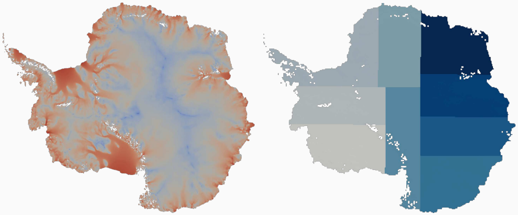
# Domain decomposition for neural networks

Alexander Heinlein[1]

Workshop on Scientific Machine Learning, Strasbourg University, France, July 8-12, 2024

[1]Delft University of Technology

# Domain Decomposition Methods



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

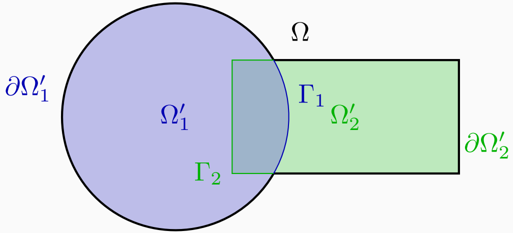
**Historical remarks:** The **alternating Schwarz method** is the earliest **domain decomposition method (DDM)**, which has been invented by **H. A. Schwarz** and published in **1870**:

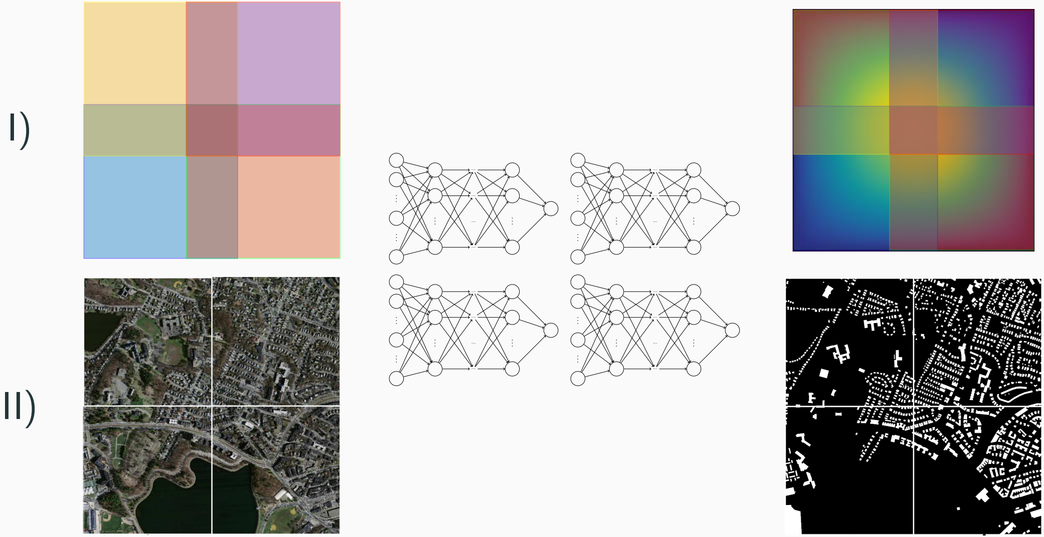
- Schwarz used the algorithm to establish the **existence of harmonic functions** with prescribed boundary values on **regions with non-smooth boundaries**.

## Idea

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

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




I)
II)

# 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, 2024); Heinlein, Howard, Beecroft, Stinis (acc. 2024 / arXiv:2401.07888)**
- **FBKANs**: **Howard, Jacob, Murphy, Heinlein, Stinis (arXiv:2406.19662)**

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

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

**Combining machine learning and domain decomposition methods for the solution of partial differential equations — A review**

GAMM-Mitteilungen. 2021.

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

A. Klawonn, M. Lanser, J. Weber

**Machine learning and domain decomposition methods – a survey**

arXiv:2312.14050. 2023

# Outline

1. 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)

2. 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)

3. Domain Decomposition for Convolutional Neural Networks

   Based on joint work with

   **Eric Cyr** (Sandia National Laboratories)

   **Corné Verburg** (Delft University of Technology)

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

# 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

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

by solving an **optimization problem**

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

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

## Construction of the trial functions

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

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

- 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**

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

## Approach

Solve a general differential equation subject to boundary conditions

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

by solving an **optimization problem**

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

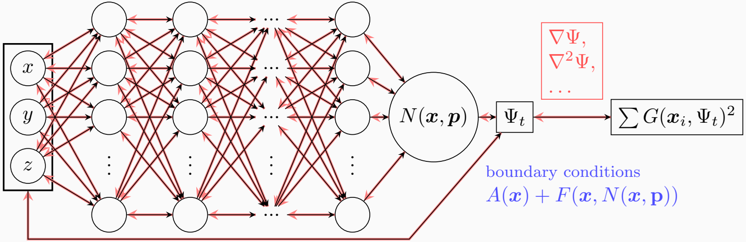
where $\Psi_t\left(\boldsymbol{x}, \boldsymbol{\theta}\right)$ is a **trial function**, $\boldsymbol{x}_i$ **sampling points inside the domain** $\Omega$ and $\boldsymbol{\theta}$ are **adjustable parameters**.

## 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**



Solve the **boundary value problem**

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

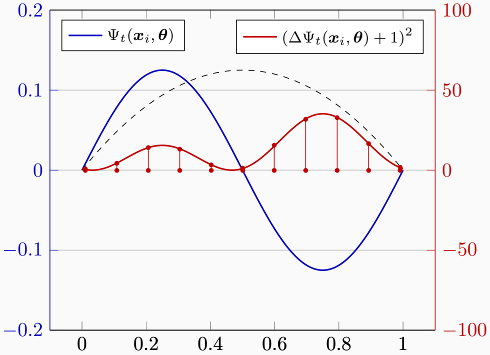
via a **collocation approach**:

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

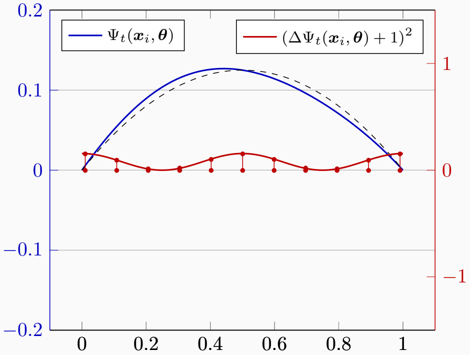
### Boundary conditions

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

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




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




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

# Physics-Informed Neural Networks (PINNs)

In the **physics-informed neural network (PINN)** approach introduced by **Raissi et al. (2019)**, a **neural network** is employed to **discretize a partial differential equation**

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

PINNs use a **hybrid loss function**:

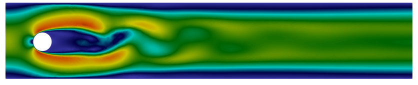
$$\mathcal{L}(\boldsymbol{\theta}) = \omega_{\text{data}} \mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}),$$

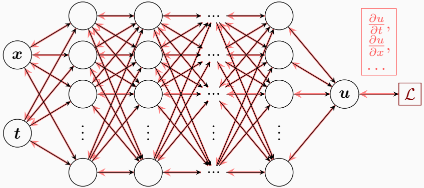
where $\omega_{\text{data}}$ and $\omega_{\text{PDE}}$ are **weights** and

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

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

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





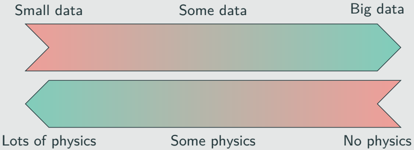

## Advantages

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

## Drawbacks

- **Training cost** and **robustness**
- **Convergence not well-understood**
- **Difficulties with scalability** and **multi-scale problems**

## Hybrid loss




- **Known solution values** can be included in $\mathcal{L}_{\text{data}}$
- **Initial and boundary conditions** are also included in $\mathcal{L}_{\text{data}}$

# Available Theoretical Results for PINNs – An Example

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

**Estimate of the generalization error**

The generalization error (or total error) satisfies

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

where

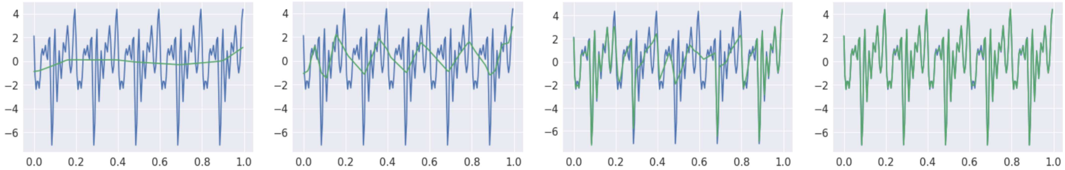
- $\mathcal{E}_G = \mathcal{E}_G(\boldsymbol{X}, \boldsymbol{\theta}) := \|\mathbf{u} - \mathbf{u}^*\|_V$ **general. error** ($V$ Sobolev space, $\boldsymbol{X}$ training data set)
- $\mathcal{E}_T$ **training error** ($l^p$ **loss** of the residual of the PDE)
- $N$ **number of the training points** and $\alpha$ **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"**

### 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)**

Solve

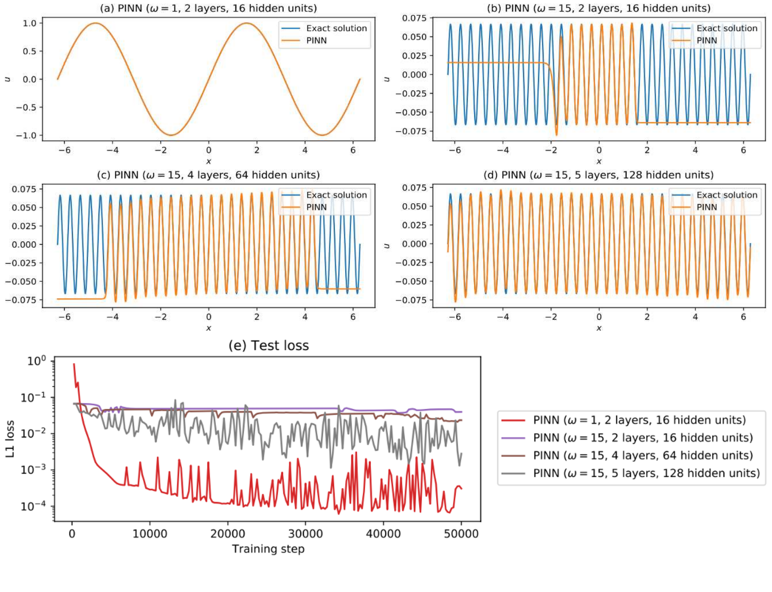
$$u' = \cos(\omega \boldsymbol{x}),$$
$$u(0) = 0,$$

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)**




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

Solve

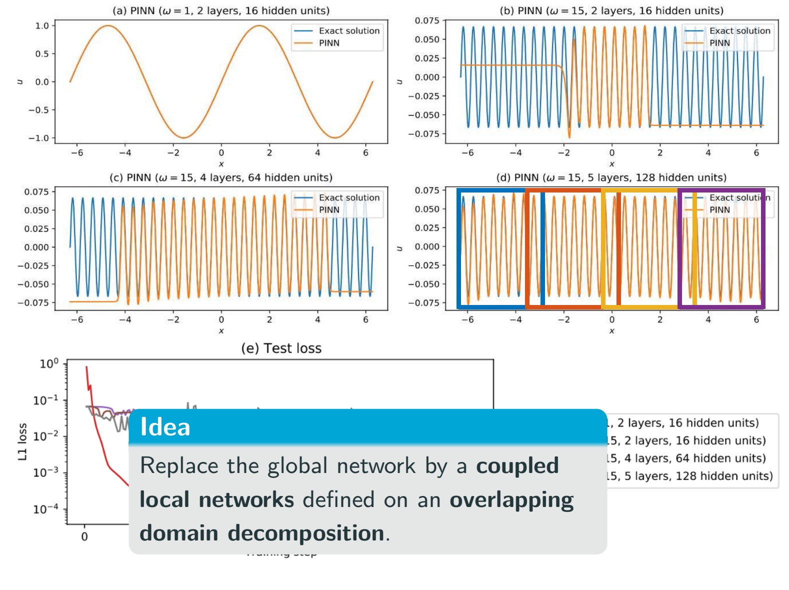
$$\begin{aligned} u' &= \cos(\omega \boldsymbol{x}), \\ u(0) &= 0, \end{aligned}$$

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)**




## 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

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**

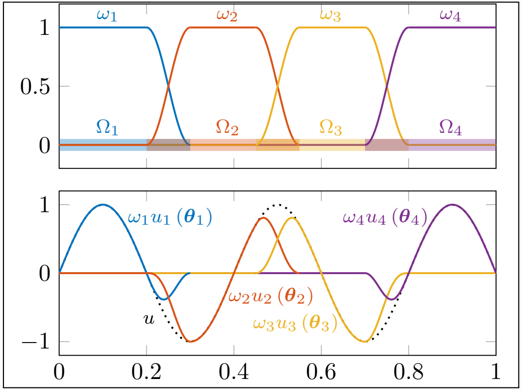
$$u(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \mathcal{C} \sum\nolimits_{j=1}^{J} \omega_j u_j \left(\boldsymbol{\theta}_j\right)$$

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** $\omega_j$ with $\mathrm{supp}(\omega_j) \subset \Omega_j$ and $\sum_{j=1}^{J} \omega_j \equiv 1$ on $\Omega$



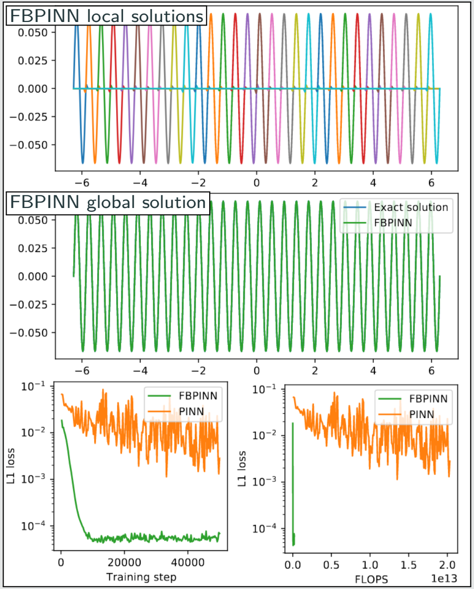

**Hard enf. of boundary conditions**

Loss function

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

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

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




## Scalability of FBPINNs

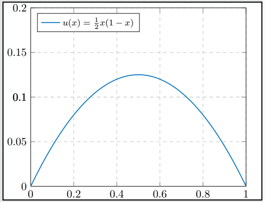
Consider the **simple boundary value problem**

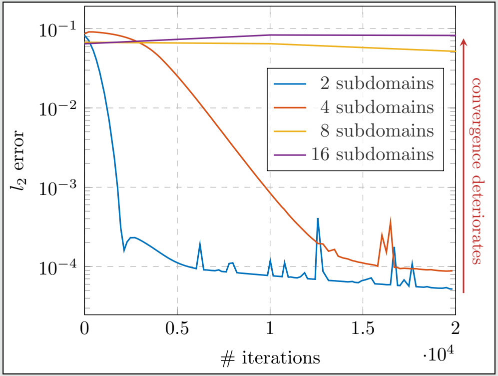
$$-u'' = 1 \text{ in } [0, 1],$$

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

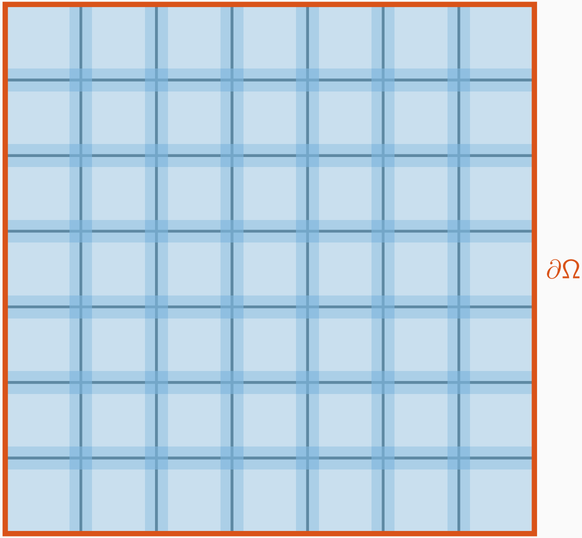
which has the **solution**

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

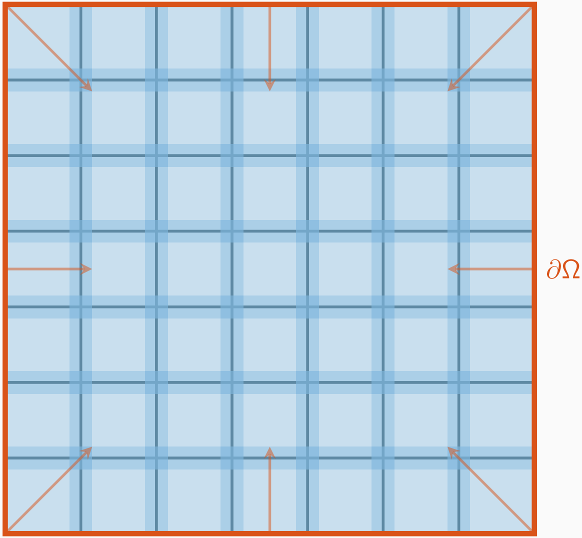


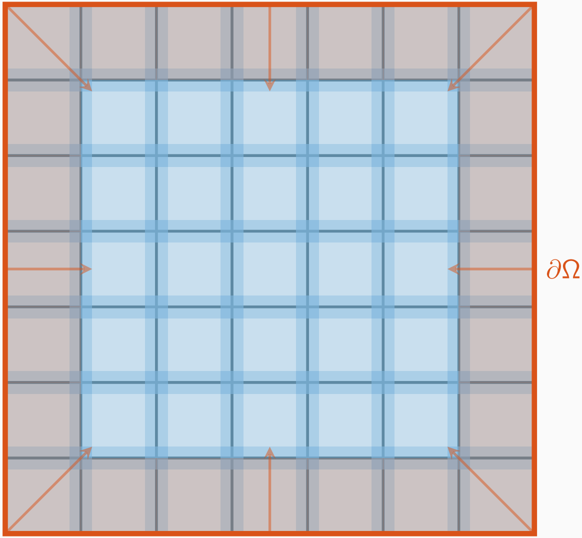
# Transport of Information – One-Level Overlapping Schwarz Methods



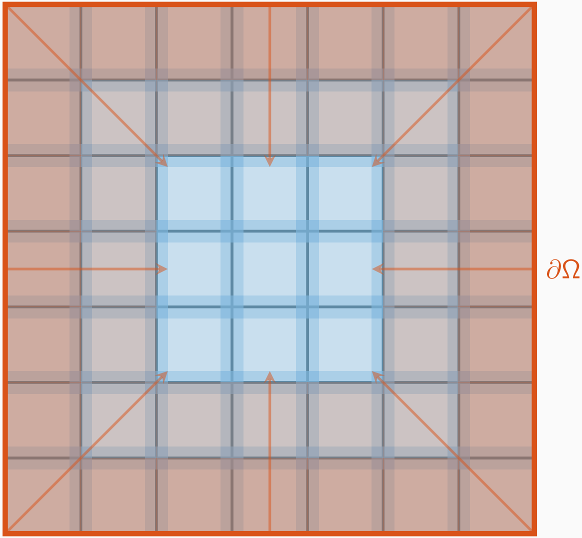
# Transport of Information – One-Level Overlapping Schwarz Methods

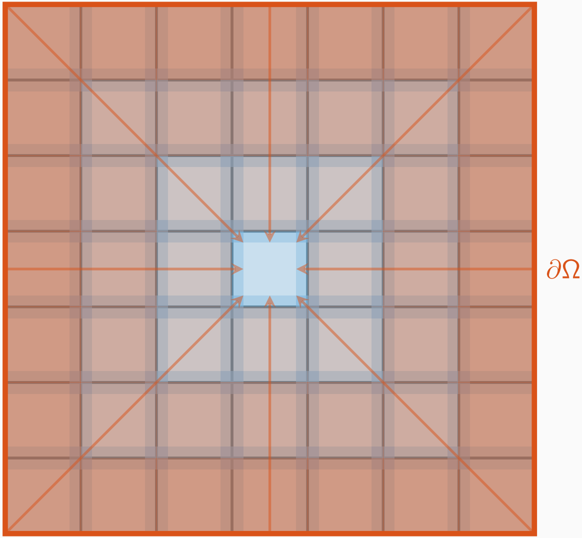


# Transport of Information – One-Level Overlapping Schwarz Methods

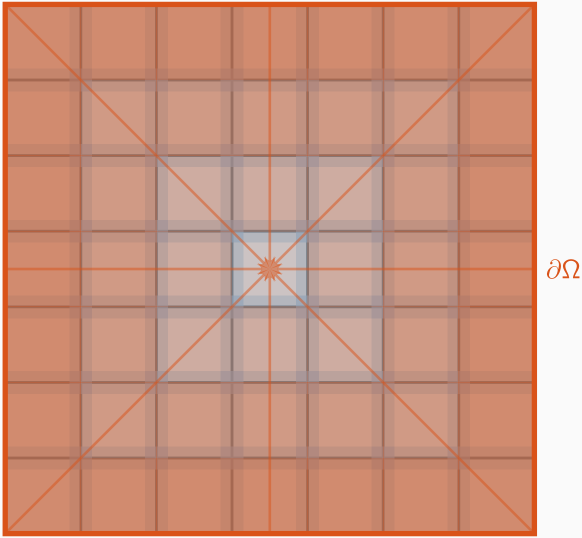


# Transport of Information – One-Level Overlapping Schwarz Methods




$\partial\Omega$

# 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

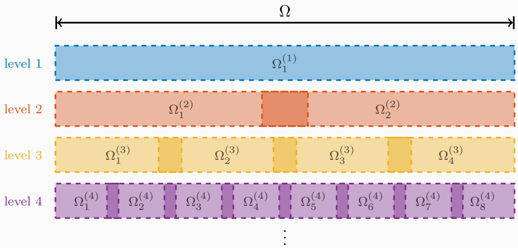
We introduce a **hierarchy of $L$ overlapping domain decompositions**

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

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

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

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

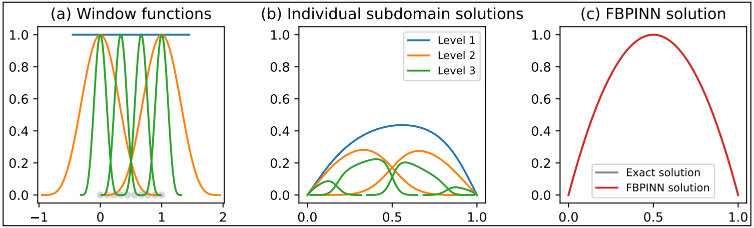



## $L$-level network architecture

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

## Loss function

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

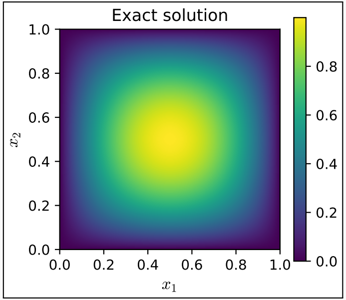


# Multilevel FBPINNs – 2D Laplace

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

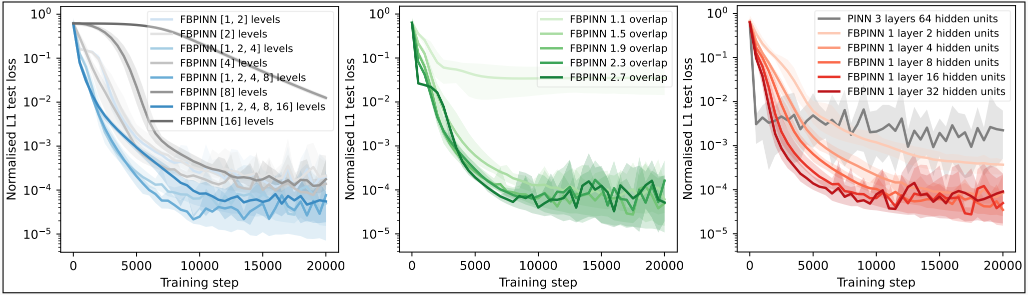
$$-\Delta u = 32(x(1-x) + y(1-y)) \quad \text{in } \Omega = [0,1]^2,$$
$$u = 0 \quad \text{on } \partial\Omega,$$

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




**Baseline model:**

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




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

Implementation using JAX

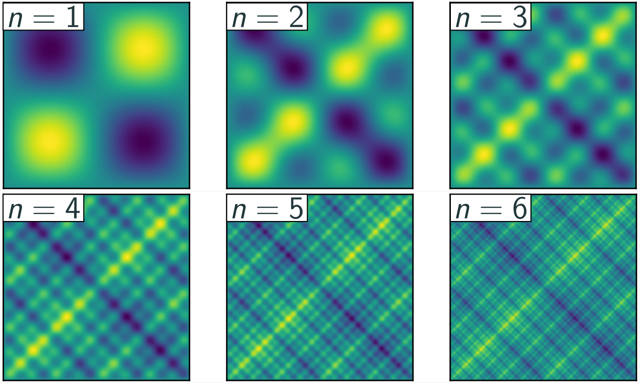
# Multi-Frequency Problem

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

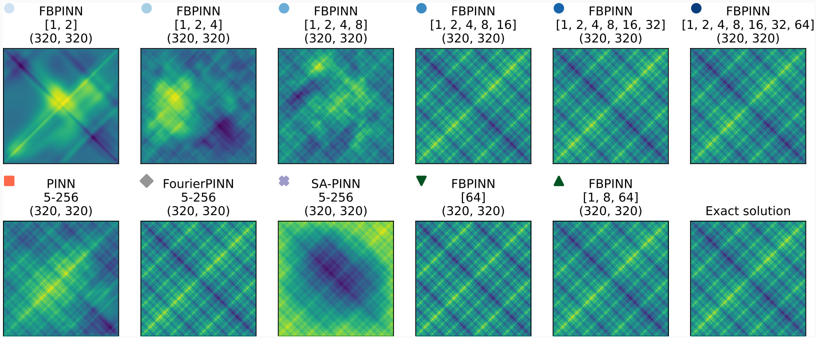
$$
\begin{aligned}
-\Delta u &= 2 \sum_{i=1}^{n} (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y) && \text{in } \Omega = [0,1]^2, \\
u &= 0 && \text{on } \partial\Omega,
\end{aligned}
$$

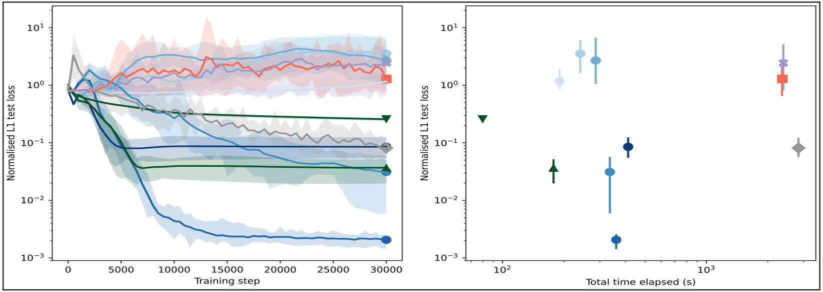
with $\omega_i = 2^i$.

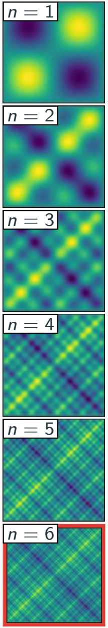
For increasing values of $n$, we obtain the **analytical solutions**:



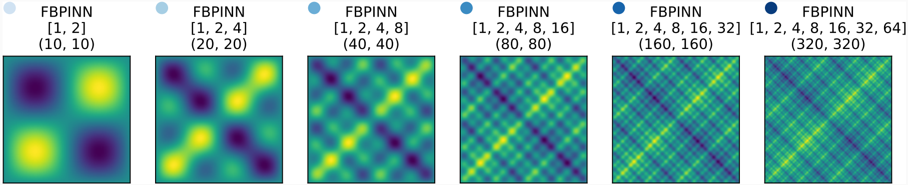
# Multi-Level FBPINNs for a Multi-Frequency Problem – Strong Scaling

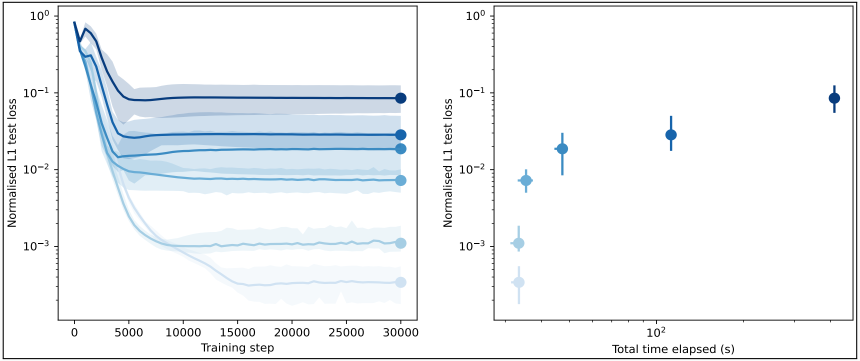


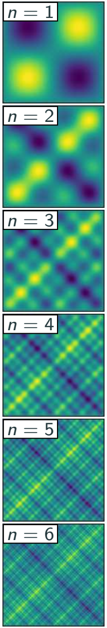



# Multi-Level FBPINNs for a Multi-Frequency Problem – Weak Scaling




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

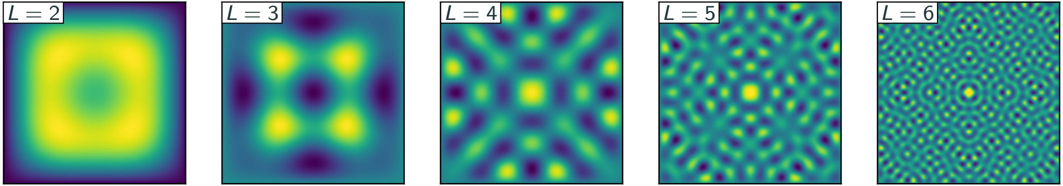
Cf. **Dolean, Heinlein, Mishra, Moseley (2024)**.

# Helmholtz Problem

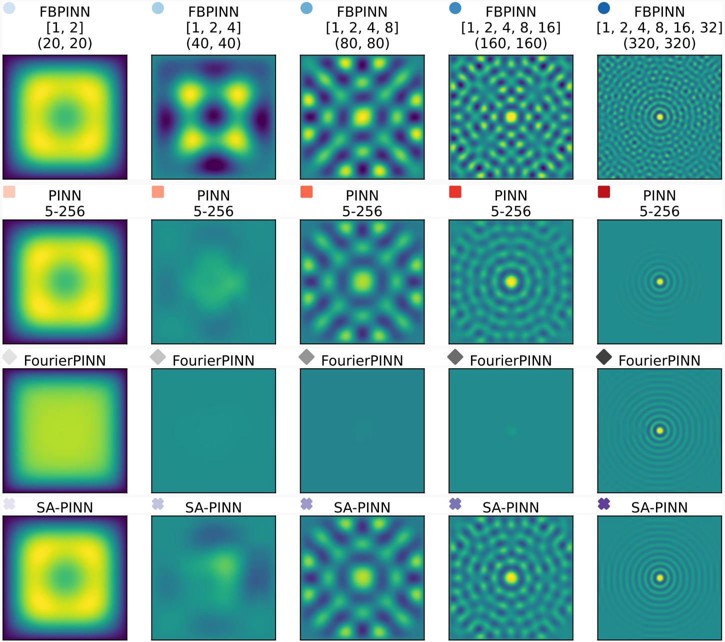
Finally, let us consider the **two-dimensional Helmholtz boundary value problem**

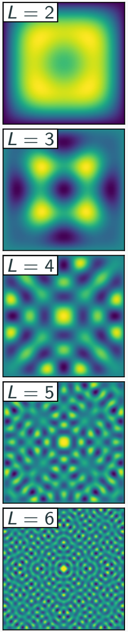
$$\Delta u - k^2 u = f \quad \text{in } \Omega = [0,1]^2,$$
$$u = 0 \quad \text{on } \partial\Omega,$$
$$f(\mathbf{x}) = e^{-\frac{1}{2}(\|\mathbf{x}-0.5\|/\sigma)^2}.$$

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

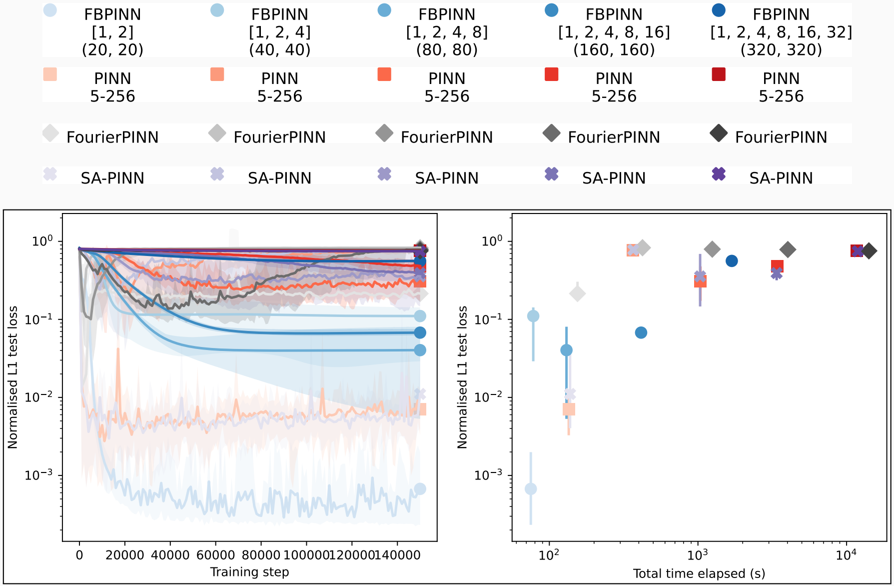


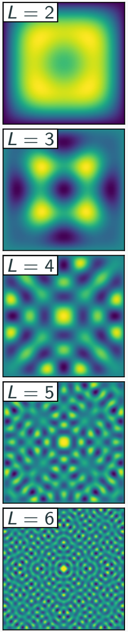
# Multi-Level FBPINNs for the Helmholtz Problem – Weak Scaling



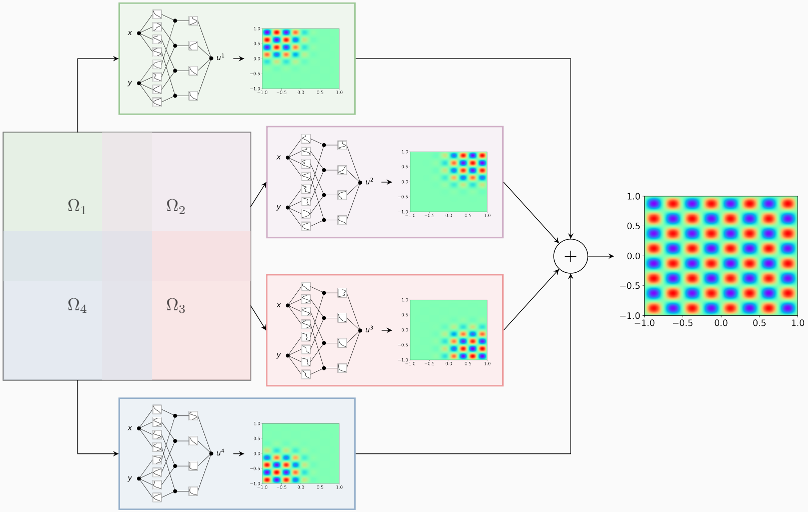
# Multi-Level FBPINNs for the Helmholtz Problem – Weak Scaling




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

# Extension to Kolmogorov–Arnold Networks




→ **Accurate results** for **noisy data** due to use of trainable **spline-based activation functions**.

Cf. **Howard, Jacob, Murphy, Heinlein, Stinis (arXiv:2406.19662)**.

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

# PINNs for Time-Dependent Problems

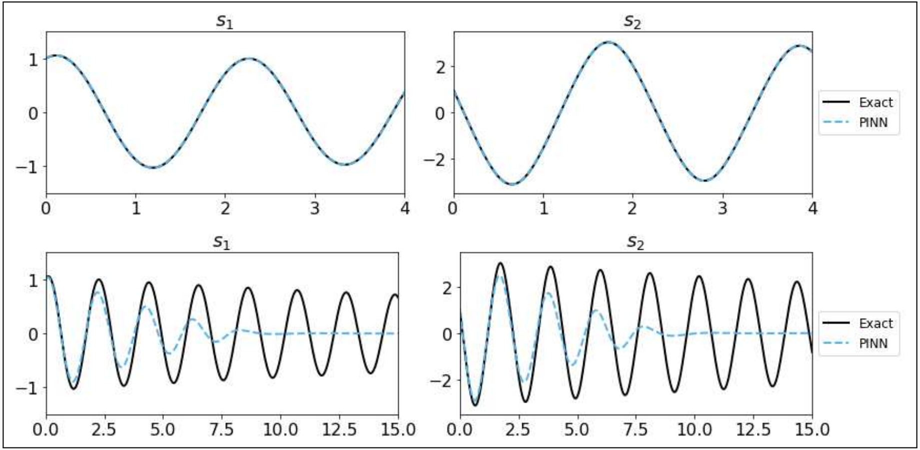
We investigate the performance of PINNs for **time-dependent problems**. Therefore, consider the simple **pedulum problem**:

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

**Problem parameters**

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

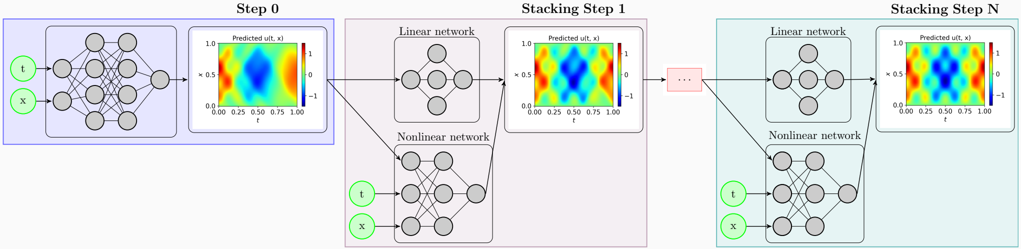
- **Top:** $T = 4$
- **Bottom:** $T = 20$



# Stacking Multifidelity PINNs

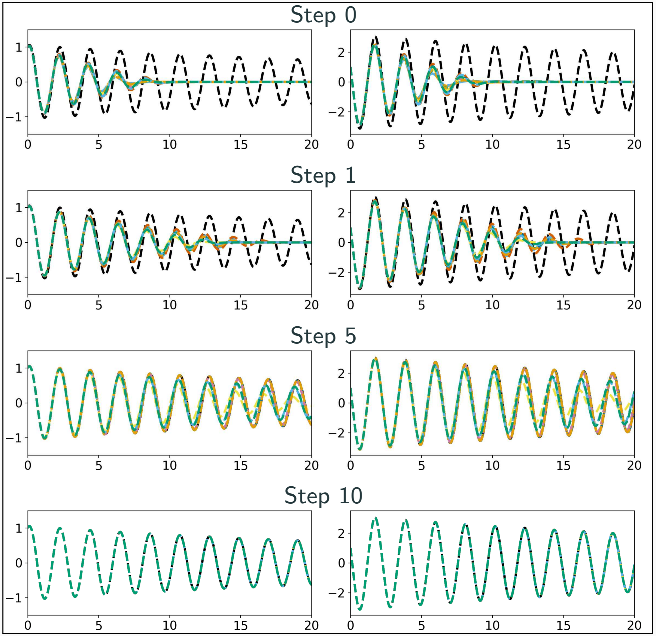
In the **stacking multifidelity PINNs approach** introduced in **Howard, Murphy, Ahmed, Stinis (arXiv 2023)**, **multiple PINNs are trained in a recursive way.** In each step, a model $u^{MF}$ is trained based on the previous model $u^{SF}$:

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




## Related works (non-exhaustive list)

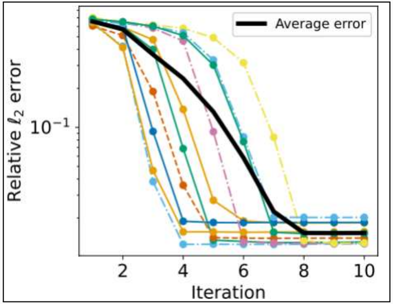
- **Cokriging & multifidelity Gaussian process regression:** E.g., **Wackernagel (1995); Perdikaris et al. (2017); Babaee et al. (2020)**
- **Multifidelity PINNs & DeepONet:** **Meng and Karniadakis (2020); Howard, Fu, and Stinis (arXiv 2023); Howard, Perego, Karniadakis, Stinis (2023); Murphy, Ahmed,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



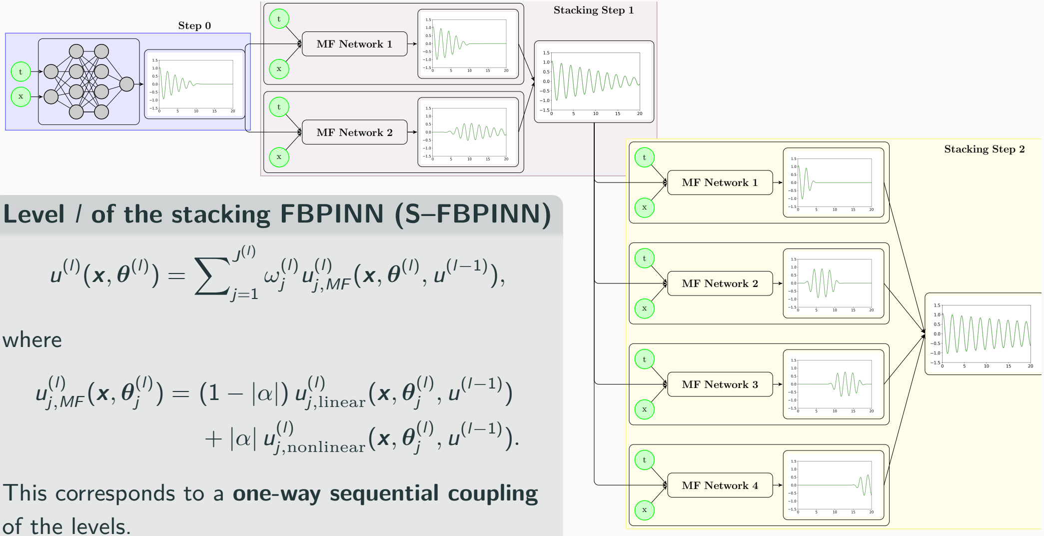

**Pendulum problem:**

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

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



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

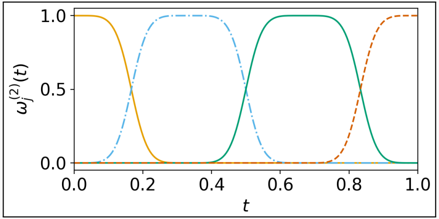
$$u_{j,MF}^{(l)}(\boldsymbol{x}, \boldsymbol{\theta}_j^{(l)}) = (1 - |\alpha|)\, u_{j,\mathrm{linear}}^{(l)}(\boldsymbol{x}, \boldsymbol{\theta}_j^{(l)}, u^{(l-1)}) + |\alpha|\, u_{j,\mathrm{nonlinear}}^{(l)}(\boldsymbol{x}, \boldsymbol{\theta}_j^{(l)}, u^{(l-1)}).$$

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

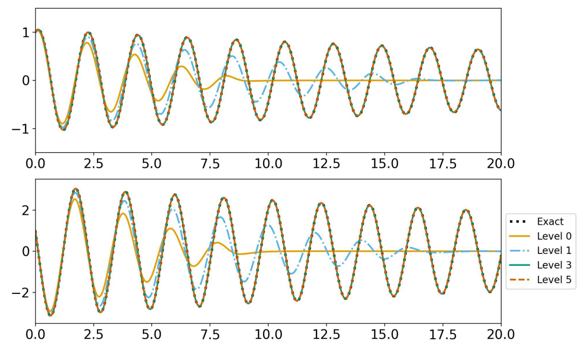
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

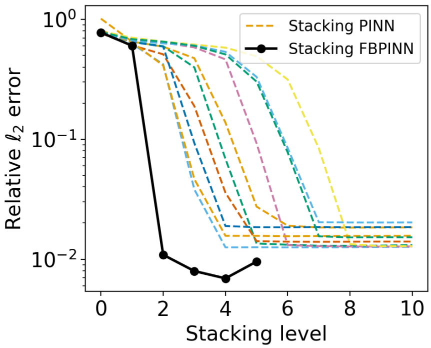
$$\frac{ds_1}{dt} = s_2,$$
$$\frac{ds_2}{dt} = -\frac{b}{m} s_2 - \frac{g}{L}\sin(s_1)$$

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




Exemplary partition of unity in time



# Numerical Results – Pendulum Problem

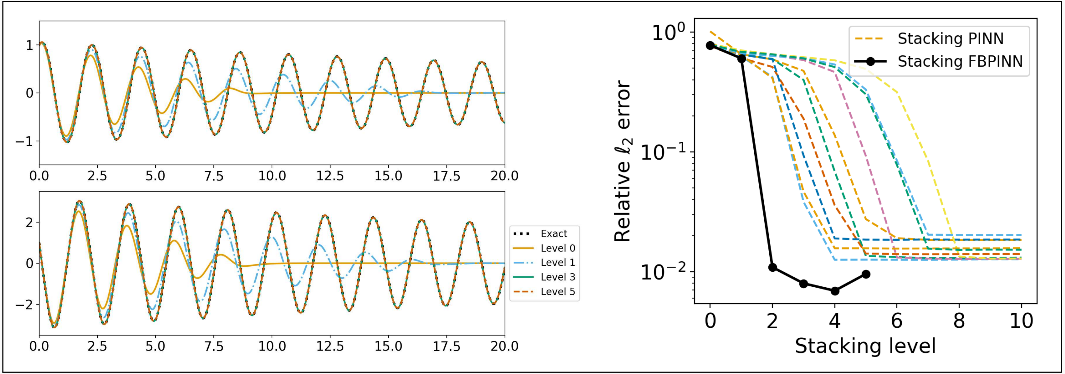
First, we consider a **pedulum problem** and **compare the stacking multifidelity PINN and FBPINN** approaches:

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

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

**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 |



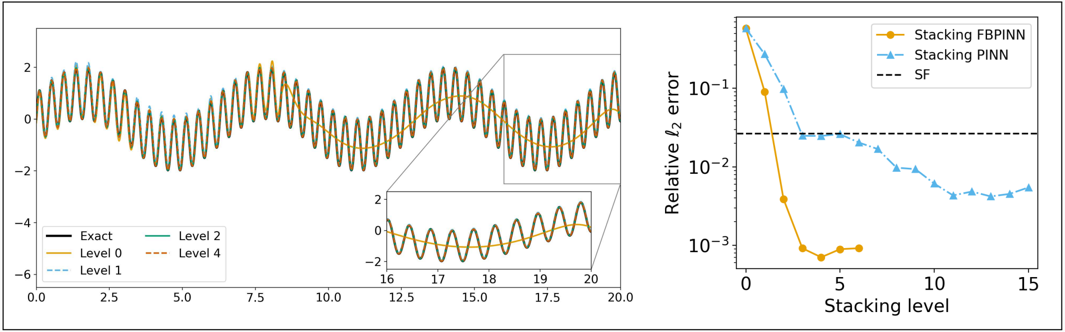
# Numerical Results – Two-Frequency Problem

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

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

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

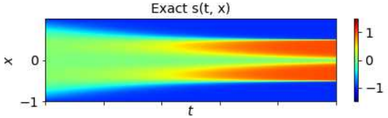
| method | arch. | # levels | # params | error |
|---|---|---|---|---|
| PINN | 4x64 | 0 | 12 673 | 0.6543 |
| PINN | 5x64 | 0 | 16 833 | 0.0265 |
| S–PINN | 4x16, 1x5 | 3 | 4900 | 0.0249 |
| S–PINN | 4x16, 1x5 | 10 | 11 179 | 0.0061 |
| S–FBPINN | 4x16, 1x5 | 2 | 7822 | 0.00415 |
| S–FBPINN | 4x16,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



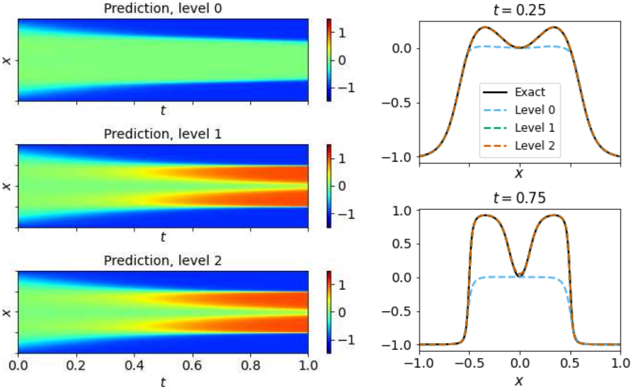

→ Due to the **multiscale structure of the problem**, the **improvements** due to the **multifidelity FBPINN approach** are **even stronger**.

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

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



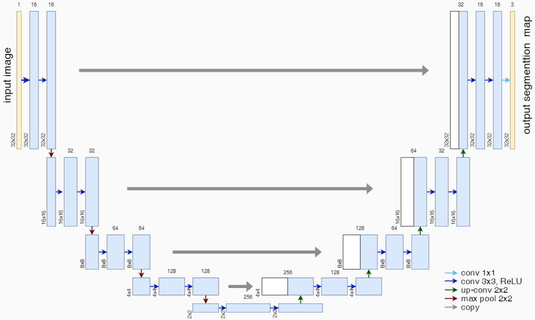

| method | arch. | # params | error |
|---|---|---|---|
| PINN | 0 levels<br>6x50 | 12 951 | 0.499 |
| S–FBPINN | 2 levels<br>5x32, 1x20 | 39 627 | 0.00594 |




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

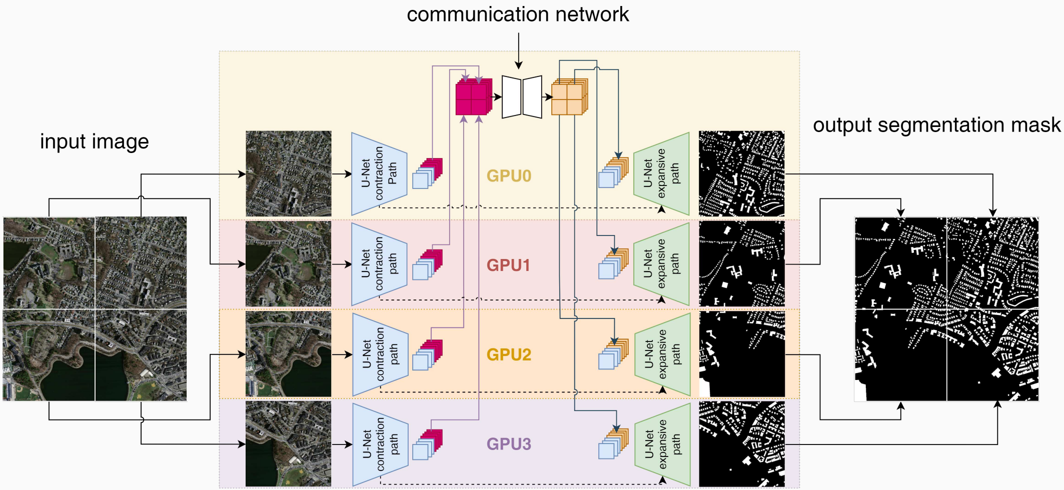



- As an example for a **convolutional neural network (CNN)**, we employ the **U-Net architecture** introduced in **Ronneberger, Fischer, and Brox (2015)**.
- The U-Net yields **state-of-the-art accuracy in semantic image segmentation** and other **image-to-image tasks**.

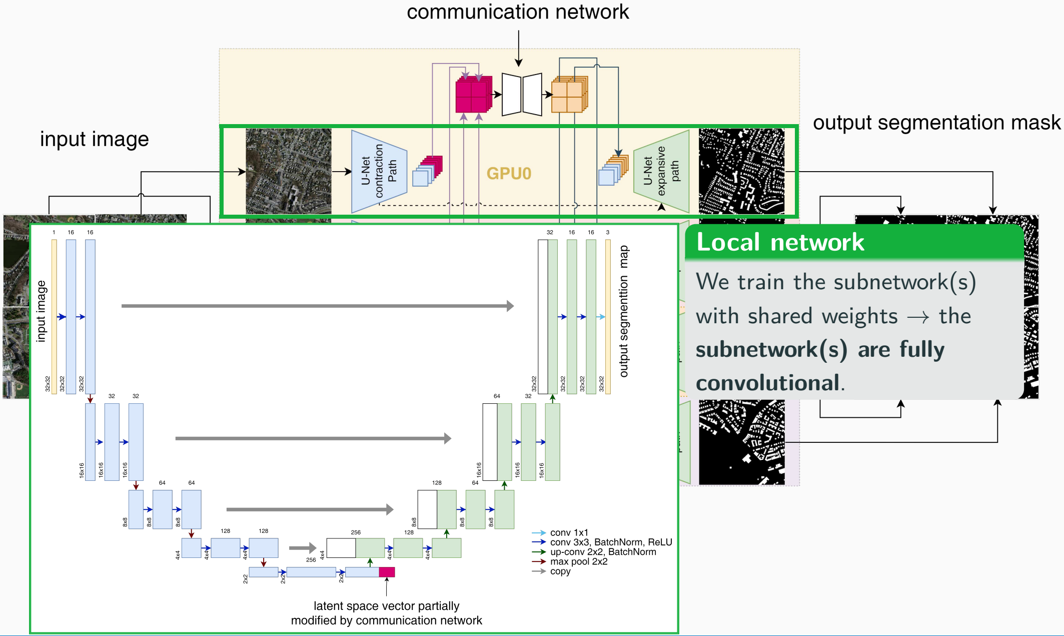
***Below:*** *memory consumption for training on a single* $1024 \times 1024$ *image.*

| name | size | # channels input | # channels output | mem. feature maps # of values | mem. feature maps MB | mem. weights # of values | mem. weights 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



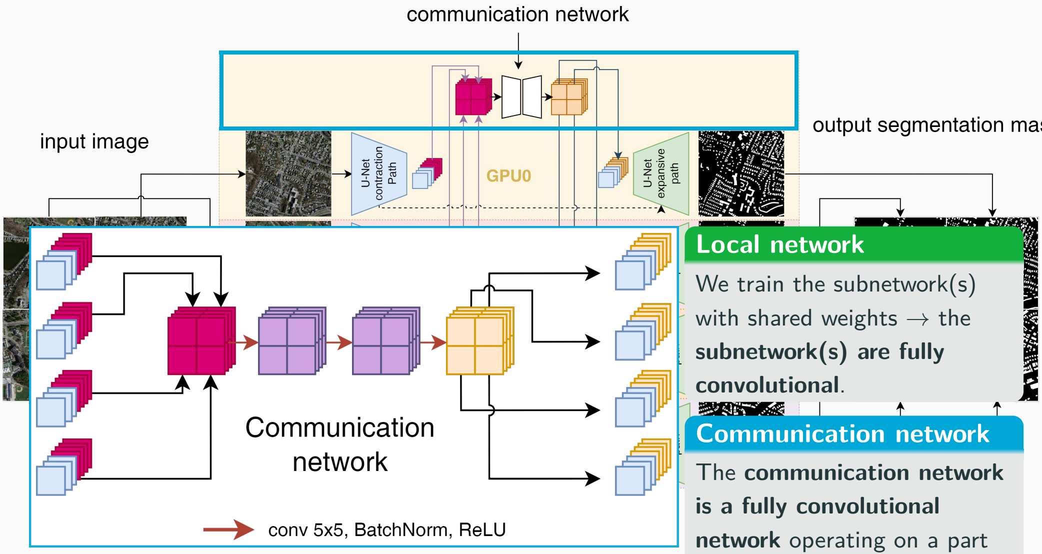

Cf. **Verburg, Heinlein, Cyr (in preparation).**




## Local network

We train the subnetwork(s) with shared weights $\rightarrow$ the **subnetwork(s) are fully convolutional**.

# Decomposing the U-Net

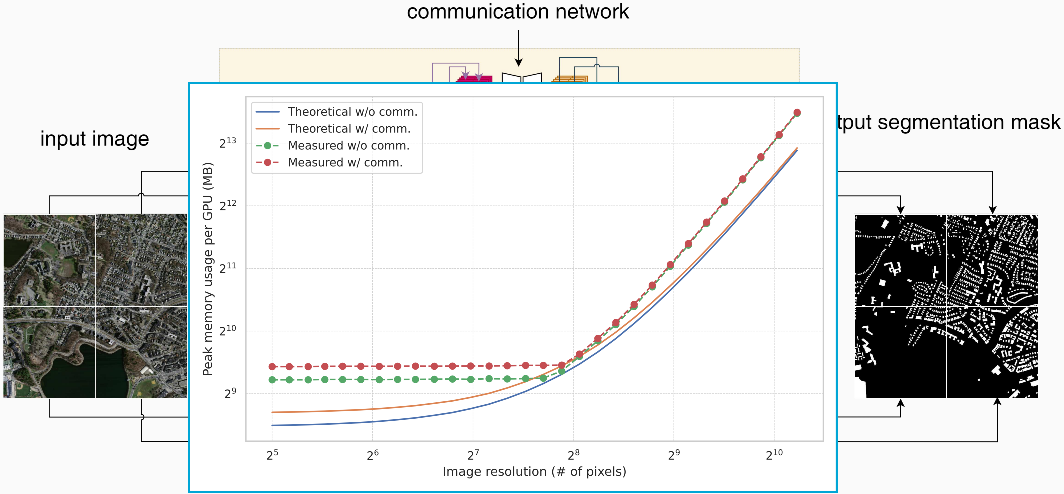



**Local network**

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

**Communication network**

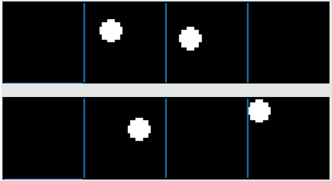
The **communication network is a fully convolutional network** operating on a part of the coarse feature maps.



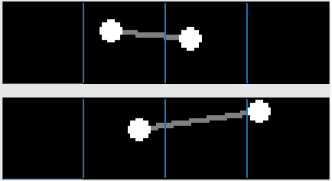

- Distribution of feature maps results in **significant reduction of memory usage on a single GPU**
- **Moderate additional memory usage** due to the **communication network**

### Task: Connect two dots via a line segment

**Input**



**Target (segmentation mask)**

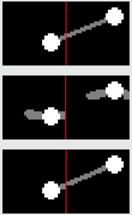


### Result: Communication

True mask

Pred. (no comm.)

Pred. (comm.)



Training & testing on # of subimages (32 × 32 pixels) vs. # feature maps communicated:

| # subimages | 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**

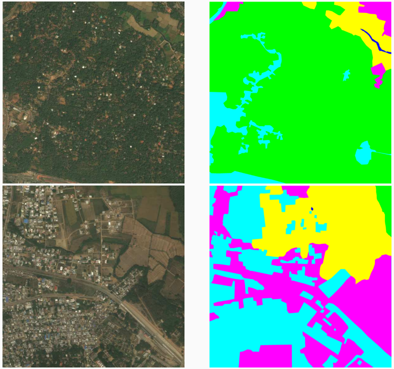
Training on # subimages (32 × 32 pixels) vs. # feature maps communicated:

| # 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 % |

**Input** **Target**

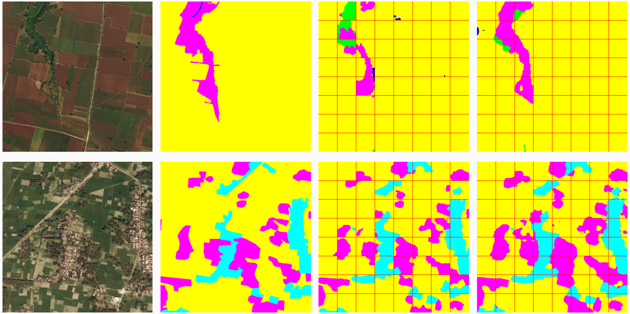


## Avoiding overfitting

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)**)

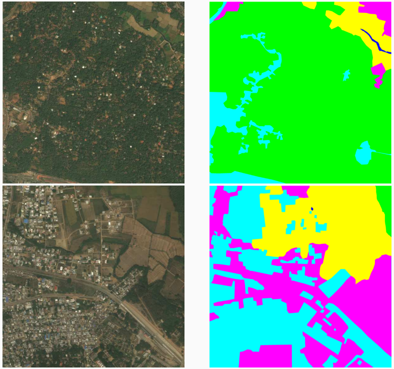
**image** **true mask** **without comm.** **with comm.**



# 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 % |

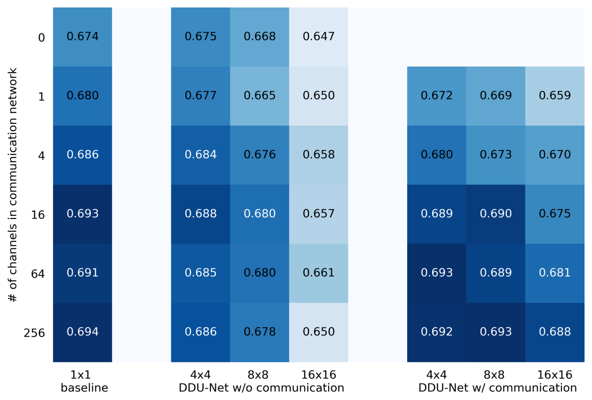
**Input** **Target**



## 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)**)



## 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**.

## Stacking Multifidelity FBPINNs

- 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!**

**Topical Activity Group**

**Scientific Machine Learning**