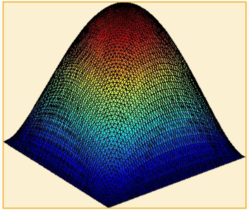
$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$$

using the **conjugate gradient (CG) iterative method.**





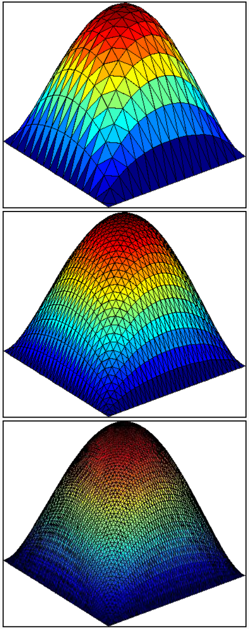




One-Level Schwarz Preconditioners

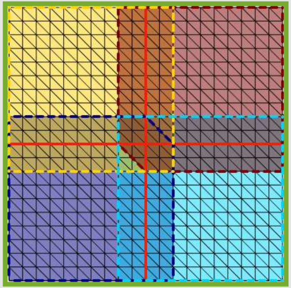
In order to improve convergence, instead of $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$, we solve

$$\boldsymbol{M}^{-1}\boldsymbol{K}\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{f}.$$

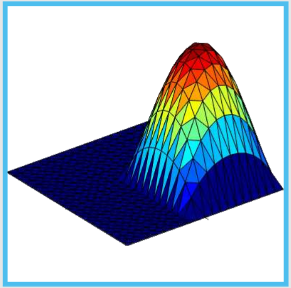


One-level Schwarz preconditioner

Overlap $\delta = 1h$



Solution of local problem



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

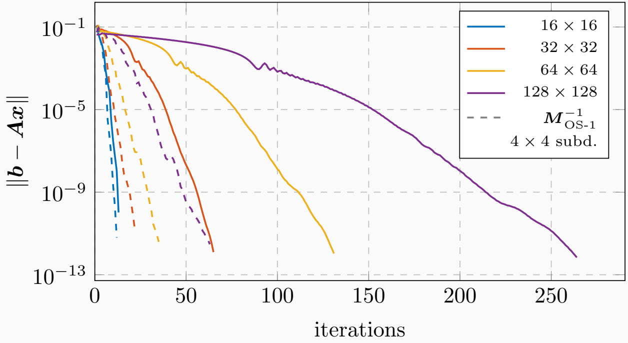
One-Level Schwarz Preconditioners

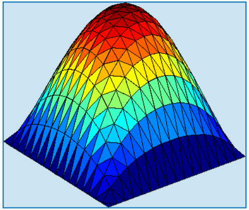
In order to improve convergence, instead of $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$, we solve

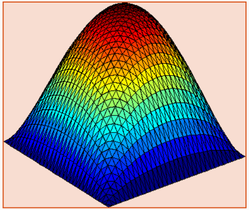
$$\boldsymbol{M}^{-1}\boldsymbol{K}\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{f}.$$

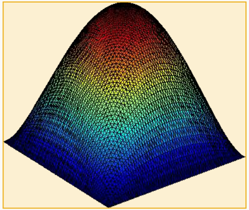
One-level Schwarz preconditioner

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

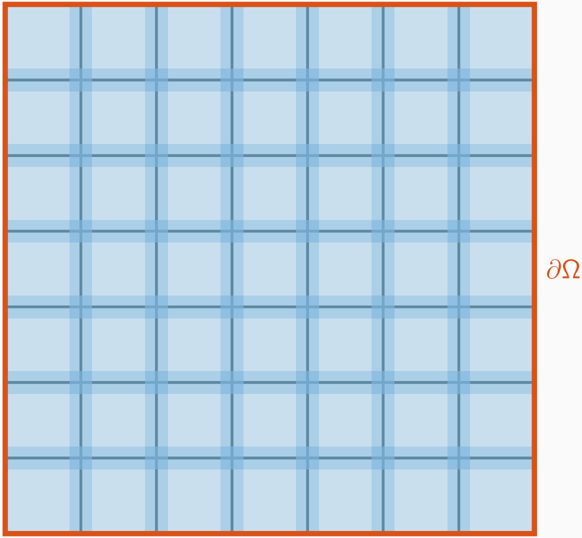




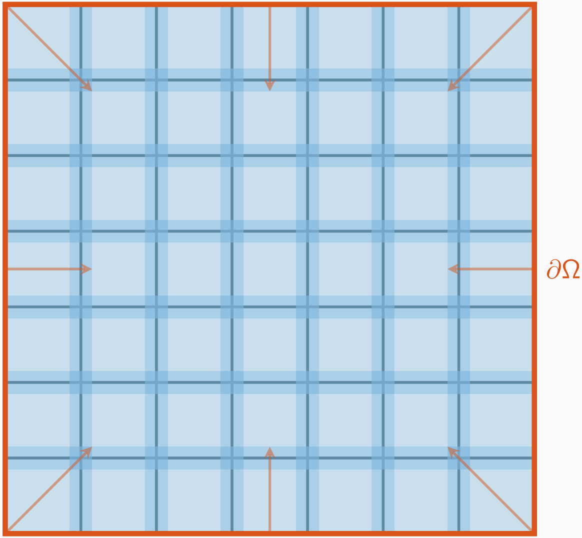




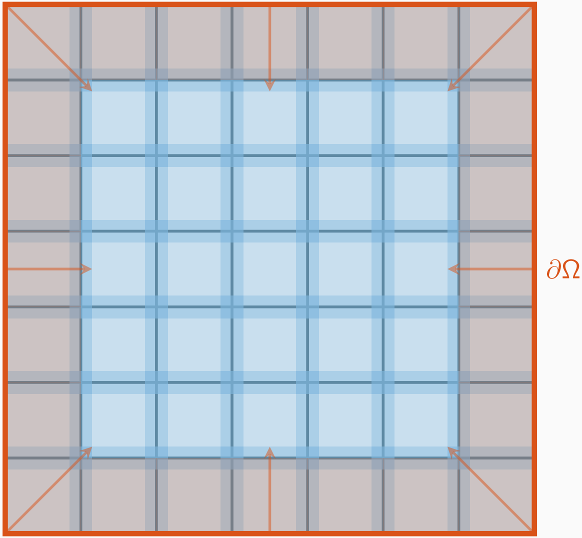
Transport of Information – One-Level Overlapping Schwarz Methods



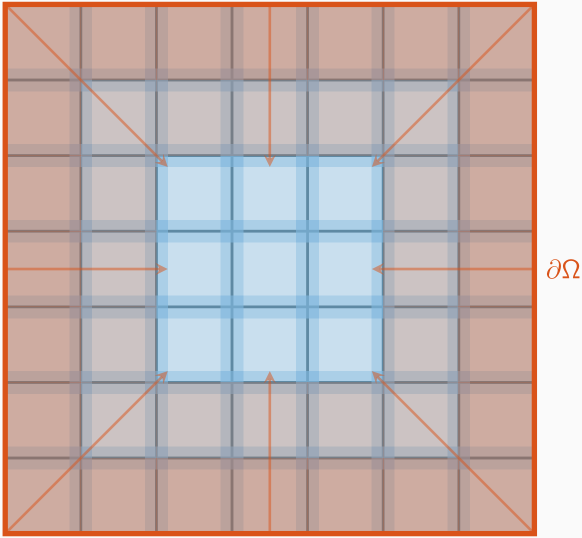
Transport of Information – One-Level Overlapping Schwarz Methods



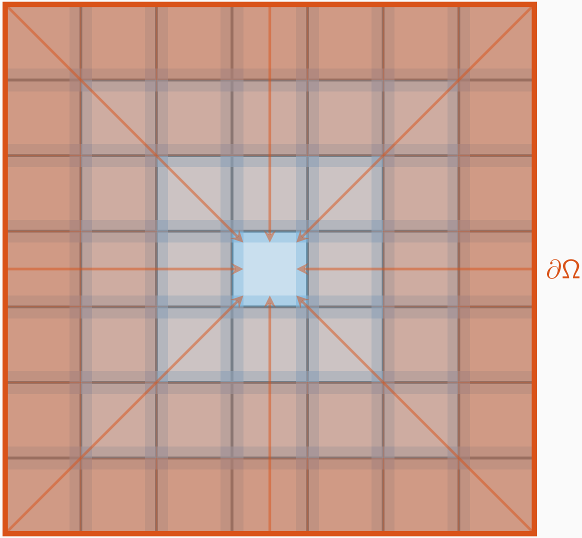
Transport of Information – One-Level Overlapping Schwarz Methods



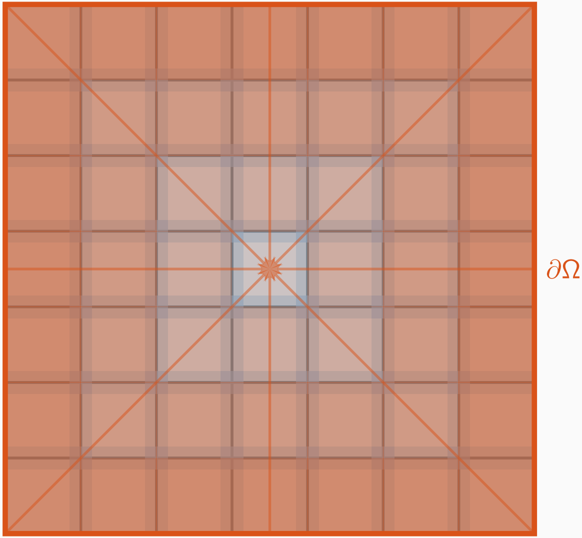
Transport of Information – One-Level Overlapping Schwarz Methods



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**.

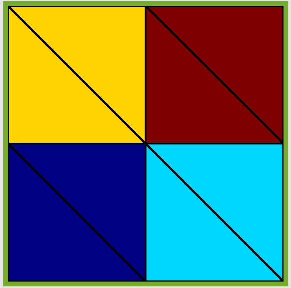
Schwarz Preconditioners

In order to improve convergence, instead of $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$, we solve

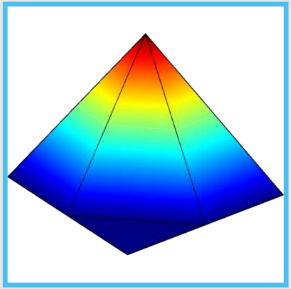
$$\boldsymbol{M}^{-1}\boldsymbol{K}\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{f}.$$

Two-level Schwarz preconditioner

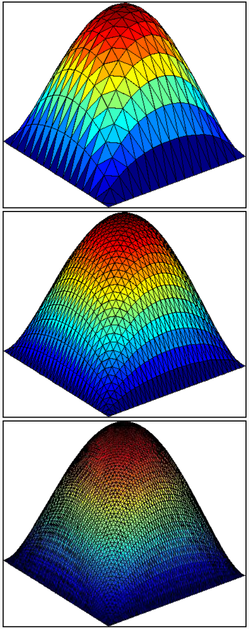
Coarse triangulation



Coarse solution



$$\boldsymbol{M}_{\text{OS-2}}^{-1}\boldsymbol{K} = \Phi \boldsymbol{K}_0^{-1} \Phi^\top \boldsymbol{K} + \sum\nolimits_{i=1}^{N} \boldsymbol{R}_i^\top \boldsymbol{K}_i^{-1} \boldsymbol{R}_i \boldsymbol{K},$$



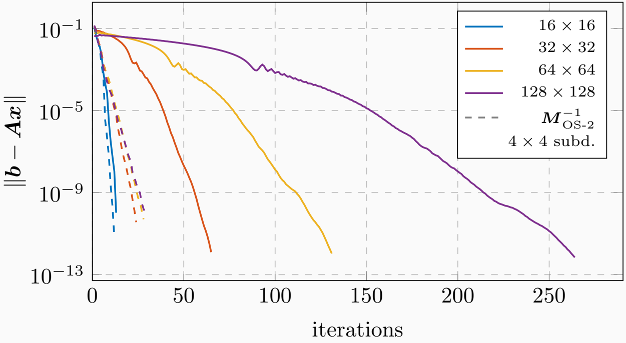
Schwarz Preconditioners

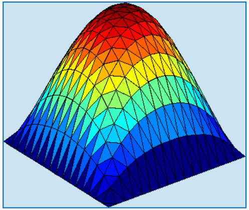
In order to improve convergence, instead of $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$, we solve

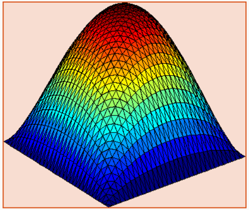
$$\boldsymbol{M}^{-1}\boldsymbol{K}\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{f}.$$

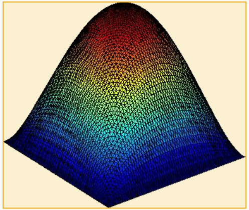
Two-level Schwarz preconditioner

$$\boldsymbol{M}_{\text{OS-2}}^{-1}\boldsymbol{K} = \Phi \boldsymbol{K}_0^{-1} \Phi^\top \boldsymbol{K} + \sum\nolimits_{i=1}^{N} \boldsymbol{R}_i^\top \boldsymbol{K}_i^{-1} \boldsymbol{R}_i \boldsymbol{K},$$



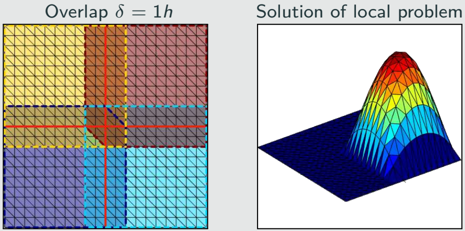






Two-Level Schwarz Preconditioners – Weak Scaling Study

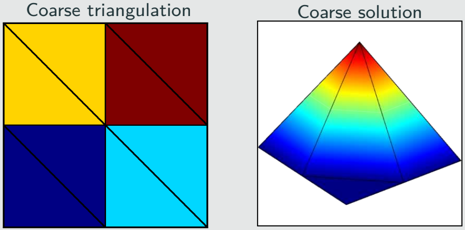
One-level Schwarz preconditioner



Overlap $\delta = 1h$

Solution of local problem

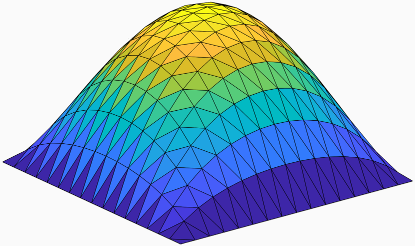
Lagrangian coarse space

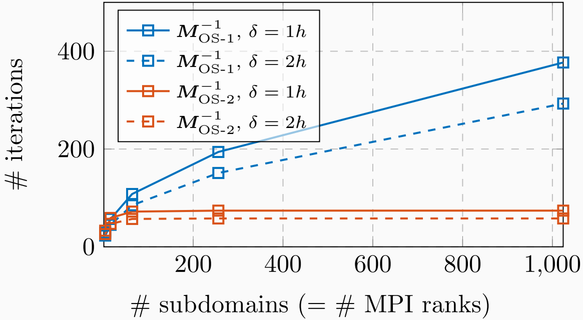


Coarse triangulation

Coarse solution

Diffusion model problem in two dimensions,
$H/h = 100$



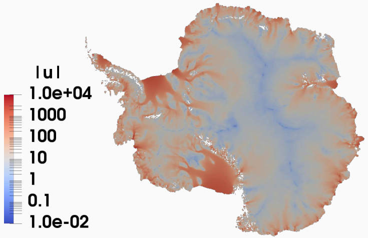


$M_{\text{OS-1}}^{-1}$, $\delta = 1h$
$M_{\text{OS-1}}^{-1}$, $\delta = 2h$
$M_{\text{OS-2}}^{-1}$, $\delta = 1h$
$M_{\text{OS-2}}^{-1}$, $\delta = 2h$
# iterations
0
200
400
200
400
600
800
1,000
# subdomains (= # MPI ranks)

Schwarz Preconditioners for Land Ice Simulations

Albany

https://github.com/SNLComputation/Albany



The velocity of the ice sheet in Antarctica and Greenland is modeled by a **first-order-accurate Stokes approximation model**,

$$-\nabla \cdot (2\mu\dot{\epsilon}_1) + \rho g \frac{\partial s}{\partial x} = 0, \quad -\nabla \cdot (2\mu\dot{\epsilon}_2) + \rho g \frac{\partial s}{\partial y} = 0,$$

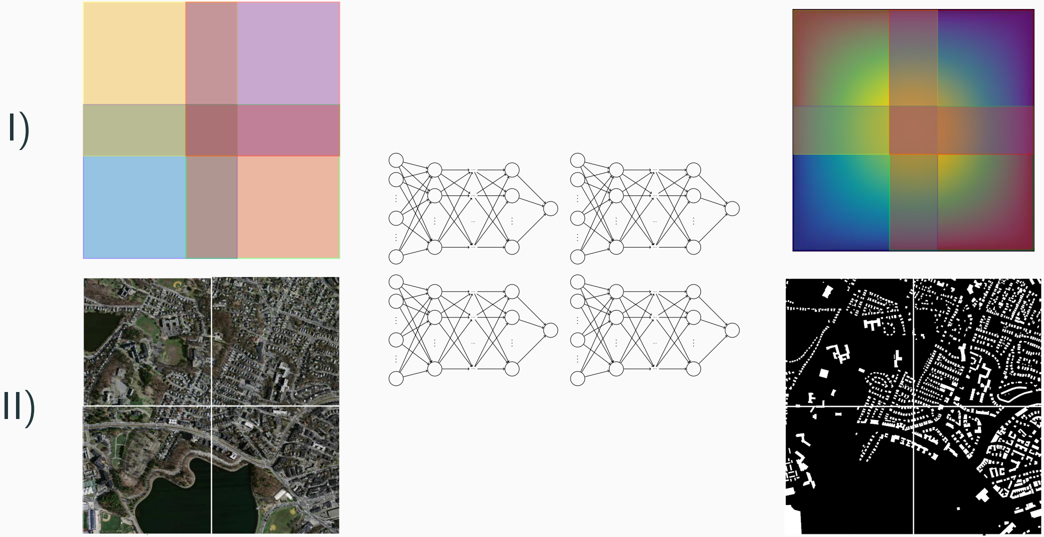
with a **nonlinear viscosity model** (Glen's law); cf., e.g., **Blatter (1995)** and **Pattyn (2003)**.

	Antarctica (**velocity**)			Greenland (**multiphysics vel. & temperature**)		
	4 km resolution, 20 layers, 35 m dofs			1-10 km resolution, 20 layers, 69 m dofs		
MPI ranks	avg. its	avg. setup	avg. solve	avg. its	avg. setup	avg. solve
512	**41.9** (11)	25.10 s	12.29 s	**41.3** (36)	18.78 s	4.99 s
1 024	**43.3** (11)	9.18 s	5.85 s	**53.0** (29)	8.68 s	4.22 s
2 048	**41.4** (11)	4.15 s	2.63 s	**62.2** (86)	4.47 s	4.23 s
4 096	**41.2** (11)	1.66 s	1.49 s	**68.9** (40)	2.52 s	2.86 s
8 192	**40.2** (11)	1.26 s	1.06 s	-	-	-

Computations performed on Cori (NERSC).

Heinlein, Perego, Rajamanickam (2022)

Domain Decomposition for Neural Networks



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 2022, arXiv 2023); Kim, Yang (2022, arXiv 2023)**
- **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:

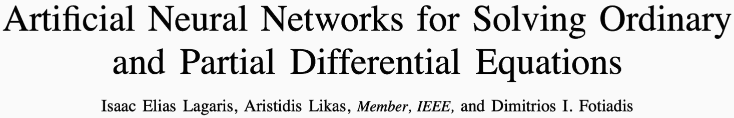
A. Klawonn, M. Lanser, J. Weber

Machine learning and domain decomposition methods – a survey

arXiv:2312.14050. 2023

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

Neural Networks for Solving Differential Equations



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)**

Neural Networks for Solving Differential Equations

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**

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

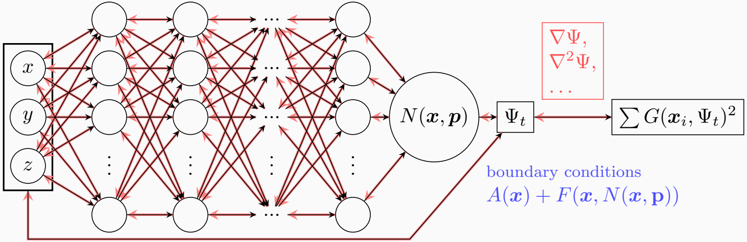
where $\Psi_t\left(\boldsymbol{x}, \boldsymbol{\theta}\right)$ is a **trial function**, $\boldsymbol{x}_i$ **sampling points inside the domain** Ω and $\boldsymbol{\theta}$ are **adjustable parameters**.

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**



x
y
z
$N(\boldsymbol{x}, \boldsymbol{p})$
Ψ_t
$\nabla\Psi,$
$\nabla^2\Psi,$
$\ldots$
$\sum G(\boldsymbol{x}_i, \Psi_t)^2$
boundary conditions
$A(\boldsymbol{x}) + F(\boldsymbol{x}, N(\boldsymbol{x}, \mathbf{p}))$

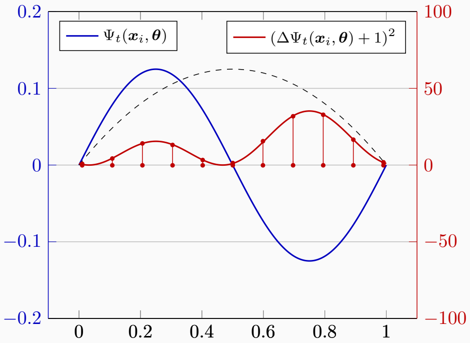
Physics-Informed Neural Networks (PINNs) – Idea

In **Lagaris et al. (1998)**, the authors solve the **boundary value problem**

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

via a **collocation approach**:

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



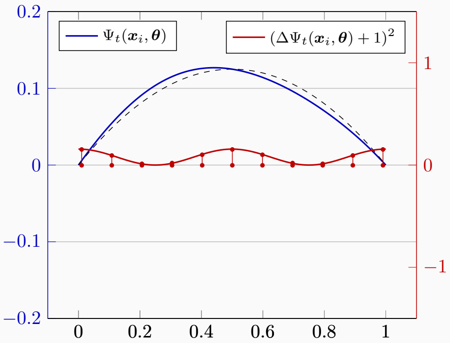
$(\Delta\Psi_t(\boldsymbol{x}_i,\boldsymbol{\theta}) + 1)^2 >> 0$

Boundary conditions ...

... can be **enforced explicitly** via the ansatz:

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

- **A satisfies the boundary conditions**
- **F does not contribute to the boundary conditions**



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

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**

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

PINNs use a **hybrid loss function**:

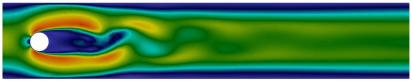
$$\mathcal{L}(\boldsymbol{\theta}) = \omega_{\text{data}} \mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}),$$

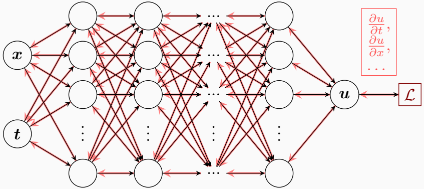
where ω_{data} and ω_{PDE} are **weights** and

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

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

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





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

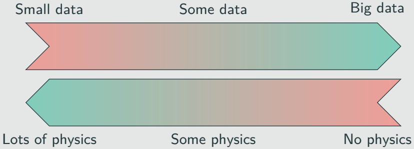
Advantages

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

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

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

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**

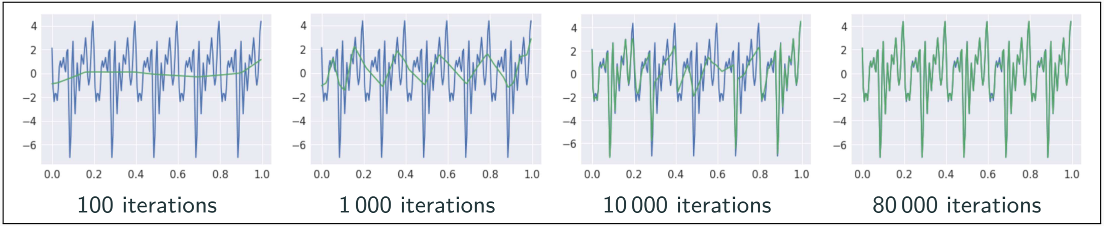
Rule of thumb:

“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.



100 iterations

1 000 iterations

10 000 iterations

80 000 iterations

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

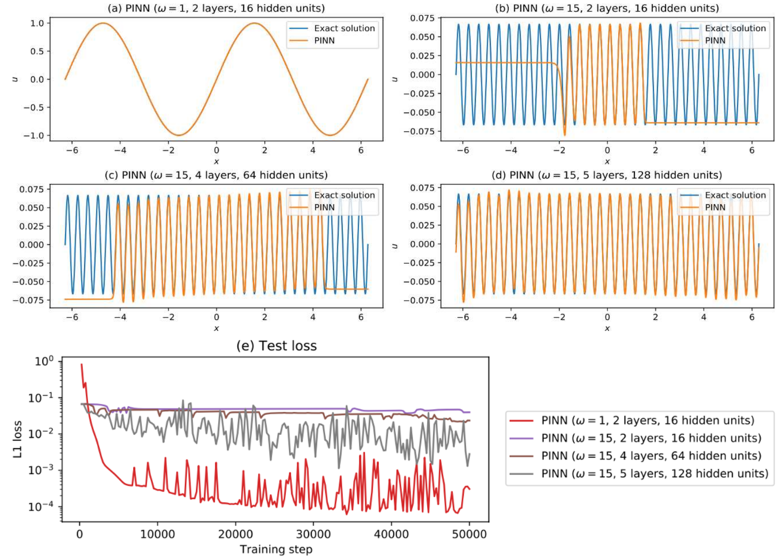
$$\begin{aligned} u' &= \cos(\omega \boldsymbol{x}), \\ u(0) &= 0, \end{aligned}$$

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

Motivation – Some Observations on the Performance of PINNs

Solve

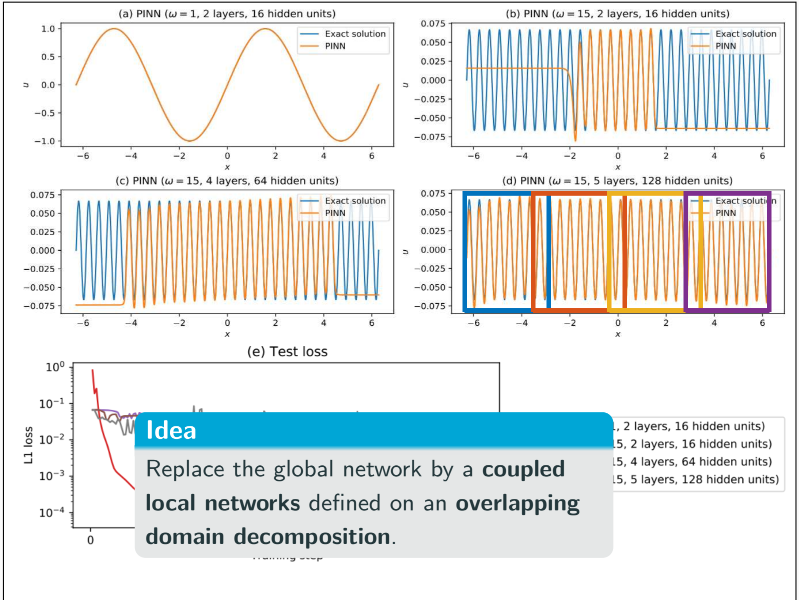
$$\begin{aligned} u' &= \cos(\omega \boldsymbol{x}), \\ u(0) &= 0, \end{aligned}$$

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

Scaling issues

- Large computational domains
- Small frequencies

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



Idea

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

(a) 321 free parameters

(d) 66 433 free parameters

Finite Basis Physics-Informed Neural Networks (FBPINNs)

In the **finite basis physics informed neural network (FBPINNs) method** introduced in **Moseley, Markham, and Nissen-Meyer (2023)**, we employ the **PINN** approach and **hard enforcement of the boundary conditions**; cf. **Lagaris et al. (1998)**.

FBPINNs use the **network architecture**

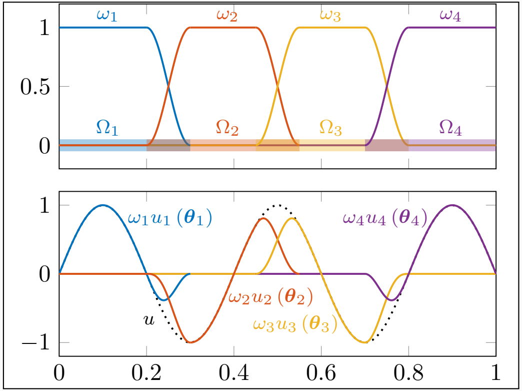
$$u(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \mathcal{C} \sum\nolimits_{j=1}^{J} \omega_j u_j \left(\boldsymbol{\theta}_j\right)$$

and the **loss function**

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

Here:

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



Hard enf. of boundary conditions

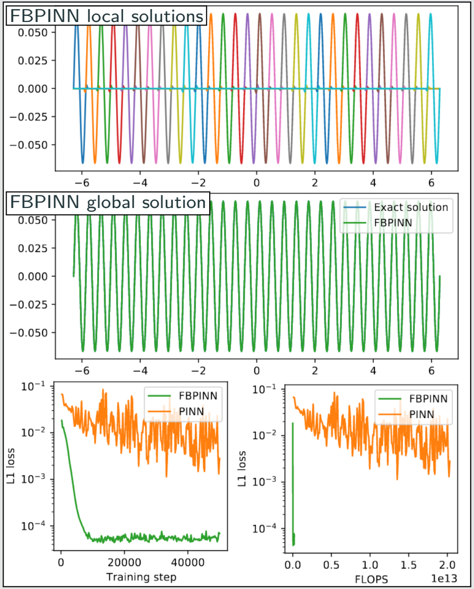
Loss function

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

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

Numerical Results for FBPINNs

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



Scalability of FBPINNs

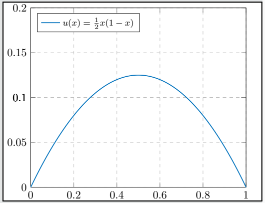
Consider the **simple boundary value problem**

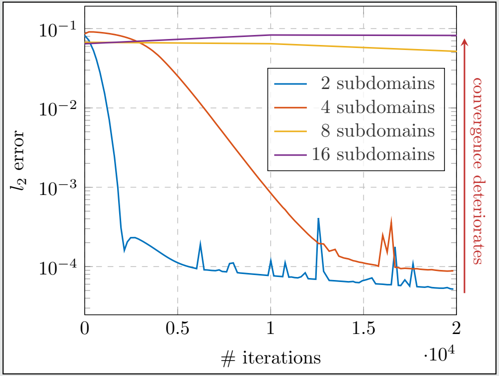
$$-u'' = 1 \text{ in } [0,1],$$

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

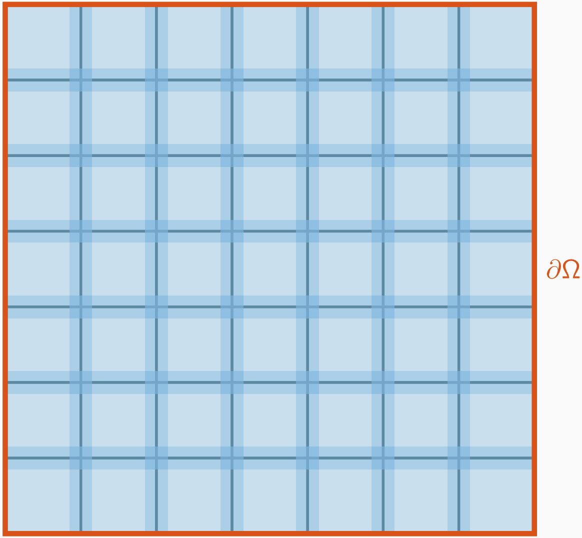
which has the **solution**

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

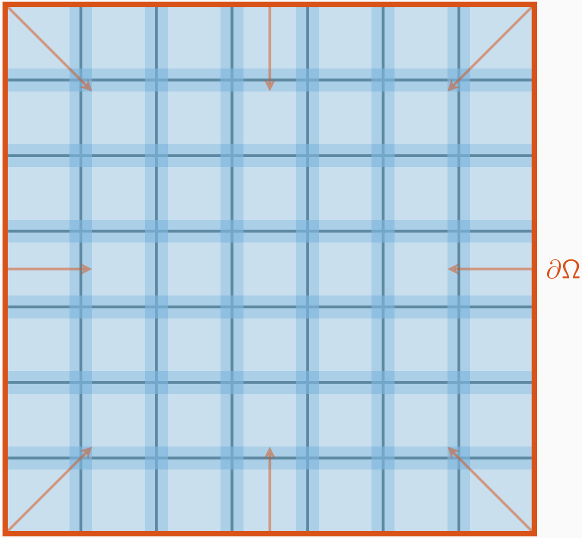




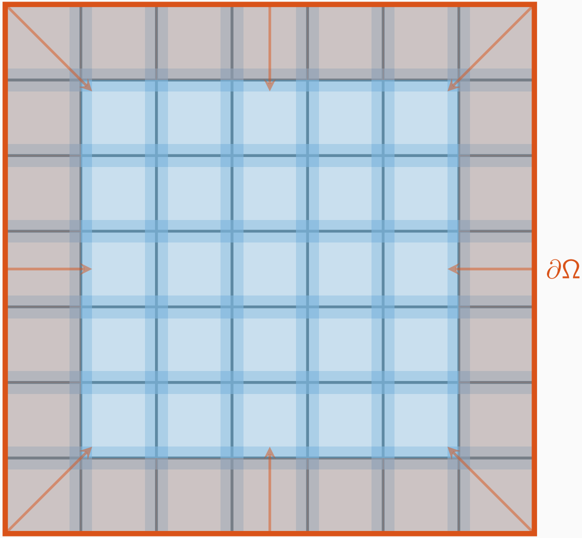
Transport of Information – One-Level Overlapping Schwarz Methods



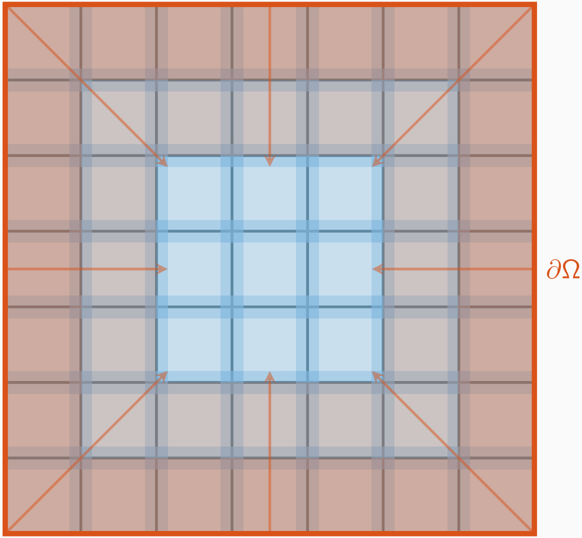
Transport of Information – One-Level Overlapping Schwarz Methods



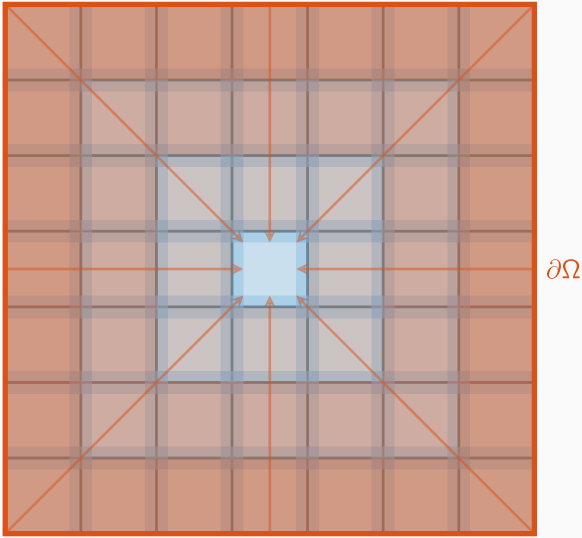
Transport of Information – One-Level Overlapping Schwarz Methods



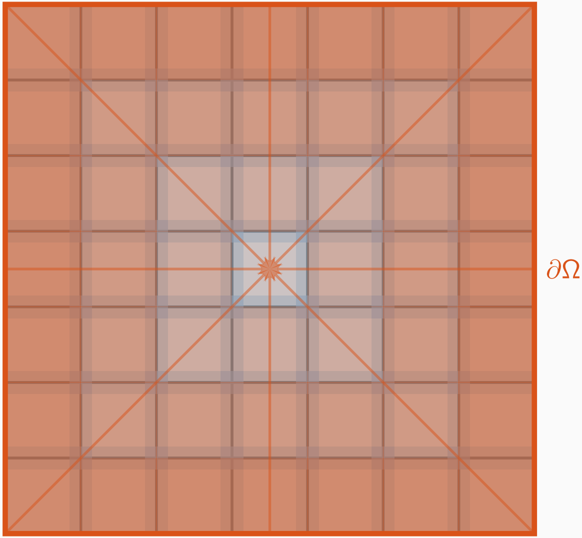
Transport of Information – One-Level Overlapping Schwarz Methods



Transport of Information – One-Level Overlapping Schwarz Methods



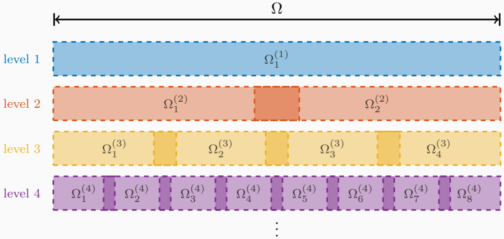
Transport of Information – One-Level Overlapping Schwarz Methods



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

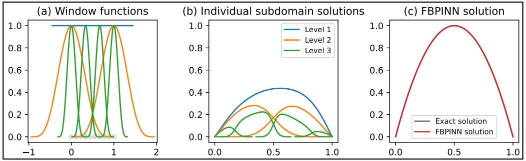
Multi-Level FBPINN Algorithm

Extension of FBPINNs to L levels; Cf. **Dolean, Heinlein, Mishra, Moseley (2024)**.



L-level network architecture

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



Multi-Frequency Problem

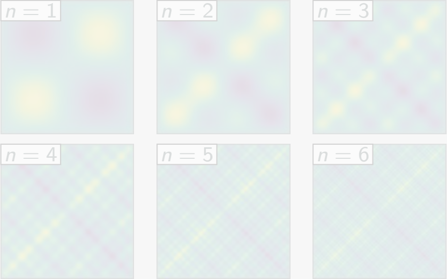
Let us now consider the **two-dimensional multi-frequency Laplace boundary value problem**

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

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

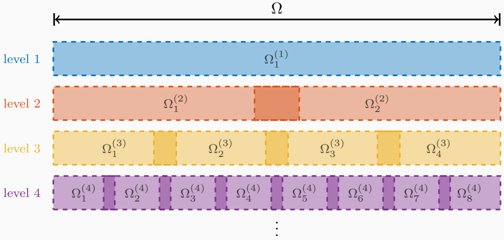
with $\omega_i = 2^i$.

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



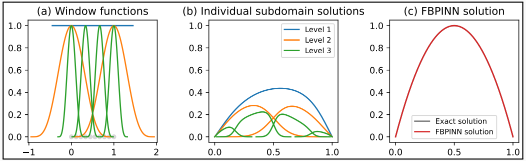
Multi-Level FBPINN Algorithm

Extension of FBPINNs to L levels; Cf. **Dolean, Heinlein, Mishra, Moseley (2024)**.



L-level network architecture

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



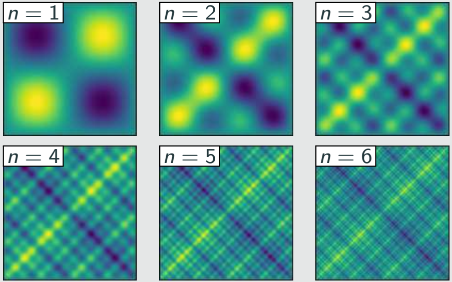
Multi-Frequency Problem

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

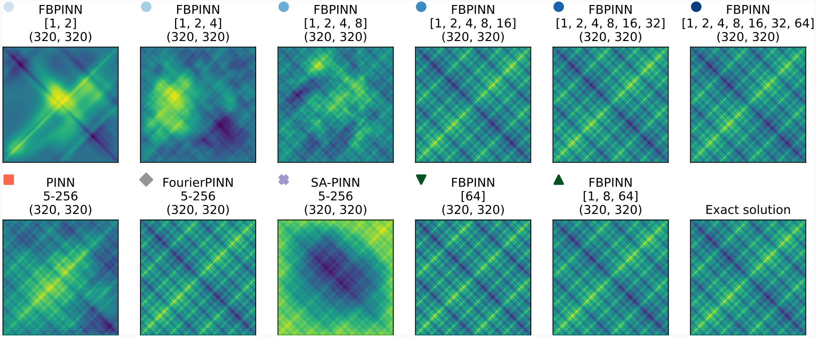
$$\begin{aligned} -\Delta u &= 2 \sum_{i=1}^{n} (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y) && \text{in } \Omega, \\ u &= 0 && \text{on } \partial\Omega, \end{aligned}$$

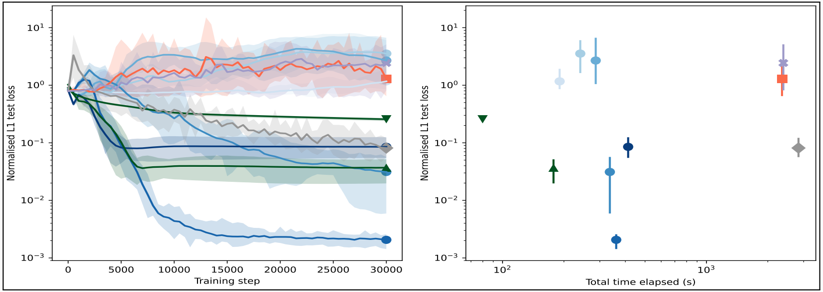
with $\omega_i = 2^i$.

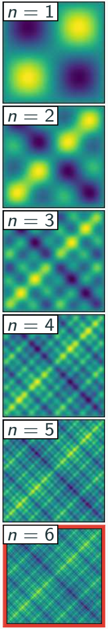
For increasing values of n, we obtain the **analytical solutions**:



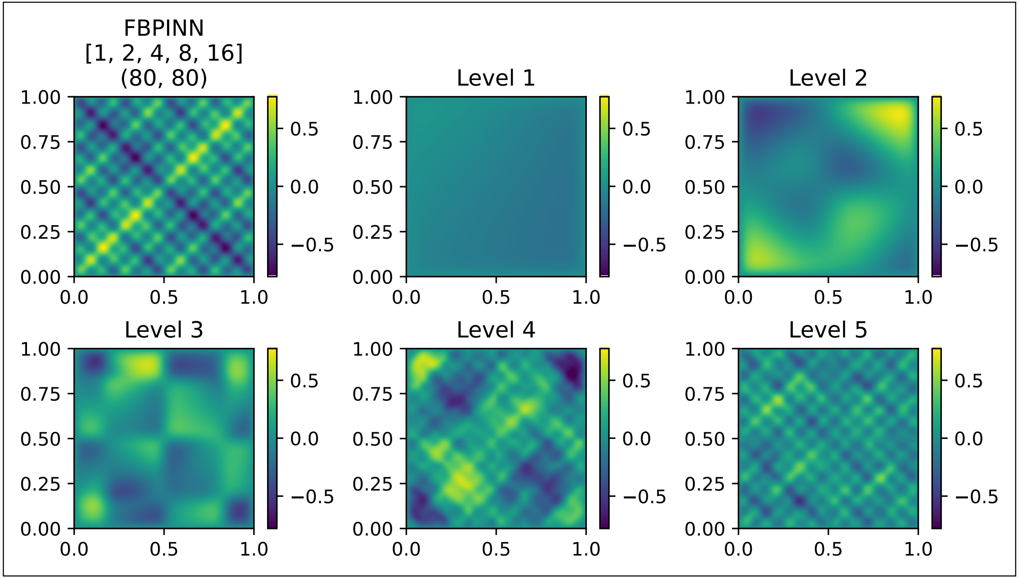
Multi-Level FBPINNs for a Multi-Frequency Problem – Strong Scaling





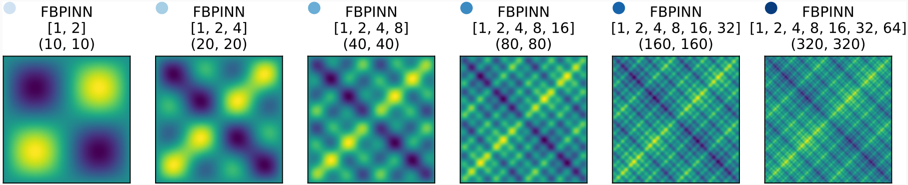


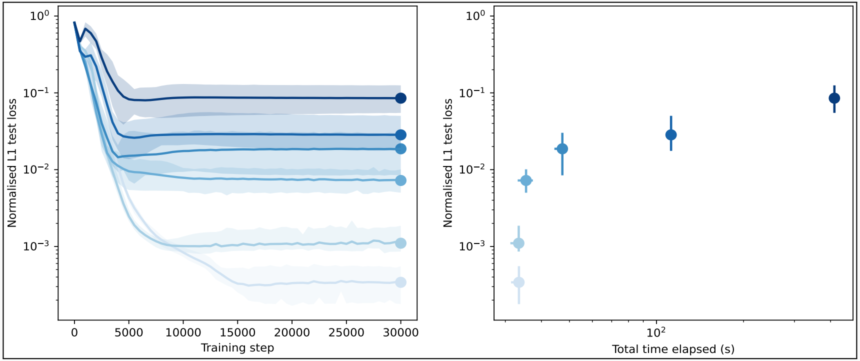
Multi-Frequency Problem – What the FBPINN Learns

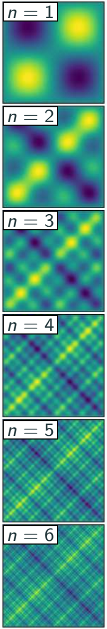


Cf. **Dolean, Heinlein, Mishra, Moseley (2024)**.

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







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

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

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

PINNs for Time-Dependent Problems

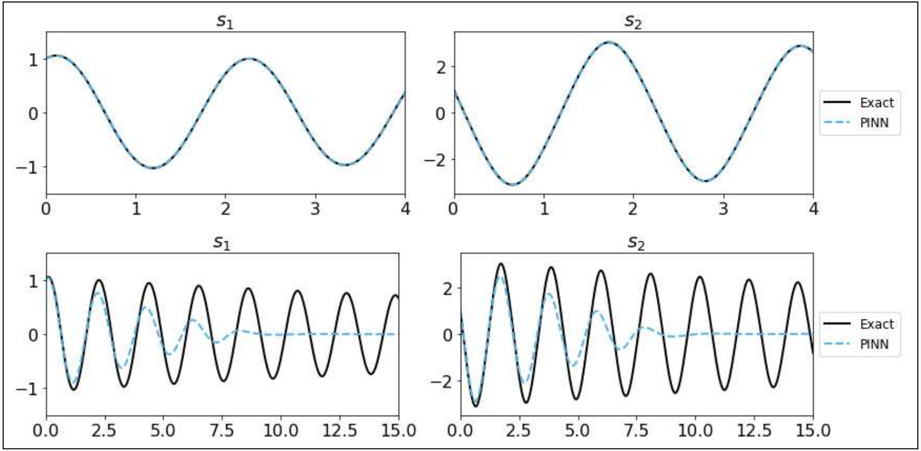
We investigate the performance of PINNs for **time-dependent problems**. Therefore, consider the simple **pedulum problem**:

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

Problem parameters

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

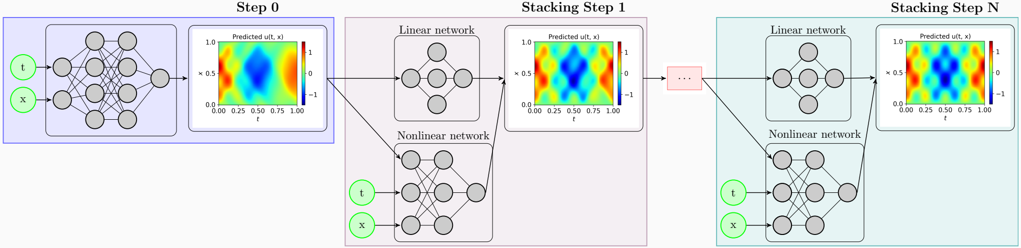
- **Top:** $T = 4$
- **Bottom:** $T = 20$



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}:

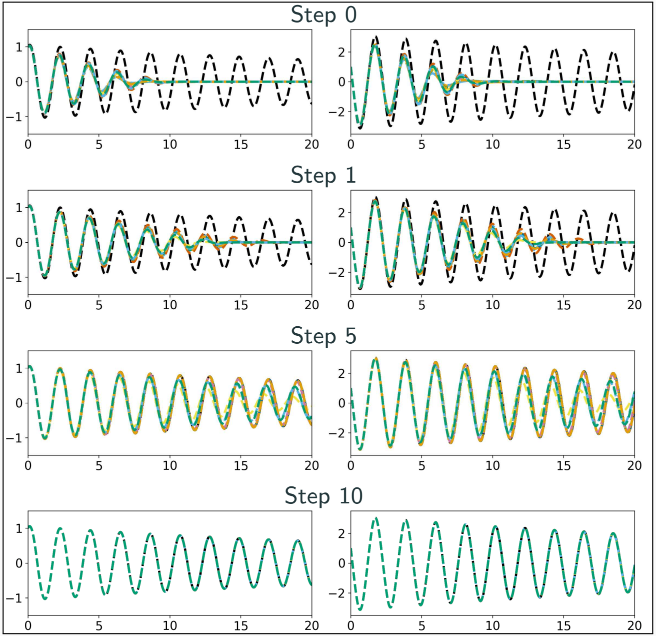
$$u^{MF}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}) = (1 - |\alpha|)\, u^{MF}_{\text{linear}}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}, u^{SF}) + |\alpha|\, u^{MF}_{\text{nonlinear}}(\boldsymbol{x}, \boldsymbol{\theta}^{MF}, u^{SF})$$



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)

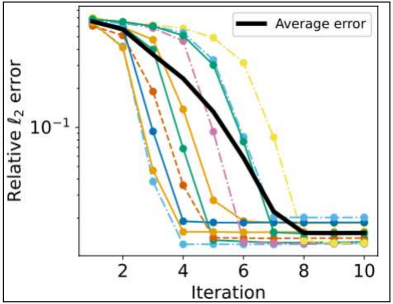
Stacking Multifidelity PINNs for the Pendulum Problem



Pendulum problem:

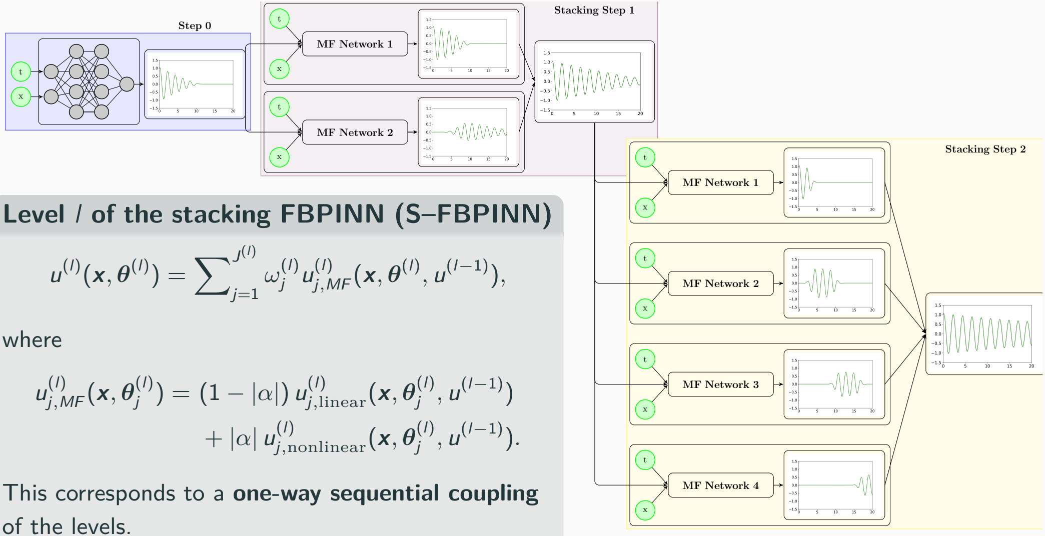
$$\frac{d\mathcal{s}_1}{dt} = \mathcal{s}_2,$$
$$\frac{d\mathcal{s}_2}{dt} = -\frac{b}{m}\mathcal{s}_2 - \frac{g}{L}\sin(\mathcal{s}_1).$$

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



Stacking Multifidelity 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

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

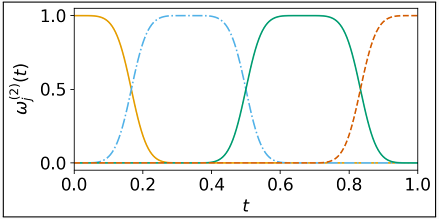
This corresponds to a **one-way sequential coupling** of the levels.

Numerical Results – Pendulum Problem

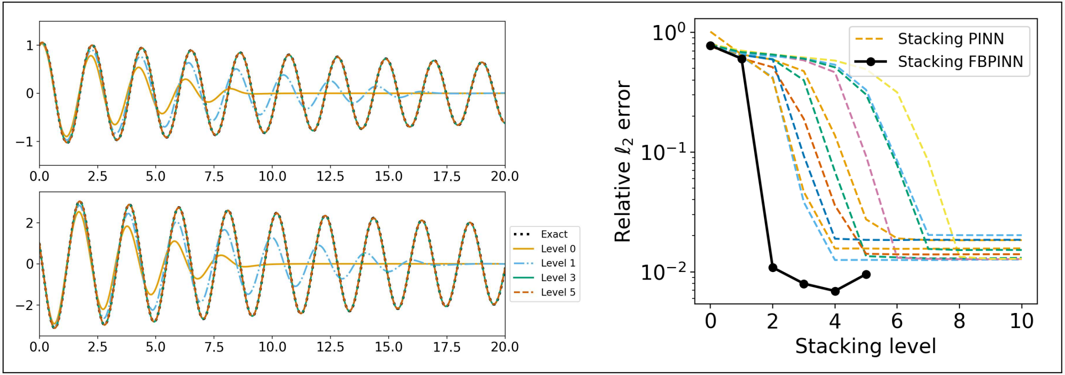
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

$$
\begin{aligned}
\frac{d\mathscr{s}_1}{dt} &= \mathscr{s}_2, \\
\frac{d\mathscr{s}_2}{dt} &= -\frac{b}{m}\mathscr{s}_2 - \frac{g}{L}\sin(\mathscr{s}_1)
\end{aligned}
$$

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



Exemplary partition of unity in time



Numerical Results – Pendulum Problem

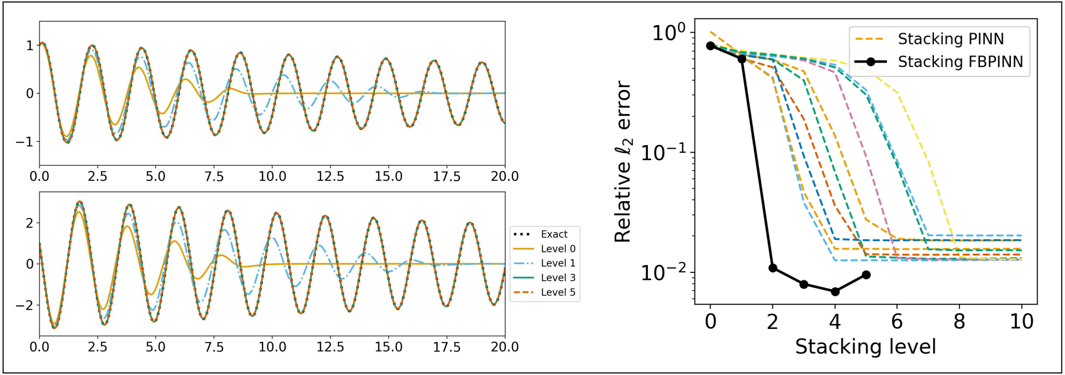
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

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

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



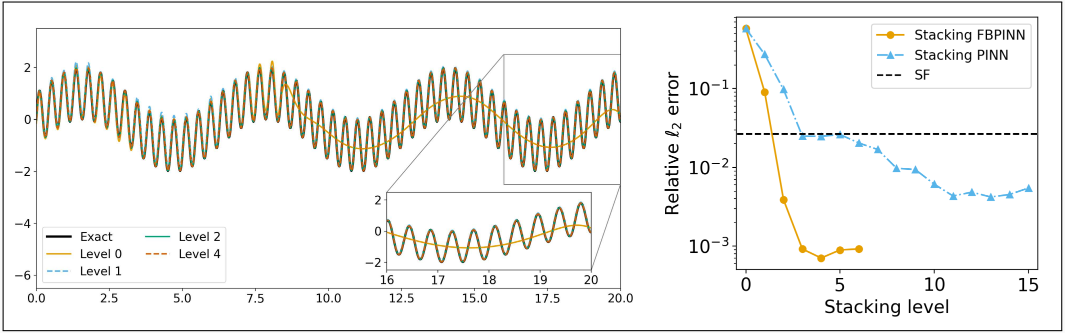
Numerical Results – Two-Frequency Problem

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

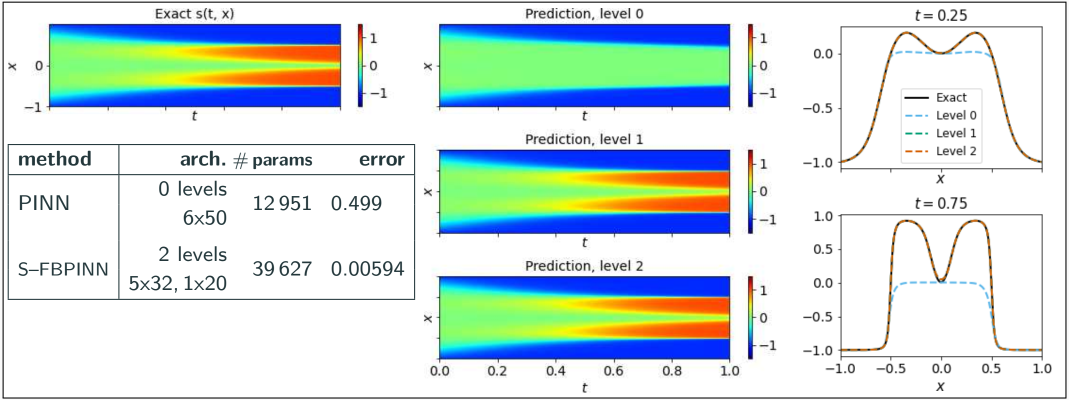


→ Due to the **multiscale structure of the problem**, the **improvements** due to the **multifidelity FBPINN approach** are **even stronger**.

Numerical Results – Allen–Cahn Equation

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

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

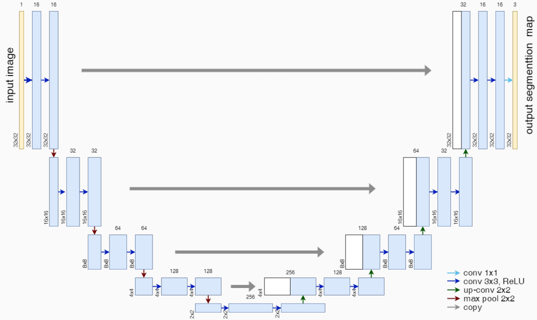


method	**arch.**	**# params**	**error**
PINN	0 levels 6x50	12 951	0.499
S–FBPINN	2 levels 5x32, 1x20	39 627	0.00594

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

Domain Decomposition for Convolutional Neural Networks

Memory Requirements for CNN Training

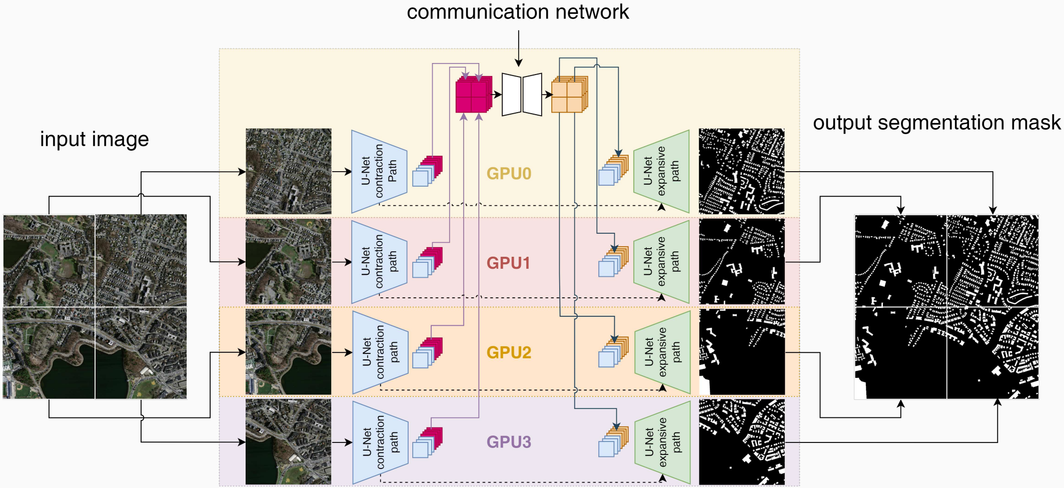


- As an example for a **convolutional neural network (CNN)**, we employ the **U-Net architecture** introduced in **Ronneberger, Fischer, and Brox (2015)**.
- The U-Net yields **state-of-the-art accuracy in semantic image segmentation** and other **image-to-image tasks**.

Below: *memory consumption for training on a single* 1024 × 1024 *image.*

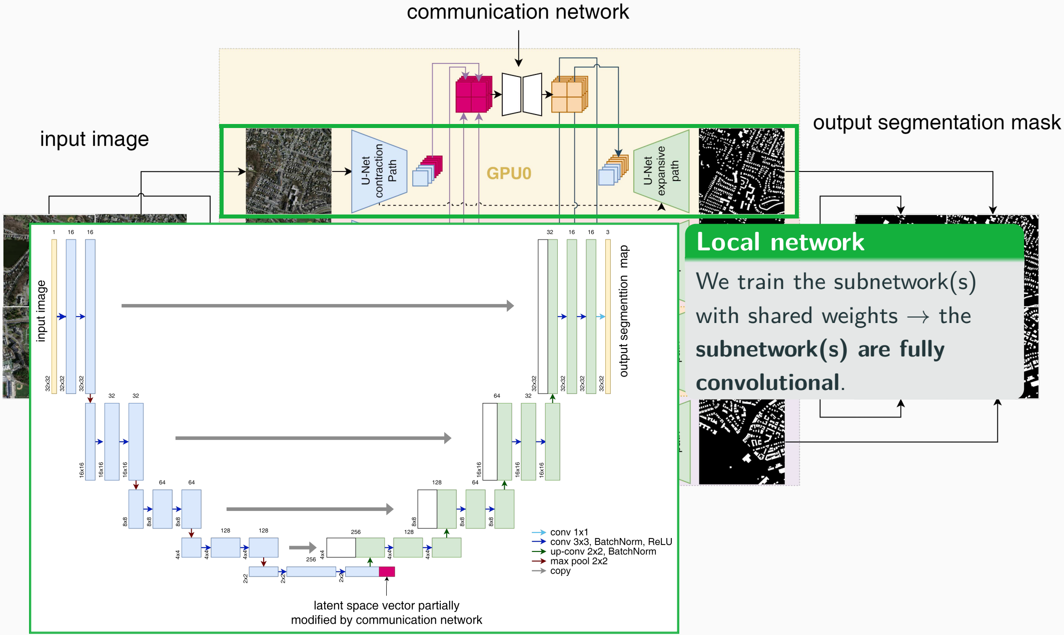
name	size	# channels		mem. feature maps		mem. weights	
		input	output	# of values	MB	# of values	MB
input block	1 024	3	64	268 M	**1 024.0**	38 848	**0.148**
encoder block 1	512	64	128	167 M	**704.0**	221 696	**0.846**
encoder block 2	256	128	256	84 M	**352.0**	885 760	**3.379**
encoder block 3	128	256	512	42 M	**176.0**	3 540 992	**13.508**
encoder block 4	64	512	1 024	21 M	**88.0**	14 159 872	**54.016**
decoder block 1	64	1,024	512	50 M	**192.0**	9 177 088	**35.008**
decoder block 2	128	512	256	101 M	**384.0**	2 294 784	**8.754**
decoder block 3	256	256	128	201 M	**768.0**	573 952	**2.189**
decoder block 4	512	128	64	402 M	**1 536.0**	143 616	**0.548**
output block	1 024	64	3	3.1 M	**12.0**	195	**0.001**

Decomposing the U-Net



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

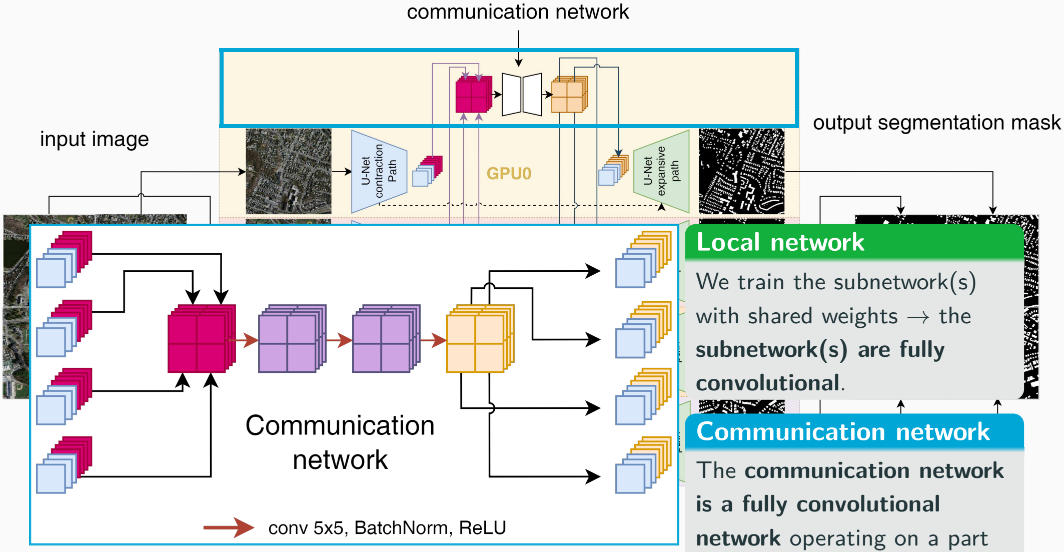
Decomposing the U-Net



Local network

We train the subnetwork(s) with shared weights → the **subnetwork(s) are fully convolutional**.

Decomposing the U-Net



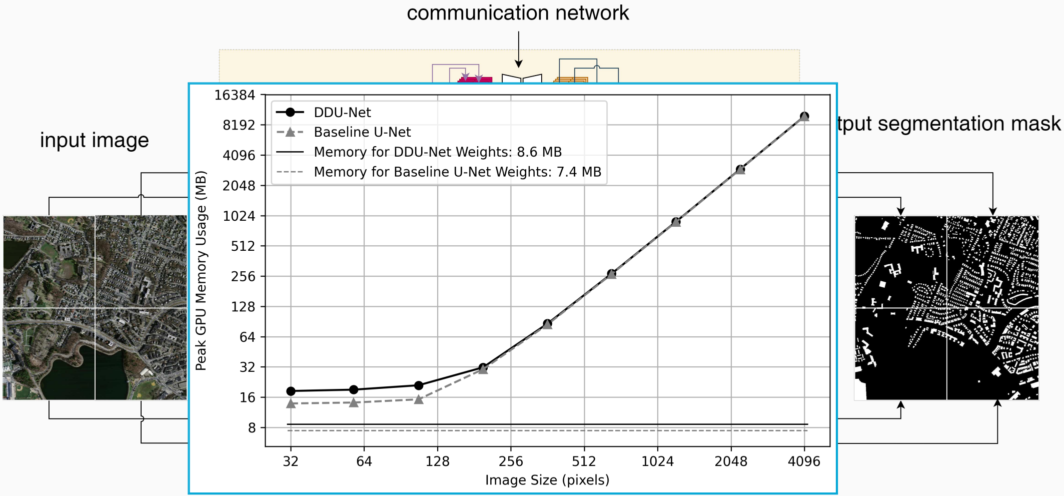
Local network

We train the subnetwork(s) with shared weights → the **subnetwork(s) are fully convolutional**.

Communication network

The **communication network is a fully convolutional network** operating on a part of the coarse feature maps.

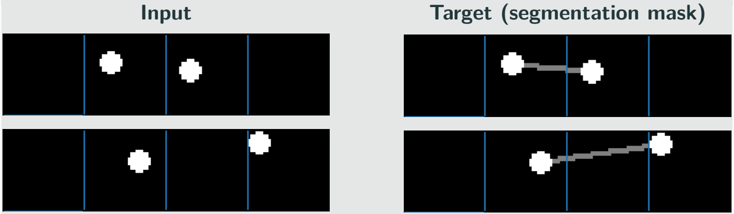
Decomposing the U-Net



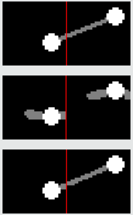
- Distribution of feature maps results in **significant reduction of memory usage on a single GPU**
- **Moderate additional memory usage** due to the **communication network**

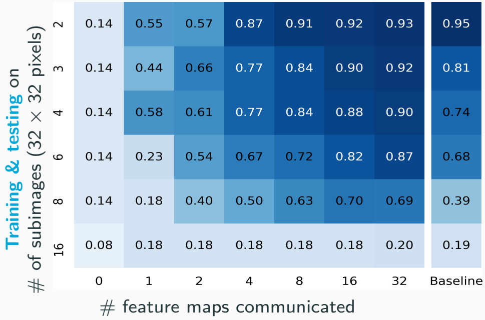
Results – Synthetic Data Set

Task: Connect two dots via a line segment

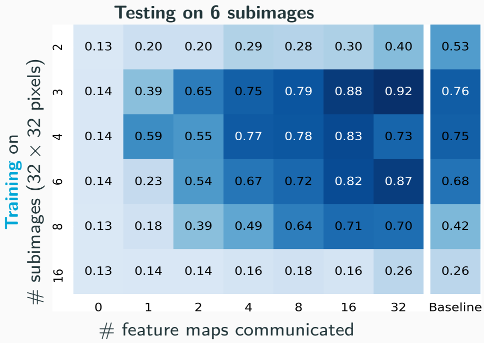


Result: Communication





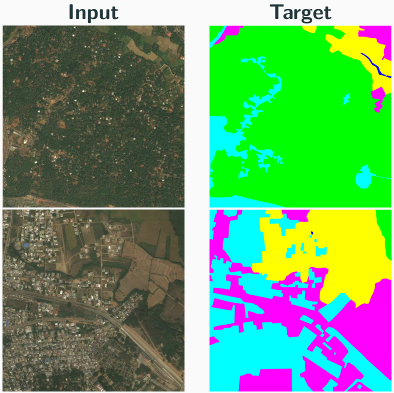
	0	1	2	4	8	16	32	Baseline
2	0.14	0.55	0.57	0.87	0.91	0.92	0.93	0.95
3	0.14	0.44	0.66	0.77	0.84	0.90	0.92	0.81
4	0.14	0.58	0.61	0.77	0.84	0.88	0.90	0.74
6	0.14	0.23	0.54	0.67	0.72	0.82	0.87	0.68
8	0.14	0.18	0.40	0.50	0.63	0.70	0.69	0.39
16	0.08	0.18	0.18	0.18	0.18	0.18	0.20	0.19


Testing on 6 subimages

	0	1	2	4	8	16	32	Baseline
2	0.13	0.20	0.20	0.29	0.28	0.30	0.40	0.53
3	0.14	0.39	0.65	0.75	0.79	0.88	0.92	0.76
4	0.14	0.59	0.55	0.77	0.78	0.83	0.73	0.75
6	0.14	0.23	0.54	0.67	0.72	0.82	0.87	0.68
8	0.13	0.18	0.39	0.49	0.64	0.71	0.70	0.42
16	0.13	0.14	0.14	0.16	0.18	0.16	0.26	0.26

DeepGlobe 2018 Satellite Image Data Set (Demir et al. (2018))

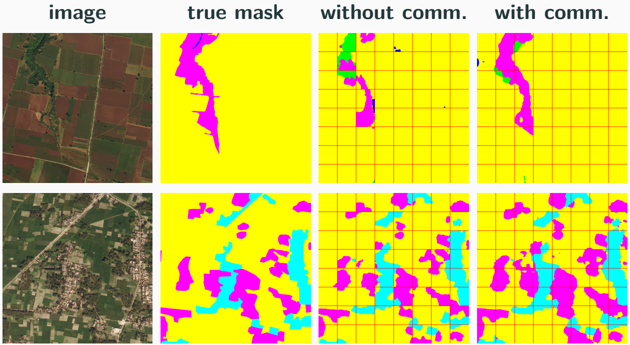
class	pixel count	proportion
urban	642.4M	9.35 %
agriculture	3898.0M	56.76 %
rangeland	701.1M	10.21 %
forest	944.4M	13.75 %
water	256.9M	3.74 %
barren	421.8M	6.14 %
unknown	3.0M	0.04 %



Avoiding overfitting

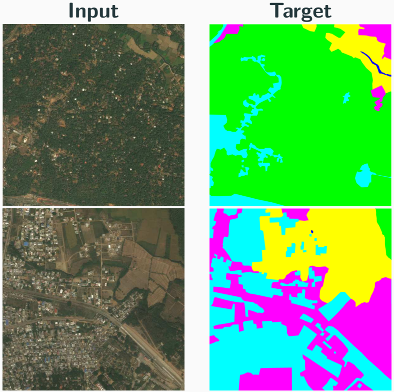
The data set includes **only 803 images**. To **avoid overfitting**, we

- apply **batch normalization**, use **random dropout** layers and **data augmentation**, and
- **initialize the encoder** using the **ResNet-18** (**He, Zhang, Ren, and Sun (2016)**)



DeepGlobe 2018 Satellite Image Data Set (Demir et al. (2018))

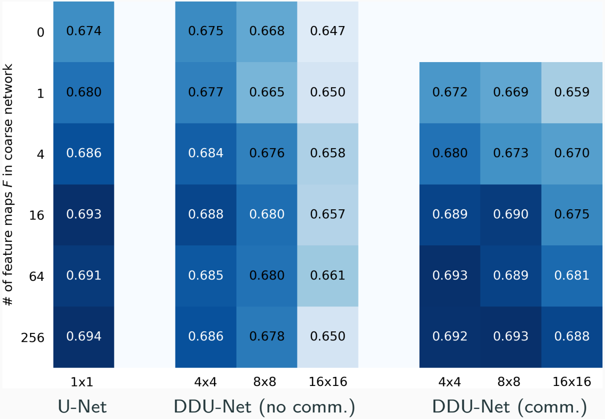
class	pixel count	proportion
urban	642.4M	9.35 %
agriculture	3898.0M	56.76 %
rangeland	701.1M	10.21 %
forest	944.4M	13.75 %
water	256.9M	3.74 %
barren	421.8M	6.14 %
unknown	3.0M	0.04 %



Avoiding overfitting

The data set includes **only 803 images**. To **avoid overfitting**, we

- apply **batch normalization**, use **random dropout** layers and **data augmentation**, and
- **initialize the encoder** using the **ResNet-18** (He, Zhang, Ren, and Sun (2016))



# of feature maps F in coarse network	U-Net 1x1	DDU-Net (no comm.) 4x4	DDU-Net (no comm.) 8x8	DDU-Net (no comm.) 16x16	DDU-Net (comm.) 4x4	DDU-Net (comm.) 8x8	DDU-Net (comm.) 16x16
0	0.674	0.675	0.668	0.647			
1	0.680	0.677	0.665	0.650	0.672	0.669	0.659
4	0.686	0.684	0.676	0.658	0.680	0.673	0.670
16	0.693	0.688	0.680	0.657	0.689	0.690	0.675
64	0.691	0.685	0.680	0.661	0.693	0.689	0.681
256	0.694	0.686	0.678	0.650	0.692	0.693	0.688

FBPINNs – Domain Decomposition for Physics-Informed Neural Networks

- **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.

Stacking Multifidelity FBPINNs for Time-Dependent Problems

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

DDU-Net – Domain Decomposition for CNNs

- The **memory requirements for training of high-resolution images** using CNNs can be large, In particular, the U-Net model requires **storing intermediate feature maps**.
- Our **novel DDU-Net** approach **decouples the training on the sub-images**, allowing us to distribute the memory load among **multiple GPUs**. It limits communication to **deepest level** of the U-Net architecture using a **communication network**.

Thank you for your attention!