Scientific Machine Learning

Alexander Heinlein[1]

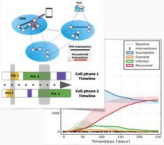
VORtech Lunch Lecture, Delft, September 29, 2025

[1]Delft University of Technology

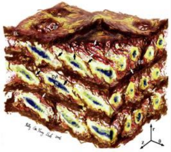
Artificial Intelligence in Science and Engineering

Medical Imaging

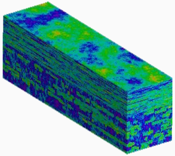
Epidemiology



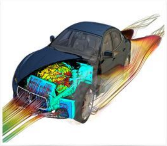
Biomedical Engineering



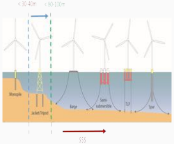
Geoscience



Fluid Mechanics



Civil Engineering



Unsupervised Learning
Use weak inductive biases to uncover structure from data

Inverse Problems
Use strong inductive biases to infer variables from data

Inference
Predict complex, nonlinear relations from data

Challenges

- Data are often **scarce** and **noisy**, yet we must deliver **accuracy**, **reliability**, and **robustness**
- **Multi-physics**, **multi-scale** systems require robust coupling, **scalability**, and **scale adaptation** of learning algorithms

Opportunities

- Integrating **physics** with **data-driven models** enhances **generalization**, **interpretability**, and **reliability**
- Encoding **physical laws** enables **trustworthy predictions** and efficient surrogates for **real-time predictions and control** (digital twins)

Scientific Machine Learning as a Standalone Field

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

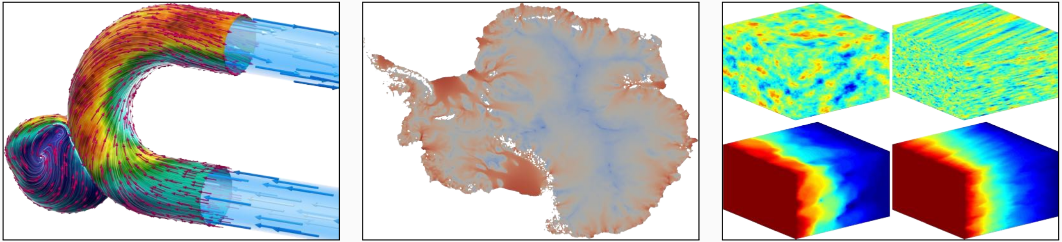
Foundational research themes:

- 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

Scientific Computing and Machine Learning



Numerical methods

Based on physical models

- \+ Robust and generalizable
- − Require availability of mathematical models

Machine learning models

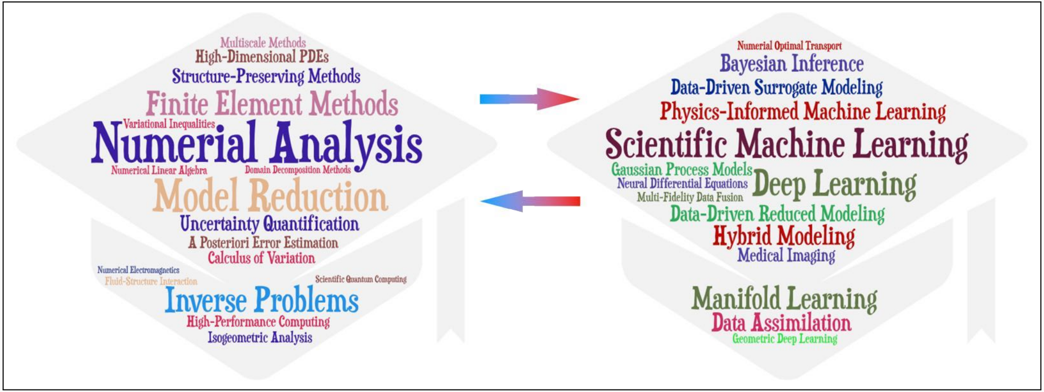
Driven by data

- \+ 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


Multiscale Methods
High-Dimensional PDEs
Structure-Preserving Methods
Finite Element Methods
Variational Inequalities
Numerial Analysis
Numerical Linear Algebra
Domain Decomposition Methods
Model Reduction
Uncertainty Quantification
A Posteriori Error Estimation
Calculus of Variation
Numerical Electromagnetics
Fluid-Structure Interaction
Scientific Quantum Computing
Inverse Problems
High-Performance Computing
Isogeometric Analysis
Numerial Optimal Transport
Bayesian Inference
Data-Driven Surrogate Modeling
Physics-Informed Machine Learning
Scientific Machine Learning
Gaussian Process Models
Neural Differential Equations
Multi-Fidelity Data Fusion
Deep Learning
Data-Driven Reduced Modeling
Hybrid Modeling
Medical Imaging
Manifold Learning
Data Assimilation
Geometric Deep Learning

Scientific Machine Learning Techniques

NA-enhanced ML

Examples:

- *physics-informed machine learning*
- *preconditioned optimizers* and *gradient flow*
- *domain-decomposed and multiscale neural networks*

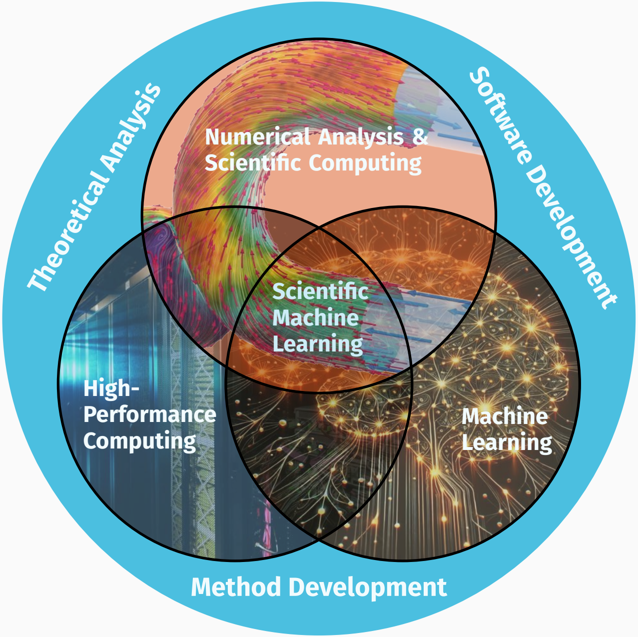
Numerical Analysis (NA) → **Hybrid Models** ← **Machine Learning (ML)**

ML-enhanced NA

Examples:

- *learned discretizations and iterative solvers*
- *inverse problems,*
- *reduced order and surrogate models*

SCaLA – Scalable Scientific Computing and Learning Algorithms



Outline

1. Physics-informed neural networks – Adaptive sampling and localization via domain decompostion

Based on joint work with

Victorita Dolean	(Eindhoven University of Technology)
Bianca Giovanardi, **Coen Visser**	(Delft University of Technology)
Amanda A. Howard and **Panos Stinis**	(Pacific Northwest National Laboratory)
Siddhartha Mishra	(ETH Zürich)
Ben Moseley	(Imperial College London)

2. Deep operator networks – Error analysis and localization via domain decompostion

Based on joint work with

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

3. Surrogate models for varying computational domains

Based on joint work with

Eric Cyr	(Sandia National Laboratories)
Matthias Eichinger, **Viktor Grimm**, **Axel Klawonn**	(University of Cologne)
Corné Verburg	(Delft University of Technology)

Physics-informed neural networks – Adaptive sampling and localization via domain decompostion

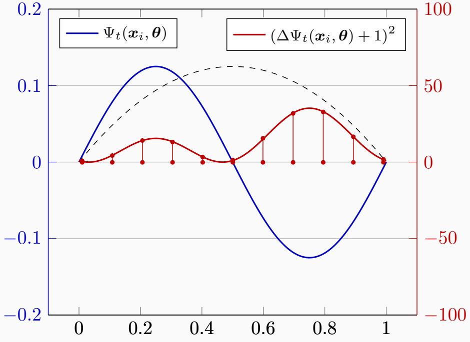
Physics-Informed Neural Networks (PINNs) – Idea

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**:

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



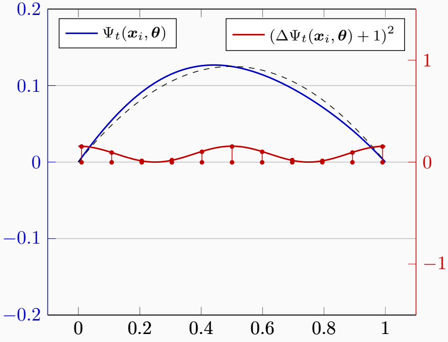
$(\Delta\Psi_t(\boldsymbol{x}_i,\boldsymbol{\theta}) + 1)^2 >> 0$

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**:

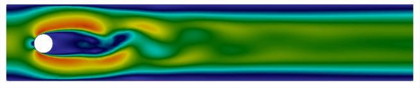
$$\mathcal{L}(\boldsymbol{\theta}) = \omega_{\text{data}} \mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}),$$

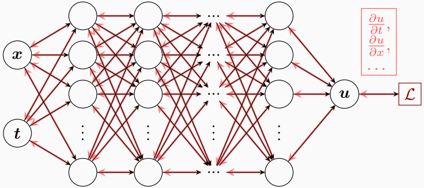
where ω_{data} and ω_{PDE} are **weights** and

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

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

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





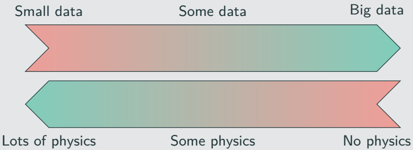
Advantages

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

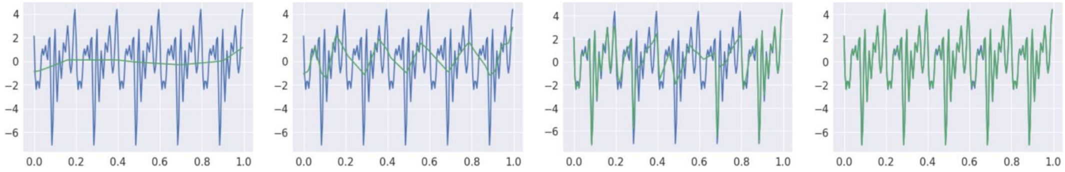
Estimate of the generalization error (Mishra and Molinaro (2022))

The generalization error (or total error) satisfies

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

- $\mathcal{E}_G = \mathcal{E}_G(\boldsymbol{X}, \theta) := \|\mathbf{u} - \mathbf{u}^*\|_V$ **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)**

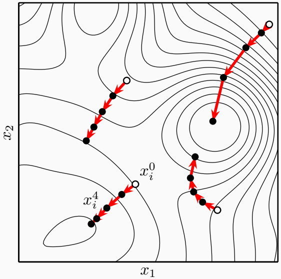
Related works: **Cao et al. (2021), Wang, et al. (2022), Hong et al. (arXiv 2022), Xu et al. (2024), . . .**

PACMANN – Point Adaptive Collocation Method for Artificial Neural Networks

In **Visser, Heinlein, and Giovanardi (subm. 2025; arXiv:2411.19632)**, the collocation points are updated by solving the **min-max problem**

$$\min_{\boldsymbol{\theta}} \left[\omega_{\text{data}} \mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \max_{\boldsymbol{X} \subset \Omega} \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}}(\boldsymbol{X}, \boldsymbol{\theta}) \right].$$

This idea was already mentioned in **Wang et al. (2022)**. Different from other residual-based adaptive sampling methods, **existing collocation points are moved** using gradient-based optimizers such as **gradient ascent**, **RMSprop** (**Hinton (2018)**), or **Adam** (**Kingma, Ba (2017)**).



Algorithm 1: PACMANN with iteration counts P and T and stepsize s

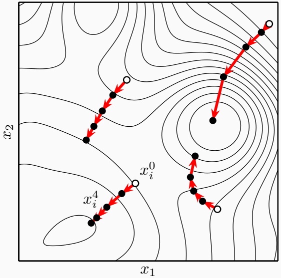
Sample a set $\boldsymbol{X}$ of N_{PDE} collocation points using a uniform sampling method;
while *stopping criterion not reached* **do**
  Train the PINN for P iterations;
  for $k = 1, \ldots, T$ **do**
    Compute squared residual $\mathcal{R}(\boldsymbol{x}_i) = (\mathcal{N}[u](\boldsymbol{x}_i, \boldsymbol{\theta}) - f(\boldsymbol{x}_i))^2$ for all $\boldsymbol{x}_i \in \boldsymbol{X}$;
    Compute gradient $\nabla_{\boldsymbol{x}} \mathcal{R}(\boldsymbol{x}_i)$ for all $\boldsymbol{x}_i \in \boldsymbol{X}$;
    Move the points in $\boldsymbol{X}$ according to the chosen optimization algorithm and stepsize s;
  end
  Resample points in $\boldsymbol{X}$ that moved outside Ω based on a uniform probability distribution;
end

PACMANN – Point Adaptive Collocation Method for Artificial Neural Networks

In **Visser, Heinlein, and Giovanardi (subm. 2025; arXiv:2411.19632)**, the collocation points are updated by solving the **min-max problem**

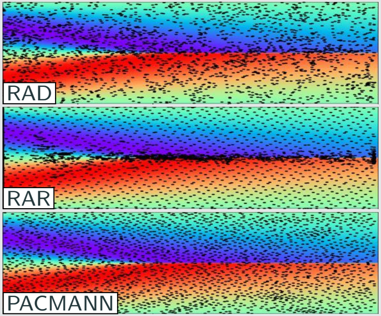
$$\min_{\boldsymbol{\theta}} \left[\omega_{\text{data}} \mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \max_{\boldsymbol{X} \subset \Omega} \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}}(\boldsymbol{X}, \boldsymbol{\theta}) \right].$$

This idea was already mentioned in **Wang et al. (2022)**. Different from other residual-based adaptive sampling methods, **existing collocation points are moved** using gradient-based optimizers such as **gradient ascent**, **RMSprop** (**Hinton (2018)**), or **Adam** (**Kingma, Ba (2017)**).



Comparison against different methods

sampling method *2500 coll. points*	L_2 relative error mean	 1 SD	mean runtime [s]
uniform grid	25.9%	14.2%	425
Hammersley grid	0.61%	0.53%	443
random resampling	0.40%	0.35%	**423**
RAR	0.11%	**0.05%**	450
RAD	0.16%	0.10%	463
RAR-D	0.24%	0.21%	503
PACMANN–Adam	**0.07%**	**0.05%**	461

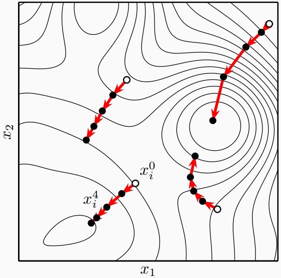


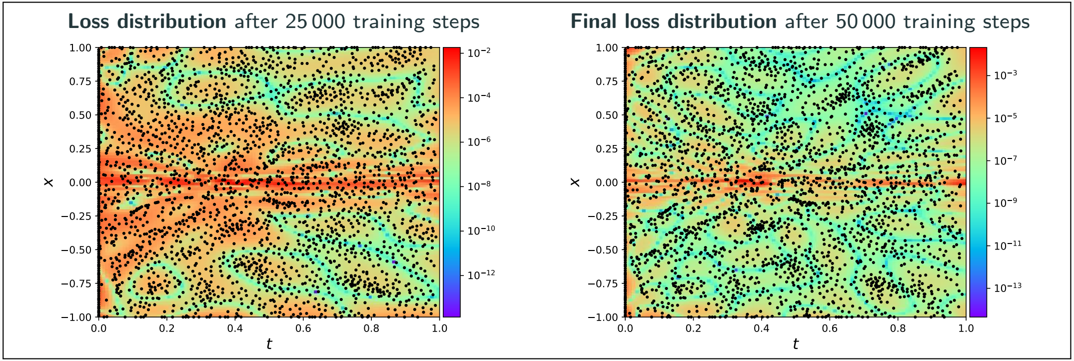
PACMANN – Point Adaptive Collocation Method for Artificial Neural Networks

In **Visser, Heinlein, and Giovanardi (subm. 2025; arXiv:2411.19632)**, the collocation points are updated by solving the **min-max problem**

$$\min_{\boldsymbol{\theta}} \left[\omega_{\text{data}} \mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \max_{\boldsymbol{X} \subset \Omega} \omega_{\text{PDE}} \mathcal{L}_{\text{PDE}}(\boldsymbol{X}, \boldsymbol{\theta}) \right].$$

This idea was already mentioned in **Wang et al. (2022)**. Different from other residual-based adaptive sampling methods, **existing collocation points are moved** using gradient-based optimizers such as **gradient ascent**, **RMSprop** (**Hinton (2018)**), or **Adam** (**Kingma, Ba (2017)**).





Solve

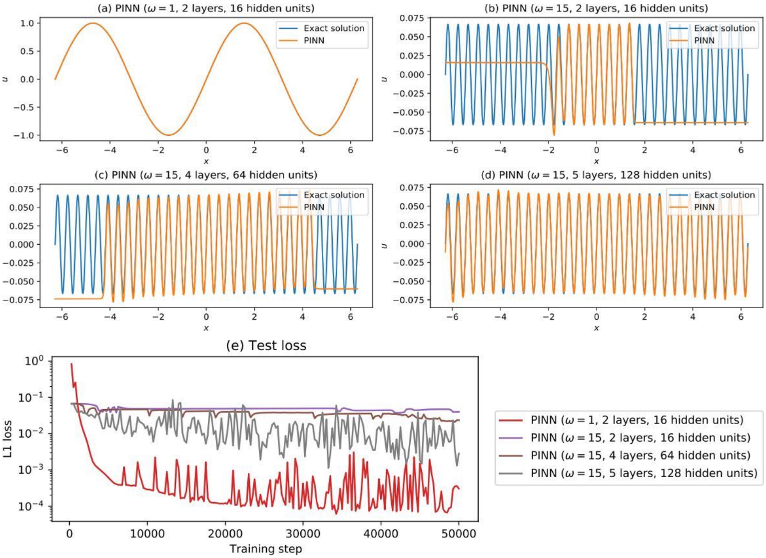
$$\begin{aligned} u' &= \cos(\omega \boldsymbol{x}), \\ u(0) &= 0, \end{aligned}$$

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

Scaling of PINNs for a Simple ODE Problem

Solve

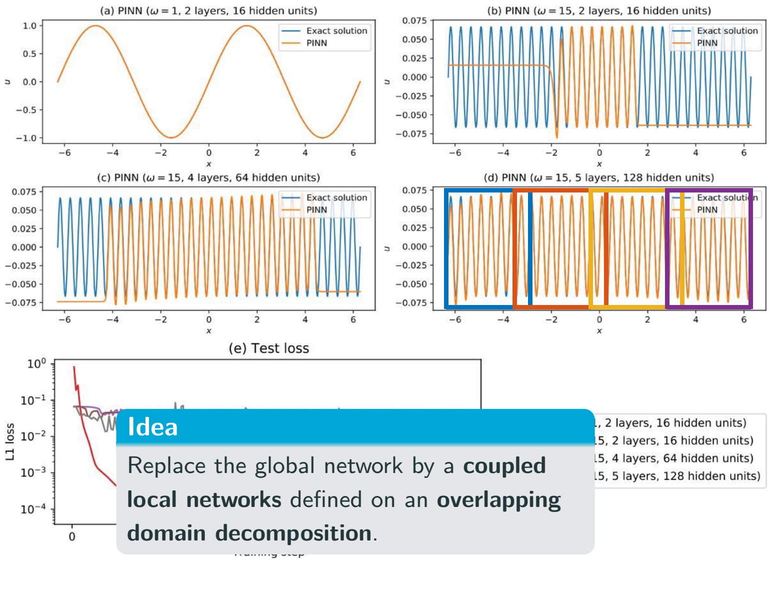
$$
\begin{aligned}
u' &= \cos(\omega \boldsymbol{x}), \\
u(0) &= 0,
\end{aligned}
$$

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)**

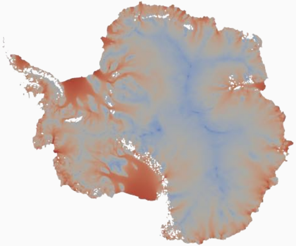


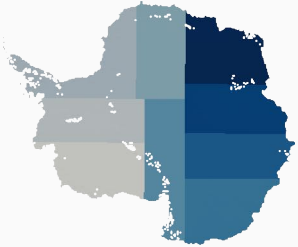
Idea

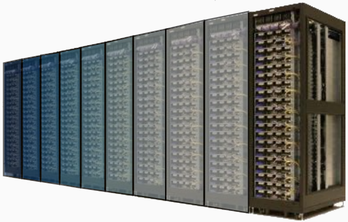
Replace the global network by a **coupled local networks** defined on an **overlapping domain decomposition**.

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

Domain Decomposition Methods



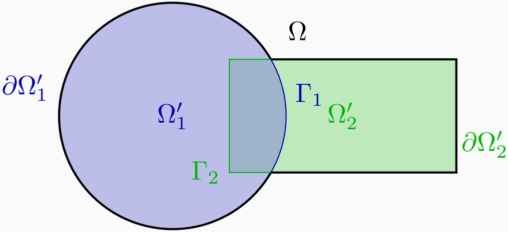




Graphics based on results from **Heinlein, Perego, Rajamanickam (2022)**

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**.



Finite Basis Physics-Informed Neural Networks (FBPINNs)

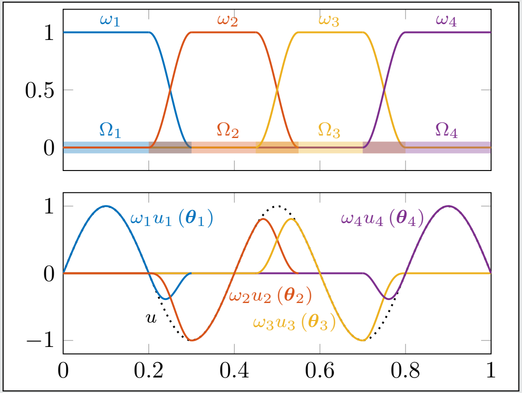
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

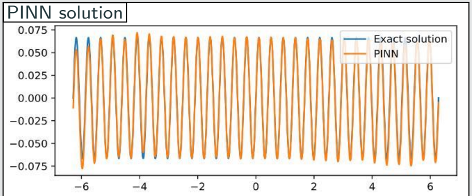
$$u(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \sum\nolimits_{j=1}^{J} \omega_j u_j(\boldsymbol{\theta}_j)$$

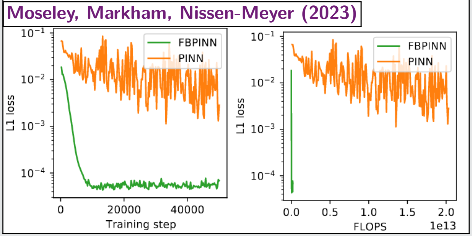
and the **loss function**

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



1D single-frequency problem





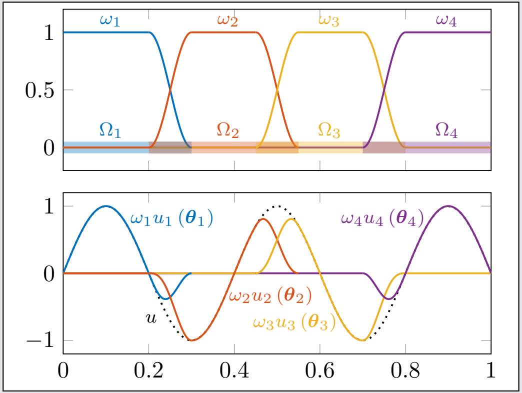
FBPINNs (Moseley, Markham, Nissen-Meyer (2023))

FBPINNs employ the **network architecture**

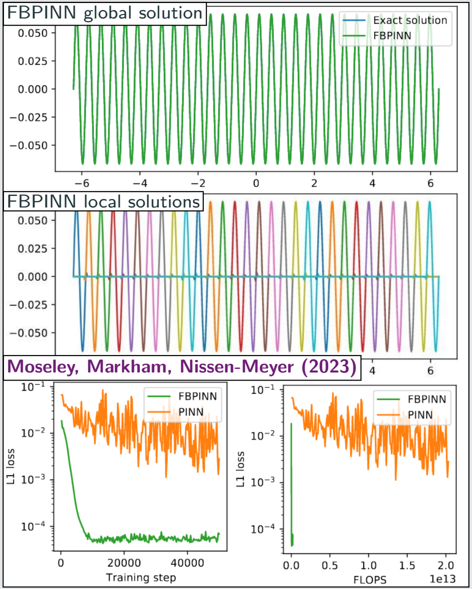
$$u(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \sum\nolimits_{j=1}^{J} \omega_j u_j\left(\boldsymbol{\theta}_j\right)$$

and the **loss function**

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



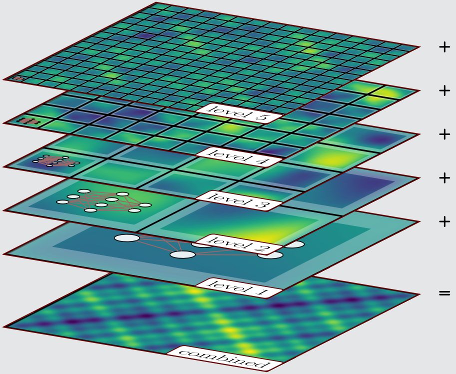
1D single-frequency problem



Multi-Level FBPINNs

Multi-level FBPINNs (ML-FBPINNs)

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

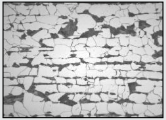


This yields the **network architecture**:

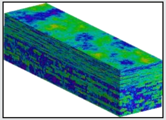
$$u\left(\theta_1^{(1)}, \ldots, \theta_{J^{(L)}}^{(L)}\right) = \sum_{l=1}^{L} \sum_{i=1}^{N^{(l)}} \omega_j^{(l)} u_j^{(l)}\left(\theta_j^{(l)}\right)$$

Multiscale problems . . .

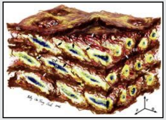
. . . appear in most areas of modern science and engineering:



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**

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

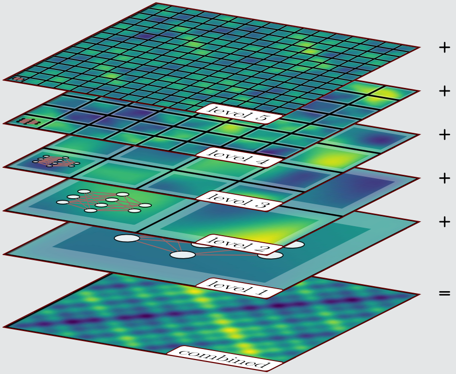
with homogeneous Dirichlet boundary conditions and $\omega_i = 2^i$.

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:**

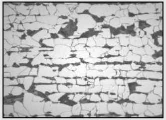


This yields the **network architecture**:

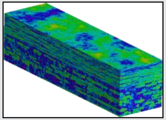
$$u\left(\theta_1^{(1)}, \ldots, \theta_{J^{(L)}}^{(L)}\right) = \sum\nolimits_{l=1}^{L} \sum\nolimits_{i=1}^{N^{(l)}} \omega_j^{(l)} u_j^{(l)}\left(\theta_j^{(l)}\right)$$

Multiscale problems ...

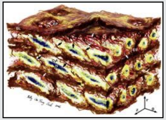
...appear in most areas of modern science and engineering:



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



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



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

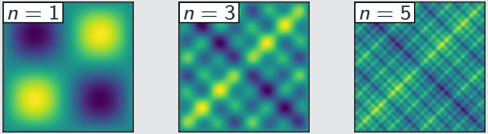
Multi-frequency problem

Consider the **multi-frequency Laplace problem**

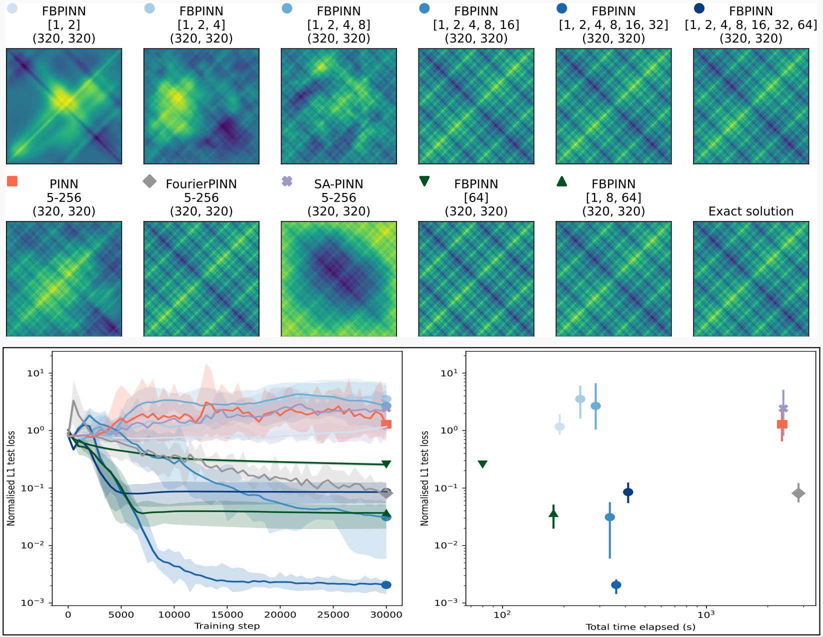
$$-\Delta u = 2 \sum\nolimits_{i=1}^{n} (\omega_i \pi)^2 \sin(\omega_i \pi x) \sin(\omega_i \pi y),$$

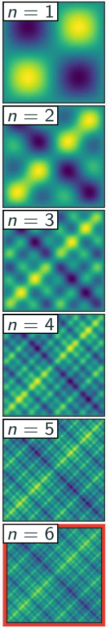
with homogeneous Dirichlet boundary conditions and $\omega_i = 2^i$.

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

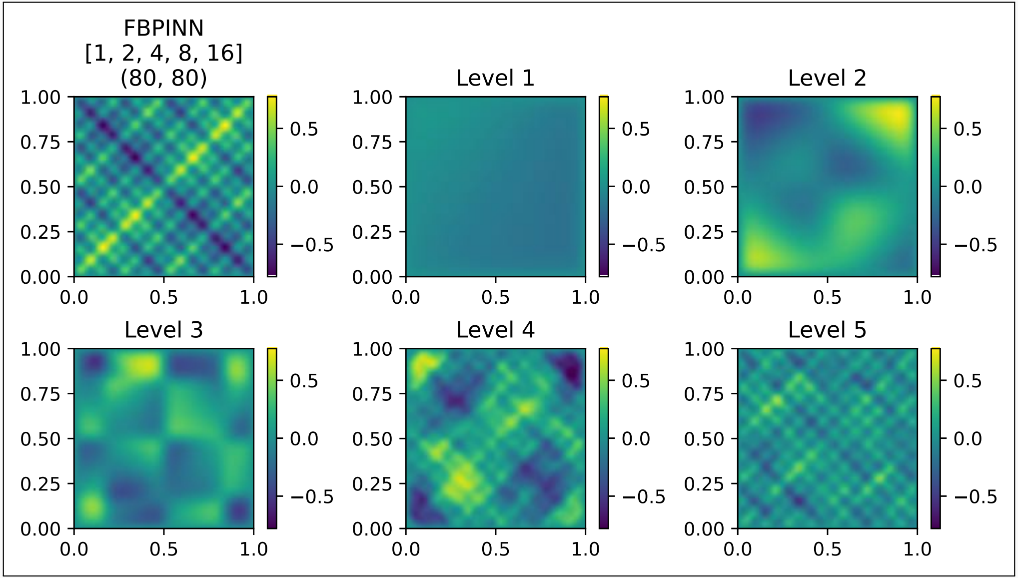


Multi-Level FBPINNs for a Multi-Frequency Problem – Strong Scaling





Multi-Frequency Problem – What the FBPINN Learns

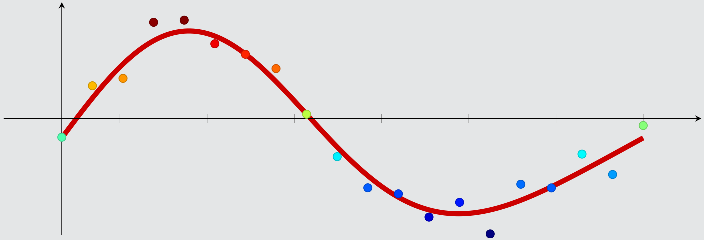


Cf. **Dolean, Heinlein, Mishra, Moseley (2024)**.

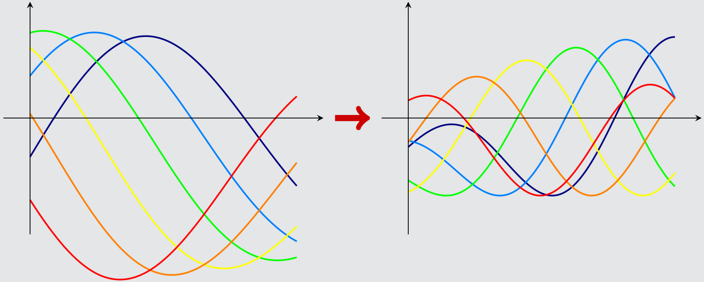
Deep operator networks – Error analysis and localization via domain decompostion

Function Versus Operator Learning

Function learning

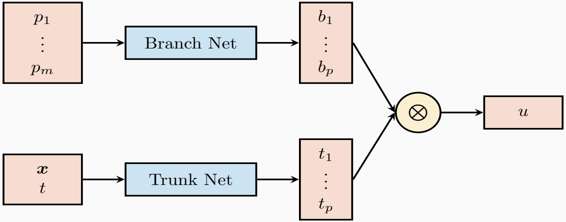


Operator learning



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

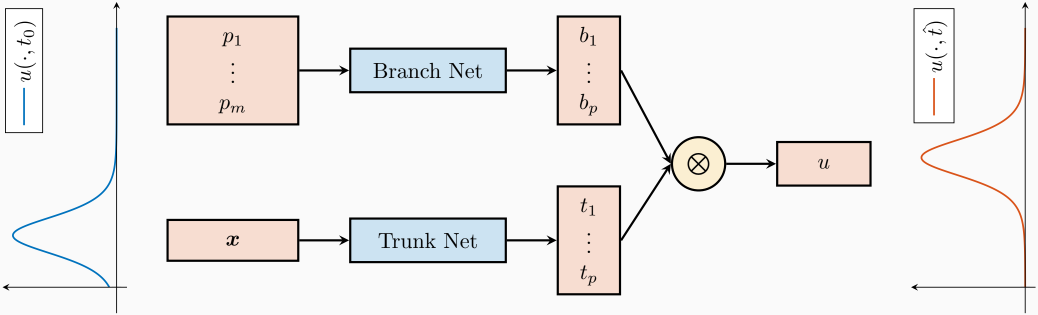
- **FNOs**: **Li et al. (2021)**
- **PCA-Net**: **Bhattacharya et al. (2021)**
- **Random features**: **Nelsen and Stuart (2021)**
- **CNOs**: **Raonić et al. (2023)**

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 $u(\cdot, t_0)$.



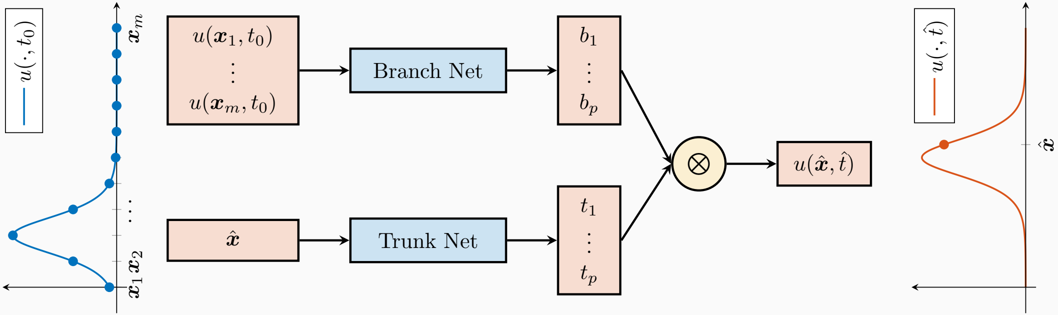
Here, the forecast time $\hat{t}$ is fixed to keep the learning task simple. A **more general neural operator** can take the target time as an additional input to the trunk network.

How a DeepONet Maps Between Function Spaces

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

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

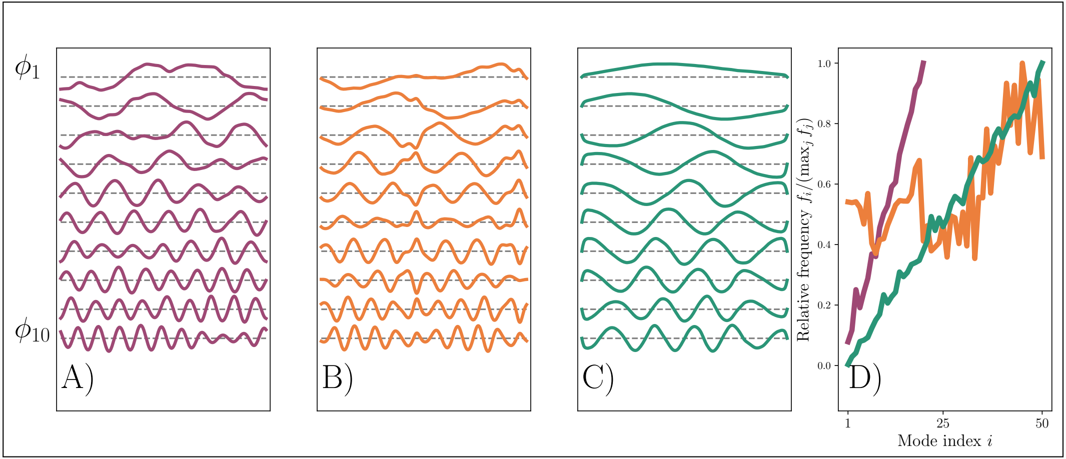
which models unidirectional waves in shallow water. Our goal is to train a DeepONet that predicts the wave profile at a future time $\hat{t}$ from the observed height profile $u(\cdot, t_0)$.



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.

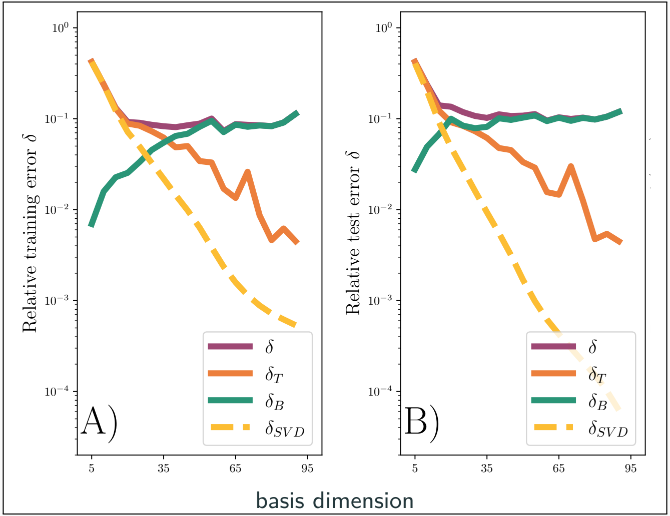
Let us consider some **examples of the left singular vectors** for **three differential equations**:

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)**.

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

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

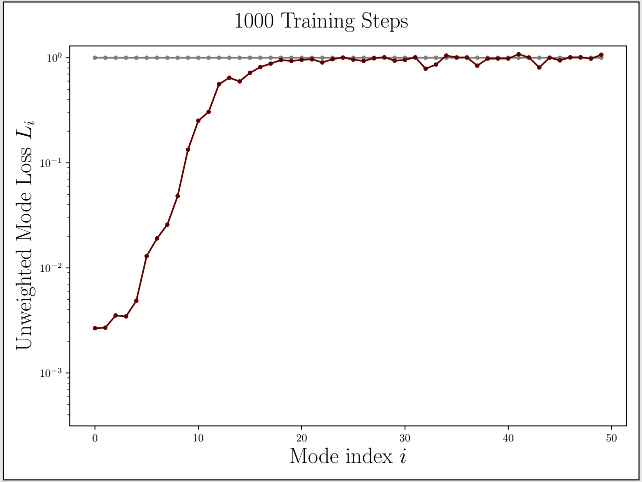
We call

$$\sigma_i^2 L_i$$

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

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

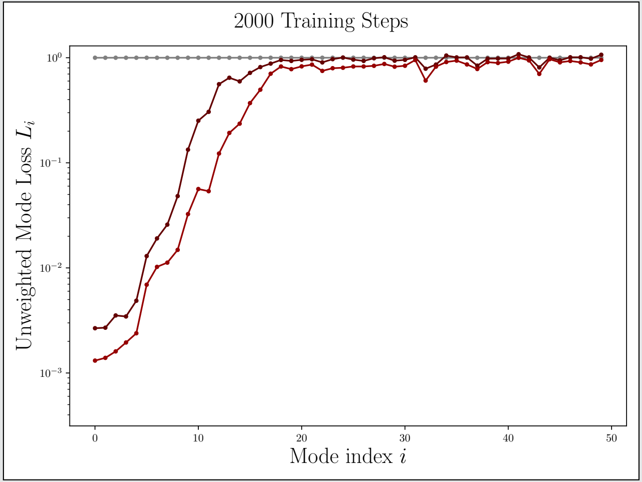
We call

$$\sigma_i^2 L_i$$

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

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

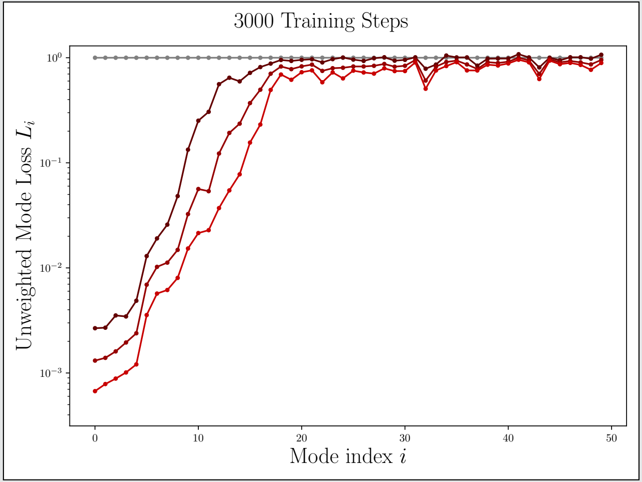
We call

$$\sigma_i^2 L_i$$

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

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

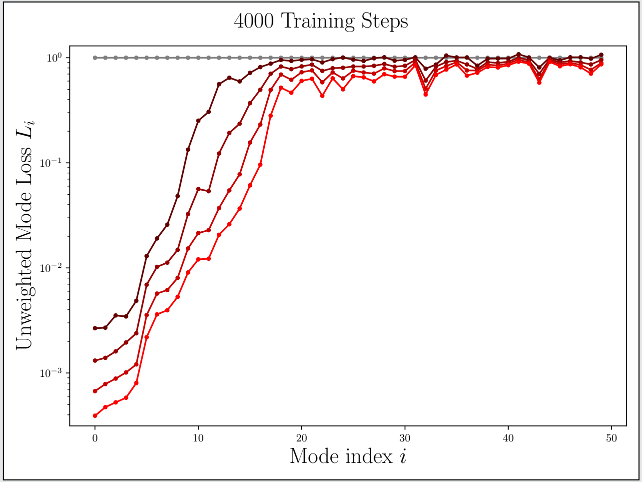
We call

$$\sigma_i^2 L_i$$

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

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

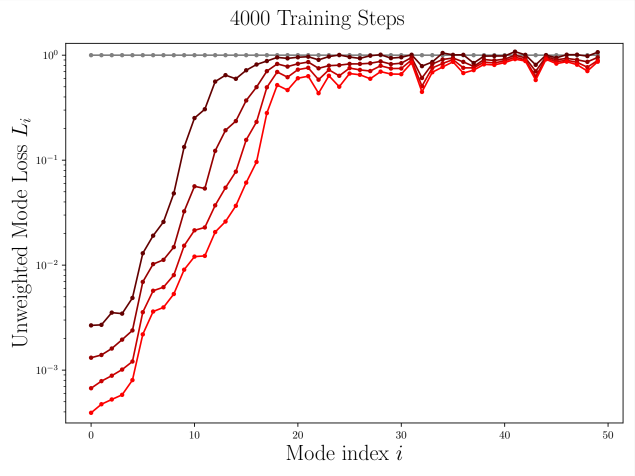
We call

$$\sigma_i^2 L_i$$

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

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

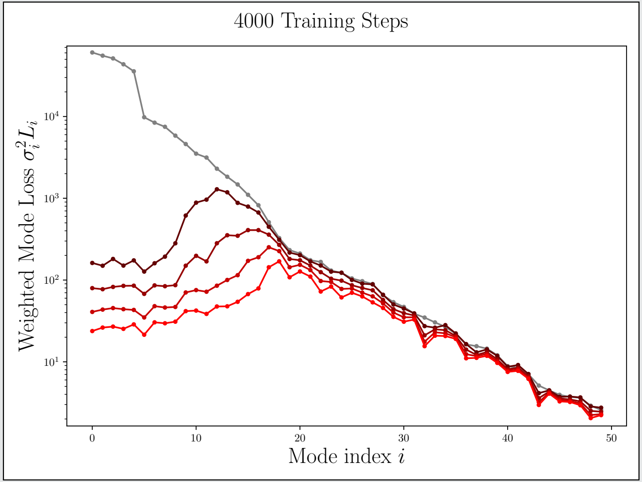
We call

$$\sigma_i^2 L_i$$

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

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

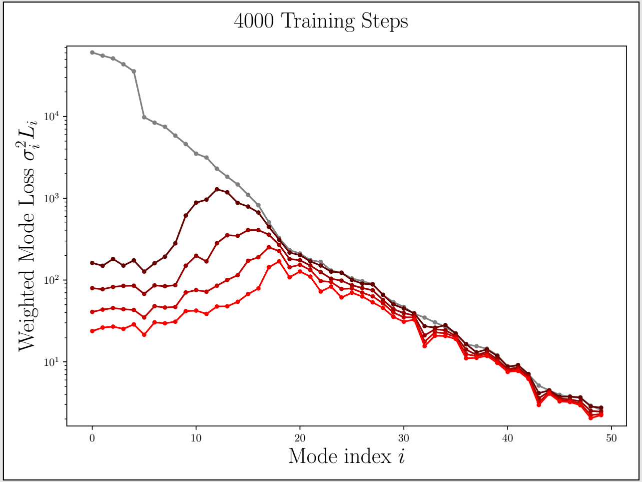
We call

$$\sigma_i^2 L_i$$

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

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

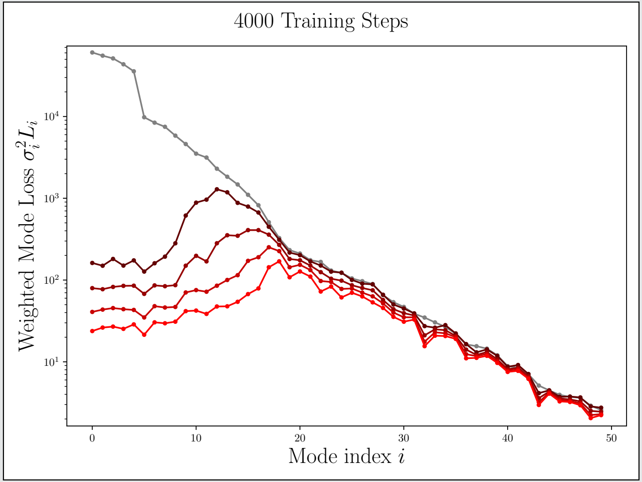
We call

$$\sigma_i^2 L_i$$

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

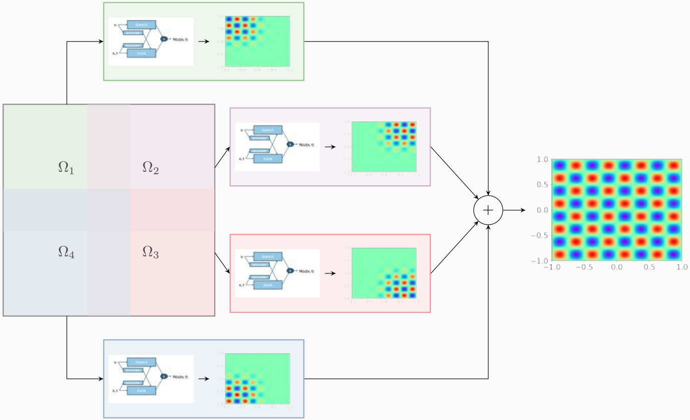
This choice of **left singular vectors as the trunk basis** is often denoted **POD-DeepONet** in **Lu et al. (2022)**.

(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

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

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

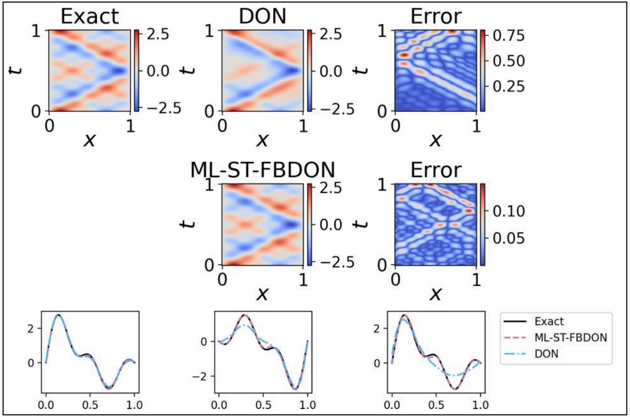
Solution: $s(x,t) = \sum_{n=1}^{5} b_n \sin(n\pi x)\cos\left(n\pi\sqrt{2}t\right)$

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

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

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

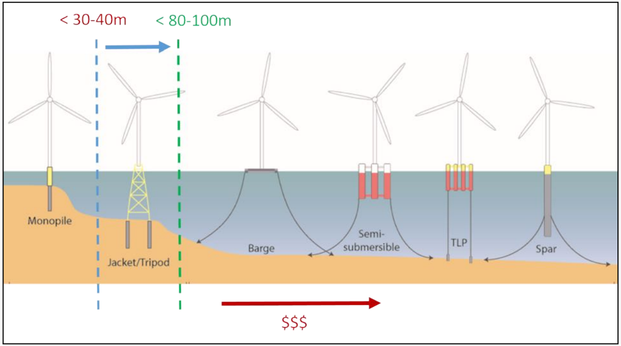
Surrogate models for varying computational domains

Designing of Perforated Monopiles for Offshore Wind Energy

Perforated monopiles

Monopiles are the **most used and cheapest** solution of support structures in **offshore wind energy**.
→ **Perforated monopiles reduce the wave load**.

What is the **optimal perforation shape**?

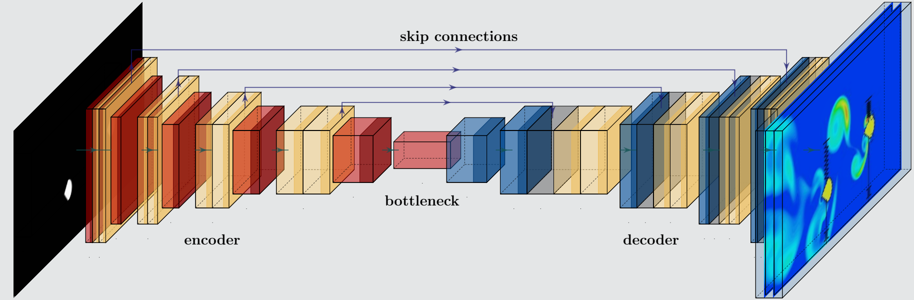


Fully resolved CFD simulations are **costly**, but **rough predictions** may be **sufficient**.

Convolutional Neural Network-Based Surrogate Model

CNN-based approach

We employ a convolutional neural network (CNN) (**LeCun (1998)**) to predict the stationary flow field, given an **image of the geometry as input**.



U-Net architecture; cf. **Ronneberger, Fischer, Brox (2015)**

Related works (non-exhaustive)

- **Guo, Li, Iorio (2016)**
- **Niekamp, Niemann, Schröder (2022)**
- **Stender, Ohlsen, Geisler, Chabchoub, Hoffmann, Schlaefer (2022)**

Operator learning (non-exhaustive)

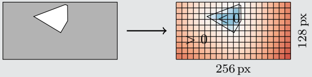
- FNOs: **Li et al. (2021)**
- PCA-Net: **Bhattacharya et al. (2021)**
- Random features: **Nelsen and Stuart (2021)**
- CNOs: **Raonić et al. (2023)**

Comparison OpenFOAM® Versus CNN (Relative Error 2 %)

We automatically generate geometries and compute the corresponding flow fields using OpenFOAM®.

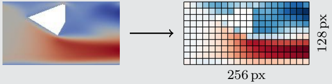
Input data

SDF (Signed Distance Function)

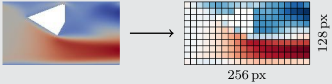


Output data

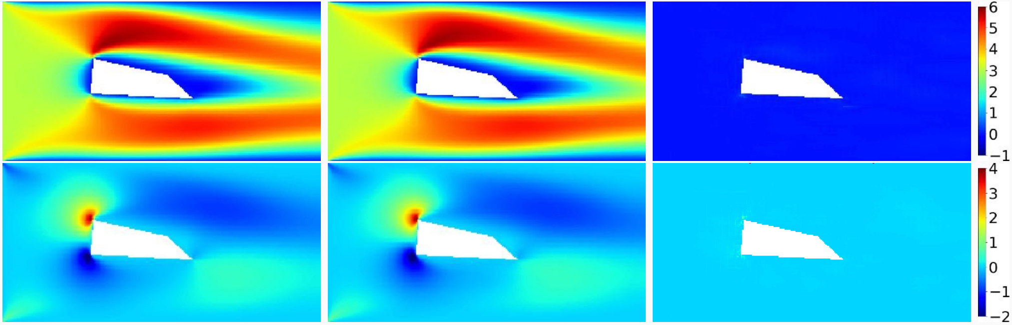
u_x



u_x



u_x CFD u_x CNN u_x ERR



u_y CFD u_y CNN u_y ERR

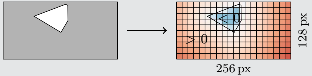
Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**.

Comparison OpenFOAM® Versus CNN (Relative Error 14 %)

We automatically generate geometries and compute the corresponding flow fields using OpenFOAM®.

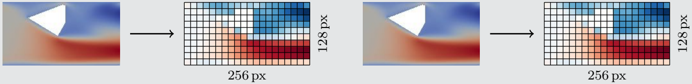
Input data

SDF (Signed Distance Function)

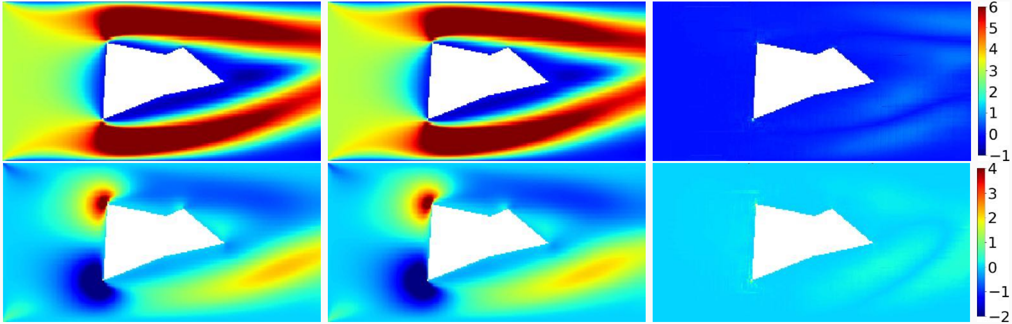


Output data

u_x u_x



u_x CFD u_x CNN u_x ERR



u_y CFD u_y CNN u_y ERR

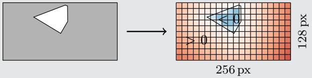
Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**.

Comparison OpenFOAM® Versus CNN (Relative Error 31 %)

We automatically generate geometries and compute the corresponding flow fields using OpenFOAM®.

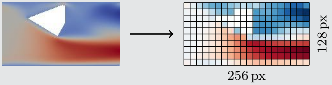
Input data

SDF (Signed Distance Function)

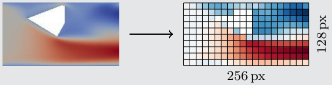


Output data

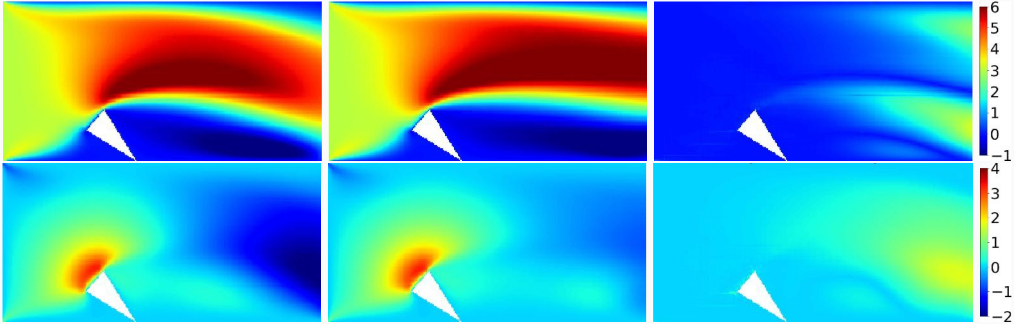
u_x



u_x



u_x CFD u_x CNN u_x ERR



u_y CFD u_y CNN u_y ERR

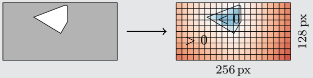
Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**.

Comparison OpenFOAM® Versus CNN

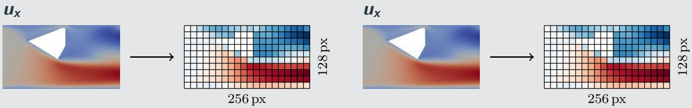
We automatically generate geometries and compute the corresponding flow fields using OpenFOAM®.

Input data

SDF (Signed Distance Function)



Output data



OpenFOAM® simulation

avg. runtime per case	
create STL	0.15 s
`snappyHexMesh`	37 s
`simpleFoam`	13 s
total time	$\approx$ 50 s

CPU: AMD Threadripper 2950X (8×3.8 Ghz), 32GB RAM
GPU: GeForce RTX 2080Ti

Cf. **Eichinger, Heinlein, Klawonn (2021, 2022)**.

U-Net

training cost		
# decoders	1	2
# parameters	$\approx$ 34 m	$\approx$ 53.5 m
time/epoch	**195 s**	**270 s**

avg. inference cost		
	CPU	GPU
avg. time	**0.092 s**	**0.0054 s**

Unsupervised Learning Approach – PDE Loss Using Finite Differences

Physics-informed loss function

We train the CNN by incorporating the **PDE residuals**, discretized via finite differences, into the **loss function**:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \|\mathcal{R}(u_{\text{CNN}}, p_{\text{CNN}})\|_2^2$$

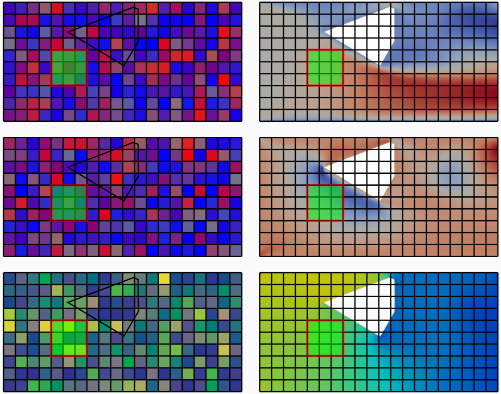
Here, N_{PDE} is the number of training configs.

Cf. **Raissi et al. (2019), Dissanayake and Phan-Thien (1994), Lagaris et al. (1998)**.

We explcitly enforce boundary conditions on the output image → **hard constraints**

error	$\frac{\|u_{NN}-u\|_2}{\|u\|_2}$	$\frac{\|p_{NN}-p\|_2}{\|p\|_2}$	mean residual moment.	mean residual mass	# epochs trained
train.	1.43 %	7.30 %	$1.0\cdot10^{-1}$	$1.5\cdot10^{0}$	500
val.	2.52 %	8.67 %	$1.2\cdot10^{-1}$	$1.5\cdot10^{0}$	
train.	5.03 %	11.63 %	$3.2\cdot10^{-2}$	$7.7\cdot10^{-2}$	**5 000**
val.	5.18 %	11.60 %	$4.2\cdot10^{-2}$	$1.1\cdot10^{-1}$	

→ Errors are **comparable to the data-based approach**, but **training requires more epochs**.



$\|\mathcal{R}(u_{\text{CNN}}, p_{\text{CNN}})\|^2 >> 0$ $\quad$ $\|\mathcal{R}(u_{\text{CNN}}, p_{\text{CNN}})\|^2 \approx 0$

Here, we consider ther **Navier–Stokes equations**:

$$\mathcal{R}(u_{\text{CNN}}, p_{\text{CNN}}) = \begin{bmatrix} -\nu\Delta\vec{u} + (u\cdot\nabla)\,\vec{u} + \nabla p \\ \nabla\cdot u \end{bmatrix}$$

Cf. **Grimm, Heinlein, Klawonn (2025)**.

A New Paradigm for Scientific Software?

Hybrid workflows determine software design

- SciML codes **combine machine learning modules** with **classical numerical algorithms** and **physics models**.
- Mainstream **ML frameworks** (e.g., PyTorch, JAX, ...) excell in **maintainance, usability, and documentation**.
- **Automatic differentiation** significantly **simplifies core kernels**, shifting major efforts towards **data processing as well as architecture and hyperparameter tuning**.

Hardware and parallelization assumptions change

- Most architectures are **not based on locality** – except for, e.g., **CNNs**, **GNNs**, or explicit **domain decomposition**. Hence, dominant kernels rely on **dense linear algebra**, favoring dense **GPU/TPU** kernels over CPU/sparse stacks.
- **Training often converges slowly** in both iterations and computing time; **training offline** is preferred.
- **Parallel scaling** is more natural through **data parallelism** than through **model decomposition**.

Robustness and reproducibility guarantees often still work-in-progress

- **Randomness in models (training)** demand infrastructure for **uncertainty quantification, ensembles, and validation**.
- **Non-convex optimization problems** make guarantees on convergence, robustness, and stability difficult. **Reproducibility** is a **major challenge**.

4TU.AMI – SRI “Bridging Numerical Analysis and Machine Learning”

UNIVERSITY OF TWENTE.


Christoph Brune


Silke Glas


Matthias Schlottbom

TU Delft Delft University of Technology


Francesca Bartolucci


Alexander Heinlein


Matthias Möller


Deepesh Toshniwal

4TU.AMI

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

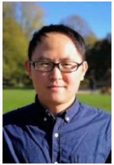

Victorita Dolean


Wil Schilders


Jemima Tabeart


Karen Veroy-Grepl

WAGENINGEN UNIVERSITY & RESEARCH


Xiaodong Cheng

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

Scientific Machine Learning (SciML)

- SciML is a young field joining **scientific computing** and **machine learning**.
- The combination of scientific computing and machine learning **comes in various forms**.
- Recent progress rides on accessible **hardware and open-source software**.

Opportunities

- **SciML techniques enhance classical numerical solvers and purely data-driven models**.
- **Offline-trained surrogates** yield **fast inference**, **speeding up or replacing costly workflow steps**.

Challenges

- Many methods are **not yet fully theoretically understood** and **lack rigorous theoretical guarantees**.
- **Achieving robust, efficient, stable training at scale**, especially with **sparse or noisy data**, is a topic of **ongoing research**.

Thank you for your attention!

Topical Activity Group

Scientific Machine Learning