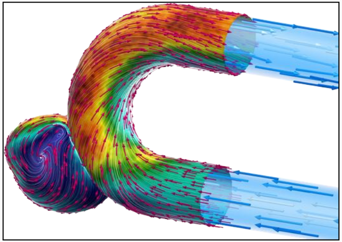
Localization of Neural Networks Using Domain Decomposition

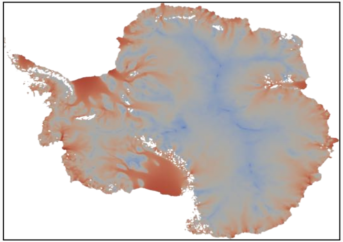
Alexander Heinlein[1]

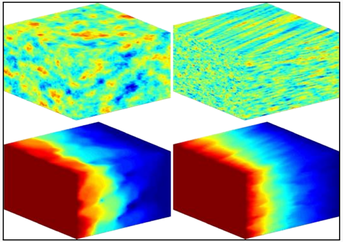
Seminar, Great Bay University, Dongguan, China, November 10, 2025

[1]Delft University of Technology

Scientific Computing and Machine Learning







Numerical methods

Based on physical models

+ Robust and generalizable
− Require availability of mathematical models

Machine learning models

Driven by data

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

Scientific machine learning

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

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

Outline

1. Domain decomposition for physics-informed neural networks

   Based on joint work with

   Victorita Dolean (Eindhoven University of Technology)
   Taniya Kapoor (Wageningen University & Research)
   Siddhartha Mishra (ETH Zürich)
   Ben Moseley (Imperial College London)
   Yong Shang and **Fei Wang** (Xi'an Jiaotong University)

2. Spectral analysis and domain decomposition for deep operator networks

   Based on joint work with

   Amanda A. Howard and **Panos Stinis** (Pacific Northwest National Laboratory)
   Johannes Taraz (Delft University of Technology)

3. Domain decomposition-based image segmentation for high-resolution image segmentation on multiple GPUs

   Based on joint work with

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

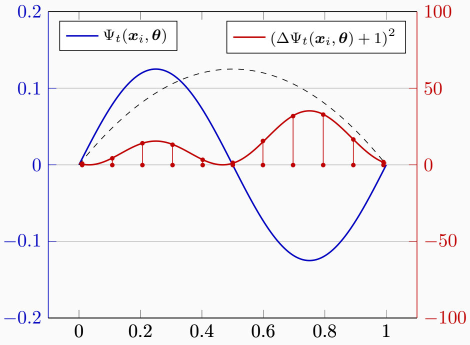
Domain decomposition for physics-informed neural networks

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**:

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



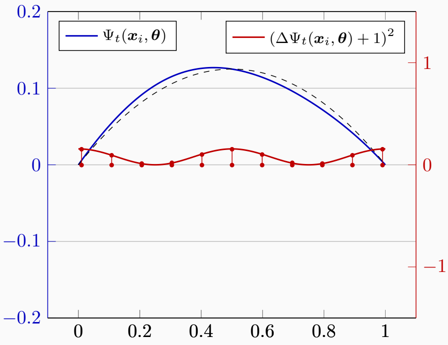
$(\Delta\Psi_t(\boldsymbol{x}_i,\theta)+1)^2 >> 0$

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**



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

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**:

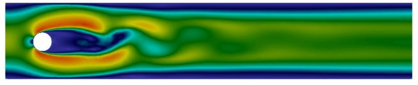
$$\mathcal{L}(\boldsymbol{\theta}) = \omega_{\text{data}} \mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}),$$

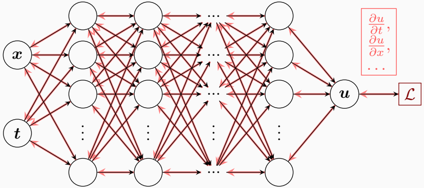
where ω_{data} and ω_{PDE} are **weights** and

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

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

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





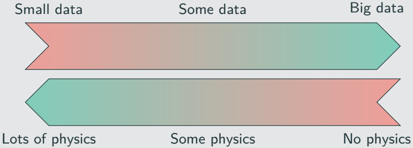
Advantages

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

Drawbacks

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

Hybrid loss



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

Error Estimate & Spectral Bias

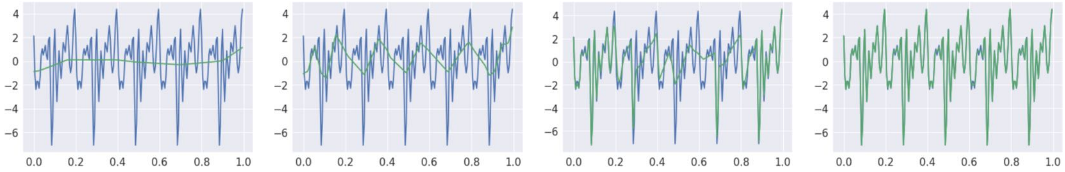
Estimate of the generalization error (Mishra and Molinaro (2022))

The generalization error (or total error) satisfies

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

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

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



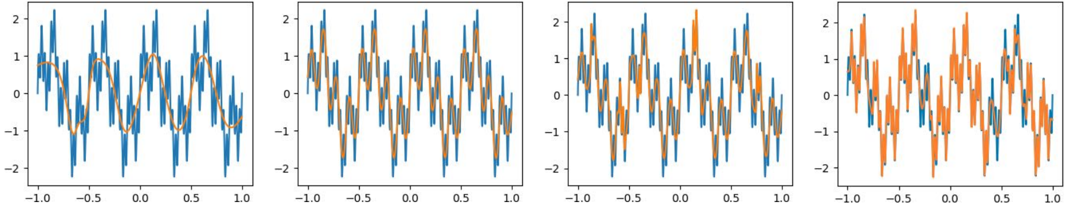
100 iterations | 1 000 iterations | 10 000 iterations | 80 000 iterations

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

Related works: **Cao et al. (2021), Wang, et al. (2022), Hong et al. (arXiv 2022), Xu et al. (2024), ...**

Spectral Bias of Neural Networks

Rahaman et al. (2019) observed the **spectral bias of neural networks**: during training, they **learn low-frequency functions faster than high-frequency functions**; see also **Cao et al. (2021), Wang, et al. (2022), Hong et al. (arXiv 2022), Xu et al. (2024), ...**



3 k iterations　　25 k iterations　　100 k iterations　　400 k iterations

This can be understood by interpreting the traning dynamics of neural networks as **gradient flow**

$$\theta_{k+1} = \theta_k - \eta_k \nabla_\theta \mathcal{C}(\mathcal{N}_{\theta_k}) \quad \rightarrow \quad \frac{\mathrm{d}x}{\mathrm{d}t}(t) = -\nabla_x \mathcal{C}(x(t))$$

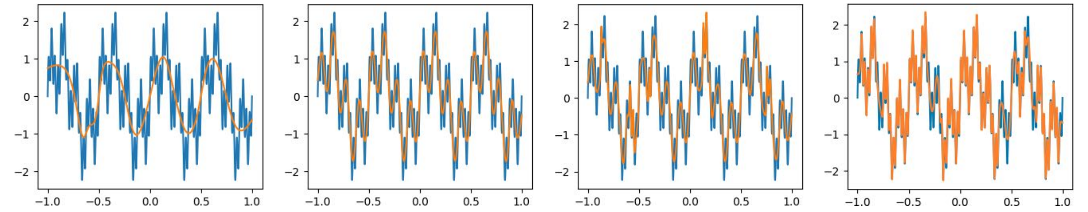
For **infinitely wide neural networks**, we obtain a **constant neural tangent kernel (NTK)** (**Jacot et al. (2018)**).

For mean squared error (MSE) loss $\mathcal{C}$, we obtain the **discretized** $\boldsymbol{K} = \left(\left\langle \frac{d\mathcal{N}_{\theta(t)}}{d\theta}(x_i), \frac{d\mathcal{N}_{\theta(t)}}{d\theta}(x_j) \right\rangle \right)_{ij}$:

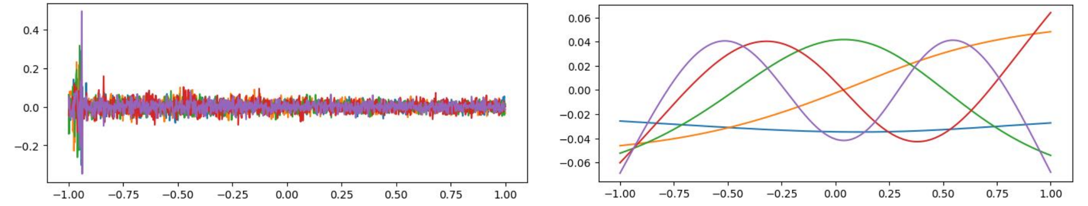
$$\frac{d\mathcal{N}_{\theta(t)}}{dt} = -\frac{2}{n} \cdot \boldsymbol{K}(t) \left(\mathcal{N}_{\theta(t)}(\mathbf{X}) - \mathbf{Y} \right) \quad \Rightarrow \quad \mathbf{U}^\top \left(\mathcal{N}_{\theta(t)}(\mathbf{X}) - \mathbf{Y} \right) \approx e^{-t\frac{2}{n}\mathbf{\Lambda}} \mathbf{U}^\top \mathbf{Y}$$

Spectral Bias of Neural Networks

Rahaman et al. (2019) observed the **spectral bias of neural networks**: during training, they **learn low-frequency functions faster than high-frequency functions**; see also **Cao et al. (2021), Wang, et al. (2022), Hong et al. (arXiv 2022), Xu et al. (2024), . . .**



3 k iterations | 25 k iterations | 100 k iterations | 400 k iterations



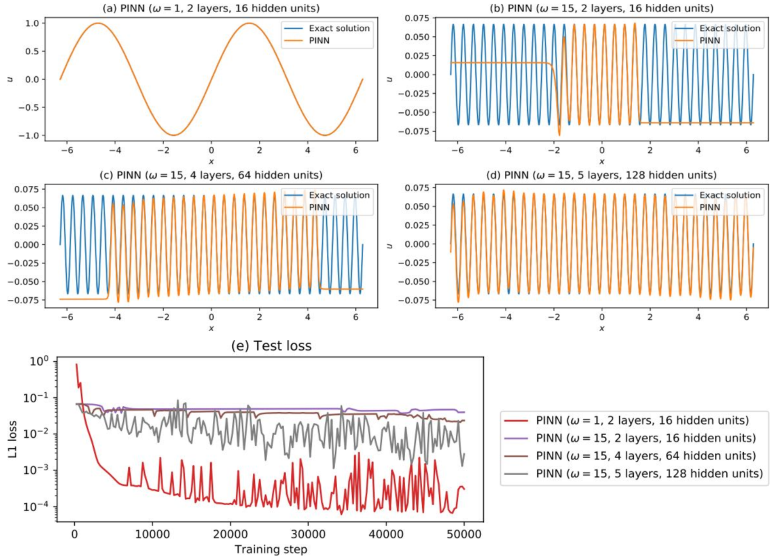
smallest eigenvalues (**slow training**) | largest eigenvalues (**fast training**)

Scaling of PINNs for a Simple ODE Problem

Solve

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

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



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

Scaling issues

- Large computational domains
- Small frequencies

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

Solve

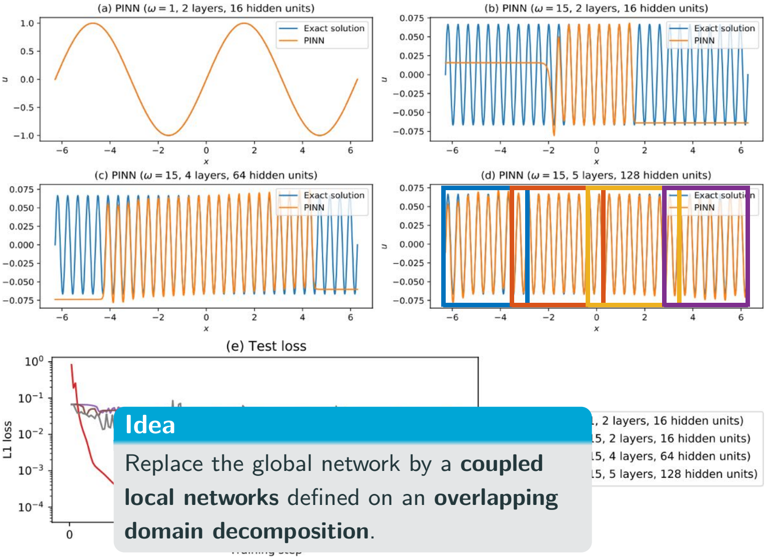
$$\begin{aligned} u' &= \cos(\omega \boldsymbol{x}), \\ u(0) &= 0, \end{aligned}$$

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

Scaling issues

- Large computational domains
- Small frequencies

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



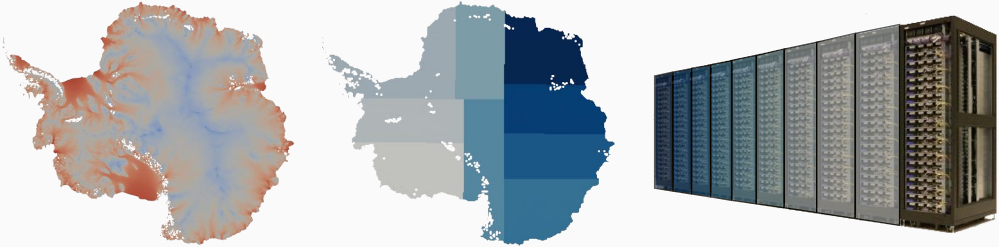
Idea

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

(a) 321 free parameters

(d) 66 433 free parameters

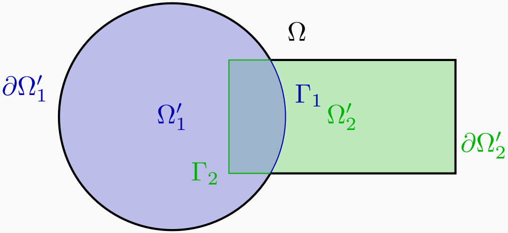
Domain Decomposition Methods



Graphics based on results from **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**.



Domain Decomposition Methods and Machine Learning – Literature

A **non-exhaustive literature overview**:

- **ML for adaptive BDDC, FETI–DP, AGDSW**: H., Klawonn, Lanser, Weber (2019, 2020, 2021, 2021, 2021, 2022); Klawonn, Lanser, Weber (2024, 2025)
- **cPINNs, XPINNs**: Jagtap, Kharazmi, Karniadakis (2020); Jagtap, Karniadakis (2020)
- **Classical Schwarz iteration for PINNs or DeepRitz:**: Li, Tang, Wu, and Liao (2019); Li, Xiang, Xu (2020); Mercier, Gratton, Boudier (arXiv 2021); Dolean, H., Mercier, Gratton (acc. 2025); Li, Wang, Cui, Xiang, Xu (2023); Sun, Xu, Yi (arXiv 2023, 2024); Kim, Yang (2023, 2024, 2024)
- **FBPINNs, FBKANs**: Moseley, Markham, Nissen-Meyer (2023); Dolean, H., Mishra, Moseley (2024, 2024); H., Howard, Beecroft, Stinis (2025); Howard, Jacob, Murphy, H., Stinis (arXiv 2024)
- **DD for randomized NNs**: Dong, Li (2021); Dang, Wang (2024); Sun, Dong, Wang (2024); Sun, Wang (2024); Chen, Chi, E, Yang (2022); Shang, H., Mishra, Wang (2025); Anderson, Dolean, Moseley, Pestana, (arXiv 2024); van Beek, Dolean, Moseley (arxiv 2025)
- **DD for Neural Operators and Surrogate Models**: H., Howard, Beecroft, Stinis (2025); Ramezankhani, Parekh, Deodhar, Birru (arXiv 2024); Wu, Kovachki, Liu (arXiv 2025); Pelzer, Verburg, H., Schulte (arXiv 2025); Klaes, Klawonn, Kubicki, Lanser, Nakajima, Shimokawabe, Weber (arXiv 2025); Howard, H., Stinis (in prep.)
- **DD for CNNs**: Gu, Zhang, Liu, Cai (2022); Lee, Park, Lee (2022); Klawonn, Lanser, Weber (2024); Verburg, H., Cyr (2025)

An overview of the state-of-the-art in 2024:

A. Klawonn, M. Lanser, J. Weber
Machine learning, domain decomposition methods – a survey
Computational Science and Engineering. 2024

Finite Basis Physics-Informed Neural Networks (FBPINNs)

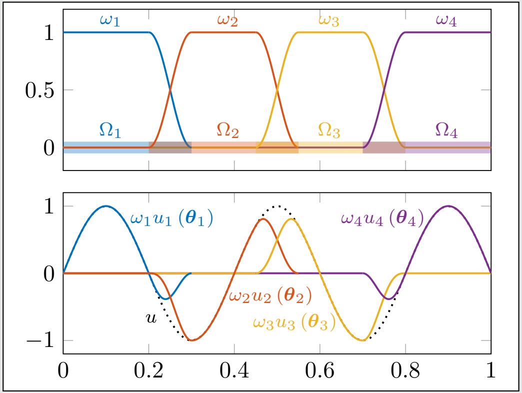
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

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

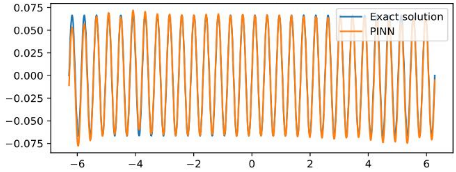
and the **loss function**

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

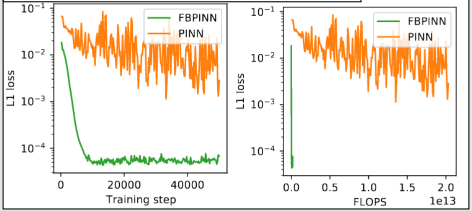


1D single-frequency problem

PINN solution



Moseley, Markham, Nissen-Meyer (2023)



Finite Basis Physics-Informed Neural Networks (FBPINNs)

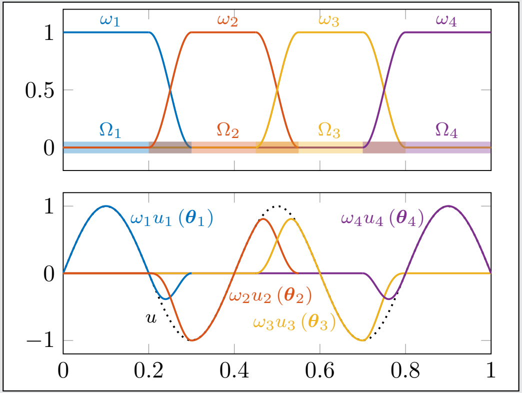
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

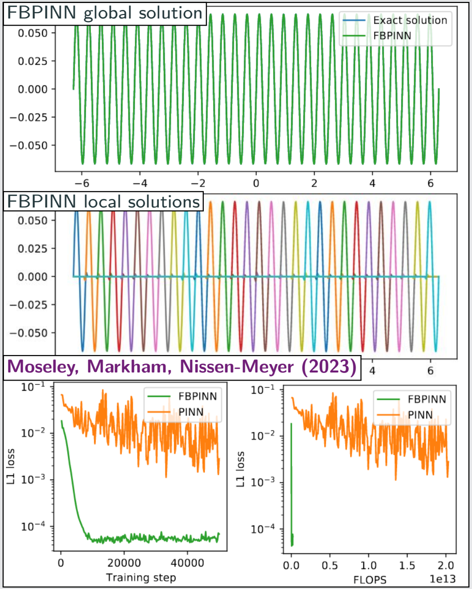
$$u(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \sum\nolimits_{j=1}^{J} \omega_j u_j\left(\boldsymbol{\theta}_j\right)$$

and the **loss function**

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



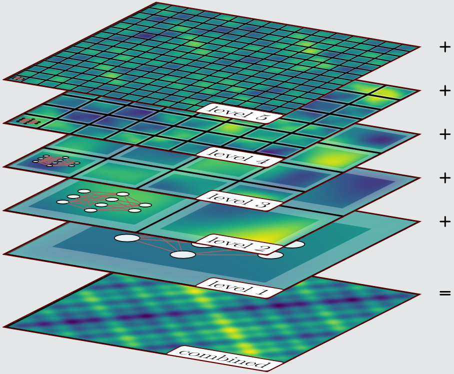
1D single-frequency problem



Multi-Level FBPINNs

Multi-level FBPINNs (ML-FBPINNs)

ML-FBPINNs (**Dolean, Heinlein, Mishra, Moseley (2024)**) are based on a **hierarchy of domain decompositions:**

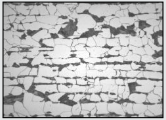


This yields the **network architecture**:

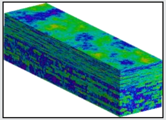
$$u\left(\theta_1^{(1)}, \ldots, \theta_{J^{(L)}}^{(L)}\right) = \sum_{l=1}^{L} \sum_{i=1}^{N^{(l)}} \omega_j^{(l)} u_j^{(l)}\left(\theta_j^{(l)}\right)$$

Multiscale problems . . .

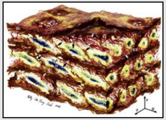
. . . appear in most areas of modern science and engineering:



Dual-phase steel; fig. courtesy of **J. Schröder**.



Groundwater flow; cf. **Christie & Blunt (2001)** (SPE10).



Arterial walls; cf. **O'Connell et al. (2008)**.

Multi-frequency problem

Consider the **multi-frequency Laplace problem**

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

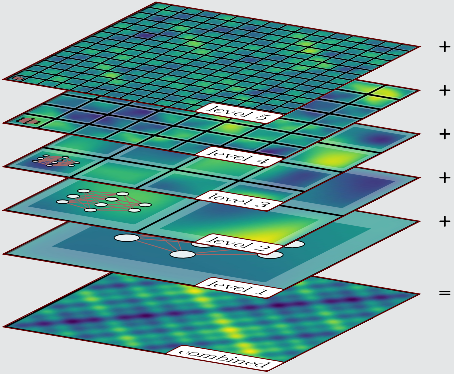
with homogeneous Dirichlet boundary conditions and $\omega_i = 2^i$.

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

Multi-Level FBPINNs

Multi-level FBPINNs (ML-FBPINNs)

ML-FBPINNs (**Dolean, Heinlein, Mishra, Moseley (2024)**) are based on a **hierarchy of domain decompositions:**

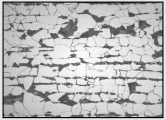


This yields the **network architecture**:

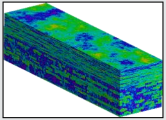
$$u\left(\theta_1^{(1)}, \ldots, \theta_{J^{(L)}}^{(L)}\right) = \sum_{l=1}^{L} \sum_{i=1}^{N^{(l)}} \omega_j^{(l)} u_j^{(l)}\left(\theta_j^{(l)}\right)$$

Multiscale problems . . .

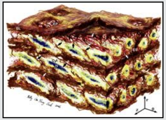
. . . appear in most areas of modern science and engineering:



Dual-phase steel; fig. courtesy of **J. Schröder**.



Groundwater flow; cf. **Christie & Blunt (2001)** (SPE10).



Arterial walls; cf. **O'Connell et al. (2008)**.

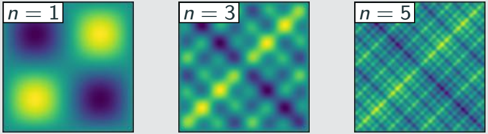
Multi-frequency problem

Consider the **multi-frequency Laplace problem**

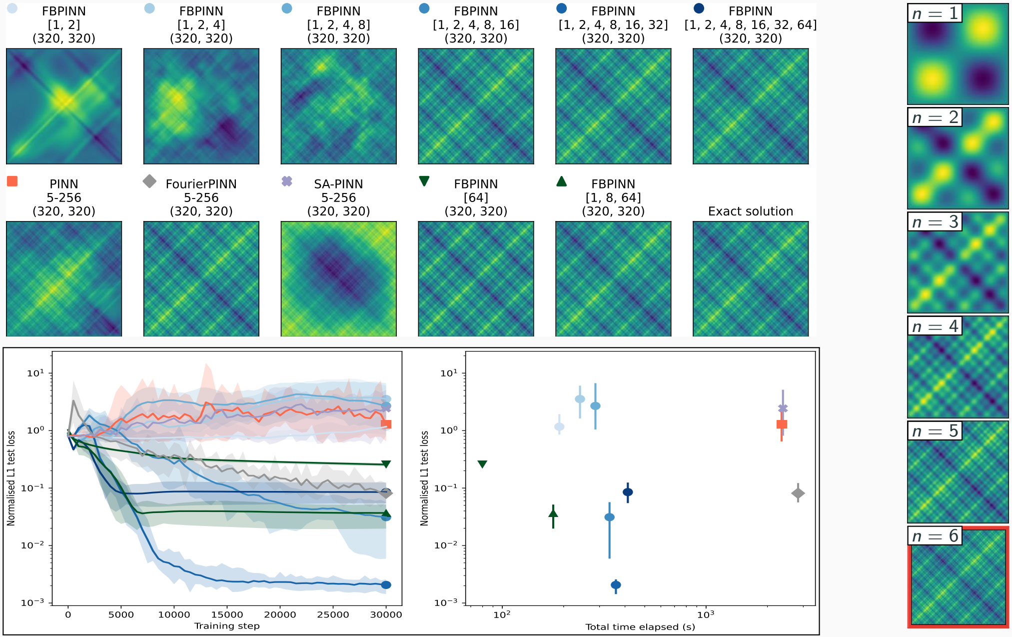
$$-\Delta u = 2 \sum_{i=1}^{n} (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y),$$

with homogeneous Dirichlet boundary conditions and $\omega_i = 2^i$.

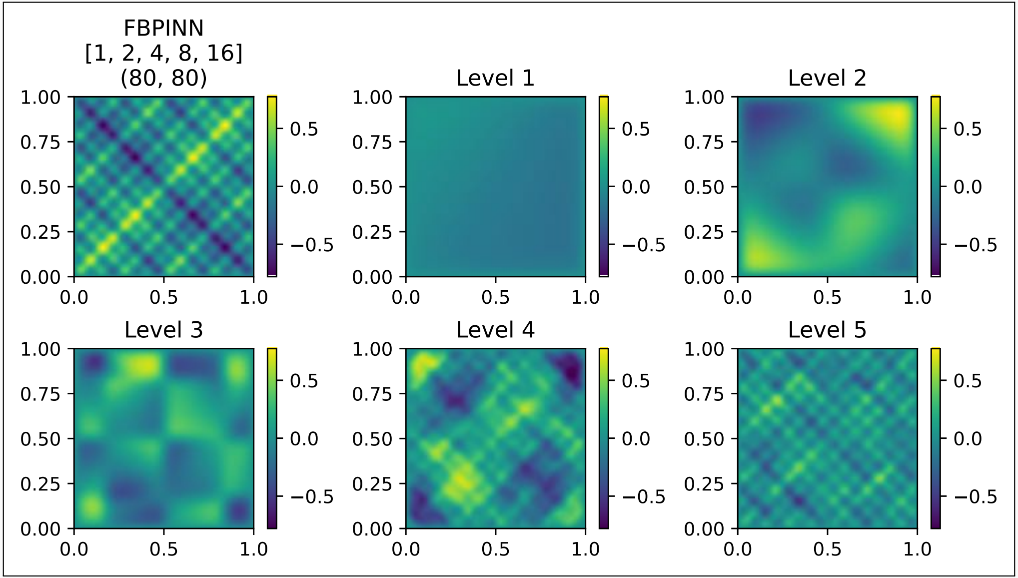
For increasing values of n, we obtain the **solutions**:



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



Multi-Frequency Problem – What the FBPINN Learns



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

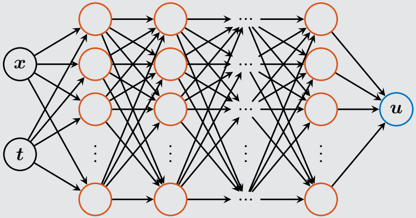
Physics-Informed Randomized Neural Networks (PIRaNNs)

Neural networks

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

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

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



In order to optimize the loss function

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

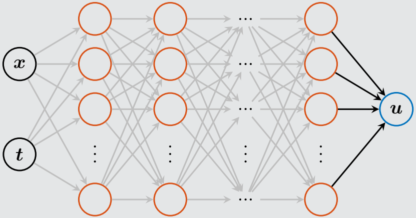
all parameters $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{W}_1, \boldsymbol{b}_1, \ldots, \boldsymbol{W}_L, \boldsymbol{b}_L)$ are **trained**.

Randomized neural networks

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

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

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



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

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

This can **simplify the training process**.

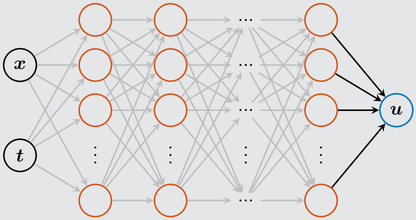
Physics-Informed Randomized Neural Networks (PIRaNNs)

Randomized neural networks

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

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

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



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

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

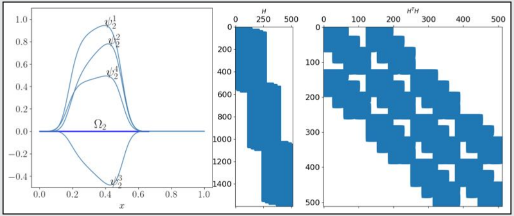
This can **simplify the training process**.

Domain decomposition for RaNNs

We employ the FBPINNs approach; cf. **Shang, Heinlein, Mishra, Wang (2025)**. This is closely related to the **random feature method (RFM)** by **Chen, Chi, E, Yang (2022)**. In particular, we solve

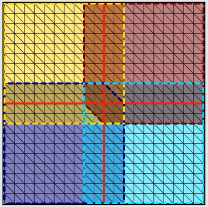
$$\mathcal{A}\left[\sum_{j=1}^{J} \omega_j u_j\left(\boldsymbol{A}_j\right)\right](\boldsymbol{x}_i) = f(\boldsymbol{x}_i),$$

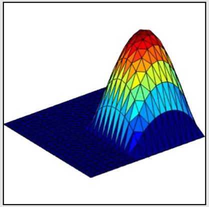
for $i = 1, \ldots, N_{\text{PDE}}$; the boundary condtions are incorporated directly into the u_j.



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

One-level Schwarz preconditioner





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

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

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

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

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

Singular Value Decomposition

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

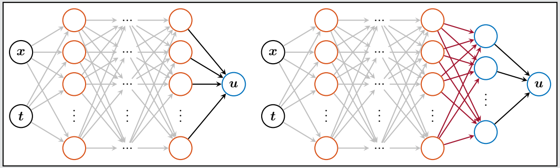
$$\begin{aligned} u_j(\boldsymbol{x}, \boldsymbol{A}_j) &= F_{L+1}^{\boldsymbol{A}} \cdot F_L^{W_L, b_L} \circ \ldots \circ F_1^{W_1, b_1}(\boldsymbol{x}) \\ &= \boldsymbol{A}_j \begin{bmatrix} \Phi_1(\boldsymbol{x}) & \cdots & \Phi_k(\boldsymbol{x}) \end{bmatrix}^\top, \end{aligned}$$

where k is the width of the last hidden layer and the Φ_l are the randomized basis functions.

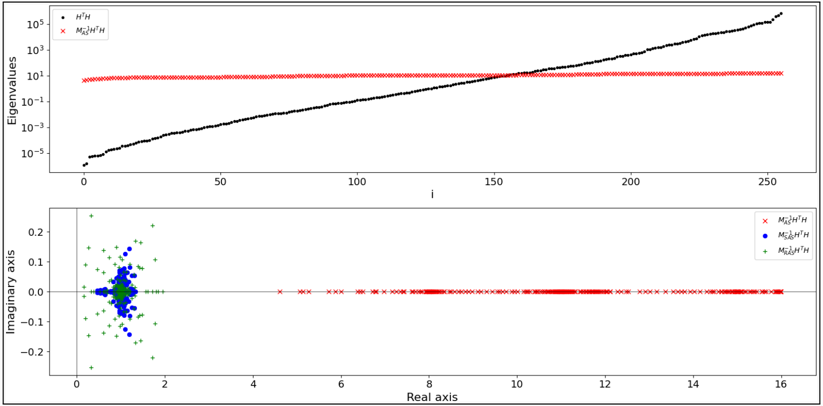
Consider a **reduced SVD** $\boldsymbol{\Phi} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, where the entries of the matrix are $\boldsymbol{\Phi}_{i,l} = \Phi_l(\boldsymbol{x}_i)$. Then, we consider

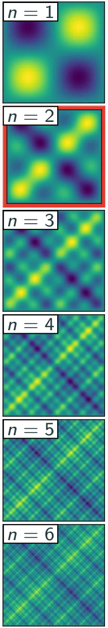
$$\hat{u}_j(\boldsymbol{x}, \boldsymbol{A}_j) = \boldsymbol{A}_j \hat{\boldsymbol{V}}^\top \begin{bmatrix} \Phi_1(\boldsymbol{x}) & \cdots & \Phi_k(\boldsymbol{x}) \end{bmatrix}^\top,$$

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



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

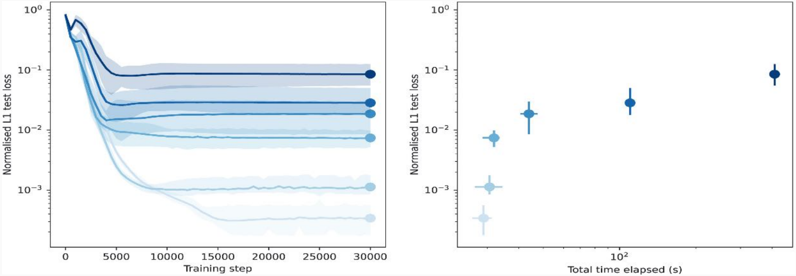




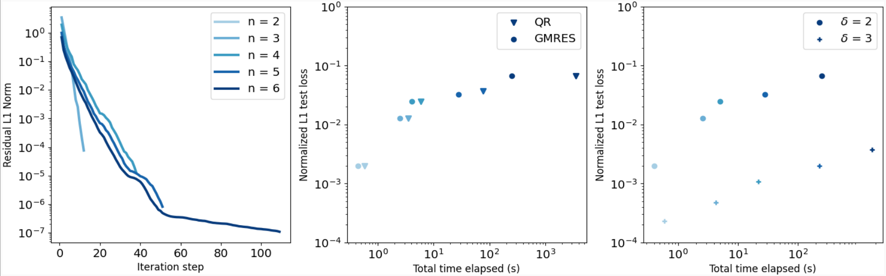
	$M^{-1}=I$		$M^{-1}=M_{AS}^{-1}$		$M^{-1}=M_{RAS}^{-1}$		$M^{-1}=M_{SAS}^{-1}$	
	iter	e_{L^2}	iter	e_{L^2}	iter	e_{L^2}	iter	e_{L^2}
CG	>2000	$1.95\cdot10^{-2}$	8	$5.03\cdot10^{-3}$	—	—	—	—
CGS	>2000	$2.63\cdot10^{-2}$	4	$5.04\cdot10^{-3}$	24	$5.03\cdot10^{-3}$	6	$5.04\cdot10^{-3}$
BICG	>2000	$1.03\cdot10^{-2}$	8	$5.08\cdot10^{-3}$	32	$5.05\cdot10^{-3}$	11	$5.09\cdot10^{-3}$
GMRES	>2000	$8.68\cdot10^{-2}$	13	$5.07\cdot10^{-3}$	31	$5.06\cdot10^{-3}$	11	$5.08\cdot10^{-3}$

4×4 subdomains; DoF $=256$; $N=1600$; $\boldsymbol{\theta}^0\in\mathcal{U}(-1,1)$; stop.: $\|\boldsymbol{M}^{-1}\boldsymbol{r}^k\|_{L^2}/\|\boldsymbol{M}^{-1}\boldsymbol{r}^0\|_{L^2}\leq10^{-5}$

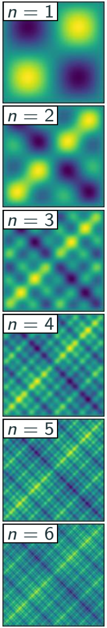
Results for the Multi-Frequency Problem



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



DD-PIRaNNs; cf. Shang, Heinlein, Mishra, Wang (2025)



Gauss–Newton Training

Many popular optimizers are based on **gradient descent**:

$$\theta^{(m+1)} = \theta^{(m)} - \alpha \nabla_\theta \mathcal{L}(\theta^{(m)}),$$

where $\theta^{(m)}$ are the parameters at iteration m, α is the learning rate, and $\mathcal{L}$ is the loss function; a commonly used example is the **Adam** optimizer (**Kingma (2017)**).

The **Gauss–Newton method** is based on **Newton's method** but uses the **Gramian** G as an **approximate Jacobian**:

$$\theta^{(m+1)} = \theta^{(m)} - \alpha\, \boldsymbol{G}^{+}(\theta^{(m)})\, \nabla_\theta \mathcal{L}(\theta^{(m)}),$$

where $G^{+}(\theta)$ is the pseudoinverse of G, as it may **not** be **invertible** but **only symmetric positive semidefinite**. Therefore, we may regularize G and consider $G + \mu I$ instead, making the matrix **symmetric positive definite**.

Mean squared error (MSE) loss

$$\boldsymbol{G}(\theta)_{ij} = \sum_{\boldsymbol{x}_i \in \Omega} (\partial_{\theta_i} u_\theta(\boldsymbol{x}_i))\, (\partial_{\theta_j} u_\theta(\boldsymbol{x}_i))$$

Physics-informed loss function

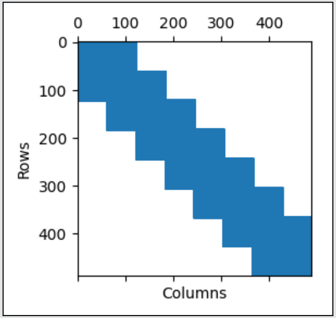
$$\boldsymbol{G}(\theta)_{ij} = \sum_{\boldsymbol{x}_i \in \Omega} \partial_{\theta_i} \mathcal{N}[u]_\theta(\boldsymbol{x}_i)\, \partial_{\theta_j} \mathcal{N}[u]_\theta(\boldsymbol{x}_i)$$

See, for instance, **Müller and Zeinhofer (2023)** and **Cai et al. (arXiv 2024)**.

Sparsity

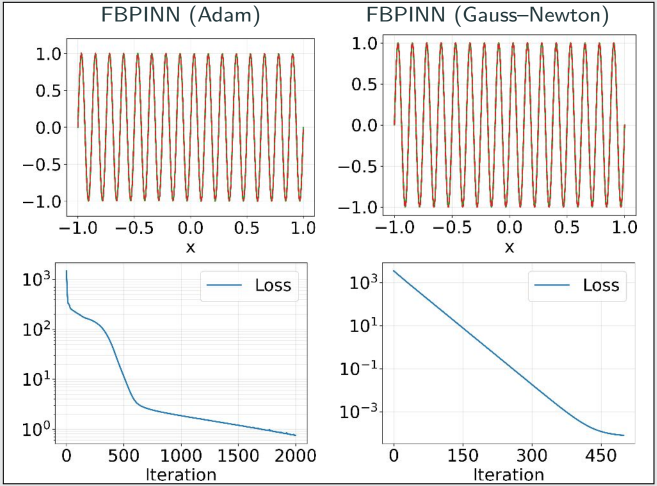
The **domain decomposition introduces sparsity** in $\boldsymbol{G}$:

- 8 subdomains
- **coupling blocks** due to **overlap between subdomains**



Cf. **Heinlein and Kapoor (arXiv 2025)**.

Comparison of Adam annd Gauss–Newton training

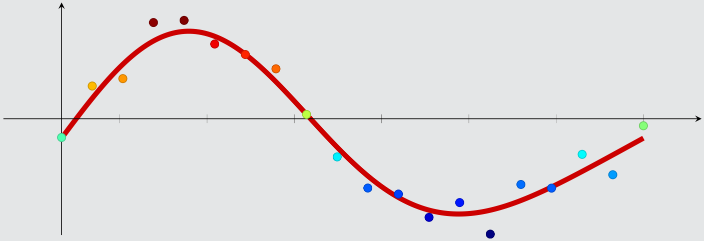


	FBPINN (Adam)	FBPINN (Gauss–Newton)
test MSE	7.8×10^{-3}	8.0×10^{-4}

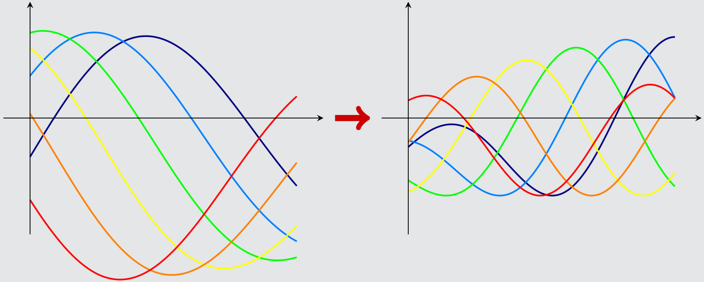
Spectral analysis and domain decomposition for deep operator networks

Function Versus Operator Learning

Function learning

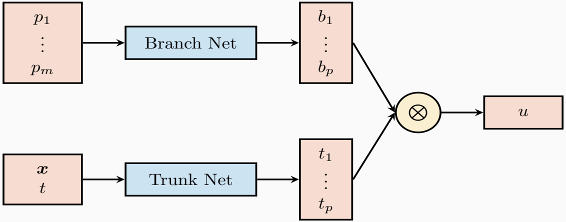


Operator learning



Deep Operator Networks (DeepONets / DONs)

Neural operators learn operators between function spaces using neural networks. Here, we learn the **solution operator** of a initial-boundary value problem parametrized with $p_1, \ldots, p_m$ using **DeepONets** as introduced in **Lu et al. (2021)**.



Single-layer case

The DeepONet architecture is based on the **single-layer case** analyzed in **Chen and Chen (1995)**. In particular, the authors show **universal approximation properties for continuous operators**.

The architecture is based on the following ansatz for presenting the parametrized solution

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

Physics-informed DeepONets

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

Other operator learning approaches

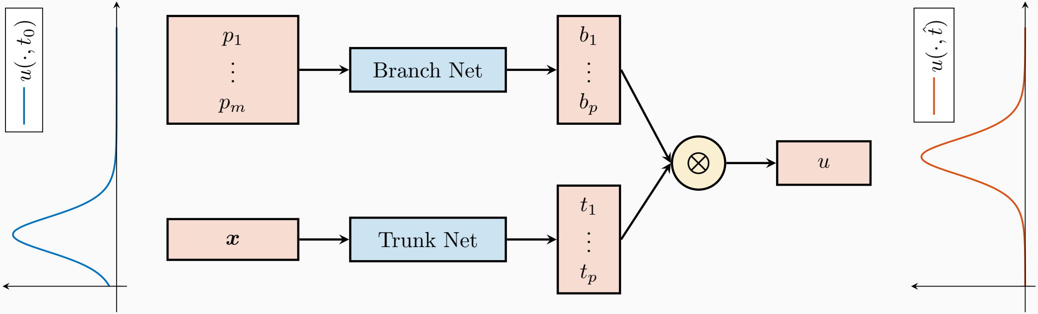
- **FNOs**: **Li et al. (2021)**
- **PCA-Net**: **Bhattacharya et al. (2021)**
- **Random features**: **Nelsen and Stuart (2021)**
- **CNOs**: **Raonić et al. (2023)**

How a DeepONet Maps Between Function Spaces

To illustrate how a DeepONet operates, we consider the **Korteweg–de Vries (KdV) equation**

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - 0.01\frac{\partial^3 u}{\partial x^3},$$

which models unidirectional waves in shallow water. Our goal is to train a DeepONet that predicts the wave profile at a future time $\hat{t}$ from the observed height profile $u(\cdot, t_0)$.



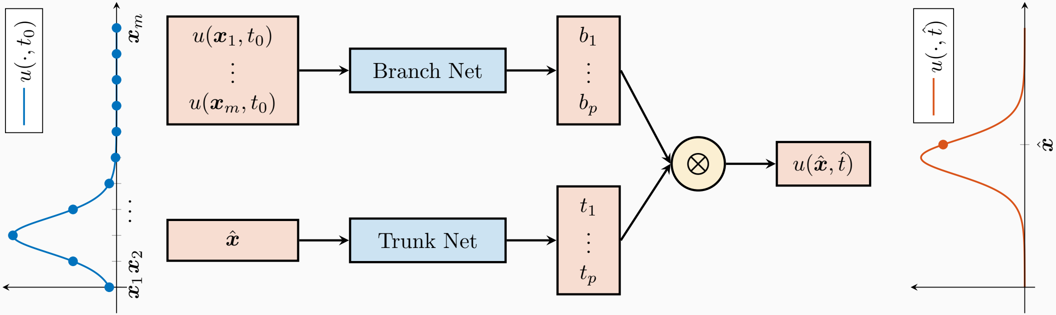
Here, the forecast time $\hat{t}$ is fixed to keep the learning task simple. A **more general neural operator** can take the target time as an additional input to the trunk network.

How a DeepONet Maps Between Function Spaces

To illustrate how a DeepONet operates, we consider the **Korteweg–de Vries (KdV) equation**

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - 0.01\frac{\partial^3 u}{\partial x^3},$$

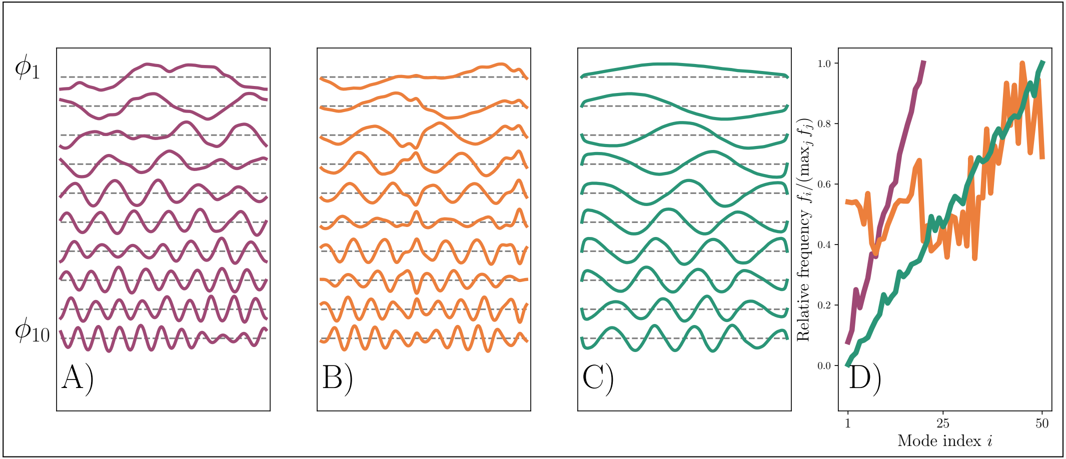
which models unidirectional waves in shallow water. Our goal is to train a DeepONet that predicts the wave profile at a future time $\hat{t}$ from the observed height profile $u(\cdot, t_0)$.



Here, the forecast time $\hat{t}$ is fixed to keep the learning task simple. A **more general neural operator** can take the target time as an additional input to the trunk network.

Let us consider some **examples of the left singular vectors** for **three differential equations**:

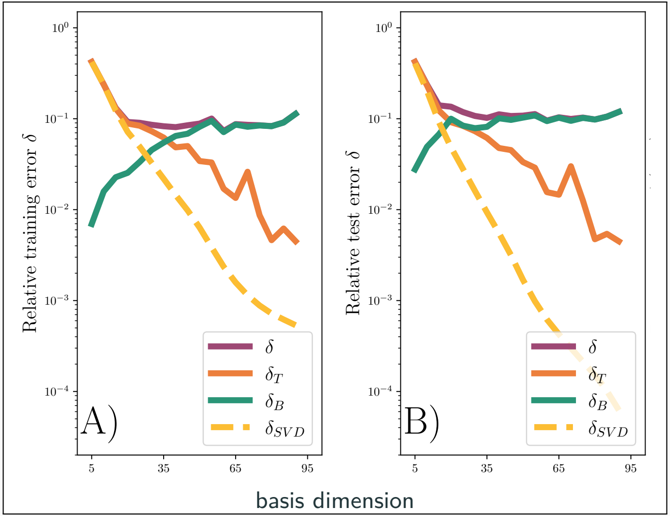
A) advection-diffusion equation **B) KdV equation** **C) Burgers equation**



The learned trunk bases have been investigated in more detail in **Williams et al. (2024)**.

DeepONet – Error Decomposition Results for the KdV Equation

Results for the **KdV equation** with $t_0 = 0.0$ and $\hat{t} = 0.2$.   900 training and 100 test configurations.



total error δ **trunk error** δ_T **branch error** δ_B **SVD truncation error** δ_{SVD}

DeepONet – Branch Error

Using the left singular vectors, the **branch error** becomes

$$\mathcal{E}_B = \sum_{i=1}^{m} \sigma_i^2 \underbrace{\|b_i - v_i\|_2^2}_{=:L_i}.$$

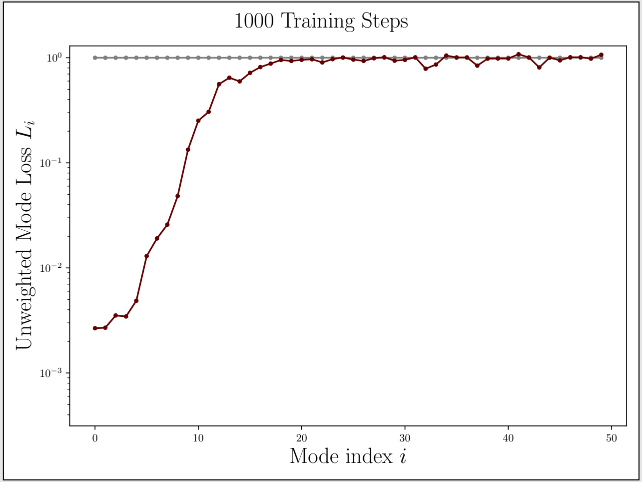
We call

$$\sigma_i^2 L_i$$

the **(weighted) mode loss** because it equals the loss contribution of the ith mode. Accordingly, L_i is the **unweighted mode loss**.

This choice of **left singular vectors as the trunk basis** is often denoted **POD-DeepONet** in **Lu et al. (2022)**.

Unweighted mode loss



DeepONet – Branch Error

Using the left singular vectors, the **branch error** becomes

$$\mathcal{E}_B = \sum_{i=1}^{m} \sigma_i^2 \underbrace{\|b_i - v_i\|_2^2}_{=:L_i}.$$

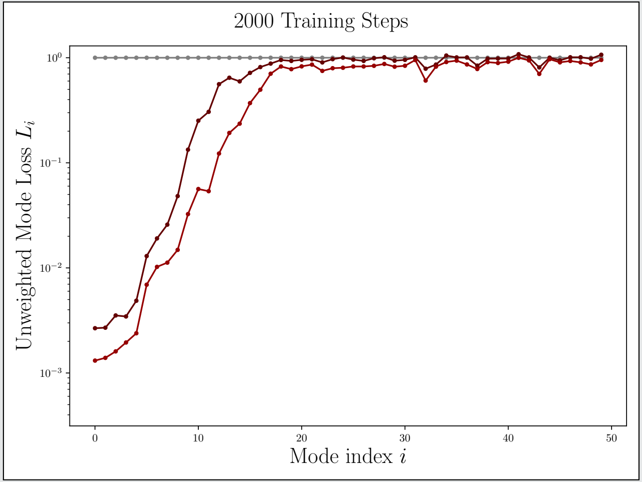
We call

$$\sigma_i^2 L_i$$

the **(weighted) mode loss** because it equals the loss contribution of the ith mode. Accordingly, L_i is the **unweighted mode loss**.

This choice of **left singular vectors as the trunk basis** is often denoted **POD-DeepONet** in **Lu et al. (2022)**.

Unweighted mode loss



DeepONet – Branch Error

Using the left singular vectors, the **branch error** becomes

$$\mathcal{E}_B = \sum_{i=1}^{m} \sigma_i^2 \underbrace{\|b_i - v_i\|_2^2}_{=:L_i}.$$

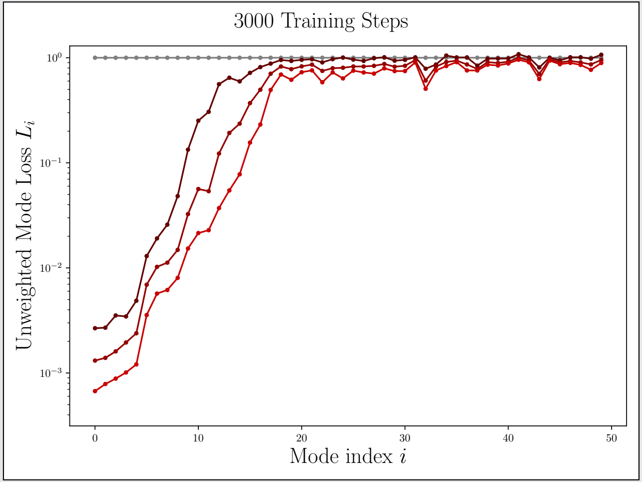
We call

$$\sigma_i^2 L_i$$

the **(weighted) mode loss** because it equals the loss contribution of the ith mode. Accordingly, L_i is the **unweighted mode loss**.

This choice of **left singular vectors as the trunk basis** is often denoted **POD-DeepONet** in **Lu et al. (2022)**.

Unweighted mode loss



DeepONet – Branch Error

Using the left singular vectors, the **branch error** becomes

$$\mathcal{E}_B = \sum_{i=1}^{m} \sigma_i^2 \underbrace{\|b_i - v_i\|_2^2}_{=: L_i}.$$

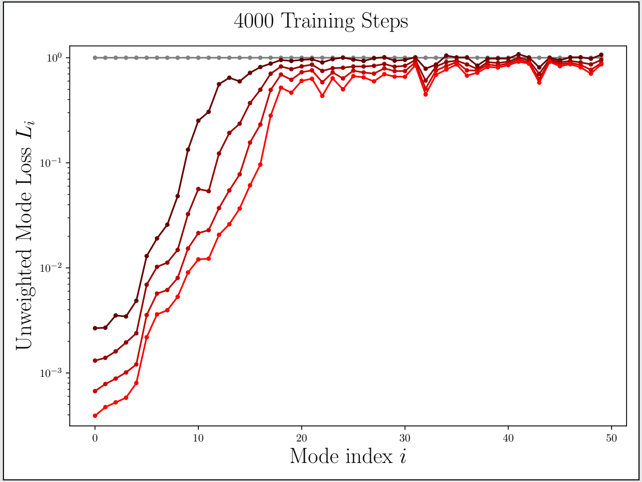
We call

$$\sigma_i^2 L_i$$

the **(weighted) mode loss** because it equals the loss contribution of the ith mode. Accordingly, L_i is the **unweighted mode loss**.

This choice of **left singular vectors as the trunk basis** is often denoted **POD-DeepONet** in **Lu et al. (2022)**.

Unweighted mode loss



DeepONet – Branch Error

Using the left singular vectors, the **branch error** becomes

$$\mathcal{E}_B = \sum_{i=1}^{m} \sigma_i^2 \underbrace{\|b_i - v_i\|_2^2}_{=:L_i}.$$

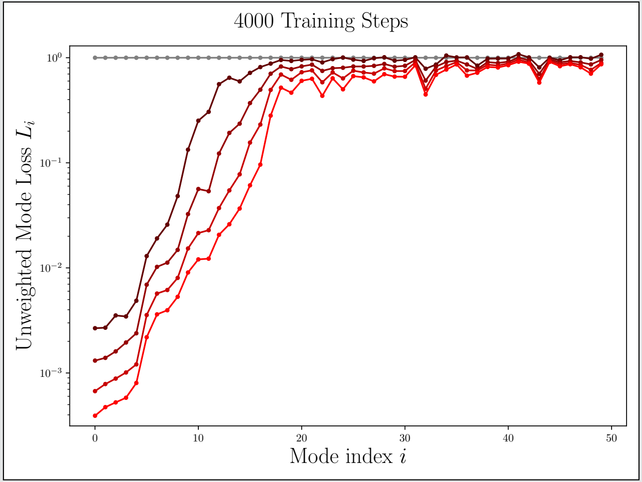
We call

$$\sigma_i^2 L_i$$

the **(weighted) mode loss** because it equals the loss contribution of the ith mode. Accordingly, L_i is the **unweighted mode loss**.

This choice of **left singular vectors as the trunk basis** is often denoted **POD-DeepONet** in **Lu et al. (2022)**.

Unweighted mode loss



→ The coefficients of modes with **large singular values** are **learned best**. Errors for modes with **small singular values remain high**.

DeepONet – Branch Error

Using the left singular vectors, the **branch error** becomes

$$\mathcal{E}_B = \sum_{i=1}^{m} \sigma_i^2 \underbrace{\|b_i - v_i\|_2^2}_{=:L_i}.$$

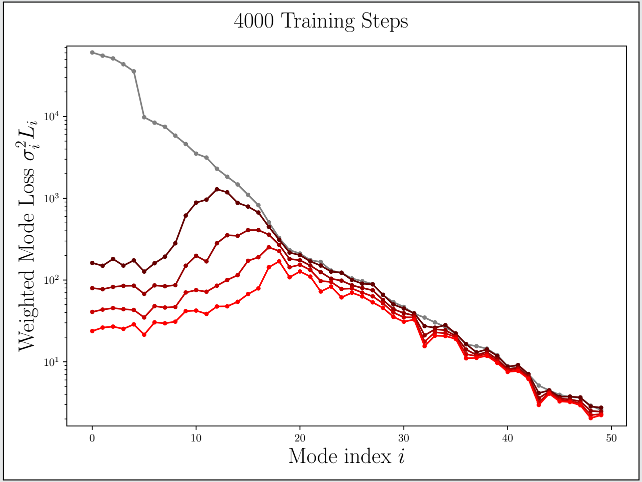
We call

$$\sigma_i^2 L_i$$

the **(weighted) mode loss** because it equals the loss contribution of the ith mode. Accordingly, L_i is the **unweighted mode loss**.

This choice of **left singular vectors as the trunk basis** is often denoted **POD-DeepONet** in **Lu et al. (2022)**.

(Weighted) mode loss



DeepONet – Branch Error

Using the left singular vectors, the **branch error** becomes

$$\mathcal{E}_B = \sum_{i=1}^{m} \sigma_i^2 \underbrace{\|b_i - v_i\|_2^2}_{=:L_i}.$$

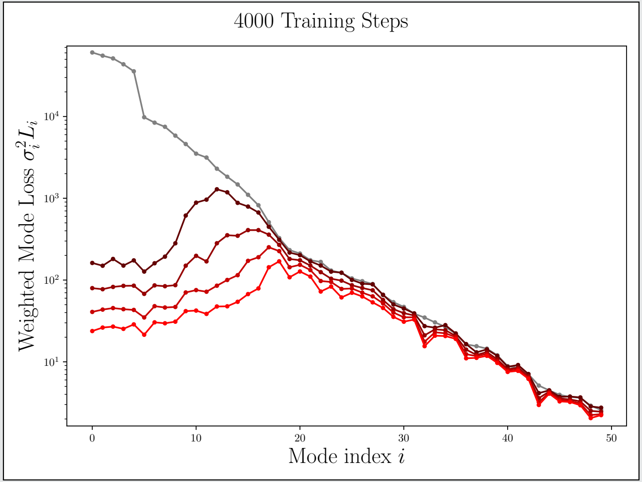
We call

$$\sigma_i^2 L_i$$

the **(weighted) mode loss** because it equals the loss contribution of the ith mode. Accordingly, L_i is the **unweighted mode loss**.

This choice of **left singular vectors as the trunk basis** is often denoted **POD-DeepONet** in **Lu et al. (2022)**.

(Weighted) mode loss



→ Analyzing the actual error contributions, the **modes with medium singular values contribute most**.

DeepONet – Branch Error

Using the left singular vectors, the **branch error** becomes

$$\mathcal{E}_B = \sum_{i=1}^{m} \sigma_i^2 \underbrace{\|b_i - v_i\|_2^2}_{=:L_i}.$$

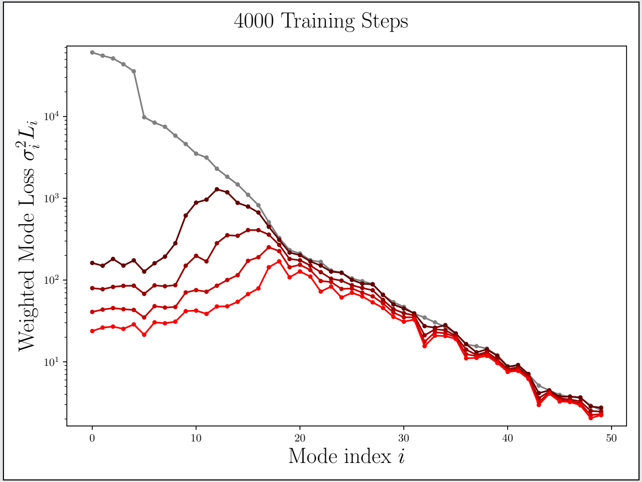
We call

$$\sigma_i^2 L_i$$

the **(weighted) mode loss** because it equals the loss contribution of the ith mode. Accordingly, L_i is the **unweighted mode loss**.

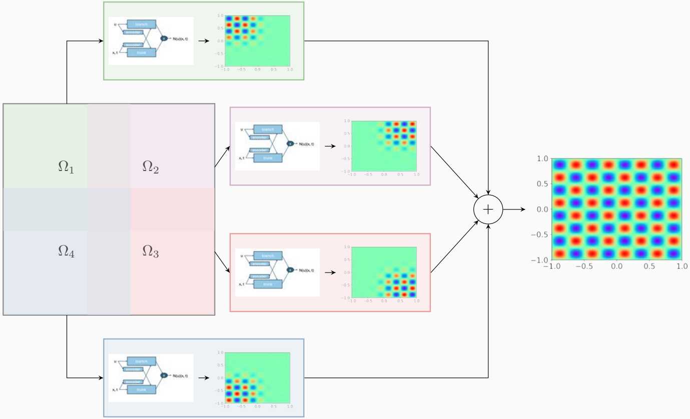
This choice of **left singular vectors as the trunk basis** is often denoted **POD-DeepONet** in **Lu et al. (2022)**.

(Weighted) mode loss



How to improve the performance on medium-sized singular value modes?

Finite Basis DeepONets (FBDONs)



Howard, Heinlein, Stinis (in prep.)

Variants:

Shared-trunk FBDONs (ST-FBDONs)

The trunk net learns spatio-temporal basis functions. In ST-FBDONs, we use the **same trunk network for all subdomains**.

Stacking FBDONs

Combination of the **stacking multifidelity approach** with FBDONs.

Heinlein, Howard, Beecroft, Stinis (2025)

FBDONs – Wave Equation

Wave equation

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

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

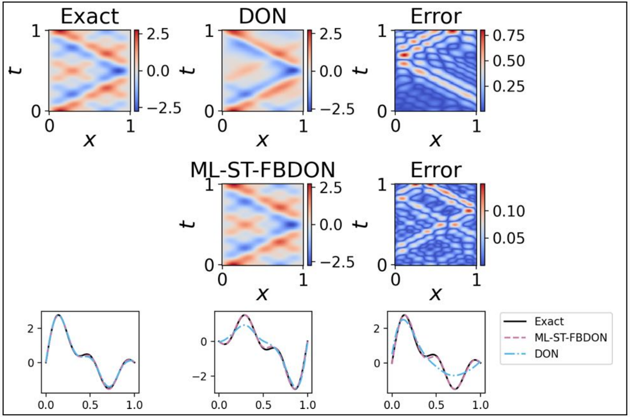
Solution: $s(x,t) = \sum_{n=1}^{5} b_n \sin(n\pi x)\cos\left(n\pi\sqrt{2}t\right)$

Parametrization

Initial conditions for s parametrized by $b = (b_1, \ldots, b_5)$ (normally distributed):

$$s(x,0) = \sum_{n=1}^{5} b_n \sin(n\pi x) \quad x \in [0,1]$$

Training on 1 000 random configurations.



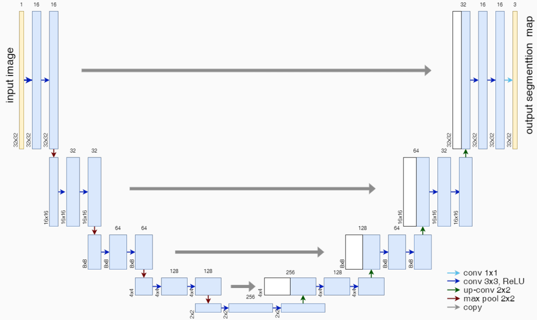
Mean rel. l_2 error on 100 config.	
DeepONet	0.30 ± 0.11
ML-ST-FBDON ([1, 4, 8, 16] subd.)	0.05 ± 0.03
ML-FBDON ([1, 4, 8, 16] subd.)	0.08 ± 0.04

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

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

Domain decomposition-based image segmentation for high-resolution image segmentation on multiple GPUs

Memory Requirements for CNN Training

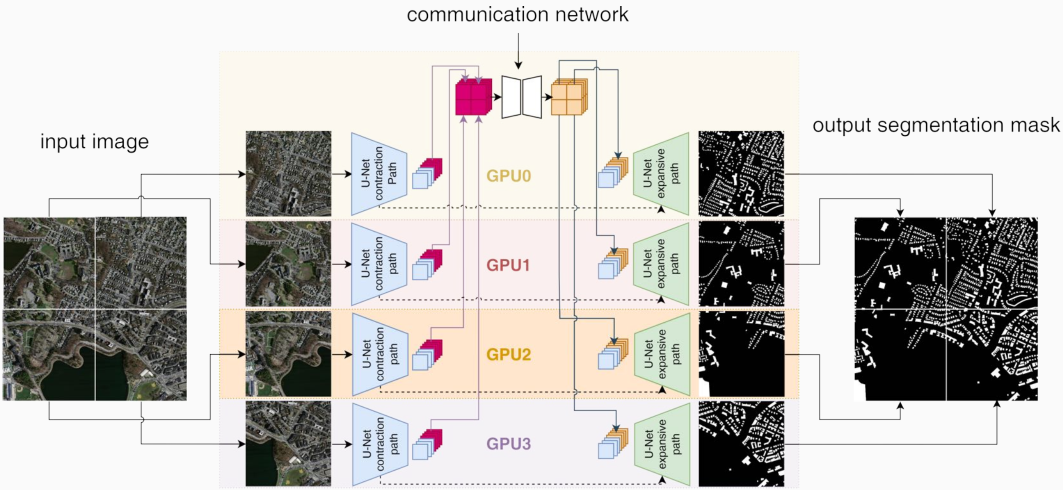


- As an example for a **convolutional neural network (CNN)**, we employ the **U-Net architecture** introduced in **Ronneberger, Fischer, and Brox (2015)**.
- The U-Net yields **state-of-the-art accuracy in semantic image segmentation** and other **image-to-image tasks**.

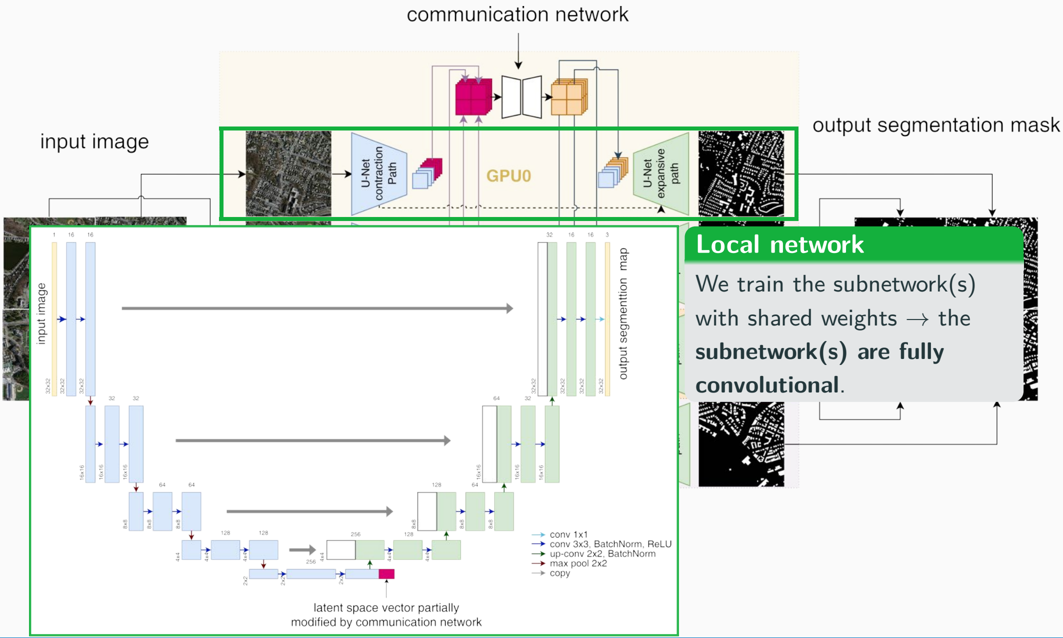
Below: *memory consumption for training on a single* 1024 × 1024 *image.*

name	size	# channels 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

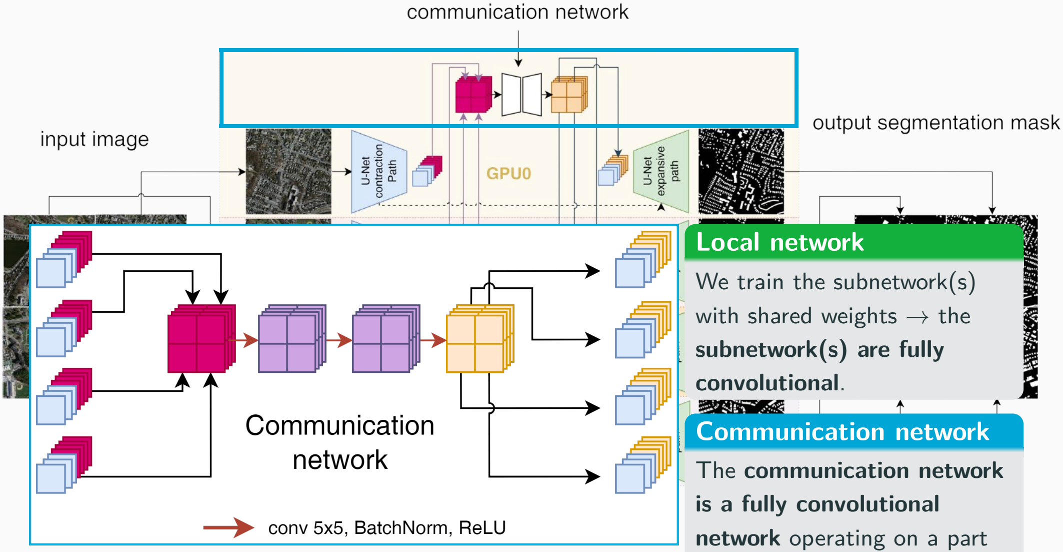


Cf. **Verburg, Heinlein, Cyr (2025).**



Local network

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



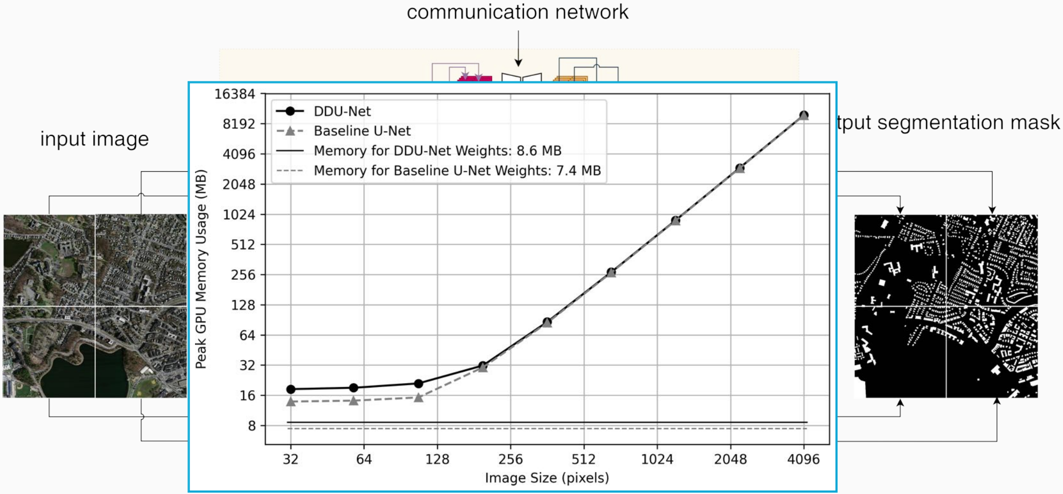
Local network

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.

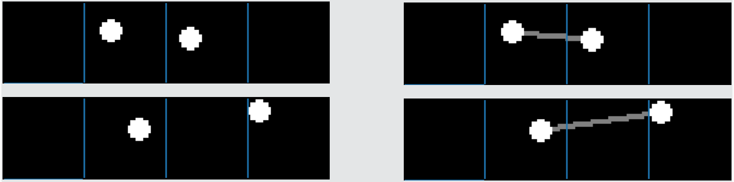
Decomposing the U-Net



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

Task: Connect two dots via a line segment

Input | Target (segmentation mask)

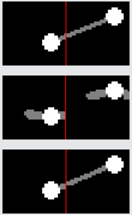


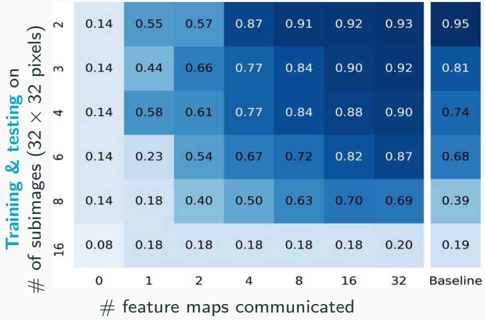
Result: Communication

True mask

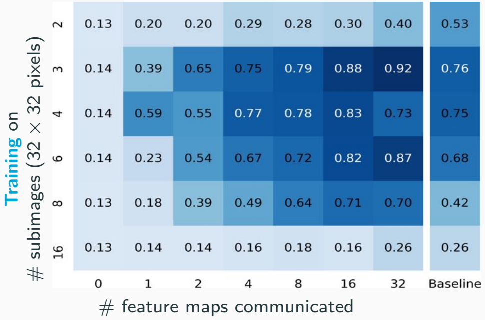
Pred. (no comm.)

Pred. (comm.)



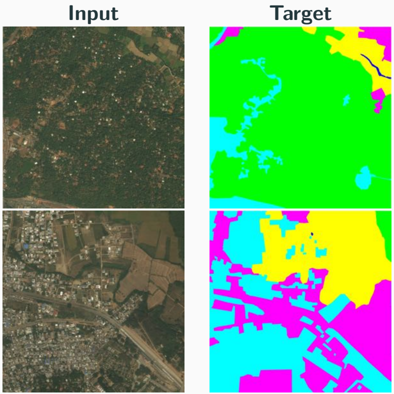


Testing on 6 subimages



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

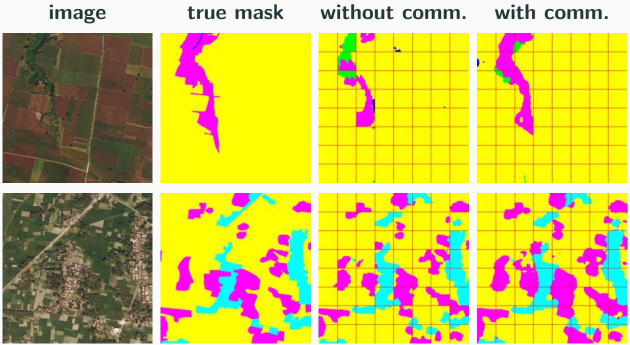
class	pixel count	proportion
urban	642.4M	9.35 %
agriculture	3898.0M	56.76 %
rangeland	701.1M	10.21 %
forest	944.4M	13.75 %
water	256.9M	3.74 %
barren	421.8M	6.14 %
unknown	3.0M	0.04 %



Avoiding overfitting

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

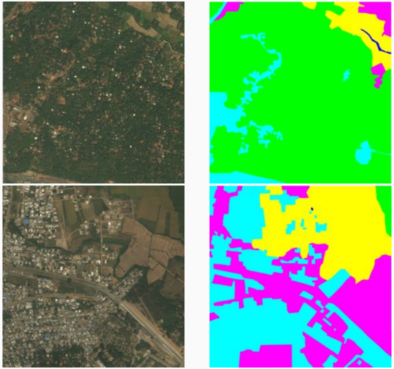
- apply **batch normalization**, use **random dropout** layers and **data augmentation**, and
- **initialize the encoder** using the **ResNet-18** (**He, Zhang, Ren, and Sun (2016)**)



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 %

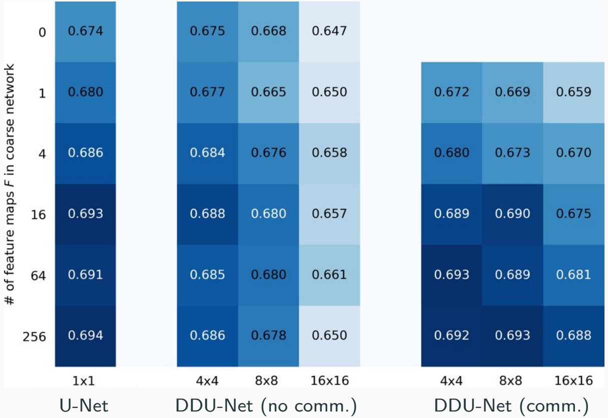
Input **Target**



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)**)



CWI Research Semester Programme:

Bridging Numerical Analysis and Scientific Machine Learning: Advances and Applications

Co-organizers: Victorita Dolean (TU/e), Alexander Heinlein (TU Delft), Benjamin Sanderse (CWI), Jemima Tabbeart (TU/e), Tristan van Leeuwen (CWI)

- **Autumn School** (October 27–31, 2025):
  - Chris Budd (University of Bath)
  - Ben Moseley (Imperial College London)
  - Gabriele Steidl (Technische Universität Berlin)
  - Andrew Stuart (California Institute of Technology)
  - Andrea Walther (Humboldt-Universität zu Berlin)
  - Ricardo Baptista (University of Toronto)
- **Workshop** (December 1–3, 2025):
  - Plenary talks (academia & industry) and panel discussion
  - **Poster session with prize sponsored by Math4NL**
  - Plenary speakers:
    - Benjamin Peherstorfer (NYU)
    - Elena Celledoni (NTNU)
    - Jakob Sauer Jørgensen (DTU)
    - Marcelo Pereyra (Heriot-Watt University)
    - Nicolas Boullé (ICL)

Centrum Wiskunde & Informatica

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

Summary

Domain Decomposition-Based Neural Network Architectures

- **Localization via domain decomposition** lets subnetworks learn **local features**, mitigates the **spectral bias** in PDE surrogates, and **captures sharp variations** more accurately.
- Partitioning the domain yields **parallelism**, promotes **sparsity**, and keeps the architecture **computationally efficient**.

Domain Decomposition Preconditioning

- The DD architecture behaves like a **discretization**, so we precondition the **least-squares** system with well-known algorithms.
- **One-level DD preconditioners** reduces **large eigenvalues**, while **SVD-based reduction removes near-linear dependencies**.

Thank you for your attention!

Topical Activity Group

Scientific Machine Learning