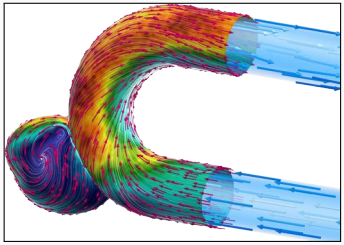
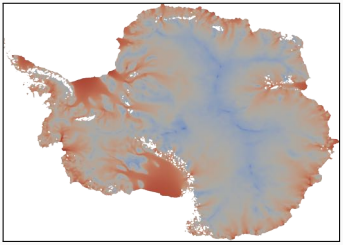
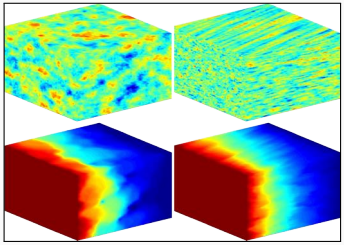
Deep Operator Networks: From Spectral Analysis to Localization and an Application to Post-Burn Wound Contraction

Alexander Heinlein[1]

International Symposium on Scientific and Bioengineering Computing, University of Macau, Macau, November 6–8, 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. Deep operator networks – Error analysis and localization via domain decompostion

   Based on joint work with

   Johannes Taraz (Delft University of Technology)

2. Domain decomposition-based physics-informed deep operator networks

   Based on joint work with

   Amanda A. Howard and **Panos Stinis** (Pacific Northwest National Laboratory)

3. Deep operator network models for predicting post-burn contraction

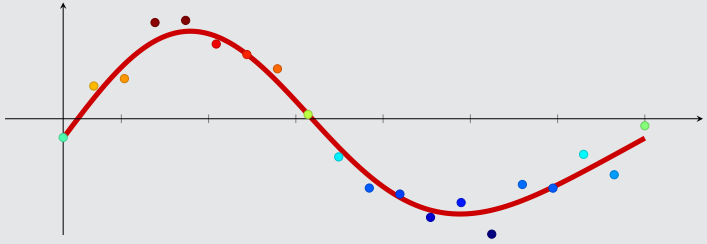
   Based on joint work with

   Selma Husanovic (form. Delft University of Technology)
   Ginger Egberts & **Fred Vermolen** (University of Hasselt)

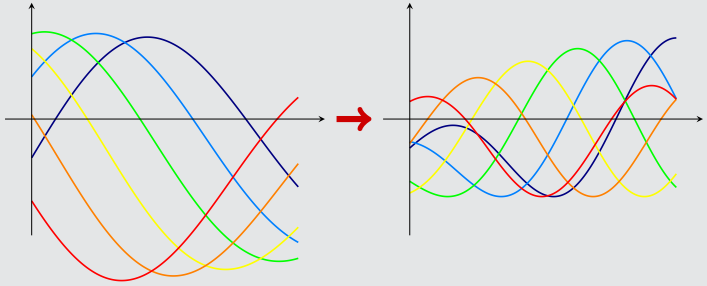
Deep operator networks – Error analysis and localization via domain decompostion

Function Versus Operator Learning

Function learning

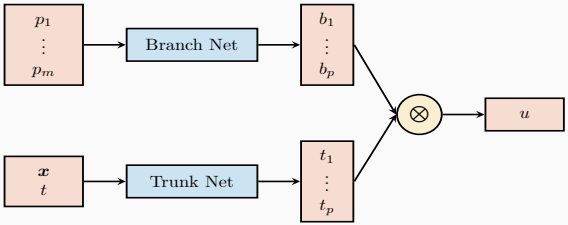


Operator learning



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

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

Physics-informed DeepONets

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

Other operator learning approaches

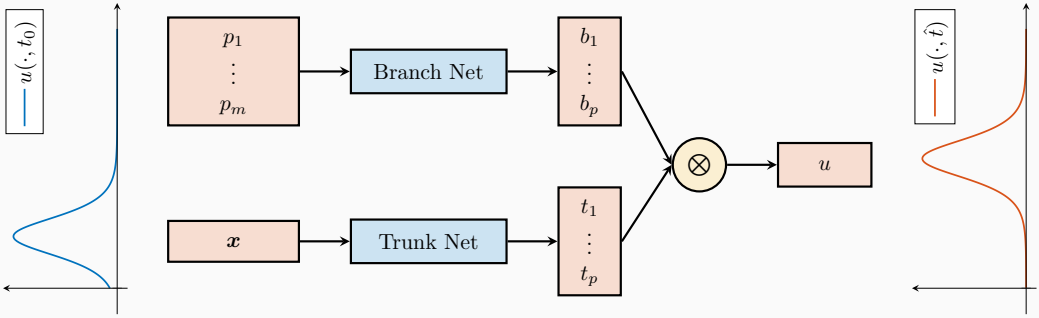
- **FNOs**: **Li et al. (2021)**
- **PCA-Net**: **Bhattacharya et al. (2021)**
- **Random features**: **Nelsen and Stuart (2021)**
- **CNOs**: **Raonić et al. (2023)**

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 $u(\cdot, t_0)$.



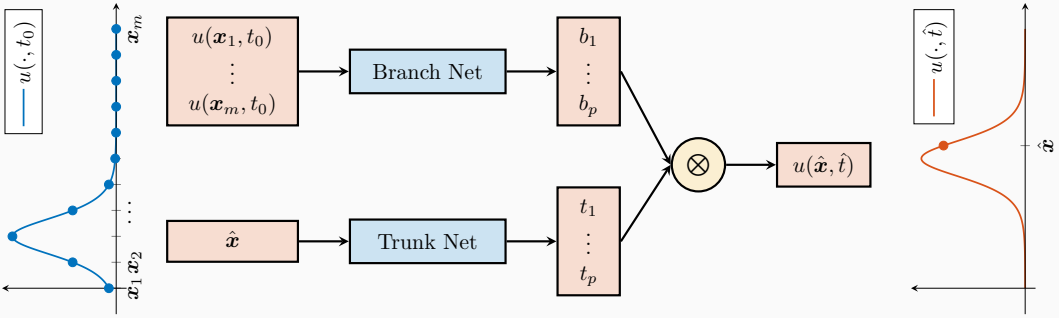
Here, the forecast time $\hat{t}$ is fixed to keep the learning task simple. A **more general neural operator** can take the target time as an additional input to the trunk network.

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 $u(\cdot, t_0)$.

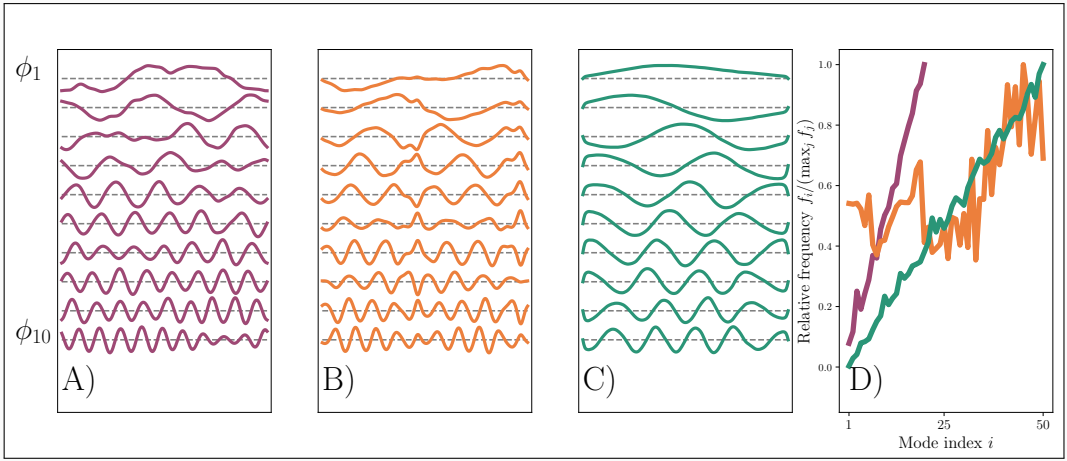


Here, the forecast time $\hat{t}$ is fixed to keep the learning task simple. A **more general neural operator** can take the target time as an additional input to the trunk network.

DeepONet Trunk Basis – Examples

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

Given **fixed parametrizations** $\mathscr{P}_1, \dots, \mathscr{P}_M$ and **evaluation points** $\hat{\boldsymbol{x}}_1, \dots, \hat{\boldsymbol{x}}_N$, the reference targets form the matrix

$$A = \begin{bmatrix} u_{\mathscr{P}_1}(\hat{\boldsymbol{x}}_1) & \cdots & u_{\mathscr{P}_M}(\hat{\boldsymbol{x}}_1) \\ \vdots & \ddots & \vdots \\ u_{\mathscr{P}_1}(\hat{\boldsymbol{x}}_N) & \cdots & u_{\mathscr{P}_M}(\hat{\boldsymbol{x}}_N) \end{bmatrix}$$

and the outputs of the DeepONet can be written as

$$\tilde{A} = \underbrace{\begin{bmatrix} t_1(\hat{\boldsymbol{x}}_1) & \cdots & t_m(\hat{\boldsymbol{x}}_1) \\ \vdots & \ddots & \vdots \\ t_1(\hat{\boldsymbol{x}}_N) & \cdots & t_m(\hat{\boldsymbol{x}}_N) \end{bmatrix}}_{=:T} \underbrace{\begin{bmatrix} b_1(\mathscr{P}_1) & \cdots & b_1(\mathscr{P}_M) \\ \vdots & \ddots & \vdots \\ b_m(\mathscr{P}_1) & \cdots & b_m(\mathscr{P}_M) \end{bmatrix}}_{=:B^\top} = \begin{bmatrix} t_1 & \cdots & t_m \end{bmatrix} \begin{bmatrix} b_1^\top \\ \vdots \\ b_m^\top \end{bmatrix},$$

where B and T are discretizations of the branch and trunk net outputs. Similar to **Lanthaler et al. (2022)**, we derive an **error decomposition into branch and trunk errors**

$$\mathscr{E} := \|A - \tilde{A}\|_F^2 = \underbrace{\|T(T^+A - B^\top)\|_F^2}_{=:\mathscr{E}_B} + \underbrace{\|(I - TT^+)A\|_F^2}_{=:\mathscr{E}_T},$$

where $\|\cdot\|_F$ is the **Frobenius norm** and T^+ is the **Moore–Penrose pseudoinverse** of T.

Analyzing the DeepONet Using the Singular Value Decomposition

A **singular value decomposition (SVD)** of the target matrix A (rank k) reads as follows:

$$A = \Phi \Sigma V^\top = \begin{bmatrix} \phi_1 & \cdots & \phi_k \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_k \end{bmatrix} \begin{bmatrix} v_1^\top \\ \vdots \\ v_k^\top \end{bmatrix} = \sum_{j=1}^{k} \sigma_j \phi_j v_j^\top,$$

where Φ and V are semi-orthogonal matrices. The **Eckart–Young theorem** states that the **best low-rank approximation of a matrix in the Frobenius norm is given by truncating its SVD**.

The structure is the same for the discretized DeepONet

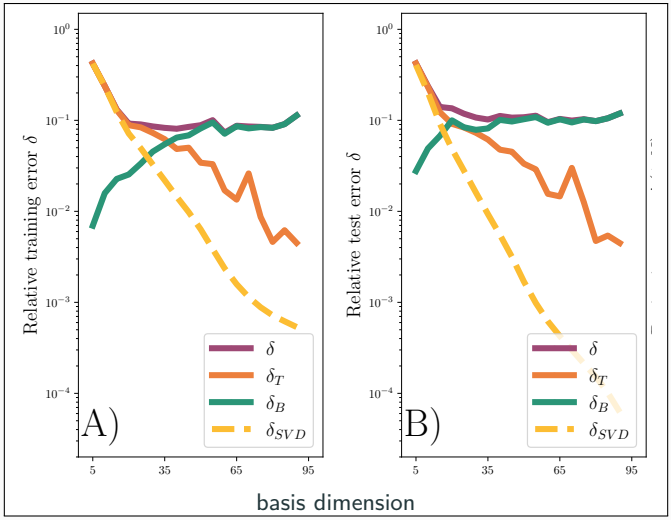
$$\tilde{A} = \begin{bmatrix} t_1 & \cdots & t_m \end{bmatrix} \begin{bmatrix} b_1^\top \\ \vdots \\ b_m^\top \end{bmatrix}.$$

Therefore, on the training data, we can minimize the discrete approximation error on the training data when choosing:

$$t_i = \phi_i \quad \text{and} \quad b_i = \sigma_i v_i.$$

DeepONet – Error Decomposition Results for the KdV Equation

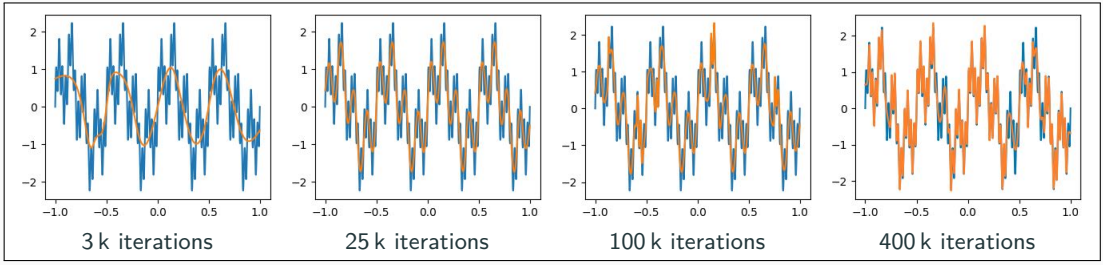
Results for the **KdV equation** with $t_0 = 0.0$ and $\hat{t} = 0.2$. 900 training and 100 test configurations.



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

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), ...**



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

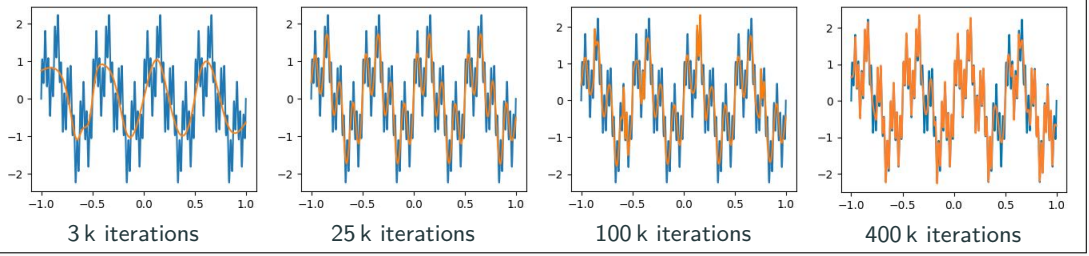
$$\theta_{k+1} = \theta_k - \eta_k \nabla_\theta \mathcal{C}(\mathcal{N}_{\theta_k}) \quad \rightarrow \quad \frac{\mathrm{d}x}{\mathrm{d}t}(t) = -\nabla_x \mathcal{C}(x(t))$$

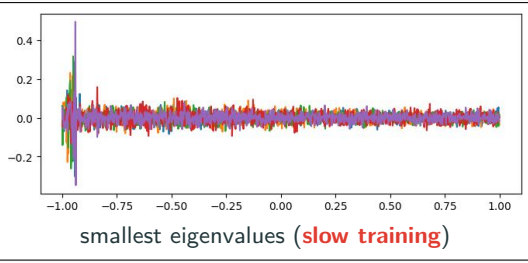
For infinite width neural networks, we obtain a constant **neural tangent kernel (NTK)** (**Jacot et al. (2018)**).
For mean squared error loss $\mathcal{C}$, we obtain with $\boldsymbol{K} = \left(\left\langle \frac{d\mathcal{N}_{\theta(t)}}{d\theta}(x_i), \frac{d\mathcal{N}_{\theta(t)}}{d\theta}(x_j) \right\rangle \right)_{ij}$:

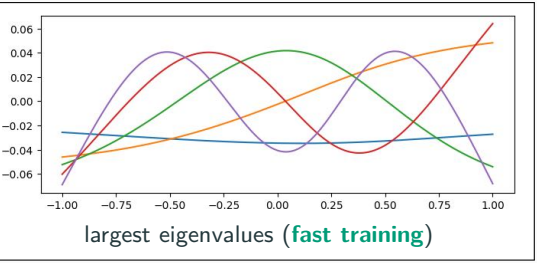
$$\frac{d\mathcal{N}_{\theta(t)}}{dt} = -\frac{2}{n} \cdot \boldsymbol{K}(t) \left(\mathcal{N}_{\theta(t)}(\mathbf{X}) - \mathbf{Y} \right) \quad \Rightarrow \quad \mathbf{U}^\top \left(\mathcal{N}_{\theta(t)}(\mathbf{X}) - \mathbf{Y} \right) \approx e^{-t\frac{2}{n}\boldsymbol{\Lambda}} \mathbf{U}^\top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

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), ...**







DeepONet – Branch Error

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

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

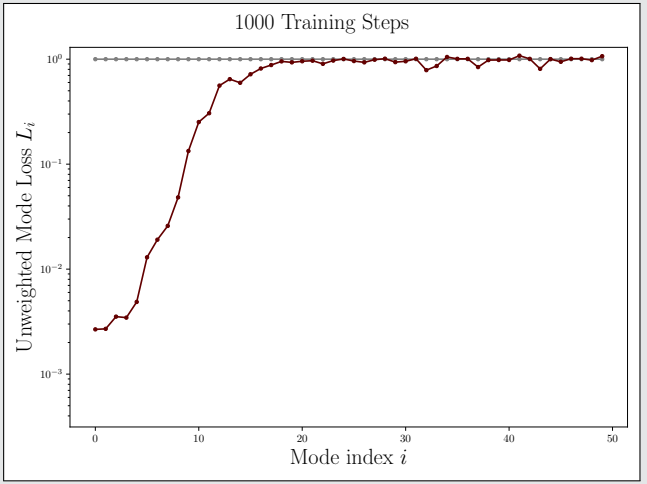
We call

$$\sigma_i^2 L_i$$

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

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

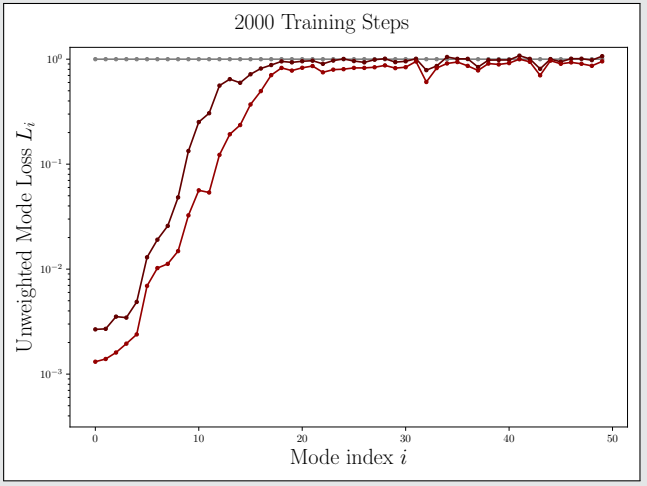
We call

$$\sigma_i^2 L_i$$

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

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

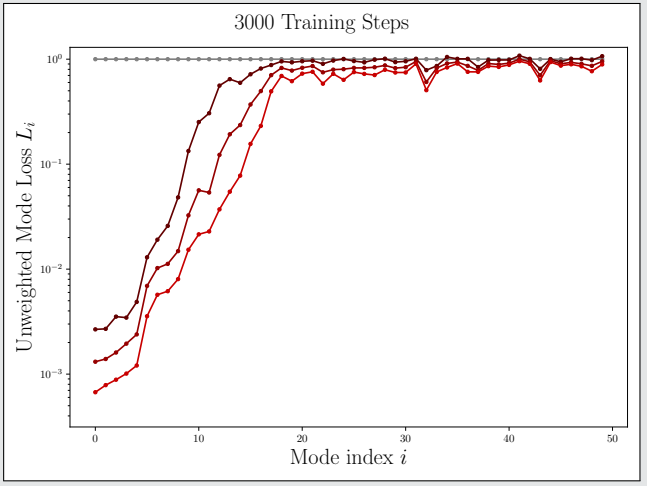
We call

$$\sigma_i^2 L_i$$

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

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

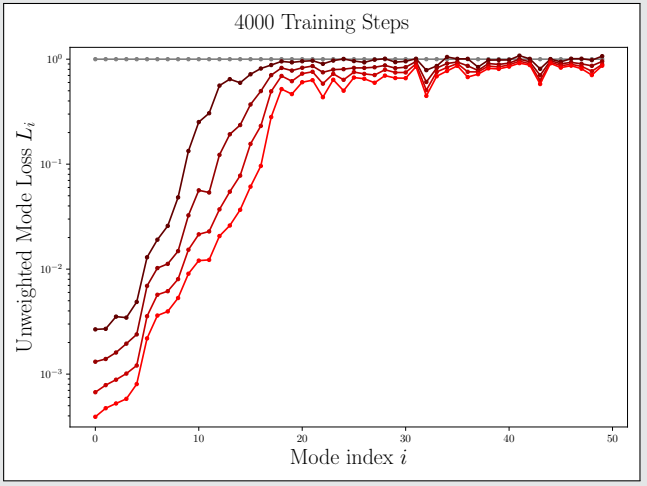
We call

$$\sigma_i^2 L_i$$

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

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

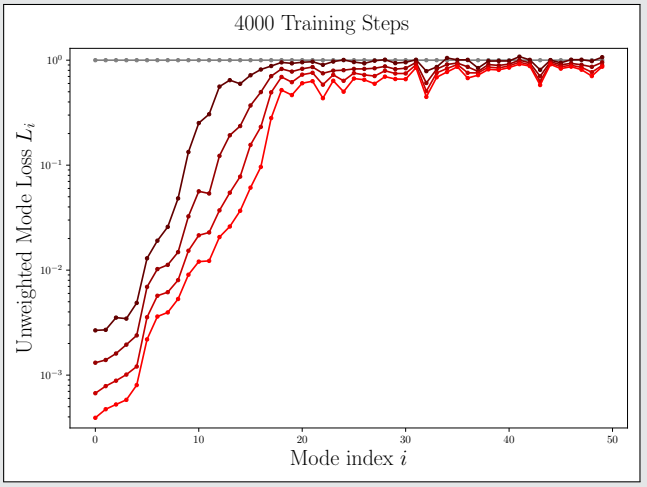
We call

$$\sigma_i^2 L_i$$

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

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

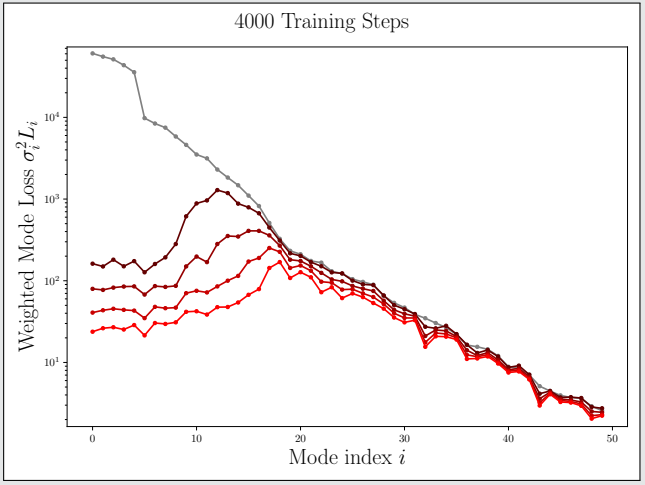
We call

$$\sigma_i^2 L_i$$

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

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

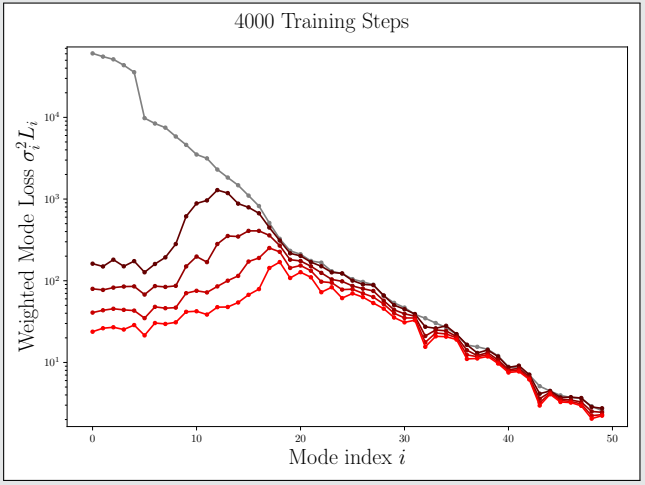
We call

$$\sigma_i^2 L_i$$

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

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

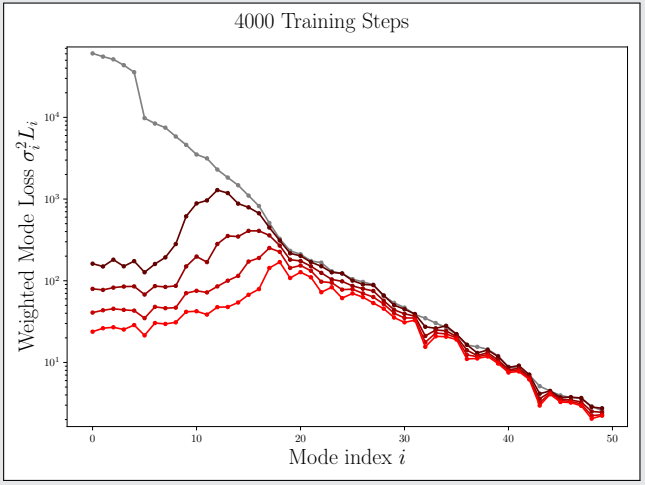
We call

$$\sigma_i^2 L_i$$

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?

Domain decomposition-based physics-informed deep operator networks

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**:

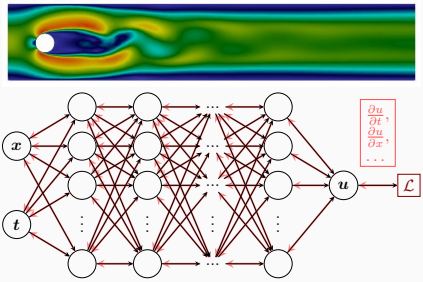
$$\mathcal{L}(\boldsymbol{\theta}) = \omega_{\text{data}}\mathcal{L}_{\text{data}}(\boldsymbol{\theta}) + \omega_{\text{PDE}}\mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}),$$

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

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

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

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



Hybrid loss

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

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

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

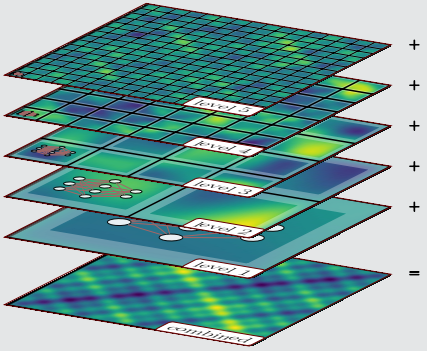
Multi-Level Finite Basis DeepONets (ML-FBDONs)

Multi-level domain decomposition

ML-FBDONs (**Howard, Heinlein, Stinis (in prep.)**) are based on

- **finite basis PINNs (FBPINNs)** (**Moseley et al. (2023)**) and
- **multi-level FBPINNs (ML-FBPINNs)** (**Dolean et al. (2024)**).

We use a **hierarchy of domain decompositions**:

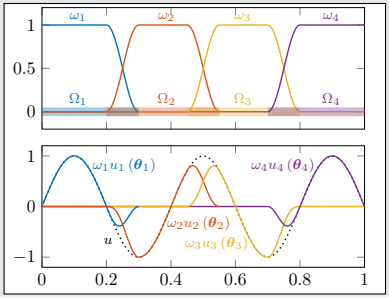


Neural network architecture

This yields the following **network architecture**:

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

where the $u_j^{(l)}$ are **subdomain DeepONets** and the $\omega_j^{(l)}$ form a **partition of unity** on each level:



Variants

- Shared-trunk FBDONs (ST-FBDONs)
- Stacking multifidelity FBDONs; cf. **Heinlein, Howard, Beecroft, Stinis (2025)**

FBDONs – Wave Equation

Wave equation

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

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

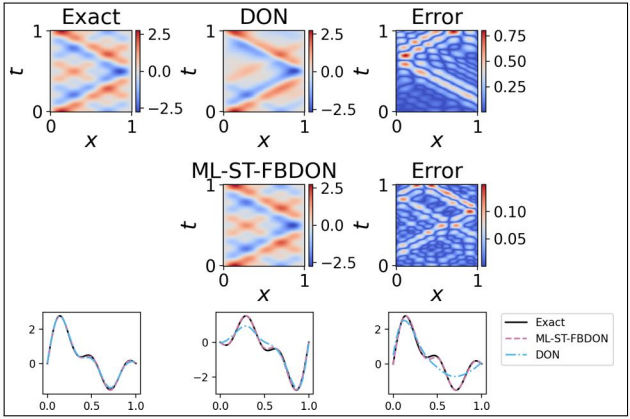
Solution: $s(x,t) = \sum_{n=1}^{5} b_n \sin(n\pi x)\cos\left(n\pi\sqrt{2}t\right)$

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

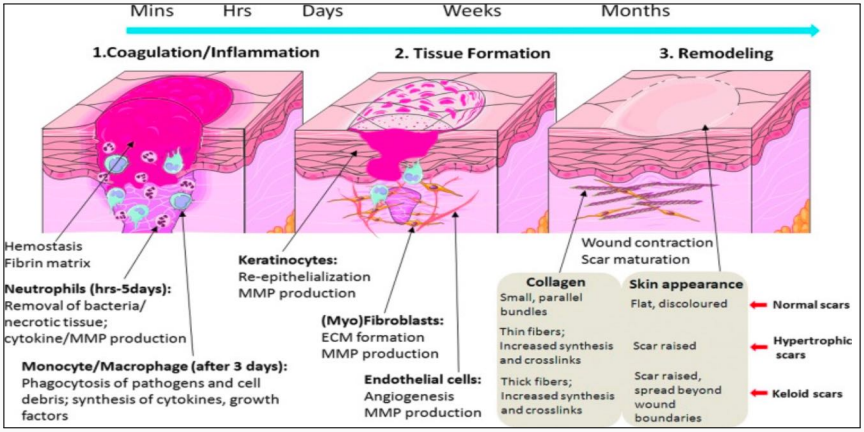
→ Sharing the trunk network does not only save in the number of parameters but even yields **better performance**

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

Deep operator network models for predicting post-burn contraction

Motivation: Post-Burn Contractures

- Burn injuries remain a massive burden, and **contractures** impose lifelong disability.
- Clinicians need **early wound forecasts** to time grafts, splints, and rehab.
- Yet **mechanistic models** remain **too slow for bedside iteration**.
- **Goal**: deploy surrogate models and deep learning to deliver **fast, actionable healing trajectories**.



Taken from **Xue and Jackson (2015)**.

Mathematical Model for Post-Burn Contraction

We use the **morphoelastic model for post-burn contraction** developed by **Koppenol et al. (2017)**:

$$\frac{\mathrm{D}z_i}{\mathrm{D}t} + z_i(\nabla \cdot \mathbf{v}) = -\nabla \cdot \mathbf{J}_i + R_i, \tag{1}$$

$$\rho_t \left(\frac{\mathrm{D}\mathbf{v}}{\mathrm{D}t} + \mathbf{v}(\nabla \cdot \mathbf{v}) \right) = \nabla \cdot \boldsymbol{\sigma} + \mathbf{f}, \tag{2}$$

$$\frac{\mathrm{D}\boldsymbol{\varepsilon}}{\mathrm{D}t} + \boldsymbol{\varepsilon}\,\mathrm{skw}(\nabla \mathbf{v}) - \mathrm{skw}(\nabla \mathbf{v})\,\boldsymbol{\varepsilon} + \big(\mathrm{tr}(\boldsymbol{\varepsilon}) - 1\big)\mathrm{sym}(\nabla \mathbf{v}) = -\mathbf{G}. \tag{3}$$

Eq. (1): **Conservation of cell density/concentration** for the biological constituents (fibroblasts (N), myofibroblasts (M), a generic signaling molecule (c), collagen (ρ)).

Eq. (2): **Conservation of linear momentum**.

Eq. (3): **Evolution of the infinitesimal effective strain $\boldsymbol{\varepsilon}$**, capturing dermal morphoelasticity, permanent deformation, and residual stresses; cf. **Hall (2008)**.

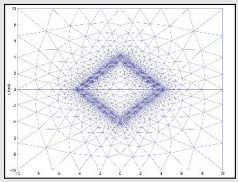
See **Koppenol et al. (2017)** and **Husanovic et al. (2025)** for more details.

Notation

- z_i concentration
- $\mathbf{J}_i$ flux per unit area
- R_i **reaction term**
- ρ_t total mass density
- $\mathbf{v} = \frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t}$ velocity vector ($\frac{\mathrm{D}}{\mathrm{D}t}$ material derivative)
- $\boldsymbol{\sigma}$ stress tensor
- $\mathbf{f}$ total body force
- $\boldsymbol{\varepsilon}$ infinitesimal effective strain

Finite element simulations

Exploiting symmetry of the domain:



Egberts et al. (2023)

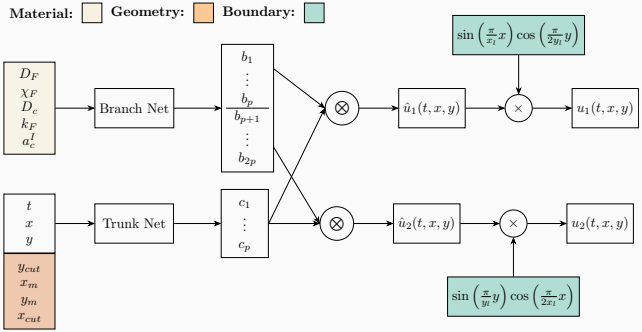
Horizon: 100 days (100 time steps)

DeepONet Architecture, Data, and Training

The architecture separates **material parameters** (branch)

- D_F **cell diffusion constant**,
- χ_F **chemotactic constant**,
- D_c **signaling diffusion constant**,
- k_F **cell differentiation rate**,
- a_c^I **half-maximum cell division enhancement rate**

and **spatio-temporal variables and parameters** (trunk):



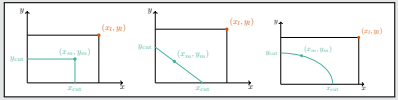
See **Husanovic et al. (2025)** for more details.

Training data

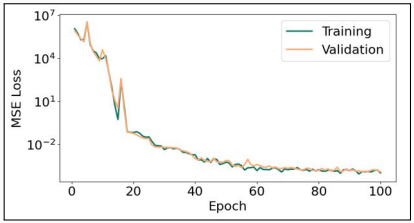
Material parameters:

param.	range	dim.
D_F	$(7.6 - 12.0)\cdot 10^{-7}$	$\mathrm{cm}^5/(\mathrm{cells\ day})$
χ_F	$(2.0 - 3.0)\cdot 10^{-3}$	$\mathrm{cm}^5/(\mathrm{g\ day})$
D_c	$(2.2 - 3.2)\cdot 10^{-3}$	$\mathrm{cm}^2/\mathrm{day}$
k_F	$(8.0 - 10.8)\cdot 10^{6}$	$\mathrm{cm}^3/(\mathrm{g\ day})$
a_c^I	$(0.9 - 1.1)\cdot 10^{-8}$	$\mathrm{g/cm}^3$

Geometric shapes:

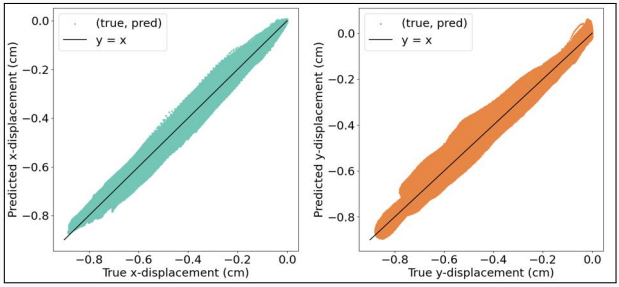


$\sum$ 150 000 data points



Overall Performance Results

Overall, we obtain **good agreement between predictions and ground truth** across **all test data points**:

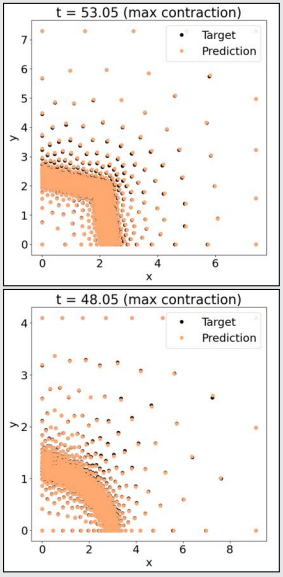


R^2	0.99
avg. relative RMSE	0.07
avg. relative error	0.03

Test data

150 finite element simulations covering geometries with mixed round/straight wound edges and varied material parameters.

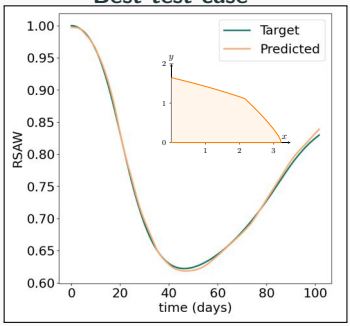
Exemplary predictions



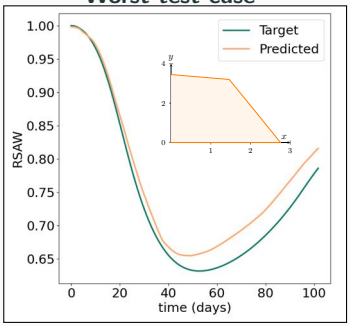
Results on Temporal Evolution

The **temporal evalution** is shown in the following plots:

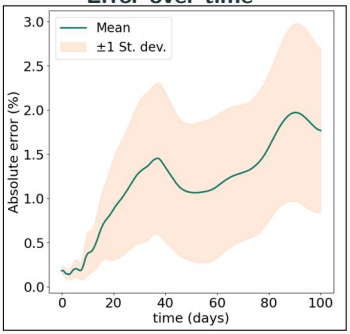
Best test case



Worst test case



Error over time



Error increases over time, but the **mean plus standard deviation error** remains **below 3 %**.

Speedup for one-year prediction

FEM (CPU): on average 4.7 minutes

DeepONet (CPU): on average 2.2 seconds

DeepONet (GPU): on average 1.2 seconds

DelftBlue supercomputer: Intel Xeon Gold 6248R CPU, NVIDIA A100 GPU.

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)

CWI

Centrum Wiskunde & Informatica

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

Summary

Approximation Properties and Error Decomposition in DeepONets

- DeepONet error **splits** into **branch** and **trunk** components.
- For **larger dimensions** of the **bases**, the **branch error** is typically **more critical**.
- Branch error is **dominated by singular vectors with medium singular values**.

Extension of FBPINN to DeepONets: Improved Performance via Localization

- To reduce errors on **medium-to-high singular values** (i.e., **higher frequencies**) we leverage **domain decomposition**-based neural architectures.
- **Localization via FBDONs**, especially with a **shared trunk**, yields **clear gains**.

Prediction of Post-Burn Wound Contraction

- A **DeepONet surrogate** matches **FE wound evolution** ($R^2 = 0.99$) for one year.
- **Speedup**: up to 128× **on CPU** and 235× **on GPU** with FE-level accuracy.

Thank you for your attention!

Topical Activity Group

Scientific Machine Learning