Advanced ROMs in SciML

Graph-based ROMs

Federico Pichi

23 June 2025

Non-intrusive and data-driven reduced order models

- **Linear approaches**: linear expansion with cheap reduced coefficients $\mathbf{X}_{\mathcal{N}}(\boldsymbol{\mu}) \approx \mathbf{V}\mathbf{X}_{N}(\boldsymbol{\mu})$.
  $\rightsquigarrow$ POD+interpolation, POD+Neural Networks, POD+Gaussian Process Regression, POD+PINN
- **Nonlinear approaches**: fully nonlinear mapping of the latent coordinates $\mathbf{X}_{\mathcal{N}}(\boldsymbol{\mu}) \approx \psi(\mathbf{X}_{N}(\boldsymbol{\mu}))$.
  $\rightsquigarrow$ Autoencoders, Neural Operators, Nonlinear Manifold Least Square Petrov Galerkin

Recap on projection-based ROMs

We characterize the error $\mathbf{e}_h(\boldsymbol{\mu}) = \mathbf{u}_h(\boldsymbol{\mu}) - \mathsf{V}\mathbf{u}_N(\boldsymbol{\mu})$, between the solution of the Galerkin-HF and -RB problems in terms of the HF residual of the Galerkin-RB solution $\mathbf{r}_h(\mathbf{u}_N;\boldsymbol{\mu}) = \mathbf{f}_h(\boldsymbol{\mu}) - \mathsf{A}_h(\boldsymbol{\mu})\mathsf{V}\mathbf{u}_N(\boldsymbol{\mu})$:

1. $\mathsf{A}_h(\boldsymbol{\mu})\mathbf{e}_h(\boldsymbol{\mu}) = \mathbf{r}_h(\mathbf{u}_N;\boldsymbol{\mu})$
2. $\mathsf{V}^T\mathsf{A}_h(\boldsymbol{\mu})\mathbf{u}_h(\boldsymbol{\mu}) = \mathsf{V}^T\mathbf{f}_h(\boldsymbol{\mu})$
3. $\mathsf{V}^T\mathbf{r}_h(\mathbf{u}_N;\boldsymbol{\mu}) = \mathbf{0}$

Indeed, since $\mathbf{u}_h(\boldsymbol{\mu}) = \mathsf{V}\mathbf{u}_N(\boldsymbol{\mu}) + \mathbf{e}_h(\boldsymbol{\mu})$, left multiplying $\mathsf{A}_h(\boldsymbol{\mu})\mathbf{u}_h(\boldsymbol{\mu}) = \mathbf{f}_h(\boldsymbol{\mu})$ by V^T we obtain

$$\mathsf{V}^T\mathsf{A}_h(\boldsymbol{\mu})\mathsf{V}\mathbf{u}_N(\boldsymbol{\mu}) - \mathsf{V}^T\mathbf{f}_h(\boldsymbol{\mu}) = -\mathsf{V}^T\mathsf{A}_h(\boldsymbol{\mu})\mathbf{e}_h(\boldsymbol{\mu}),$$

that is

$$\mathsf{V}^T\mathsf{A}_h(\boldsymbol{\mu})\mathsf{V}\mathbf{u}_N(\boldsymbol{\mu}) - \mathsf{V}^T\mathbf{f}_h(\boldsymbol{\mu}) = -\mathsf{V}^T\mathbf{r}_h(\mathbf{u}_N;\boldsymbol{\mu});$$

requiring $\mathbf{u}_N$ to satisfy

$$\mathsf{V}^T\mathbf{r}_h(\mathbf{u}_N;\boldsymbol{\mu}) = \mathbf{0}, \quad \text{or equivalently,} \quad \mathsf{V}^T\mathsf{A}_h(\boldsymbol{\mu})\mathsf{V}\mathbf{u}_N(\boldsymbol{\mu}) = \mathsf{V}^T\mathbf{f}_h(\boldsymbol{\mu}).$$

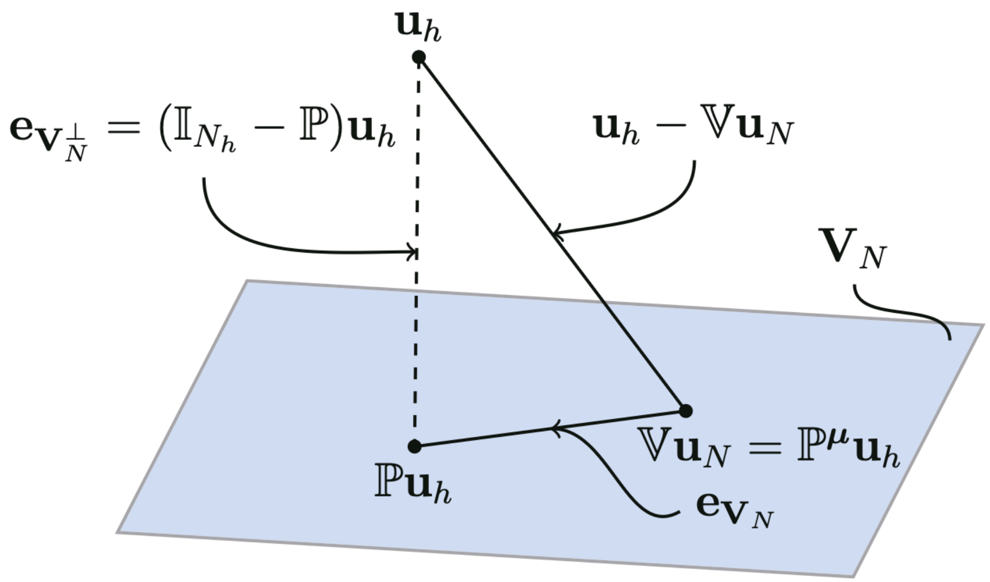
Geometric interpretation of Galerkin-RB

The matrix $\mathsf{V} \in \mathbb{R}^{\mathcal{N}\times N}$ define an orthogonal projection onto the subspace $\mathbb{V}_N = \mathrm{span}\{\boldsymbol{\zeta}_1, \dots, \boldsymbol{\zeta}_N\}$.
If the basis functions to be orthonormal with $\mathsf{V}^T\mathsf{V} = \mathbb{I}_N$

1. the matrix $\mathbb{P} = \mathsf{V}\mathsf{V}^T \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ project $\mathbb{R}^{\mathcal{N}}$ onto $\mathbb{V}_N$
2. the matrix $\mathbb{I}_{\mathcal{N}} - \mathsf{V}^T\mathsf{V} \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ project $\mathbb{R}^{\mathcal{N}}$ onto $\mathbb{V}_N^T$

The error can be additively split into two orthogonal terms

$$
\begin{aligned}
\mathbf{e}_h(\boldsymbol{\mu}) &= \mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{V}\mathbf{u}_N(\boldsymbol{\mu}) \\
&= (\mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{P}\mathbf{u}_h(\boldsymbol{\mu})) + (\mathbb{P}\mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{V}\mathbf{u}_N(\boldsymbol{\mu})) \\
&= (\mathbb{I}_{N_h} - \mathbb{P})\mathbf{u}_h(\boldsymbol{\mu}) + \mathbb{V}(\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}) - \mathbf{u}_N(\boldsymbol{\mu})) \\
&= \mathbf{e}_{\mathbf{V}_N^\perp}(\boldsymbol{\mu}) + \mathbf{e}_{\mathbf{V}_N}(\boldsymbol{\mu})
\end{aligned}
$$



Time-dependent problems and linear reduction

Given $\boldsymbol{\mu} \in \mathcal{P}$, we aim at solving the initial value problem

$$\begin{cases} \dot{\mathbf{u}}_h(t;\boldsymbol{\mu}) = \mathbf{f}(t, \mathbf{u}_h(t;\boldsymbol{\mu});\boldsymbol{\mu}) & t \in (0,T) \\ \mathbf{u}_h(0;\boldsymbol{\mu}) = \mathbf{u}_0(\boldsymbol{\mu}), \end{cases}$$

where $\mathcal{P} \subset \mathbb{R}^{n_\mu}$ is the parameter space, $\mathbf{u}_h : [0,T) \times \mathcal{P} \to \mathbb{R}^{N_h}$ is the parametrized FOM solution, $\mathbf{u}_0 : \mathcal{P} \to \mathbb{R}^{N_h}$ is the initial datum and $\mathbf{f} : (0,T) \times \mathbb{R}^{N_h} \times \mathcal{P} \to \mathbb{R}^{N_h}$ is the nonlinear dynamics.

The time-evolution of the reduced coordinates $\mathbf{u}_n(t;\boldsymbol{\mu})$ can be obtained from $\mathbf{u}_h(t;\boldsymbol{\mu}) = \mathsf{V}\mathbf{u}_n(t;\boldsymbol{\mu})$ as

$$\begin{cases} \mathsf{V}\dot{\mathbf{u}}_n(t;\boldsymbol{\mu}) = \mathbf{f}(t, \mathsf{V}\mathbf{u}_n(t;\boldsymbol{\mu});\boldsymbol{\mu}) & t \in (0,T) \\ \mathsf{V}\mathbf{u}_n(0;\boldsymbol{\mu}) = \mathbf{u}_0(\boldsymbol{\mu}), \end{cases}$$

imposing that the residual is orthogonal to a n-dimensional subspace spanned by $\mathsf{Y} \in \mathbb{R}^{N_h \times n}$, that is

$$\mathsf{Y}^T \mathbf{r}_h(\mathsf{V}\mathbf{u}_n(t;\boldsymbol{\mu})) = \mathsf{Y}^T[\mathsf{V}\dot{\mathbf{u}}_n(t;\boldsymbol{\mu}) - \mathbf{f}(t, \mathsf{V}\mathbf{u}_n(t;\boldsymbol{\mu});\boldsymbol{\mu})] = \mathbf{0}.$$

- $\mathsf{Y} = \mathsf{V}$ is a Galerkin projection, while $\mathsf{Y} \neq \mathsf{V}$ is a Petrov-Galerkin projection.
- Even choosing Y such that $\mathsf{Y}^T\mathsf{V} = I$ the ROM stability on long time intervalsis not guaranteed.

Nonlinear model order reduction

Issues: Exploit low-dimensional subspaces of dimension $n \gg n_\mu + 1$ much larger than the intrinsic dimension of the solution manifold (*slow decay Kolmogorov n-width*), and recover the efficiency for nonlinear and non-affine problems (*hyper-reduction* to avoid computations with N_h dofs).

Examples: *registration method*, *kernel POD*, *shifted POD*, *localized models* and *Wasserstein spaces*.

$\rightsquigarrow$ nonlinear ROM to approximate as $\tilde{\mathbf{u}}_h(t;\boldsymbol{\mu}) = \boldsymbol{\Psi}_h(\mathbf{u}_n(t;\boldsymbol{\mu}))$, where $\boldsymbol{\Psi}_h : \mathbb{R}^n \to \mathbb{R}^{N_h}$ is a nonlinear and differentiable function. The solution manifold $\mathcal{S}_h$ is approximated by a reduced nonlinear manifold

$$\tilde{\mathcal{S}}_n = \{\boldsymbol{\Psi}_h(\mathbf{u}_n(t;\boldsymbol{\mu})) \mid \mathbf{u}_n(t;\boldsymbol{\mu}) \in \mathbb{R}^n,\ t \in [0,T) \text{ and } \boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^{N_h}.$$

Goal: set a ROM whose dimension n close to the intrinsic dimension $n_{\boldsymbol{\mu}} + 1$ of the solution manifold $\mathcal{S}_h$.

$\rightsquigarrow$ To model $(t,\boldsymbol{\mu}) \mapsto \mathbf{u}_n(t,\boldsymbol{\mu})$ and describe the system dynamics, one exploits the map $\mathbf{u}_n(t;\boldsymbol{\mu}) = \boldsymbol{\Phi}_n(t,\boldsymbol{\mu})$, where $\boldsymbol{\Phi}_n : [0,T) \times \mathbb{R}^{n_\mu+1} \to \mathbb{R}^n$ is a nonlinear and differentiable function.

[1] Fresca, S., Dede', L., Manzoni, A., 2021. A Comprehensive Deep Learning-Based Approach to Reduced Order Modeling of Nonlinear Time-Dependent Parametrized PDEs. JSC. https://doi.org/10.1007/s10915-021-01462-7

DL-ROMs approximation

Idea: exploit deep learning (DL) functions $\boldsymbol{\Psi}_h$ and $\boldsymbol{\Phi}_n$ parametrized by NN weights $\boldsymbol{\theta}$.

Why: approximating nonlinear maps, generalize to unseen data, non-intrusive and data-driven ROMs.

How: Two main blocks to learn the *reduced dynamics* and the *reduced nonlinear manifold*.

1. **reduced dynamics**, FNN $\boldsymbol{\phi}_n^{DF}$, such that $(t, \boldsymbol{\mu}) \mapsto \mathbf{u}_n(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}) = \boldsymbol{\phi}_n^{DF}(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF})$.
2. **reduced nonlinear manifold**, decoder function of a convolutional autoencoder $\mathbf{f}_h^D$, such that $\mathbf{u}_n(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}) \mapsto \tilde{\mathbf{u}}_h(t; \boldsymbol{\mu}, \boldsymbol{\theta}) = \mathbf{f}_h^D(\mathbf{u}_n(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}); \boldsymbol{\theta}_D)$.

$\rightsquigarrow$ the DL-ROM approximation is then given by

$$\tilde{\mathbf{u}}_h(t; \boldsymbol{\mu}, \boldsymbol{\theta}) = \mathbf{f}_h^D(\boldsymbol{\phi}_n^{DF}(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}); \boldsymbol{\theta}_D),$$

where $\boldsymbol{\phi}_n^{DF}(\cdot; \cdot, \boldsymbol{\theta}_{DF}) : \mathbb{R}^{(n_\mu+1)} \to \mathbb{R}^n$ and $\mathbf{f}_h^D(\cdot; \boldsymbol{\theta}_D) : \mathbb{R}^n \to \mathbb{R}^{N_h}$.

Computing the DL-ROM approximation for any new value of $\boldsymbol{\mu} \in \mathcal{P}$, at any given time, requires to evaluate the map $(t, \boldsymbol{\mu}) \to \tilde{\mathbf{u}}_h(t; \boldsymbol{\mu}, \boldsymbol{\theta})$ at the testing stage, once the parameters $\boldsymbol{\theta}$ have been determined.

DL-ROMs training

Task: Solving an optimization problem for $\boldsymbol{\theta}$ starting from the HF dataset $\{t^j, \boldsymbol{\mu}^i, \tilde{\mathbf{u}}_h(t^j; \boldsymbol{\mu}^i)\}$.

$\rightsquigarrow$ find the optimal parameters $\boldsymbol{\theta}^*$ solution of

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}} N_t} \sum_{i=1}^{N_{\text{train}}} \sum_{j=1}^{N_t} \frac{1}{2} \|\mathbf{u}_h(t^j; \boldsymbol{\mu}_i) - \mathbf{f}_h^D(\boldsymbol{\phi}_n^{DF}(t^j; \boldsymbol{\mu}_i, \boldsymbol{\theta}_{DF}); \boldsymbol{\theta}_D)\|^2 \to \min_{\boldsymbol{\theta}}$$

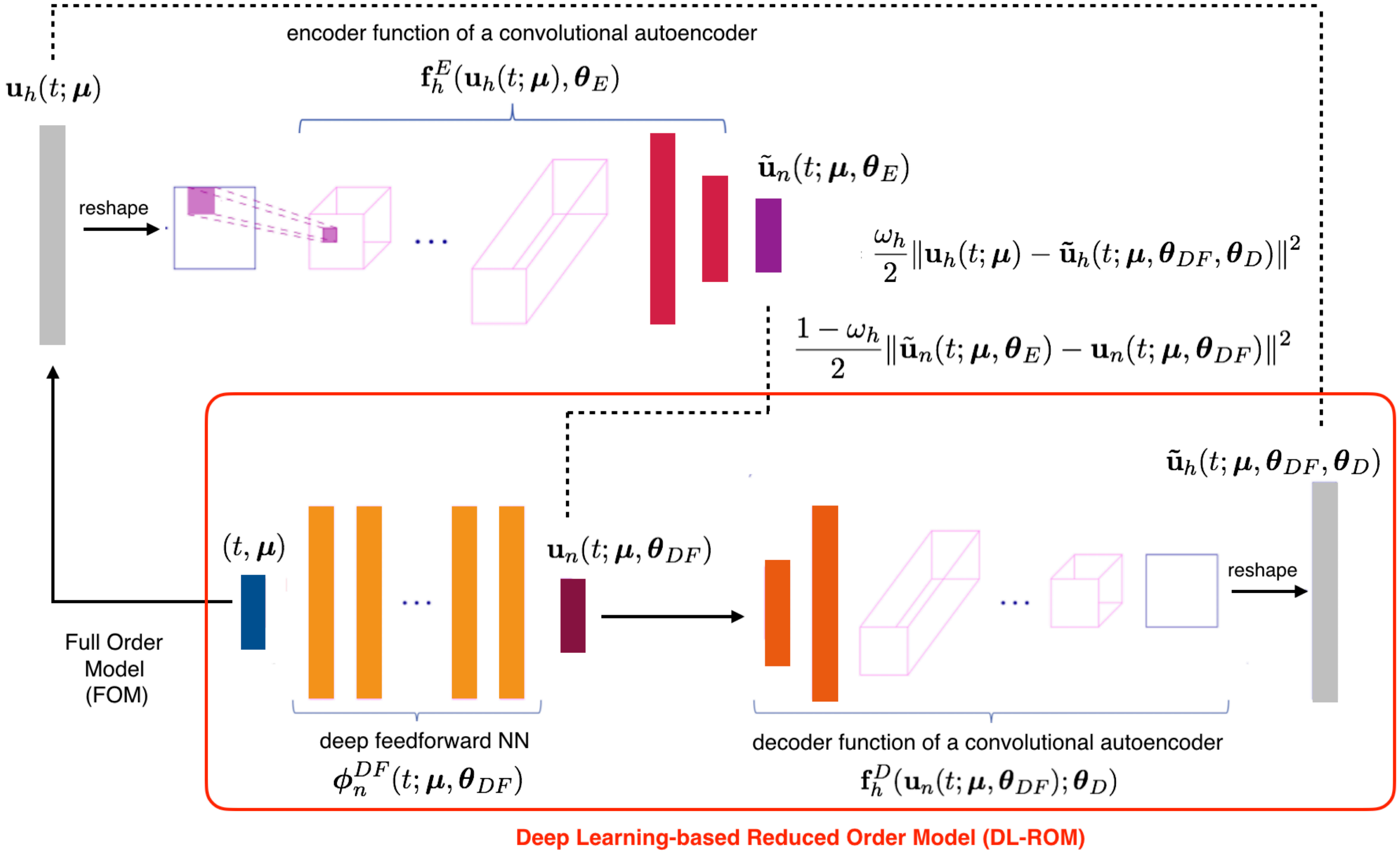
Tools: cross-validation (8:2), early-stopping, ELU, ADAM, batch, $N_s = 10^3$, $\eta = 10^{-4}$, $N_{\text{epochs}} = 10^4$.

Reduction: relying on an autoencoder we have the encoder function $\tilde{\mathbf{u}}_n(t; \boldsymbol{\mu}, \boldsymbol{\theta}_E) = \mathbf{f}_n^E(\mathbf{u}(t; \boldsymbol{\mu}); \boldsymbol{\theta}_E)$, mapping each HF solution onto a low-dimensional representation $\tilde{\mathbf{u}}_n(t; \boldsymbol{\mu}, \boldsymbol{\theta}_E)$, optimizing for

$$\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{N_{\text{train}} N_t} \sum_{j=1}^{N_{train}} \sum_{i=1}^{N_t} \frac{\omega_h}{2} \|\mathbf{u}_h(t^k; \boldsymbol{\mu}_i) - \tilde{\mathbf{u}}_h(t^k; \boldsymbol{\mu}_i, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)\|^2$$
$$+ \frac{1-\omega_h}{2} \|\tilde{\mathbf{u}}_n(t^k; \boldsymbol{\mu}_i, \boldsymbol{\theta}_E) - \mathbf{u}_n(t^k; \boldsymbol{\mu}_i, \boldsymbol{\theta}_{DF})\|^2$$

with $\boldsymbol{\theta} = (\boldsymbol{\theta}_E, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)$, and $\omega_h \in [0, 1]$.

DL-ROM architecture

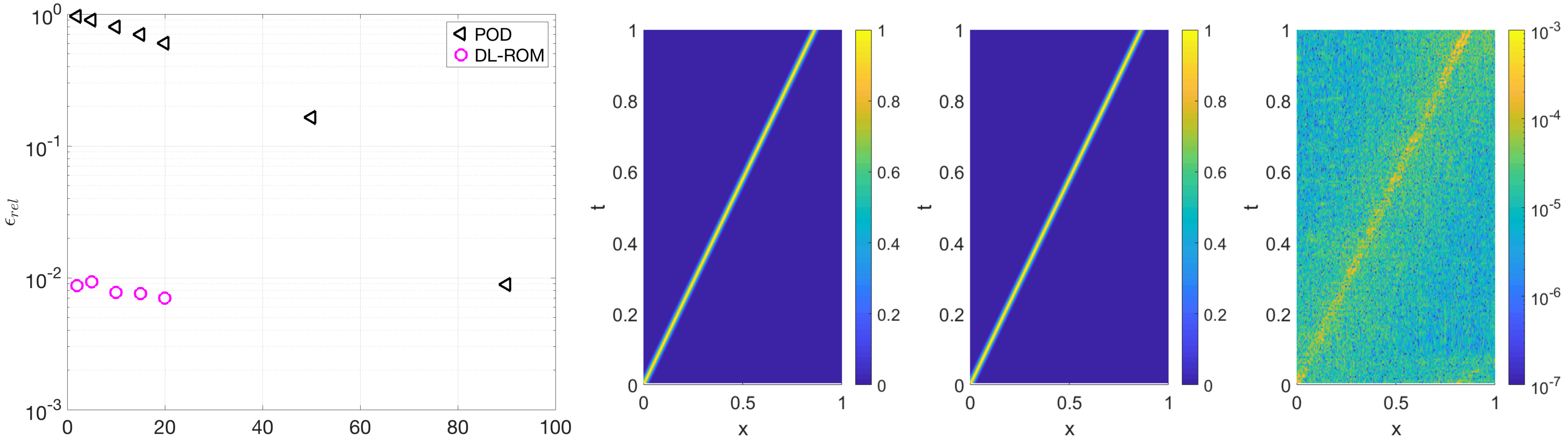


DL-ROMs results (linear transport)

Parametrized one-dimensional linear transport equation whose exact solution is $u(x,t) = u_0(x - \mu t)$:

$$\begin{cases} \dfrac{\partial u}{\partial t} + \mu \dfrac{\partial u}{\partial x} = 0, & (x,t) \in \mathbb{R} \times (0,T) \\ u(x,0) = u_0(x) = (1/\sqrt{2\pi\sigma}) e^{-x^2/2\sigma}, & x \in \mathbb{R}. \end{cases}$$

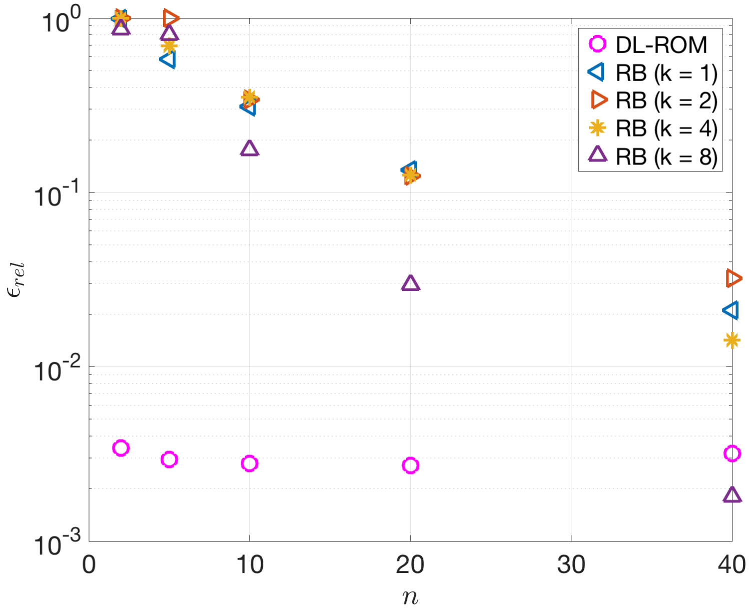
The parameter $\mu \in \mathcal{P} = [0.775, 1.25]$. represents the velocity of the travelling wave, and $T = 1$.

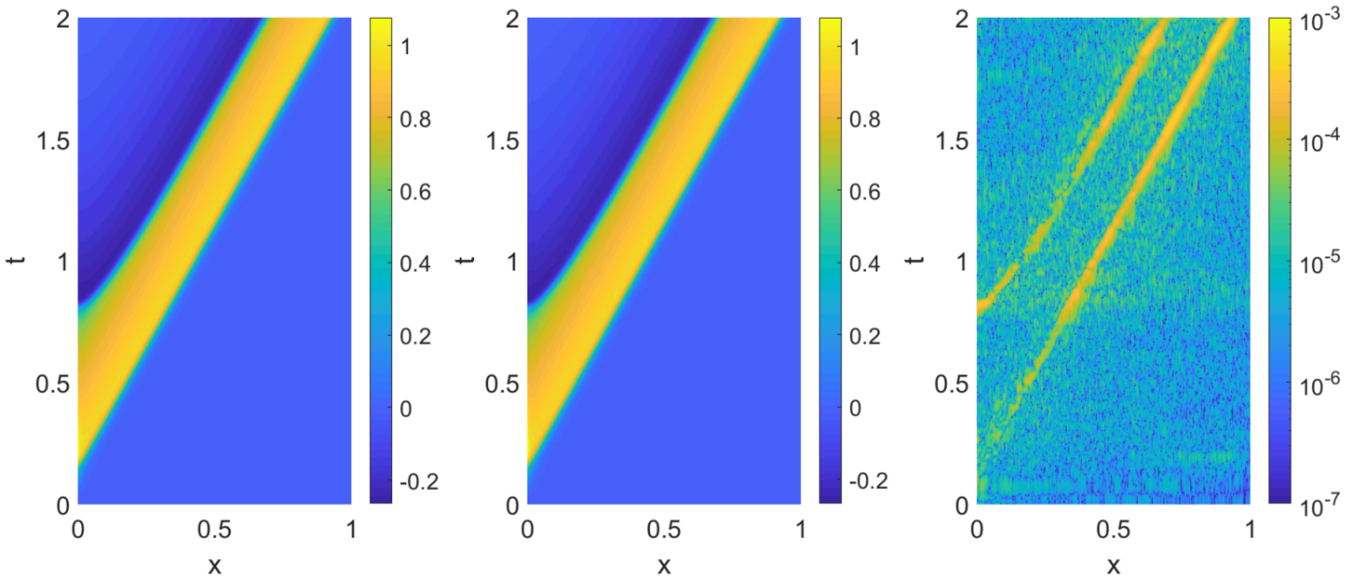


DL-ROMs results (monodomain)

One-dimensional coupled PDE-ODE nonlinear system with $\mu \in \mathcal{P} = 5 \cdot [10^{-3}, 10^{-2}]$

$$\begin{cases} \mu \frac{\partial u}{\partial t} - \mu^2 \frac{\partial^2 u}{\partial x^2} + u(u-0.1)(u-1) + w = 0, & (x,t) \in (0,L) \times (0,T) \\ \frac{dw}{dt} + (\gamma w - \beta u) = 0, & (x,t) \in (0,L) \times (0,T) \\ \frac{\partial u}{\partial x}(0,t) = 50000 t^3 e^{-15t}, & t \in (0,T) \\ \frac{\partial u}{\partial x}(L,t) = 0, & t \in (0,T) \\ u(x,0) = 0, \ w(x,0) = 0, & x \in (0,L) \end{cases}$$





POD-DL-ROMs approximation

Issue: reduce the CAE *input* dimensionality and work with *unstructured meshes*.

Aim: *faster* training for *larger* FOM dimensions, without affecting the number of networks parameters.

How: *randomized POD* as the first layer of the CAE, and a suitable *multi-fidelity* pretraining stage exploiting coarser discretizations or simplified physical models.

Idea: two-step reduction approach, comprising of a *randomized POD* and a *DL-ROM learning*.

Remarks: The dimension of the *linear susbspace* N can be taken much larger than the POD-Galerkin one, since it is used for *data compression*, to avoid to feed training data of dimension N_h.

$\rightsquigarrow$ The **POD-DL-ROM** approximation $\tilde{\mathbf{u}}_h(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D) = \mathbf{V}_N\tilde{\mathbf{u}}_N(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)$ is sought by applying the **DL-ROM** strategy to approximate $\mathbf{V}_N^T\mathbf{u}_h(t;\boldsymbol{\mu})$, rather than $\mathbf{u}_h(t;\boldsymbol{\mu})$, in the linear manifold

$$\tilde{\mathcal{S}}_h^{N,\mathrm{lin}} = \{\mathbf{V}_N\tilde{\mathbf{u}}_N(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)\ |\tilde{\mathbf{u}}_N(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D) \in \mathbb{R}^N,\ t\in[0,T)\ \text{and}\ \boldsymbol{\mu}\in\mathcal{P}\subset\mathbb{R}^{n_\mu}\} \subset \mathbb{R}^{N_h},$$

[1] Fresca, S., Manzoni, A., 2022. POD-DL-ROM: Enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition. CMAME. https://doi.org/10.1016/j.cma.2021.114181

POD-DL-ROMs strategy

1. the dimension of the reduced linear problem is decreased until to match the intrinsic dimension $n_\mu + 1$ of the parametrized problem (POD-DL-ROM vs POD-NN).

2. approximate all reduced coordinates at once, with no additional SVDs (POD-DL-ROM vs POD-GPR).

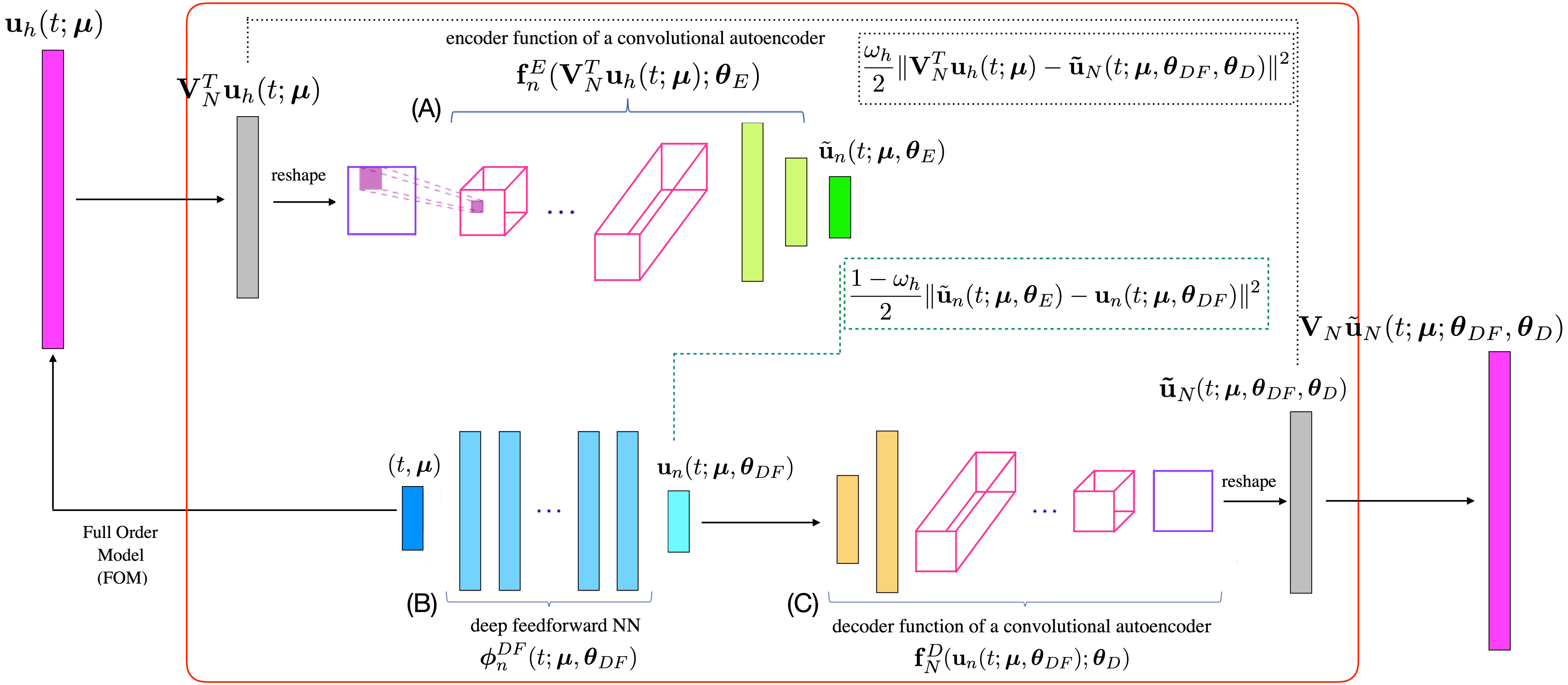
3. assuming that the input-output map is locally Lipschitz it is possible to prove convergence.

4. The SVD of $\mathsf{S} \in \mathbb{R}^{N_h \times N_s}$ can be extremely time consuming for large-scale problems. $\rightsquigarrow$ **rSVD** computes an approximated SVD, via a QR decomposition of the matrix $\mathsf{Y} = (\mathsf{S}\mathsf{S}^T)^q \mathsf{S}\Omega$, where $\Omega \in \mathbb{R}^{N_s \times m}$ is Gaussian random matrix with $N \leq m \leq N_s$. The, an SVD is performed on $\mathsf{B} = \mathsf{Q}^T\mathsf{S} = \tilde{\mathsf{V}}\tilde{\Sigma}\tilde{\mathsf{Z}}$, and a basis for S is then recovered by setting $\mathsf{V}_N = \mathsf{Q}\tilde{\mathsf{V}}$.

5. Transfer learning approach as **multi-fidelity pretraining**, train a model to solve a simpler task, then move on to the final task, combining *models* and *fidelities* (discretizations and parameter ranges).

6. For **vector problems**, this approach allows the dimensions $N_h^i, i = 1, \ldots, d$ to be different. It is the rSVD dimension N used to reduce each vector component that must be kept equal.

POD-DL-ROMs architecture

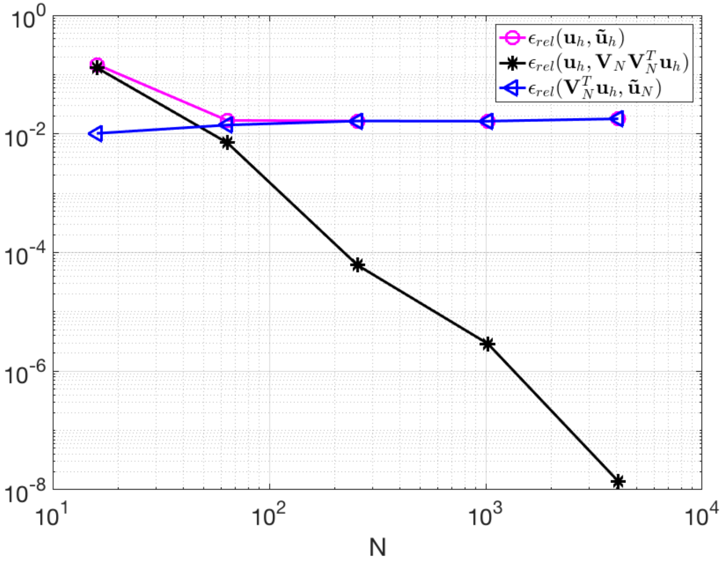


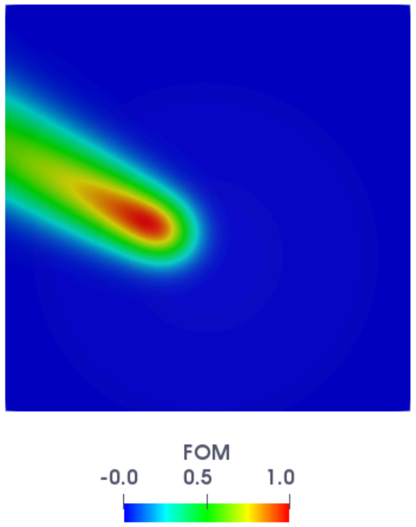
Proper Orthogonal Decomposition-enhanced Deep Learning-based Reduced Order Model (POD DL-ROM)

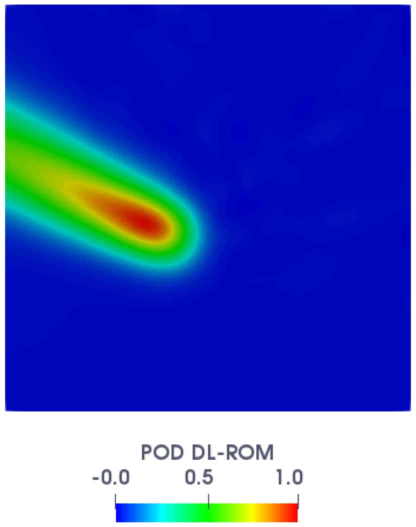
POD-DL-ROMs results (unsteady ADR equation)

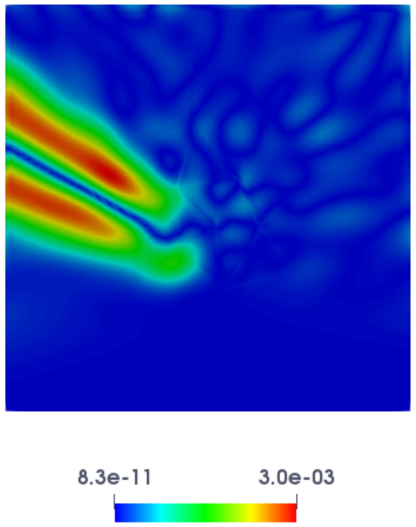
Advection-diffusion-reaction system in $\Omega = (0,1)^2$ with $n_\mu = 4$ parameters, $N_s = 5 \times 10^4$, and N=64

$$\begin{cases} \dfrac{\partial u}{\partial t} - \operatorname{div}(\mu_1 \nabla u) + \mathbf{b}(t;\mu_2) \cdot \nabla u + cu = f(\mu_3, \mu_4) & (\mathbf{x}, t) \in \Omega \times (0,T), \\ \mu_1 \nabla u \cdot \mathbf{n} = 0 & (\mathbf{x}, t) \in \partial\Omega \times (0,T), \\ u(0) = 0 & \mathbf{x} \in \Omega, \end{cases}$$







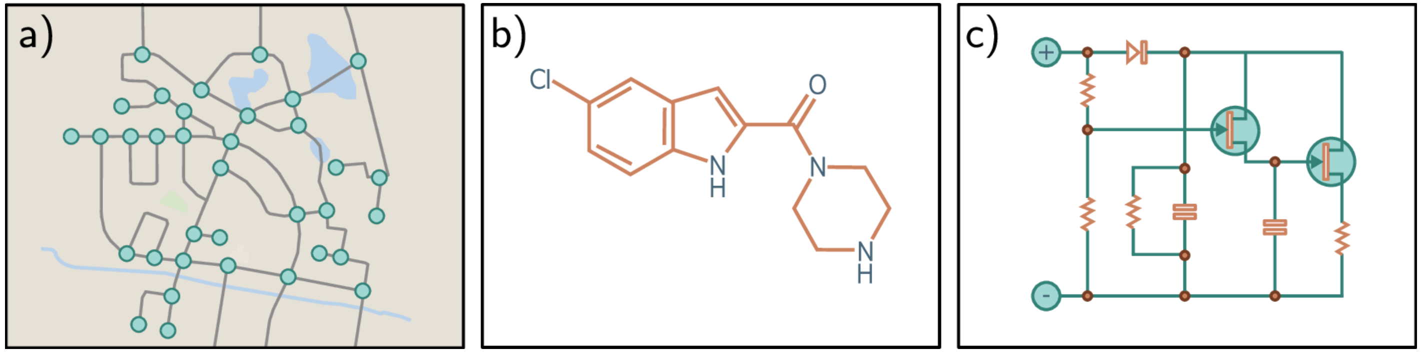


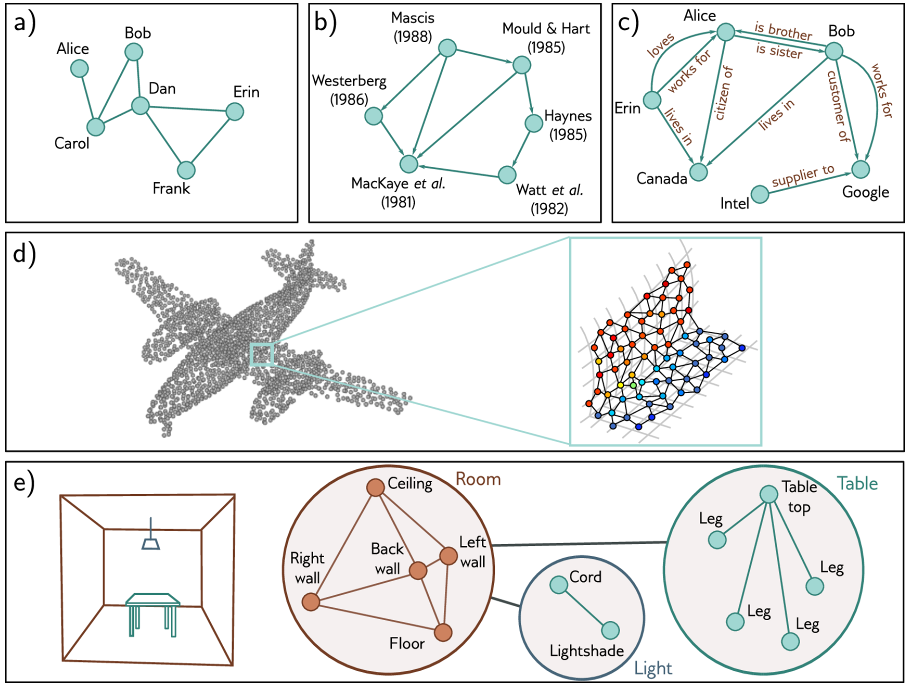
Graphs

Graphs are ubiquitous structures and consists of a set of nodes or vertices, where pairs of nodes are connected by edges.

Examples: road networks, chemical molecules are small graphs, electrical circuits and social networks, scientific literature, point clouds, but also any unordered list and images.

Types: directed or undirected graphs, heterogeneous, multigraphs, sparse, geometric, hierarchical.

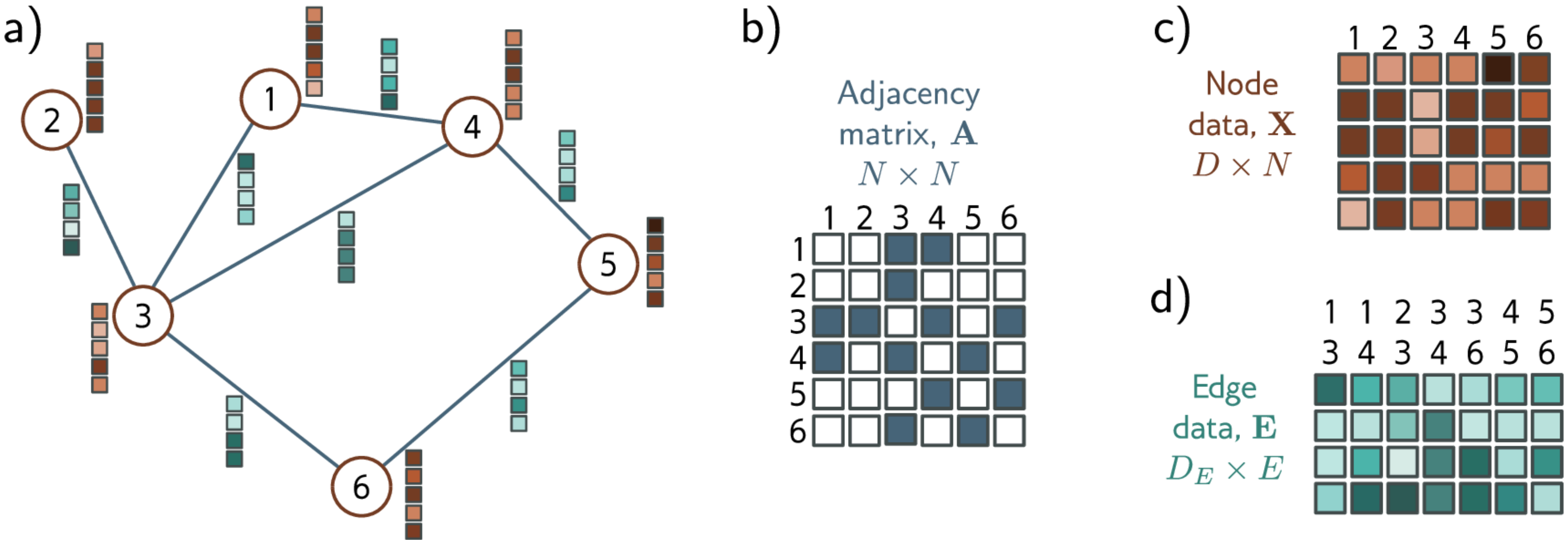




Graphs

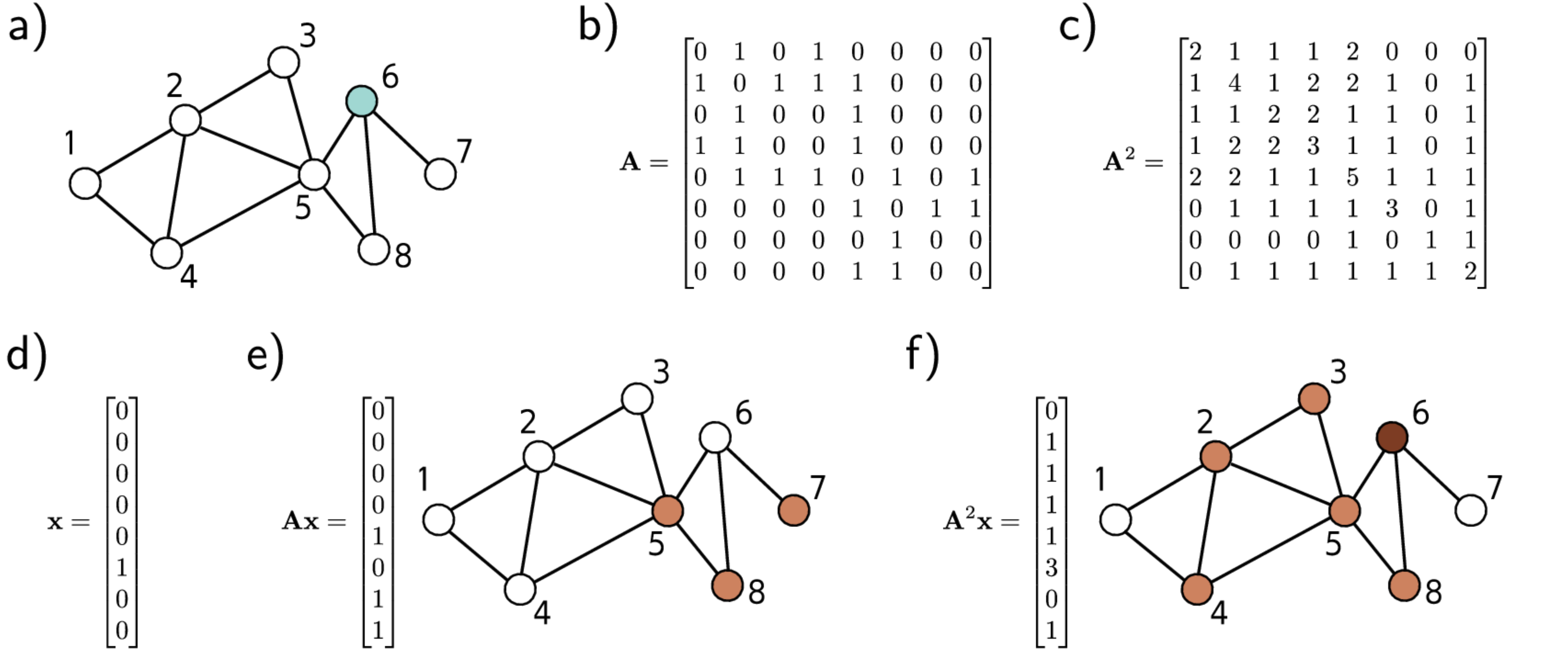
A graph consists of a set of N nodes connected by a set of E edges. The graph can be encoded by:

- $\mathbf{A} \in \mathbb{R}^{N\times N}$, the adjacency matrix such that $\mathbf{A}_{(m,n)} = \delta_{mn}$ (symmetric for undirected graphs),
- $\mathbf{X} \in \mathbb{R}^{D\times N}$, the node embedding, e.g. the n-th node has an associated embedding $\mathbf{x}^{(n)} \in \mathbb{R}^{D}$,
- $\mathbf{E} \in \mathbb{R}^{D_E\times E}$, the edge embedding, e.g. the e-th edge has an associated embedding $\mathbf{e}^{(e)} \in \mathbb{R}^{D_E}$.



Graphs

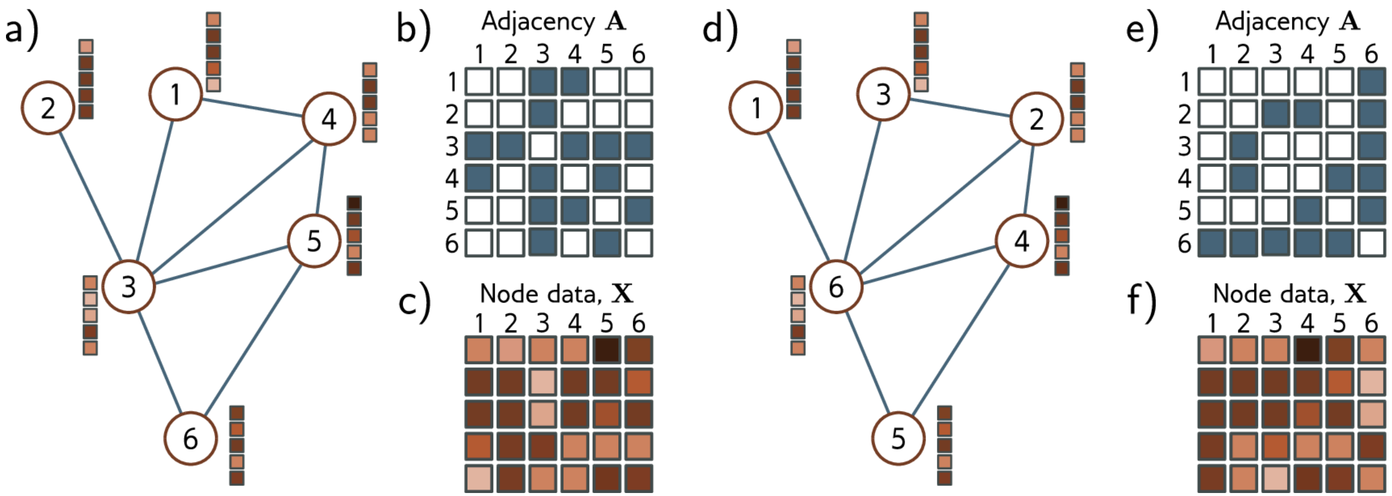
Remark: Pre-multiplying the one-hot vector for the n-th node by $\mathbf{A}$ we obtain the 1-hop distance nodes.
$\rightsquigarrow$ The entry at position (m, n) of $\mathbf{A}^L$ gives the number of unique walks of length L from node m to n.



Graphs

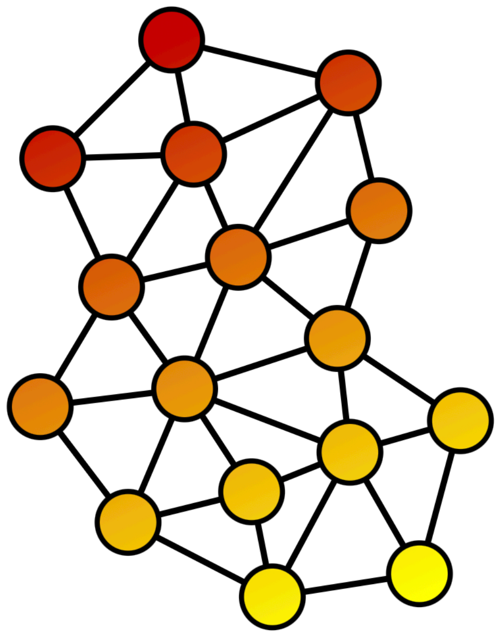
Node indexing in graphs is *arbitrary*, thus permuting node indices results in a permutation $\mathbf{P}$ of the columns of $\mathbf{X}$, and a permutation of both rows and columns of $\mathbf{A}$.

- The underlying graph is unchanged, in contrast to images or text.
- A permutation matrix is just a reordering of the identity matrix.
- If the entry (m, n) has value 1, then node m will become node n after the permutation.
- Post-multiplying by $\mathbb{P}$ permutes the columns and pre-multiplying by $\mathbb{P}^T$ permutes the rows.
- Any processing applied to the graph should also be indifferent to these permutations.



Graph Neural Networks (GNNs)

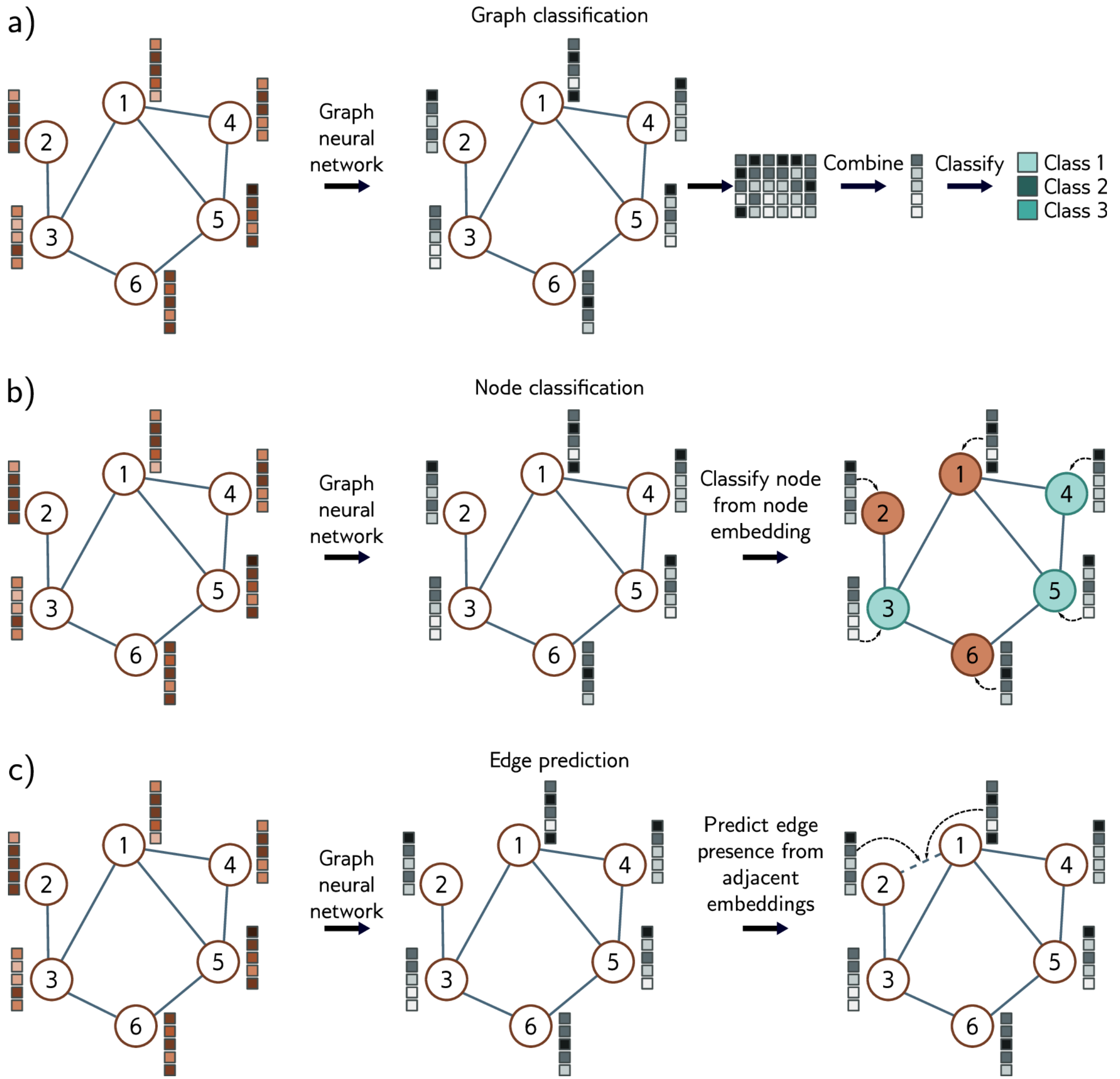
- A graph neural network is a model that takes the node embeddings $\mathbf{X}$ and the adjacency matrix $\mathbf{A}$ as inputs and passes them through a series of K layers.
- The node embeddings are updated at each layer to create intermediate “hidden” representations $\mathbf{H}_k$ before finally computing output embeddings $\mathbf{H}_K$.
- At the beginning, each column of $\mathbf{X}$ just contains information about the node itself.
- At the end, each column of $\mathbf{H}_K$ includes information about the node and its context within the graph.



Graph Neural Networks (GNNs)

Different kind of tasks can be performed:

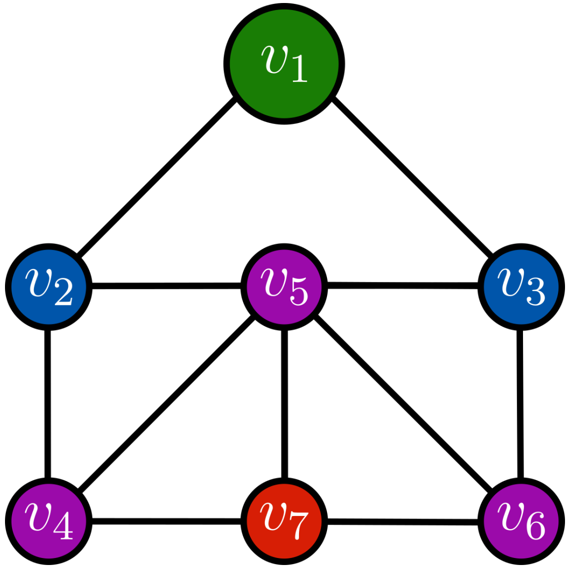
- *graph-level*, the network assigns a label or estimates one or more values from the entire graph;
- *node-level*, the network assigns a label (classification) or one or more values (regression) to each node of the graph;
- *edge-level*, the network predicts whether there should be an edge between nodes n and m.



Graph Neural Networks (GNNs) for unstructured meshes

Issue: Computational problems derived from physical models are naturally formulated over complex and possibly parametrized domains defined on unstructured meshes.
Idea: Embed geometrical information during the learning phase to obtain consistent results.
How: Geometric Deep Learning as a framework to augment NN capabilities with geometric priors.
$\rightsquigarrow$ A **GNN** is an optimizable transformation acting on all attributes of the mesh.

Let us consider the simple, undirected, and connected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$ and edges $\mathcal{E}$.



Graph Neural Networks (GNNs) for unstructured meshes

1. The nodes have associated features $\mathbf{u}$, representing the *state variables* at the vertices of the mesh. This information can be embedded in the matrix $\mathbf{U} = [\mathbf{u}_1 | \dots | \mathbf{u}_{N_h}]^T \in \mathbb{R}^{N_h \times d}$, for all the N_h nodes of the graph, which have no particular order, and d features (scalar or vector fields).
2. The *adjacency* matrix $\mathbf{A} \in \mathbb{R}^{N_h \times N_h}$ records of the connections among nodes. A convenient way of expressing it is by means of the adjacency list, where the k-th entry, corresponding to $e_k \in \mathcal{E}$, is the pair (i, j) denoting the existence of a link between $v_i, v_j \in \mathcal{V}$.
3. Define operations that are *permutation-invariant*. A random permutation of the original ordering of the nodes induced by the mesh labelling does not change the final output. This is a key difference with standard CNNs, where a fixed filter produces different outputs if two pixels are swapped.

Message passing layers

- Propagate the information to local neighbors of each node v, denoted as $\mathsf{N}(v)$ with degree $|\mathsf{N}(v)|$.
- Exchange messages between nodes at different k-hops (layers), and update them via NNs.
- At the k-th layer, we compute the hidden embedding $\mathbf{h}_u^{(k)} \in \mathbb{R}^{d^{(k)}}$ of the node u, representing a transformation of its original features by means of differentiable aggregate and update computations.

1. At a node $u \in \mathcal{V}$, one assembles the messages to be sent through the operation $\mathfrak{m}^{(k)}$ as

$$\mathbf{m}_v^{(k)} = \mathfrak{m}^{(k)}(\mathbf{h}_v^{(k-1)}), \quad \text{from each node } v \in \mathsf{N}(u)$$

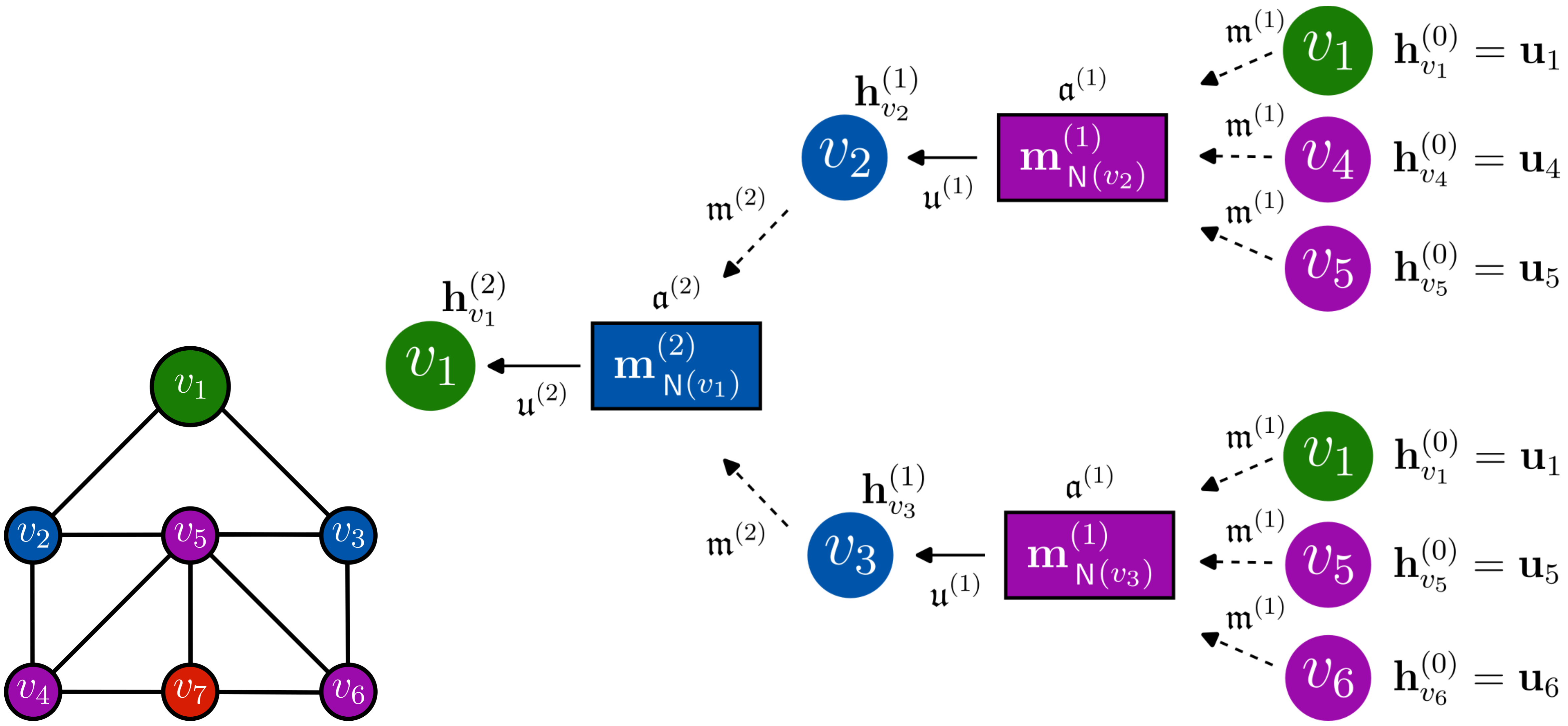
2. aggregates them with $\mathfrak{a}^{(k)}$ in

$$\mathbf{m}_{\mathsf{N}(u)}^{(k)} = \mathfrak{a}^{(k)}(\{\mathbf{m}_v^{(k)}, \ \forall v \in \mathsf{N}(u)\}),$$

3. and finally updates the hidden embedding by means of the function $\mathfrak{u}^{(k)}$

$$\mathbf{h}_u^{(k)} = \mathfrak{u}^{(k)}(\mathbf{m}_{\mathsf{N}(u)}^{(k)}).$$

Message passing layers



Message passing layers

Remarks

- For each node $v_j \in \mathcal{V}$, the initialization of the hidden embedding are defined as $\mathbf{h}_{v_j}^{(0)} = \mathbf{u}_j$.
- Residual information by considering ghost self-edges, such that $\mathsf{N}(u)$ contains the node u itself.
- The simplest example of message function $\mathfrak{m}^{(k)}$ is the multiplication of the hidden embedding with a weight matrix $\mathbf{W}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$, that is $\mathbf{m}_v^{(k)} = \mathbf{W}^{(k)} \mathbf{h}_v^{(k-1)}$ (shared weights).
- For the aggregate function $\mathfrak{a}^{(k)}$, one usually considers the normalized sum of the neighbor embeddings given by $\mathbf{m}_{\mathsf{N}(u)}^{(k)} = \sum_{v \in \mathsf{N}(u)} \frac{\mathbf{m}_v^{(k)}}{|\mathsf{N}(u)|}$, to better balance nodes with much higher degree.
- The update operation $\mathfrak{u}^{(k)}$ can be seen as the nonlinear activation function $\mathbf{h}_u^{(k)} = \sigma(\mathbf{m}_{\mathsf{N}(u)}^{(k)})$ in the NN context, with $\sigma(x) = \mathrm{ReLU}(x)$ or $\sigma(x) = \tanh(x)$.

Message passing layers

A basic GNN with K layers can be described by the iteration to update the node embeddings

$$\mathbf{h}_u^{(k)} = \sigma\left(\frac{1}{|\mathsf{N}(u)|}\sum_{v\in\mathsf{N}(u)} \mathbf{W}^{(k)}\mathbf{h}_v^{(k-1)}\right) \quad \text{for } k = 1,\ldots,K, \text{ and } u \in \mathcal{V},$$

or in compact graph-level notation

$$\mathbf{H}^{(k)} = \sigma\left(\mathbf{D}^{-1}(\mathbf{A}+\mathbf{I})\mathbf{H}^{(k-1)}{\mathbf{W}^{(k)}}^T\right) \quad \text{for } k = 1,\ldots,K,$$

where $\mathbf{H}^{(k)} = \left[\mathbf{h}_{v_1}^{(k)}|\ldots|\mathbf{h}_{v_{N_h}}^{(k)}\right]^T \in \mathbb{R}^{N_h\times d^{(k)}}$ expresses the matrix with the hidden embedding for each node taken as row, $\mathbf{D}^{-1}$ is the diagonal matrix with the inverse of the degree of each node $|\mathsf{N}(v)|$, and $\mathbf{A}, \mathbf{I}$ are the adjacency and the identity matrices (representing the self-edges), respectively.

Graph Convolutional Networks

- A graph convolutional network (GCN) is a specific type of GNN extending the concept of CNNs.
- Instead of having a fixed filter sliding over the pixels (with non invariant operations), here we have different cardinality of each node's neighbors.
- Two classes of convolutional layers: *spectral* and *spatial* ones.
- **Spectral convolutions** apply spectral/Fourier analysis of the graph Laplacian matrix (signal processing). They build the filters as $g_{\mathbf{w}} * \mathbf{H} \doteq \mathbf{U} g_{\mathbf{w}} \mathbf{U}^T \mathbf{H}$, where $g_{\mathbf{w}}$ is the filter, and $\mathbf{U}$ the eigenvector matrix of the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$ [*GCN*, *ChebNet*].
- **Spatial convolutions** are local rather than global, they do not depend on the graph's dimensionality, and result in faster and more efficient methods [*GraphSage*, *GAT*, *MoNet*].

MoNet spatial convolution

Let's consider the **MoNet** spatial framework, which can be interpreted as a *Gaussian Mixture Model* (GMM), generalizing convolutions in non-Euclidean domains.

- MoNet builds a set of pseudo-coordinates $\mathbf{e}$ used to define the weights of an optimizable Gaussian kernel with Q filters, through the iteration procedure

$$\mathbf{h}_u = \frac{1}{|\mathsf{N}(u)|} \sum_{v \in \mathsf{N}(u)} \frac{1}{Q} \sum_{q=1}^{Q} \boldsymbol{\omega}^q(\mathbf{e}_u) \odot \mathbf{W}^q \mathbf{h}_v,$$

where $\odot$ is the element-wise multiplication, and $\boldsymbol{\omega}^q$ is the weighting function defined in terms of a trainable mean vector $\boldsymbol{\mu}_q$ and a diagonal covariance matrix $\boldsymbol{\Sigma}_q$ as

$$\boldsymbol{\omega}^q(\mathbf{e}_u) = \exp\left(-\frac{1}{2}(\mathbf{e}_u - \boldsymbol{\mu}_q)^T \boldsymbol{\Sigma}_q^{-1} (\mathbf{e}_u - \boldsymbol{\mu}_q)\right).$$

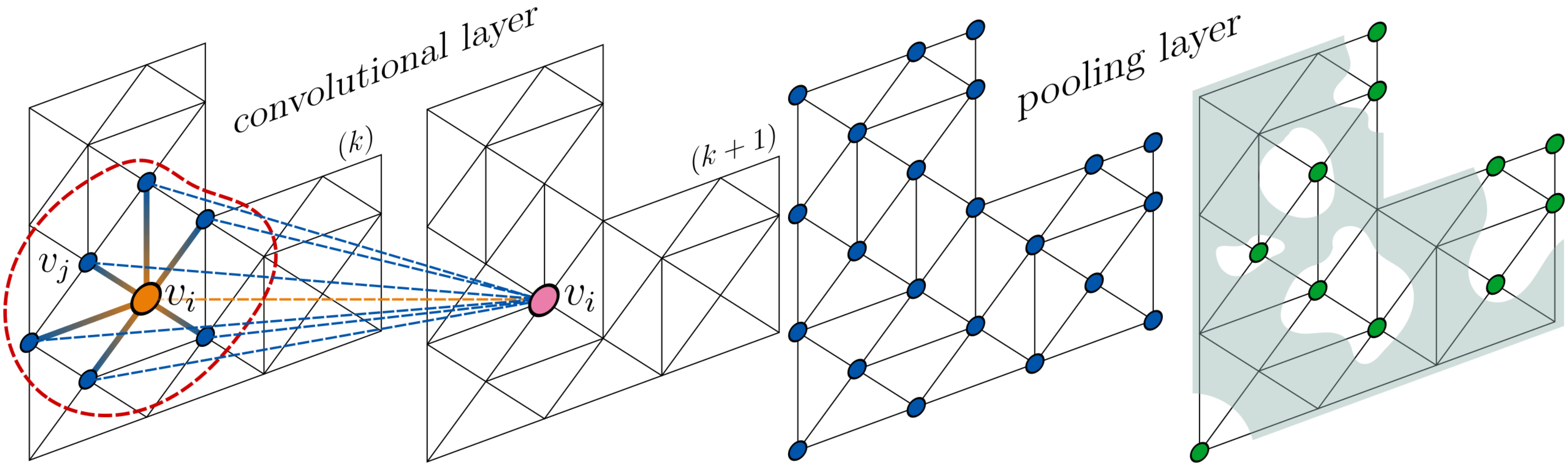
- Introducing a geometric bias in the learning process by edge attributes given by the distance between two connected nodes.

Pooling and Unpooling

- A key difference between CNNs and GNNs consists in the property of the former to naturally *reduce* the spatial dimension of the input. Indeed, convolutional layers only define an updated state for each node of the original graph/mesh.
- For autoencoding purposes one need to perform *down-sampling* and *up-sampling*, to obtain coarser and finer representation of the mesh.
- **Pooling** is a way to down-sample the size of the input by aggregating information.
- Not straightforward due to issue with hierarchy, but many ways to perform it [*random mask*, *clustering*, *attention*, *algebraic multigrid*, *top-k*].
- **Un-pooling** is the operation to reconstruct the original signal, but one way to perform it.
- PointNet++ is based on a k-NN interpolation such that, given a node at position $\mathbf{x}_i$, we define its feature vector $\mathbf{u}_i$ as the weighted interpolation w.r.t. its k neighbors given by

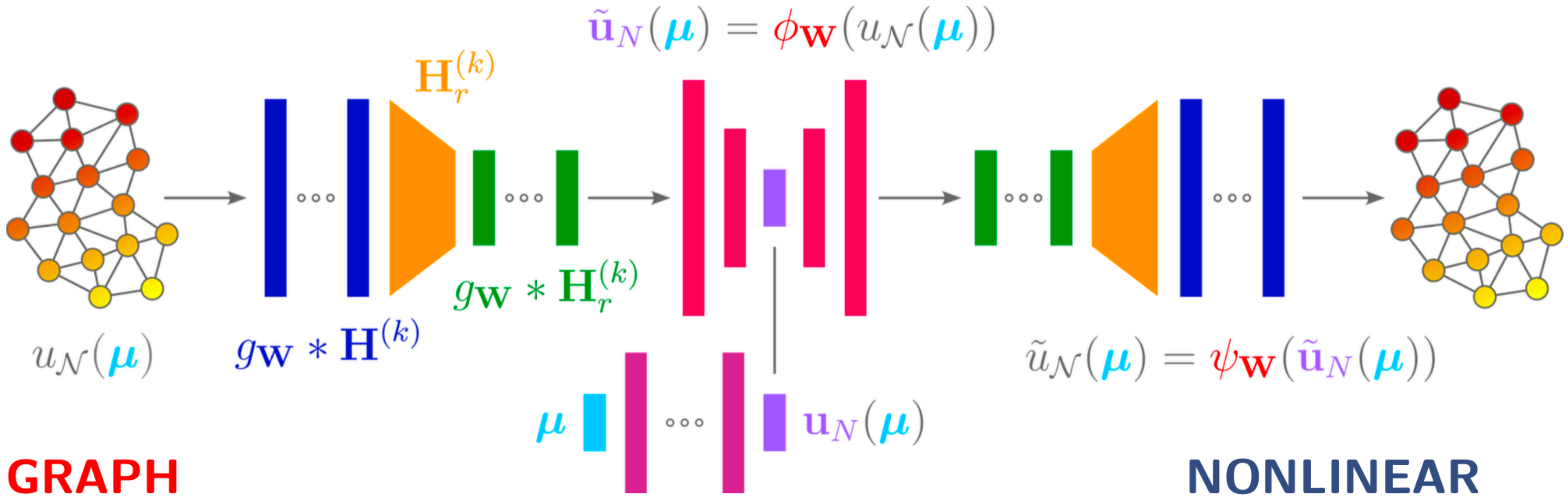
$$\mathbf{u}_i = \frac{\sum_{j=1}^{k} \xi(\mathbf{x}_j)\mathbf{u}_j}{\sum_{j=1}^{k} \xi(\mathbf{x}_j)}, \qquad \text{where} \quad \xi(\mathbf{x}_j) = \frac{1}{d(\mathbf{x}_i, \mathbf{x}_j)^2}.$$

Convolutional and pooling layers



Graph Convolutional Autoencoder

Goal: Efficient and geometrically-consistent nonlinear ROM for parametrized PDEs defined on unstructured meshes of complex and varying domains.



GRAPH CONVOLUTIONAL AUTOENCODER

NONLINEAR REDUCTION

Graph Convolutional Autoencoder

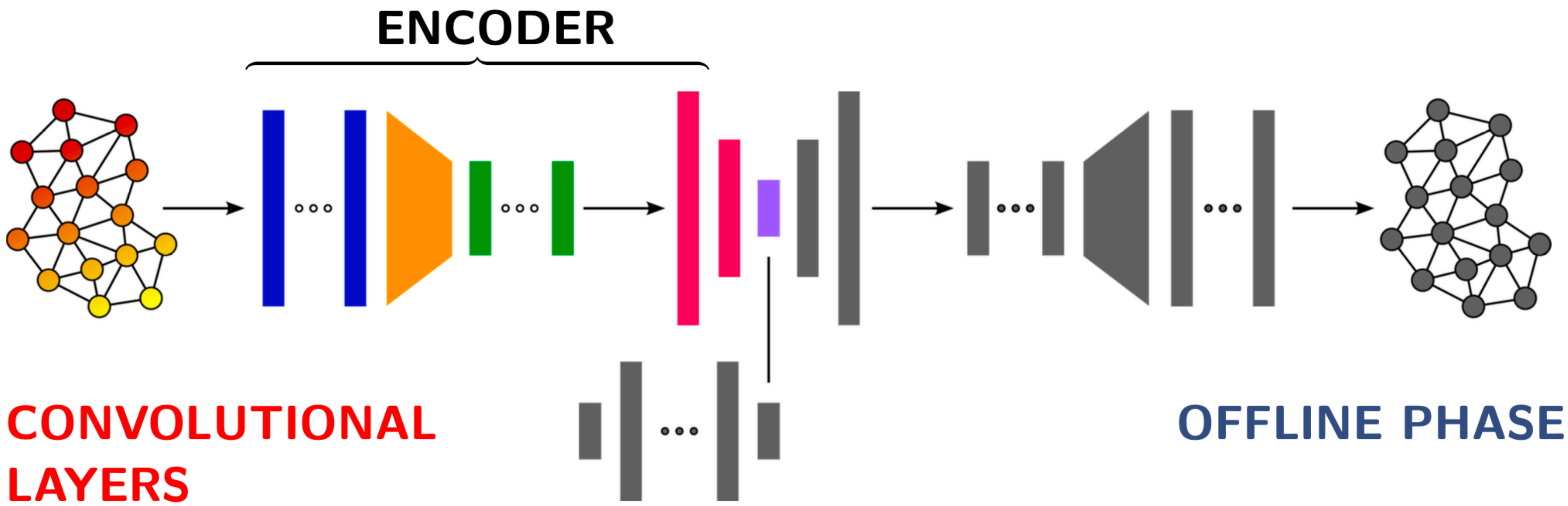
GRAPH
CONVOLUTIONAL
AUTOENCODER

OFFLINE PHASE

Graph Convolutional Autoencoder

$$\{u_{\mathcal{N}}(\boldsymbol{\mu}^i), \Omega_{\mathcal{N}}(\boldsymbol{\mu}^i)\}_{i=1}^{M}$$

Graph Convolutional Autoencoder

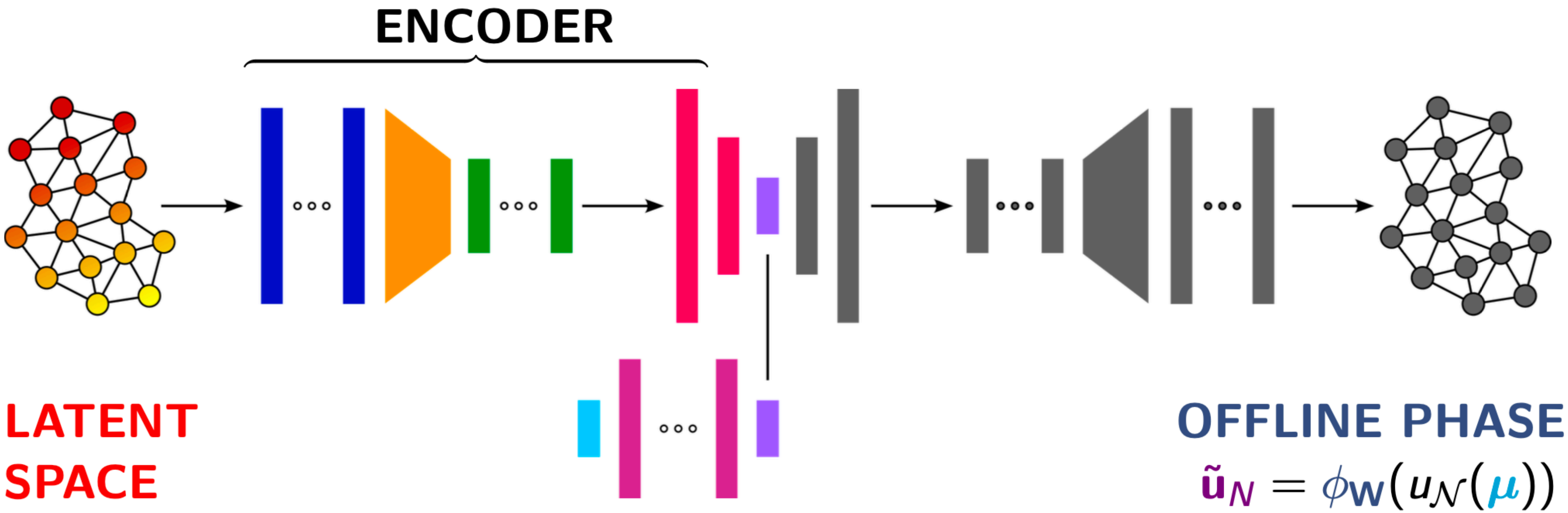


CONVOLUTIONAL LAYERS

OFFLINE PHASE

$$\tilde{\mathbf{u}}_N = \phi_{\mathbf{w}}(u_{\mathcal{N}}(\boldsymbol{\mu}))$$

Graph Convolutional Autoencoder



LATENT SPACE MAP

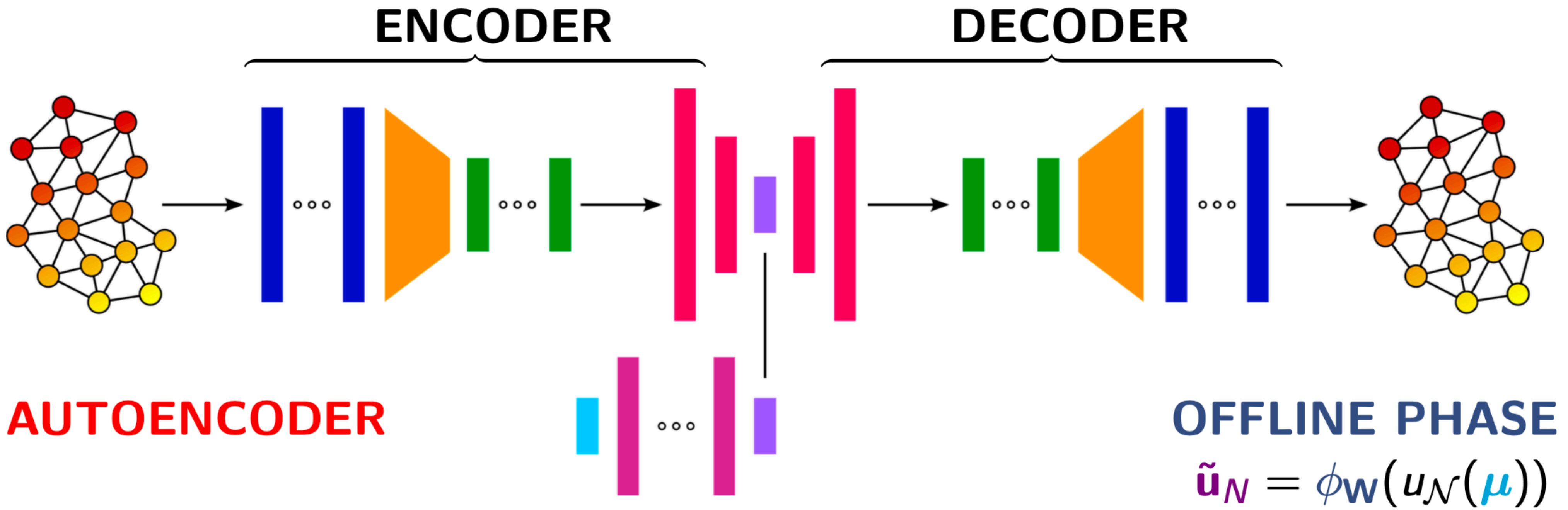
OFFLINE PHASE

$$\tilde{\mathbf{u}}_N = \phi_{\mathbf{W}}(u_{\mathcal{N}}(\boldsymbol{\mu}))$$

$$\mathbf{u}_N = \mathrm{MLP}(\boldsymbol{\mu})$$

LOSS: $\mathcal{L} = \frac{1}{N_{\mathrm{tr}}} \sum_{i=1}^{N_{\mathrm{tr}}} \left\| \mathbf{u}_N(\boldsymbol{\mu}^i) - \tilde{\mathbf{u}}_N(\boldsymbol{\mu}^i) \right\|_2^2$

Graph Convolutional Autoencoder

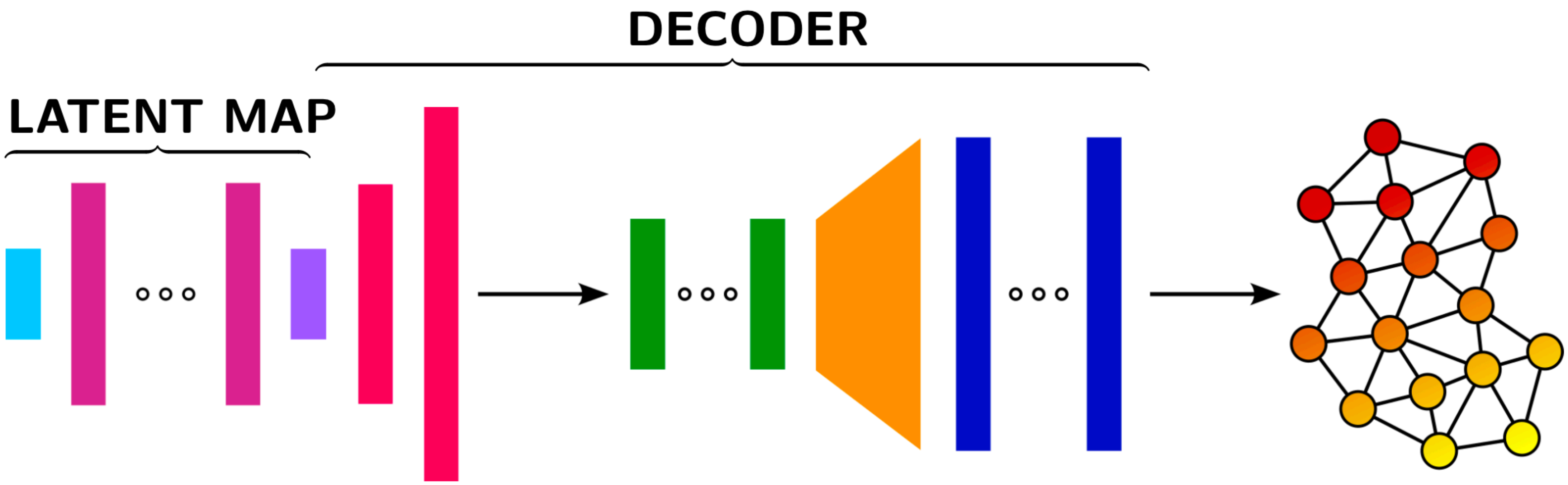


$$\tilde{\mathbf{u}}_N = \phi_{\mathbf{W}}(u_{\mathcal{N}}(\boldsymbol{\mu}))$$

$$\tilde{u}_{\mathcal{N}}(\boldsymbol{\mu}) = \psi_{\mathbf{W}}(\tilde{\mathbf{u}}_N(\boldsymbol{\mu}))$$

LOSS: $$\mathcal{L} = \frac{1}{N_{\text{tr}}} \sum_{i=1}^{N_{\text{tr}}} \| \mathbf{u}_N(\boldsymbol{\mu}^i) - \tilde{\mathbf{u}}_N(\boldsymbol{\mu}^i) \|_2^2 + \frac{1}{N_{\text{tr}}} \sum_{i=1}^{N_{\text{tr}}} \| \tilde{u}_{\mathcal{N}}(\boldsymbol{\mu}^i) - u_{\mathcal{N}}(\boldsymbol{\mu}^i) \|_2^2$$

Graph Convolutional Autoencoder



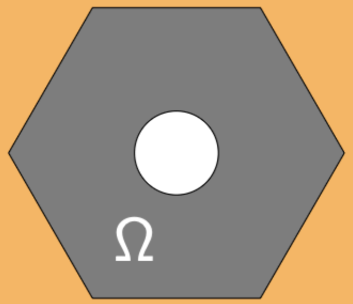
NONLINEAR ROM

ONLINE PHASE

$$\boldsymbol{\mu} \mapsto \mathbf{u}_N(\boldsymbol{\mu}) \quad \rightsquigarrow \quad \tilde{u}_{\mathcal{N}}(\boldsymbol{\mu}) = \psi_{\mathbf{W}}(\mathbf{u}_N(\boldsymbol{\mu}))$$

Nonlinear Poisson problem

$$\begin{cases} -\Delta u(\boldsymbol{\mu}) + \mu_0 \frac{e^{\mu_1 u(\boldsymbol{\mu})}-1}{\mu_1} = g & \text{in } \Omega, \\ u(\boldsymbol{\mu}) = 0 & \text{on } \partial\Omega, \end{cases}$$



Parameters:

- $\mu_0 \in [0.01, 10]$ nonlin. magnitude
- $\mu_1 \in [0.01, 10]$ sink's strength
- $N_h = 2562$ dofs
- $M = 100$ snapshots

Keywords:

- nonlinear term
- holed domain

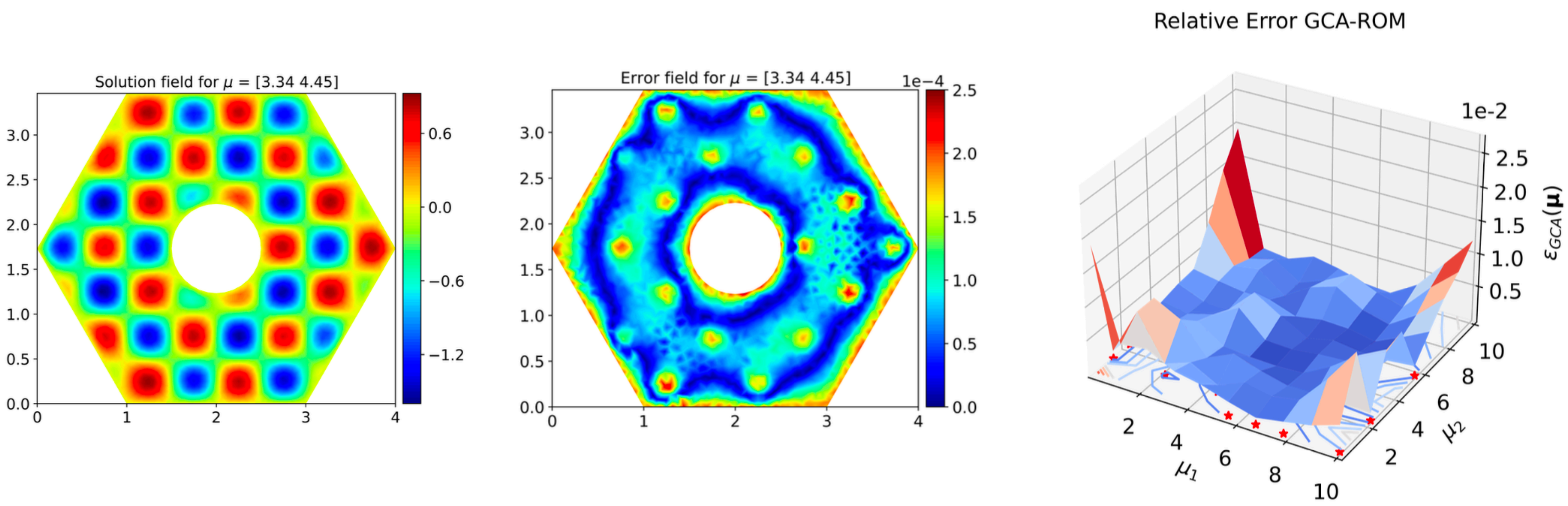
Nonlinear Poisson problem

Hyperparameters:

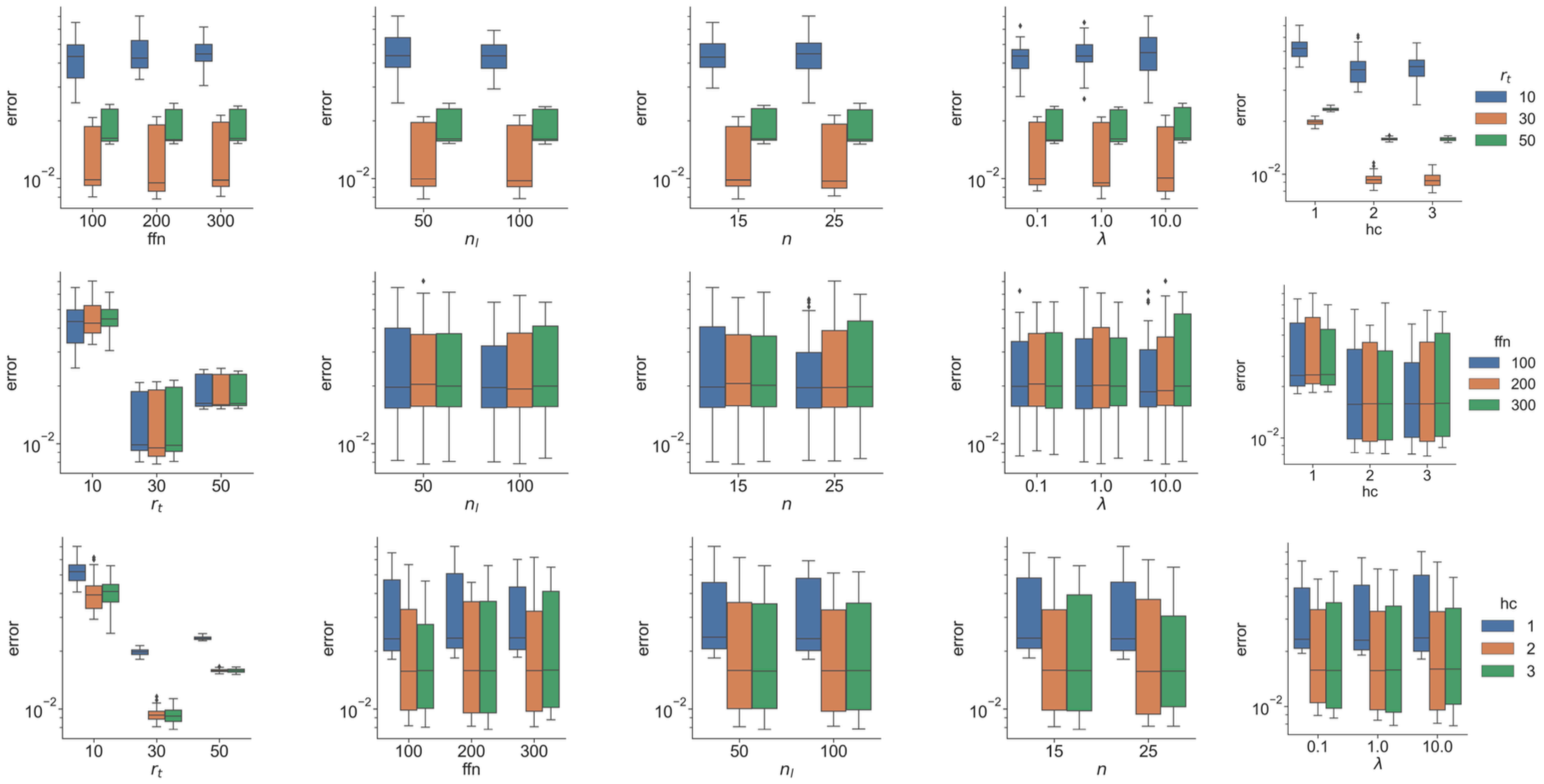
- train rate $r_t = 30\%$,
- latent $n = 15$, epochs $N_{\text{ep}} = 5000$

Relative errors:

- mean: $7.82 \cdot 10^{-3}$
- max: $2.08 \cdot 10^{-2}$



Nonlinear Poisson problem



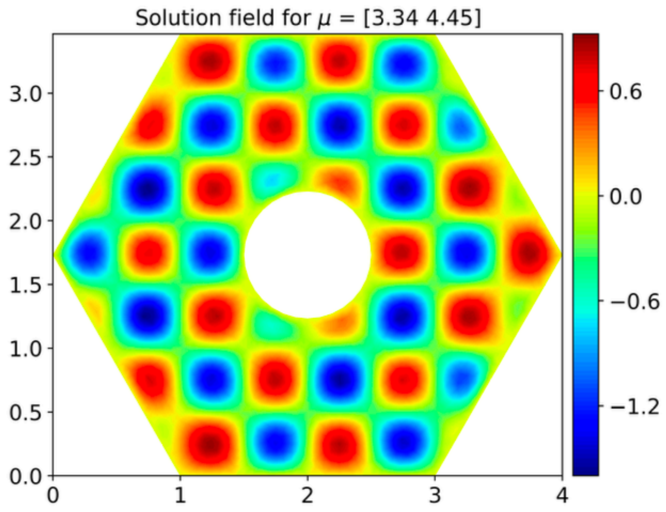
Nonlinear Poisson problem - with pooling

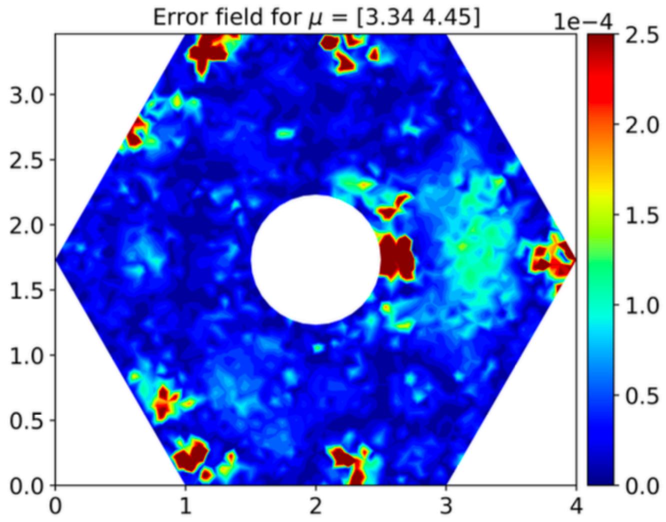
Hyperparameters:

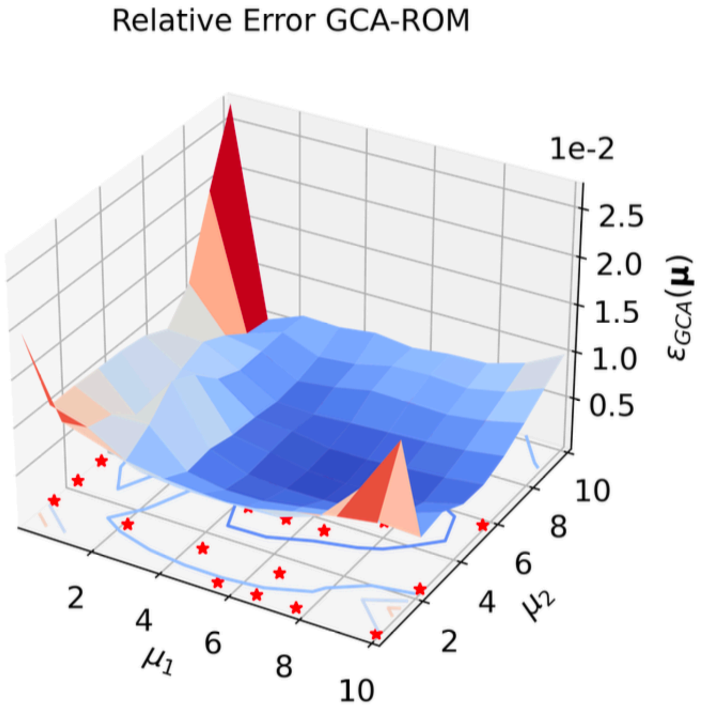
- train rate $r_t = 30\%$, pool rate $r_p = 70\%$
- latent $n = 15$, epochs $N_{ep} = 5000$

Relative errors:

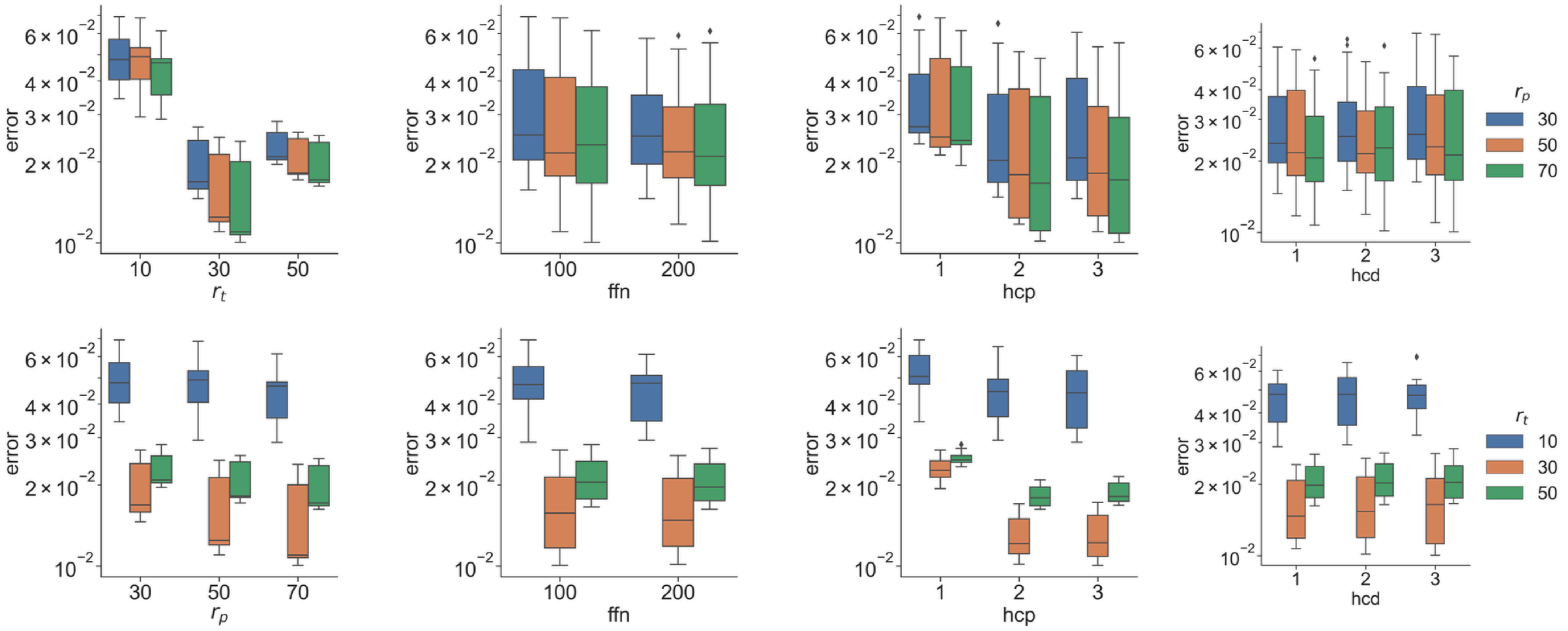
- mean: $1.01 \cdot 10^{-2}$
- max: $2.10 \cdot 10^{-2}$







Nonlinear Poisson problem - with pooling



Advection dominated problem

$$\begin{cases} -\frac{1}{Pe(\mu_0)}\Delta\, u(\boldsymbol{\mu}) + \boldsymbol{\beta}(\mu_1)\cdot\nabla u(\boldsymbol{\mu}) = g & \text{in } \Omega, \\ u(\boldsymbol{\mu}) = 0 & \text{on } \partial\Omega, \end{cases}$$

Parameters:

- $Pe(\mu_0) = 10^{\mu_0} \in [1, 10^6]$ Péclet
- $\mu_1 \in [-1, 1]$ advection's direction
- $N_h = 3967$ dofs

Keywords:

- slow K-decay
- boundary layers

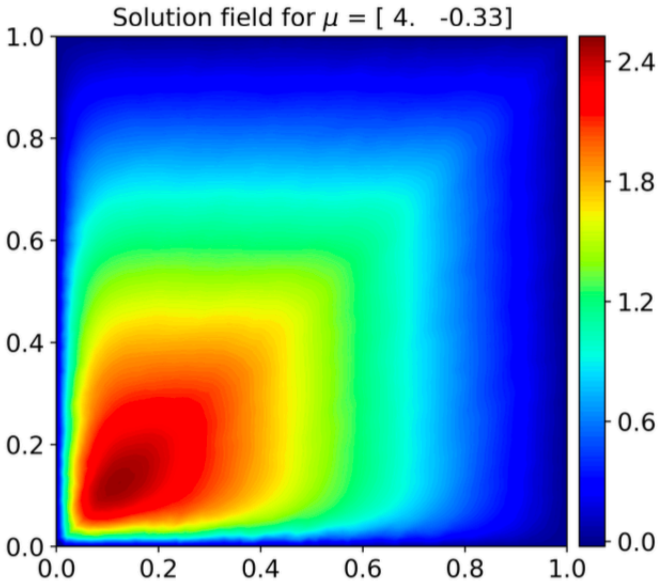
Advection dominated problem

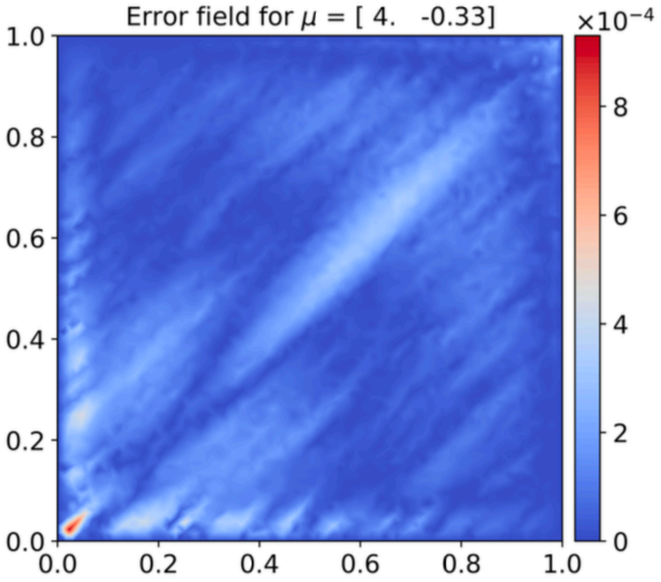
Hyperparameters:

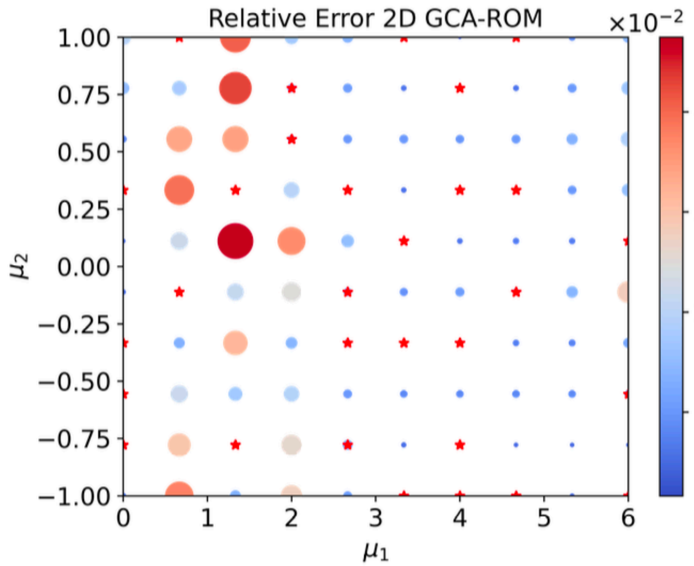
- $M = 100$ snapshots, train rate $r_t = 30\%$
- latent $n = 15$, epochs $N_{ep} = 5000$

Relative errors:

- mean: $2.37 \cdot 10^{-2}$
- max: $4.83 \cdot 10^{-2}$







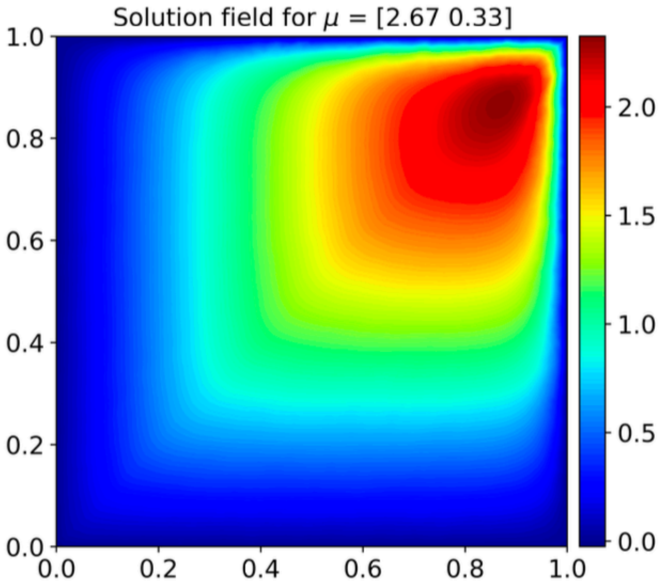
Advection dominated problem

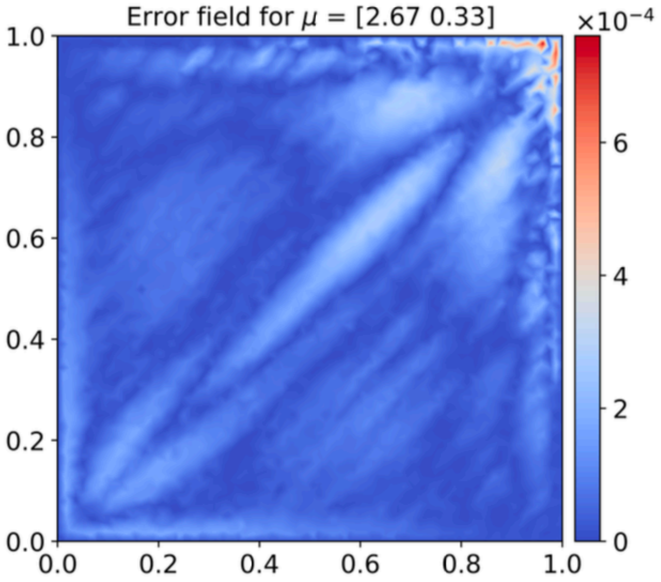
Hyperparameters:

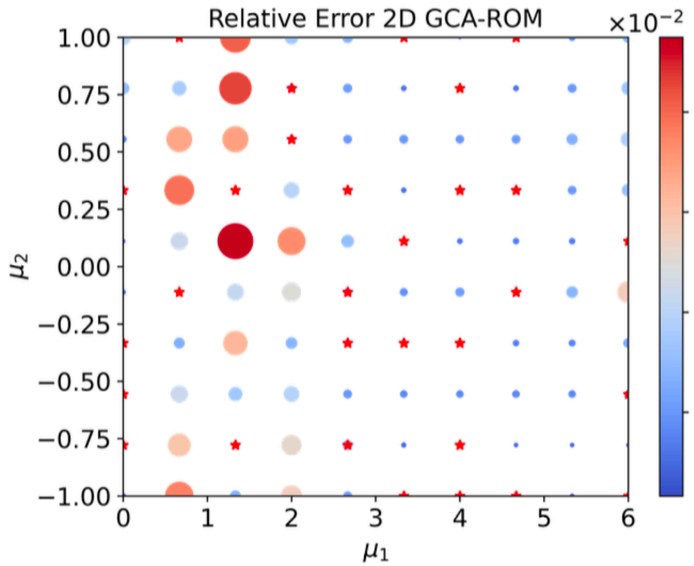
- $M = 100$ snapshots, train rate $r_t = 30\%$
- latent $n = 15$, epochs $N_{ep} = 5000$

Relative errors:

- mean: $2.37 \cdot 10^{-2}$
- max: $4.83 \cdot 10^{-2}$







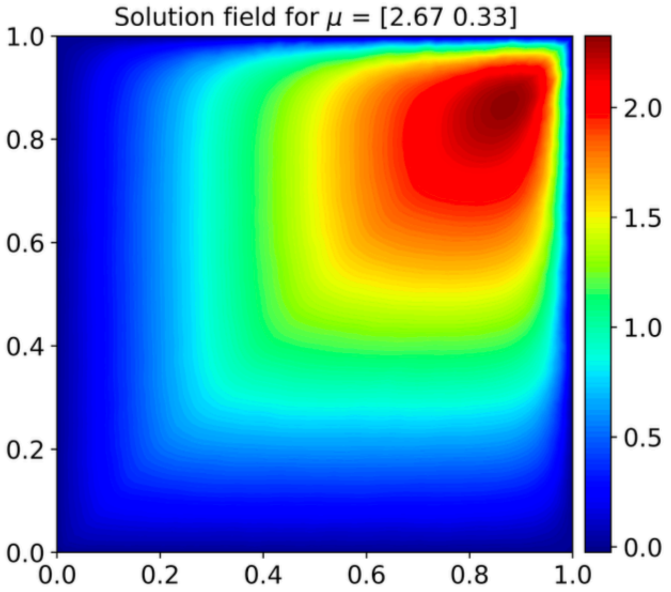
Advection dominated problem - with pooling

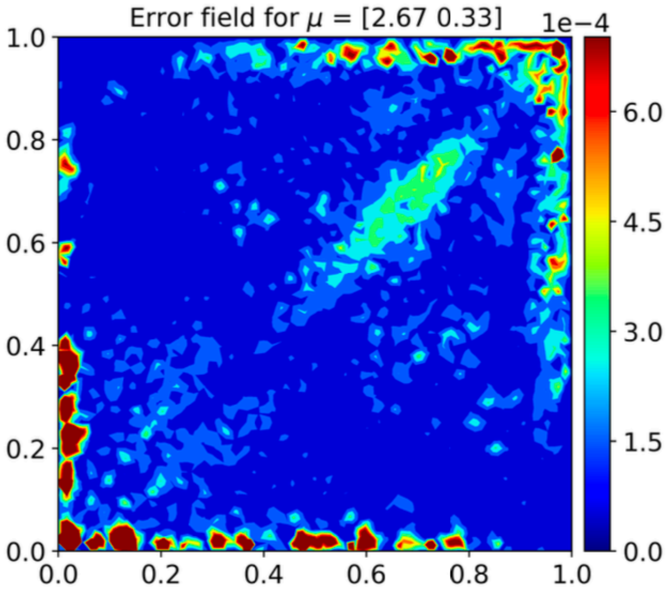
Hyperparameters:

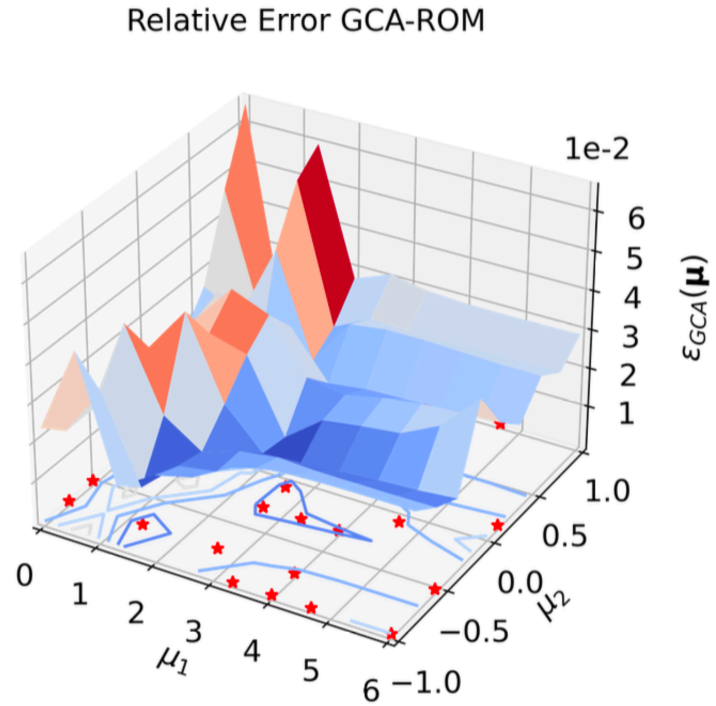
- train rate $r_t = 30\%$, pool rate $r_t = 70\%$
- latent $n = 15$, epochs $N_{ep} = 5000$

Relative errors:

- mean: $3.32 \cdot 10^{-2}$
- max: $7.15 \cdot 10^{-2}$

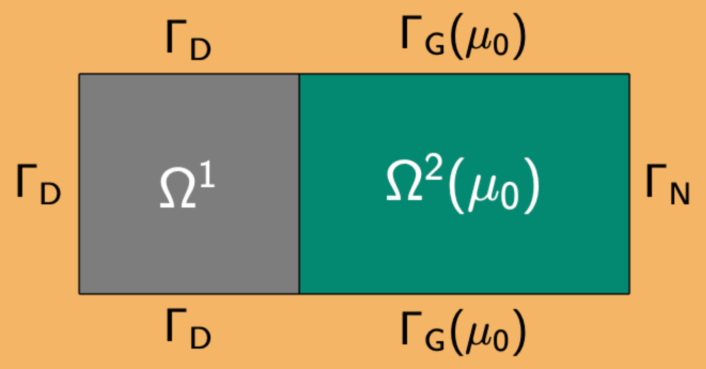






Graetz problem: parametrized geometry

$$
\begin{cases}
-\mu_1 \Delta u(\boldsymbol{\mu}) + x_1(1-x_1)\partial_{x_0} u(\boldsymbol{\mu}) = 0 & \text{in } \Omega(\mu_0),\\
u(\boldsymbol{\mu}) = 0 & \text{on } \Gamma_D,\\
u(\boldsymbol{\mu}) = 1 & \text{on } \Gamma_G,\\
\frac{\partial u}{\partial \boldsymbol{n}} = 0 & \text{on } \Gamma_N,
\end{cases}
$$



Parameters:

- $\mu_0 \in [1, 3]$ length of Ω_2
- $\mu_1 \in [0.01, 0.1]$ diffusivity constant
- $N_h = 5160$ dofs
- $M = 200$ snapshots

Keywords:

- param. geometry
- non-affine problem

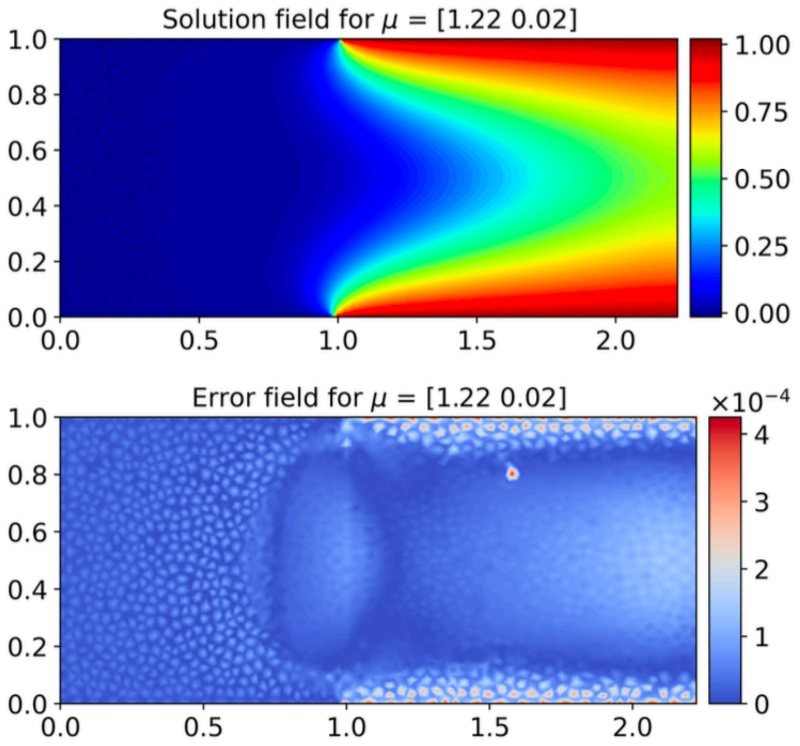
Graetz problem: parametrized geometry

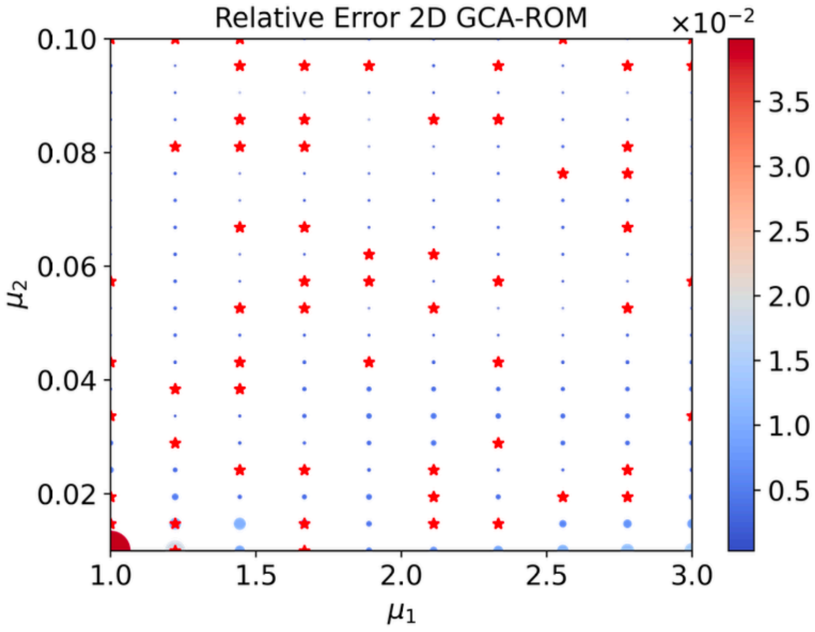
Hyperparameters:

- train rate $r_t = 30\%$
- latent $n = 25$, epochs $N_{ep} = 5000$

Relative errors:

- mean: $5.39 \cdot 10^{-3}$
- max: $3.98 \cdot 10^{-2}$





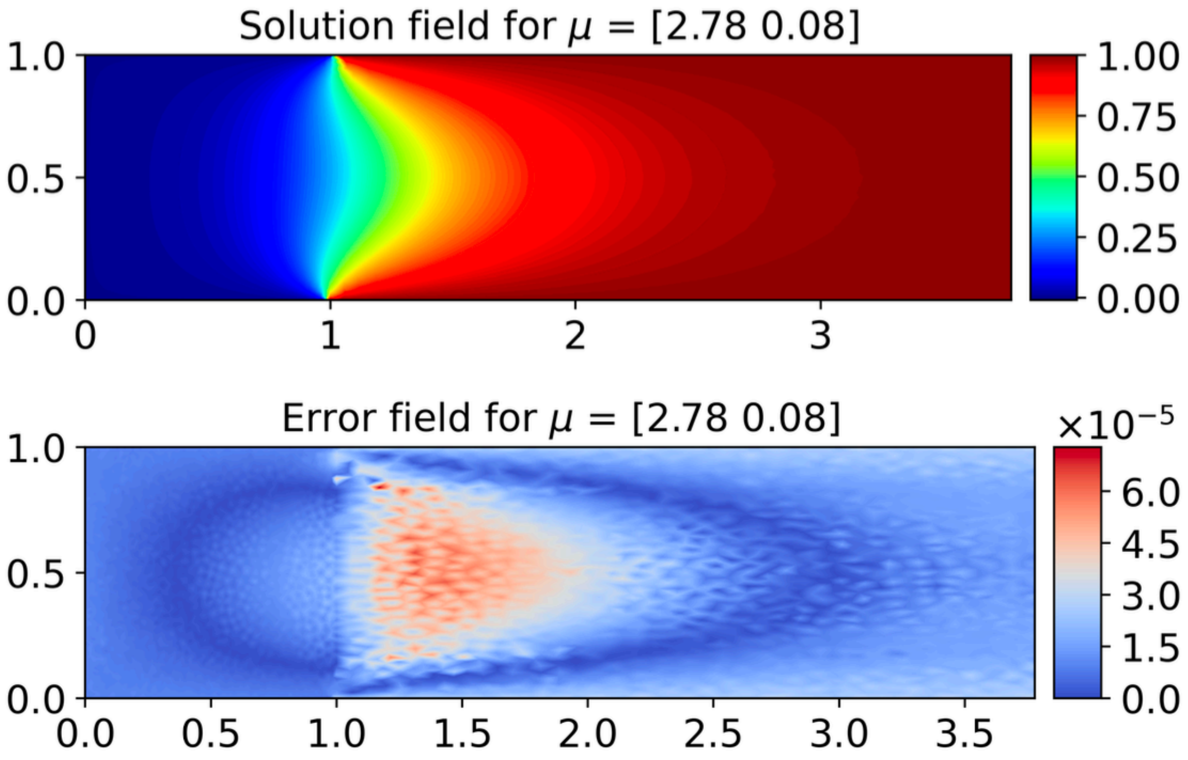
Graetz problem: parametrized geometry

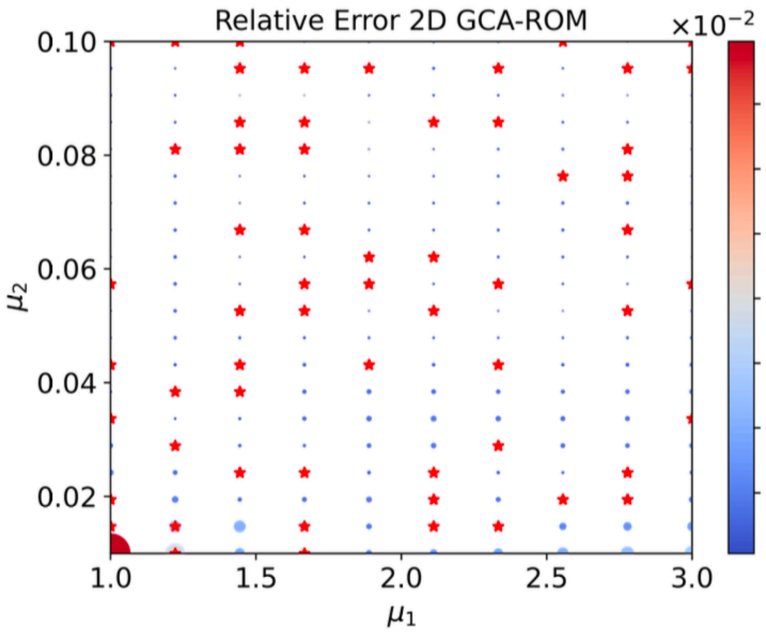
Hyperparameters:

- train rate $r_t = 30\%$
- latent $n = 25$, epochs $N_{ep} = 5000$

Relative errors:

- mean: $5.39 \cdot 10^{-3}$
- max: $3.98 \cdot 10^{-2}$





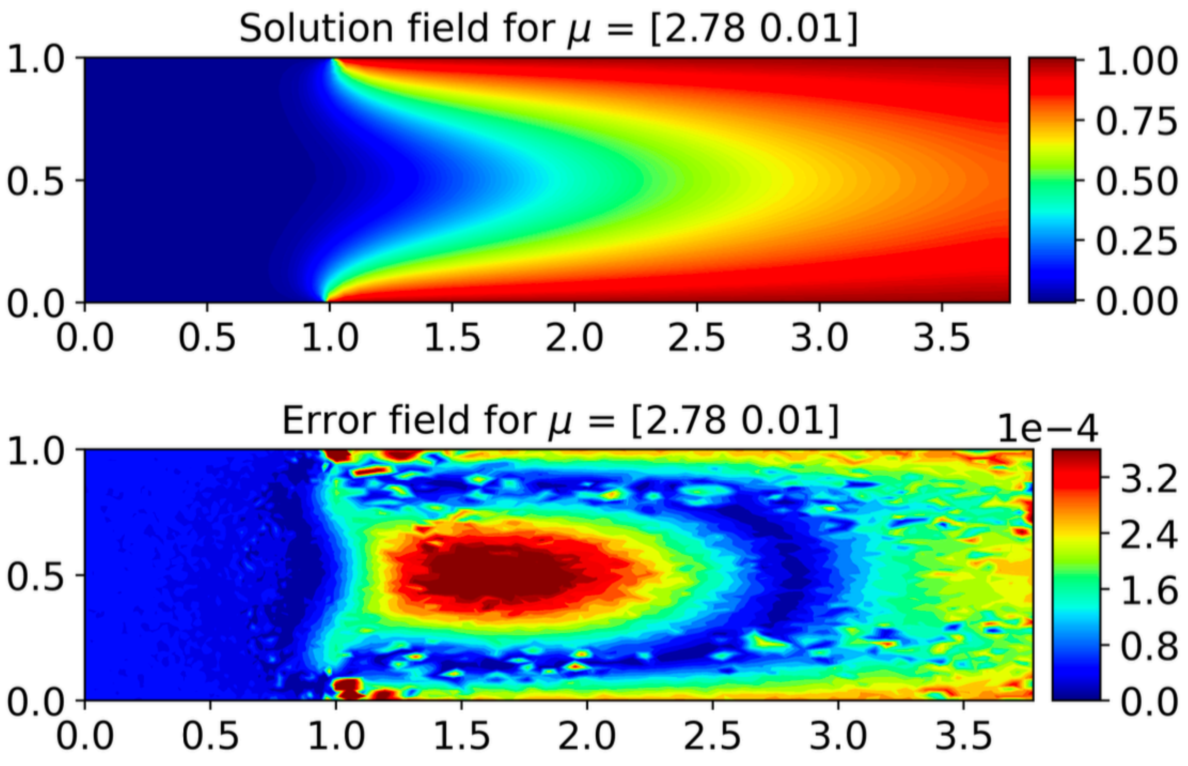
Graetz problem: parametrized geometry - with pooling

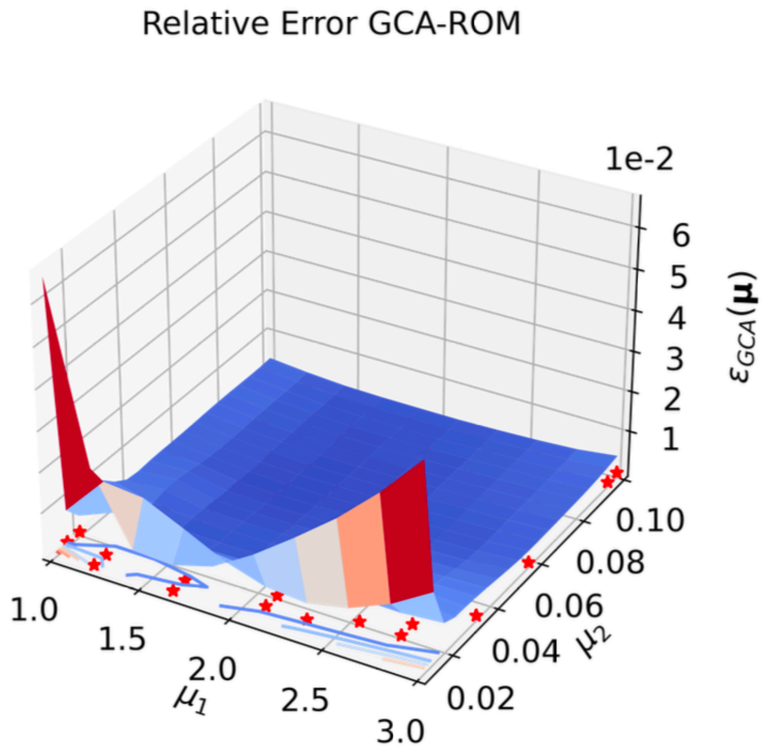
Hyperparameters:

- train rate $r_t = 30\%$, pool rate $r_p = 70\%$
- latent $n = 25$, epochs $N_{ep} = 5000$

Relative errors:

- mean: $6.97 \cdot 10^{-3}$
- max: $9.26 \cdot 10^{-2}$





Comparison with linear and nonlinear approaches

Table: Mean relative errors over Ξ_{te} with reduced/latent dimension $n = 15$.

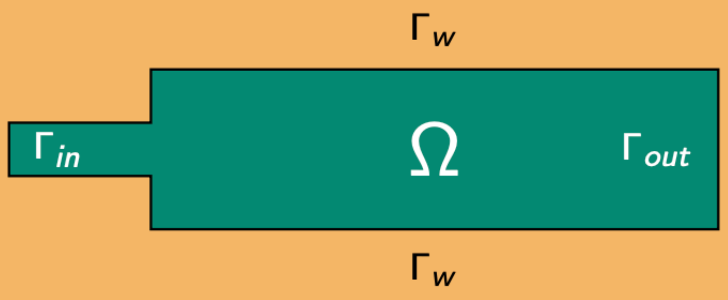
Application	POD	POD-G	DL-ROM	GCA-ROM
Poisson	9.9×10^{-5}	1.0×10^{-4}	1.5×10^{-2}	7.8×10^{-3}
Advection	3.1×10^{-2}	4.0×10^{-2}	5.0×10^{-2}	2.4×10^{-2}
Graetz	2.3×10^{-4}	2.4×10^{-4}	1.4×10^{-2}	6.8×10^{-3}

Table: Comparison between DL-ROM and GCA-ROM.

Method	Device	Filters	Parameters	Training time (s)	Testing time (s)
DL-ROM	CPU	3x3	8 476 109	66 518	9.49
		5x5	6 592 461	62 060	9.74
	GPU	3x3	8 476 109	1172	8.93
		5x5	6 592 461	1323	9.29
GCA-ROM	CPU	3	2 088 682	3967	13.94
		5	2 088 694	6944	14.86
	GPU	3	2 088 682	553	15.43
		5	2 088 694	714	14.92

Navier-Stokes: bifurcating problem

$$\begin{cases} -\mu_0 \Delta \mathbf{u}(\boldsymbol{\mu}) + (\mathbf{u}(\boldsymbol{\mu}) \cdot \nabla)\, \mathbf{u}(\boldsymbol{\mu}) + \nabla p(\boldsymbol{\mu}) = 0 & \text{in } \Omega(\mu_1), \\ \nabla \cdot \mathbf{u}(\boldsymbol{\mu}) = 0 & \text{in } \Omega(\mu_1), \\ \mathbf{u}(\boldsymbol{\mu}) = \mathbf{u}_{\text{in}} & \text{on } \Gamma_{\text{in}}, \\ \mathbf{u}(\boldsymbol{\mu}) = \mathbf{0} & \text{on } \Gamma_{\text{w}}(\mu_1), \\ \mu_0 \frac{\partial \mathbf{u}}{\partial \mathbf{n}}(\boldsymbol{\mu}) - p(\boldsymbol{\mu})\, \mathbf{n} = 0 & \text{on } \Gamma_{\text{out}}, \end{cases}$$



Parameters:

- $\mu_0 \in [0.5, 2]$ kinematic viscosity
- $\mu_1 \in [0.5, 2]$ inlet's width
- $N_h = 8157$ dofs

Keywords:

- param. geometry
- nonlinear terms
- vector problem

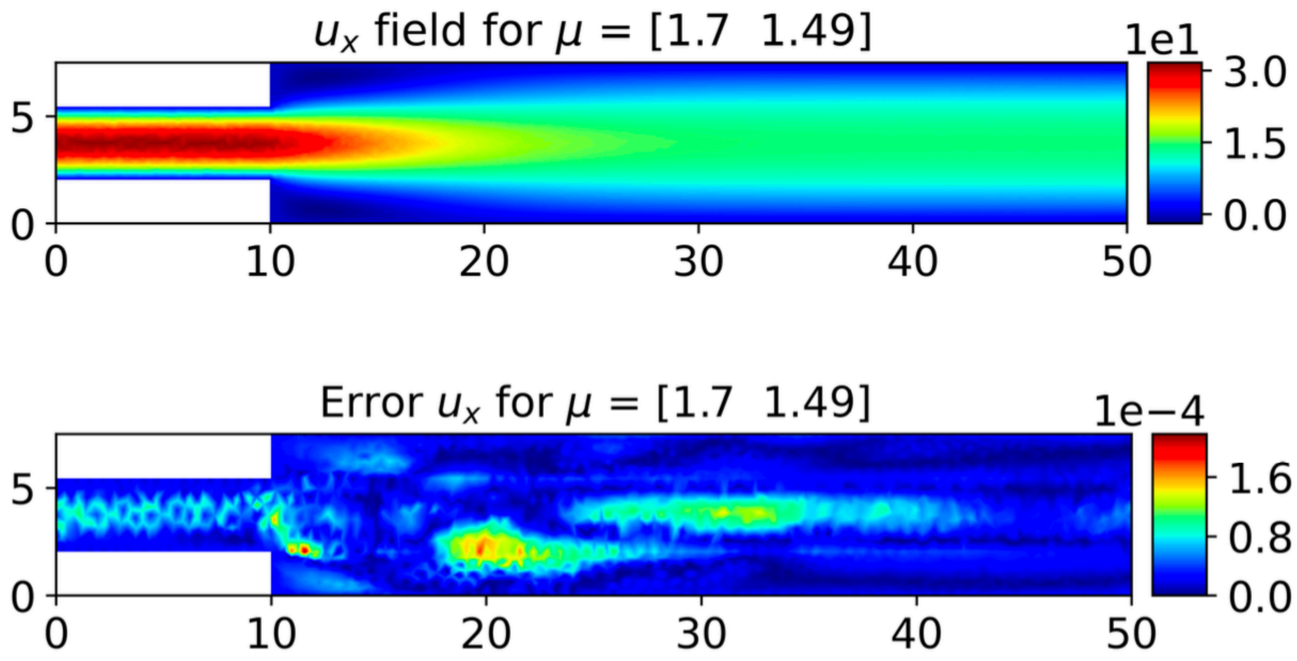
Navier-Stokes: bifurcating problem - u_x

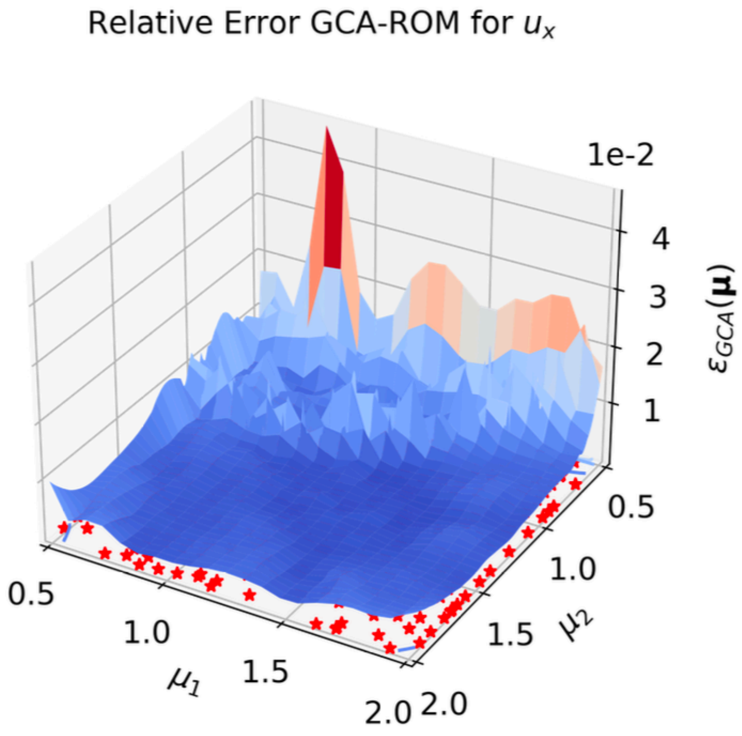
Hyperparameters:

- $M = 3171$ snapshots, train rate $r_t = 10\%$
- latent $n = 25$, epochs $N_{ep} = 5000$

Relative errors:

- mean: $4.62 \cdot 10^{-3}$
- max: $4.55 \cdot 10^{-2}$





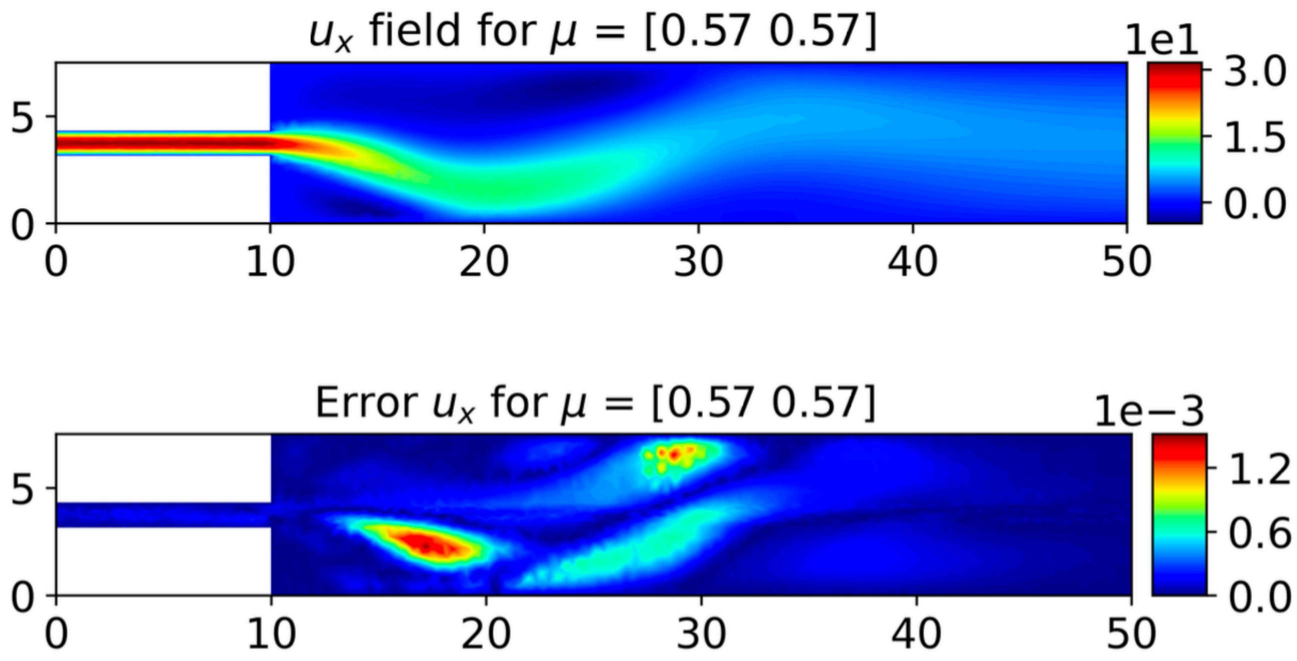
Navier-Stokes: bifurcating problem - u_x

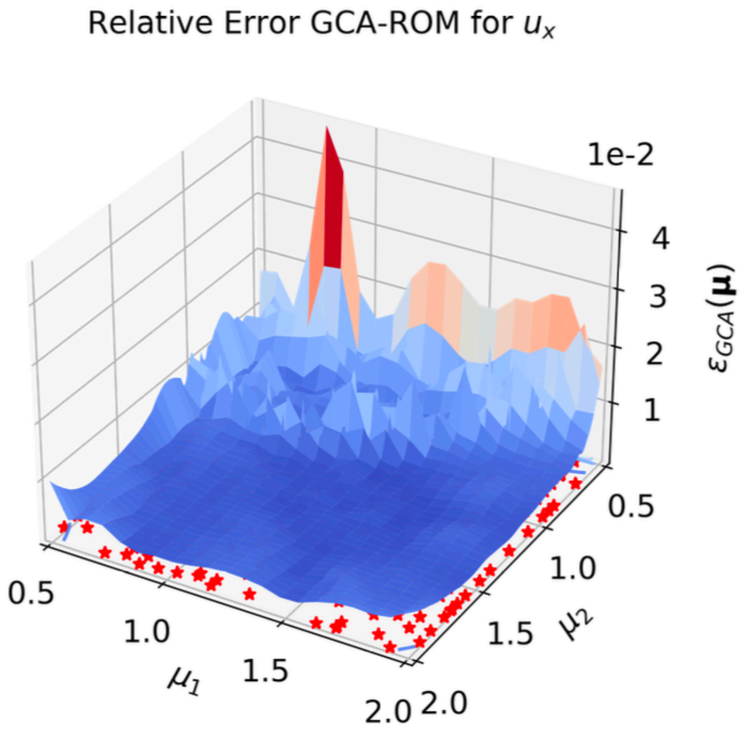
Hyperparameters:

- $M = 3171$ snapshots, train rate $r_t = 10\%$
- latent $n = 25$, epochs $N_{ep} = 5000$

Relative errors:

- mean: $4.62 \cdot 10^{-3}$
- max: $4.55 \cdot 10^{-2}$





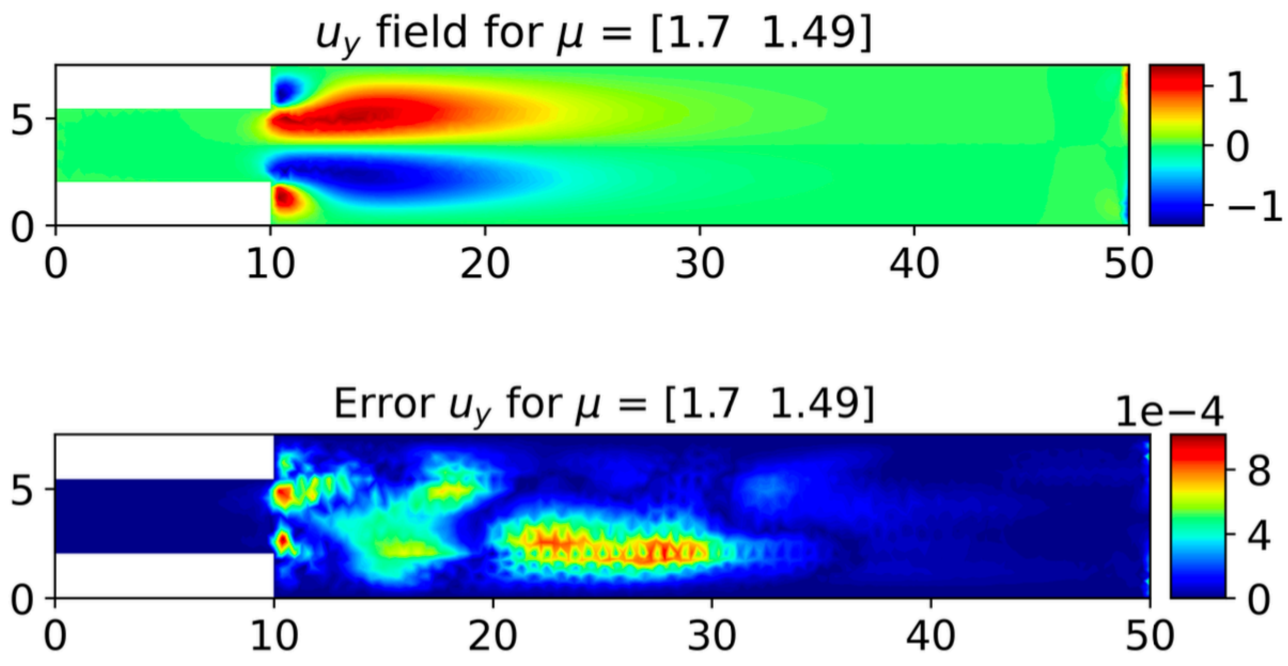
Navier-Stokes: bifurcating problem - u_y

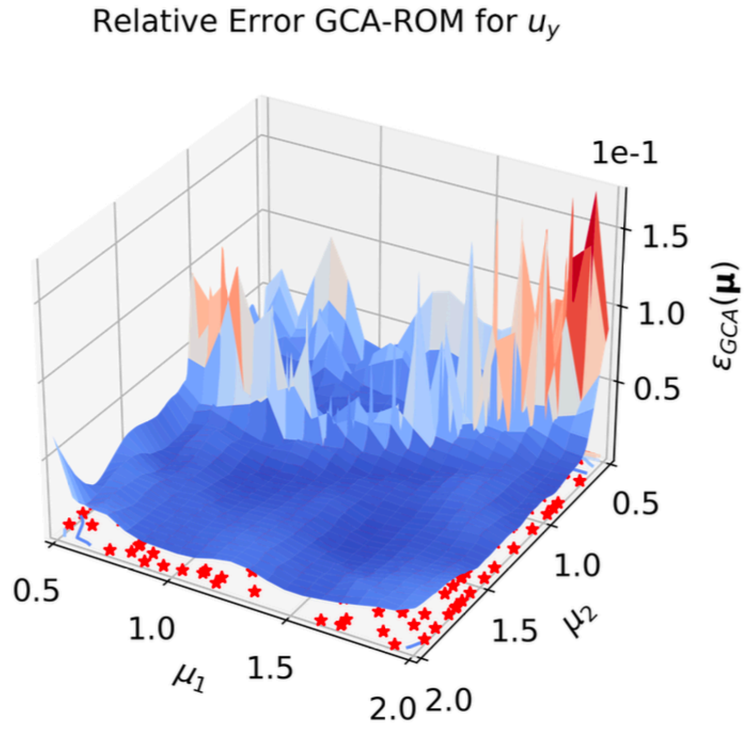
Hyperparameters:

- $M = 3171$ snapshots, train rate $r_t = 10\%$
- latent $n = 25$, epochs $N_{ep} = 5000$

Relative errors:

- mean:
- max:





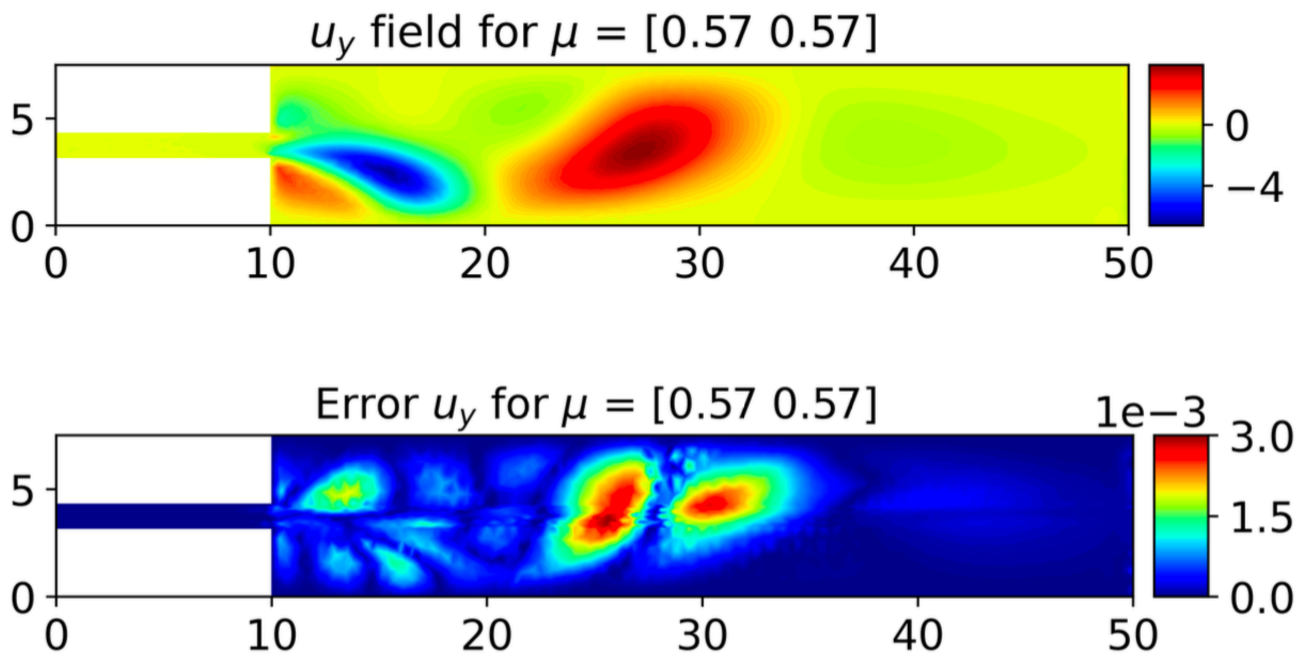
Navier-Stokes: bifurcating problem - u_y

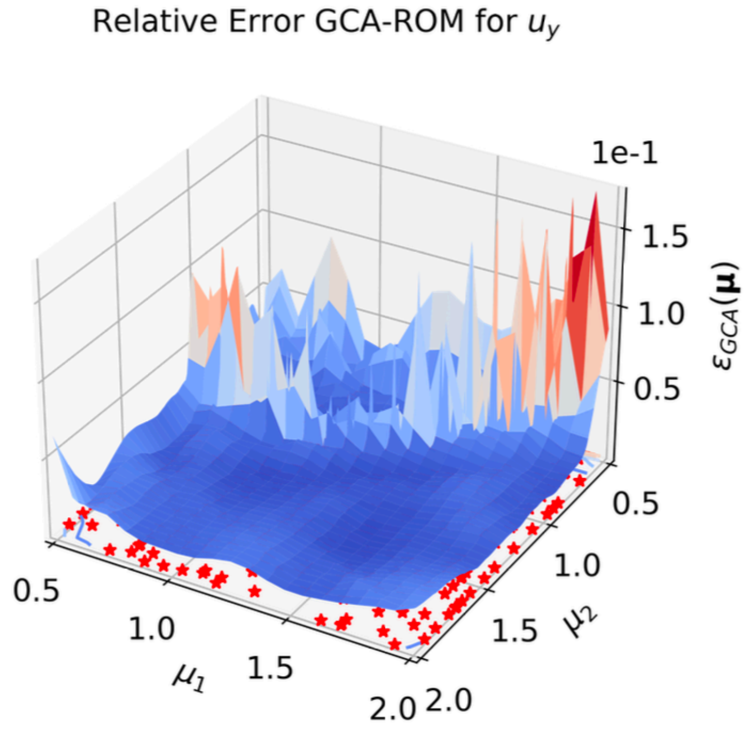
Hyperparameters:

- $M = 3171$ snapshots, train rate $r_t = 10\%$
- latent $n = 25$, epochs $N_{ep} = 5000$

Relative errors:

- mean:
- max:





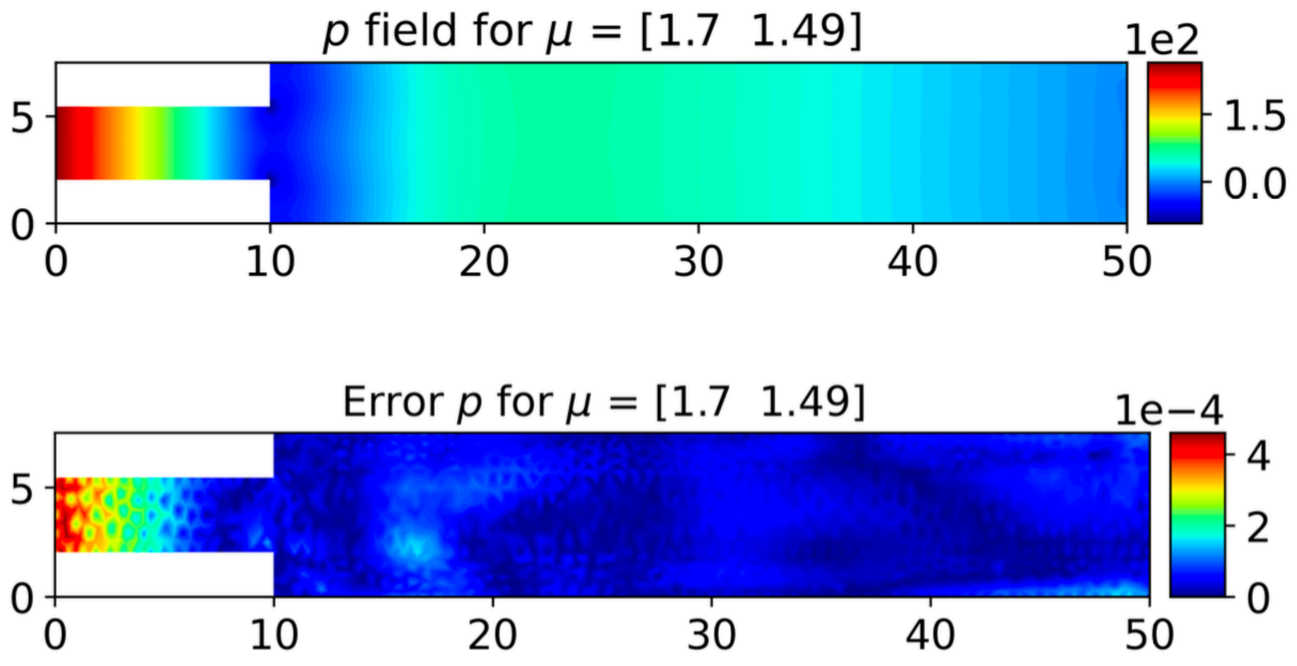
Navier-Stokes: bifurcating problem - p

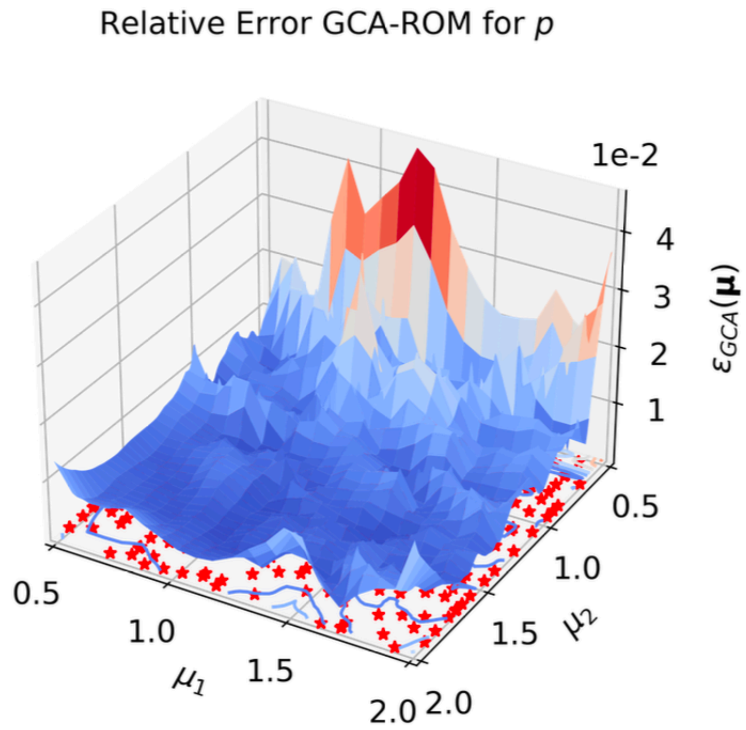
Hyperparameters:

- $M = 3171$ snapshots, train rate $r_t = 10\%$
- latent $n = 25$, epochs $N_{ep} = 5000$

Relative errors:

- mean:
- max:





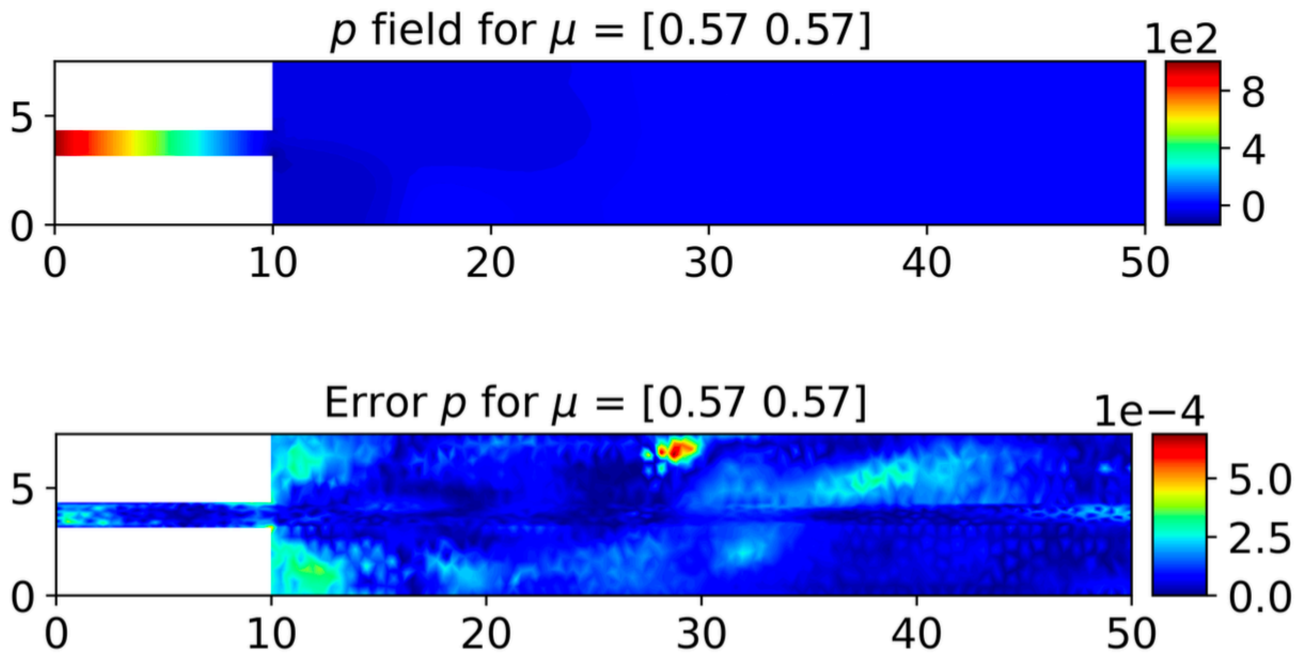
Navier-Stokes: bifurcating problem - p

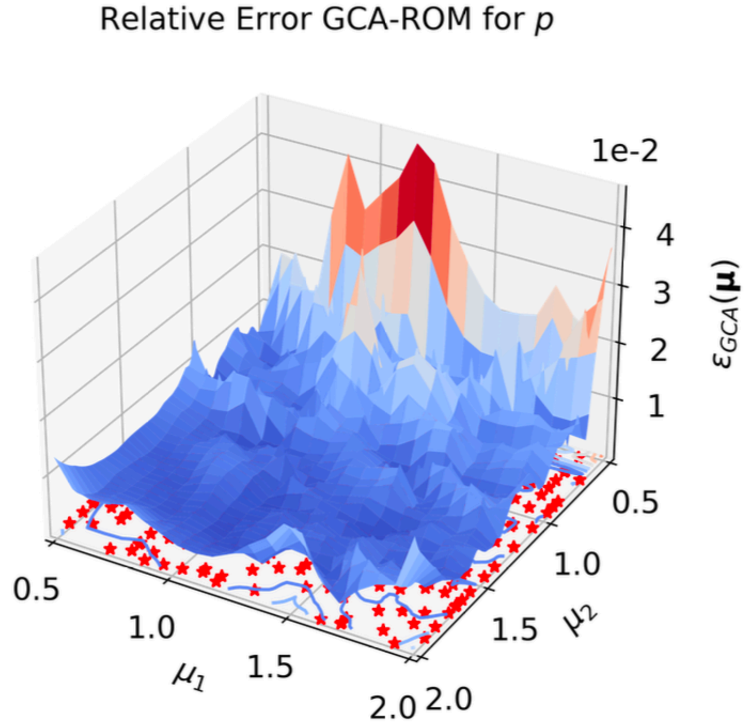
Hyperparameters:

- $M = 3171$ snapshots, train rate $r_t = 10\%$
- latent $n = 25$, epochs $N_{ep} = 5000$

Relative errors:

- mean:
- max:





Detection of bifurcation points and KNN clustering

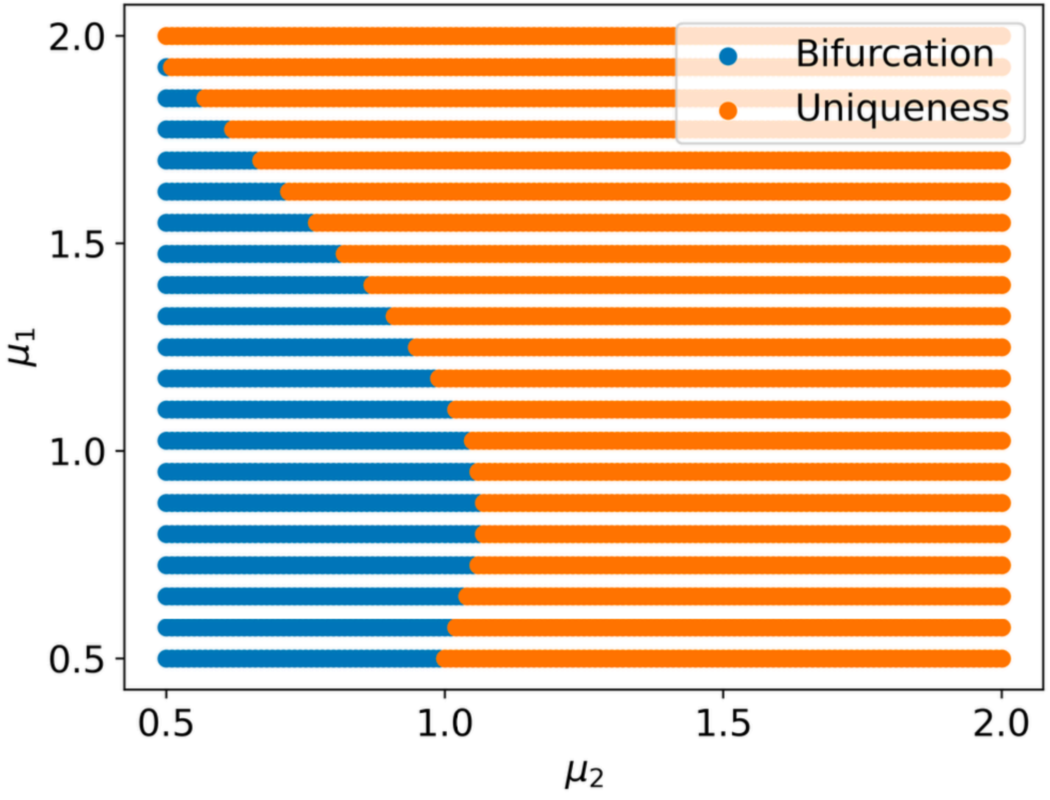


Figure: GCA-ROM

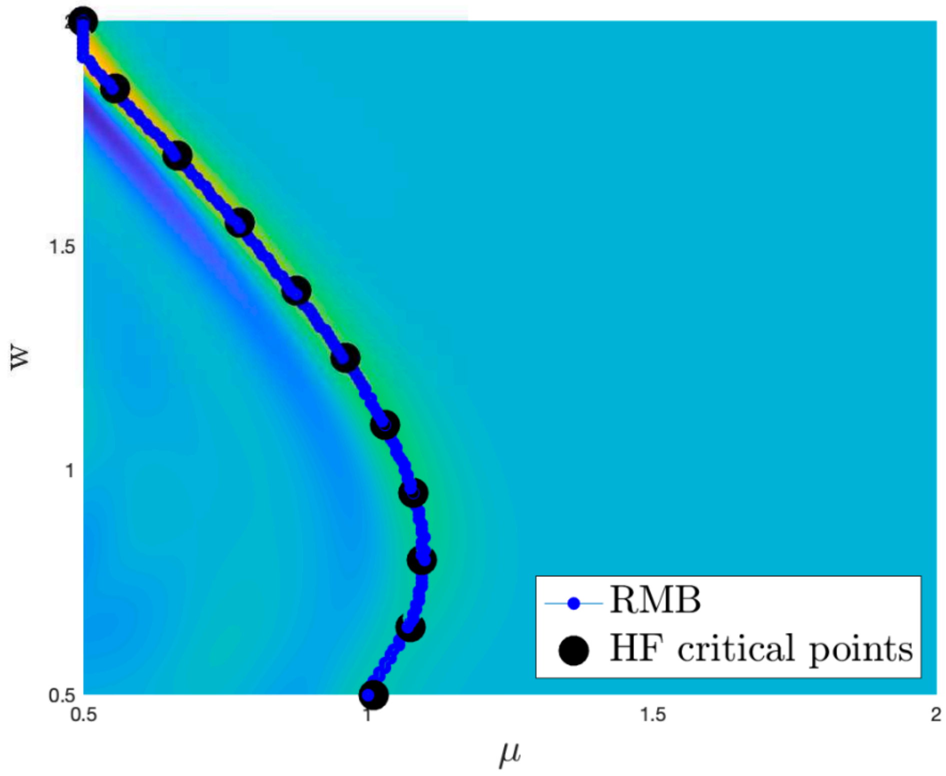


Figure: POD-NN

Advantages and disadvantages

Pros:

- *Trainable* pooling to down-sample in the encoder
- *k-NN interpolation* to up-sample in the decoder
- *Skip-connection* to create alternative learning paths
- *Low-data regime*, high generalization capability
- Handle *geometrical parametrization* and vectorial problems
- *Non-intrusive* and fast reconstruction on the mesh configuration
- Easy extension for *time-dependent* and *3D problems*

Advantages and disadvantages

Cons:

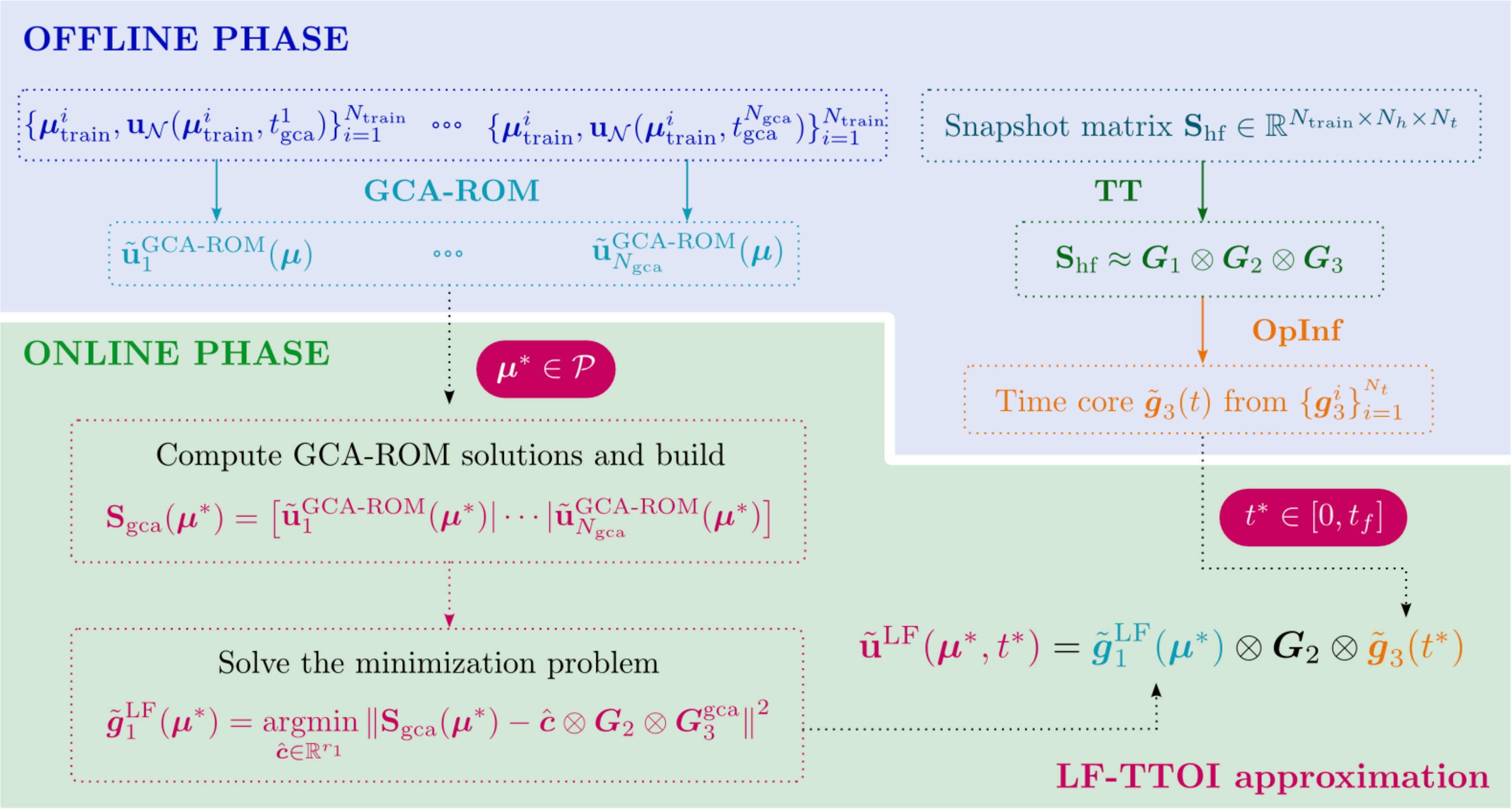
- Training and testing on *single-fidelity* data
- Cannot extend to *spatially-dependent* parameters
- Complex and costly *pooling/unpooling* procedures
- Lack of *physical* information
- No imposition of *boundary conditions*
- *Large architectures* with high *memory* requirements

Issue with GNN-based autoencoders

- *Message passing procedure*: over-smoothing, over-squashing, over-fitting, scalability and expressivity.
  $\rightsquigarrow$ Geometric Deep Learning theory
- *Extrapolation in time of dynamical systems*: non-causality.
  $\rightsquigarrow$ Time-integration and autoregressive approaches
- *Down-sampling and up-sampling strategies needed for autoencoders*: compress data and learn across fidelities.
  $\rightsquigarrow$ Multi-fidelity training and testing

Low-fidelity Tensor Train approach with Operator Inference

Idea: Exploiting **Tensor Train** decomposition to decouple the low-rank parameter and time cores, and generalize thanks to **GCA-ROM** and **Operator Inference**.



Low-fidelity Tensor Train approach with Operator Inference

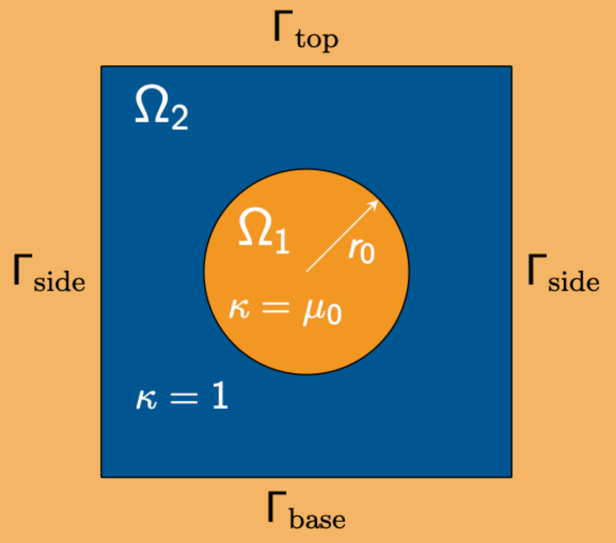
- **Offline phase**: given the snapshots tensor $\boldsymbol{S}(\boldsymbol{\mu}, t) \in \mathbb{R}^{M \times \mathcal{N}_h \times N_t}$
  - *TT-decomposition* of $\boldsymbol{S}(\boldsymbol{\mu}, t) \approx \boldsymbol{G_1}(\boldsymbol{\mu}) \otimes \boldsymbol{G_2}(x) \otimes \boldsymbol{G_3}(t)$, with $\boldsymbol{G_i} \in \mathbb{R}^{r_{i-1} \times n_i \times r_i}$
  - Construct an *Operator Inference* model for the time core $\tilde{\boldsymbol{G}}_3(t)$
  - Train N_{GCA} different *GCA-ROMs* at fixed time instances $\{t_i\}_{i=1}^{N_{\text{GCA}}}$
- **Online phase**: for a given pair $(\boldsymbol{\mu}^*, t^*)$
  - Compute low fidelity *GCA-ROMs* solutions $\tilde{\boldsymbol{S}}(\boldsymbol{\mu}^*) = \left[\tilde{u}^1_{\mathcal{N}}(\boldsymbol{\mu}^*), \dots, \tilde{u}^{N_{\text{GCA}}}_{\mathcal{N}}(\boldsymbol{\mu}^*)\right]$
  - Solve the minimization problem

$$\tilde{\boldsymbol{G}}_1(\boldsymbol{\mu}^*) = \underset{\boldsymbol{c} \in \mathbb{R}^{r_1}}{\operatorname{argmin}} \left\| \tilde{\boldsymbol{S}}(\boldsymbol{\mu}^*) - \boldsymbol{c} \otimes \boldsymbol{G_2} \otimes \boldsymbol{G_3}^{N_{\text{GCA}}} \right\|^2 .$$

  - Exploit OpInf to predict the time core $\tilde{\boldsymbol{G}}_3(t^*)$
  - Compute the approximation as $\tilde{u}_{\mathcal{N}}(\boldsymbol{\mu}^*, t^*) = \tilde{\boldsymbol{G}}_1(\boldsymbol{\mu}^*) \otimes \boldsymbol{G_2} \otimes \tilde{\boldsymbol{G}}_3(t^*)$

Thermal block problem

$$\begin{cases} \frac{\partial u}{\partial t} - \nabla \cdot (\kappa_{\boldsymbol{\mu}} \nabla u) = 0, & \text{in } \Omega \times [0, T], \\ u(x, 0; \boldsymbol{\mu}) = 0, & \text{in } \Omega, \\ u(x, t; \boldsymbol{\mu}) = 0, & \text{on } \Gamma_{\text{top}} \times [0, T], \\ \kappa_{\boldsymbol{\mu}} \nabla u \cdot \mathbf{n} = 0, & \text{on } \Gamma_{\text{side}} \times [0, T], \\ \kappa_{\boldsymbol{\mu}} \nabla u \cdot \mathbf{n} = \mu_1, & \text{on } \Gamma_{\text{base}} \times [0, T]. \end{cases}$$



Parameters:

- $\mu_0 \in [0.1, 10]$ thermal conductivity
- $\mu_1 \in [-1, 1]$ heat flux
- $N_h = 304$ dofs

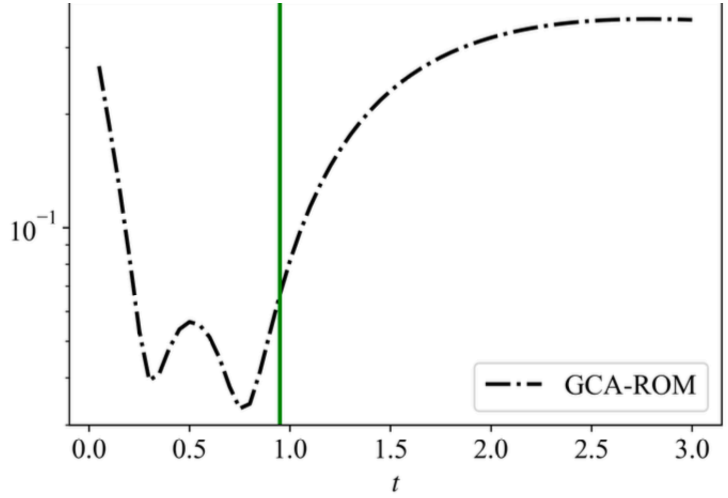
Keywords:

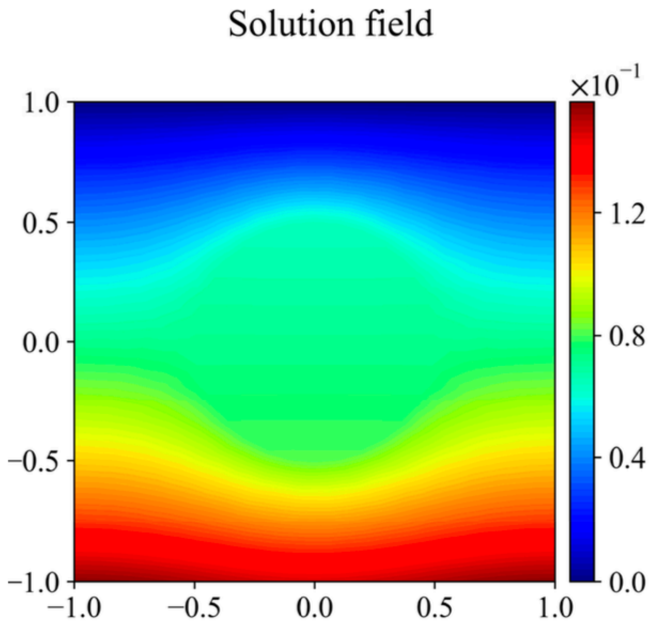
- time dependent
- multi-param
- small-data

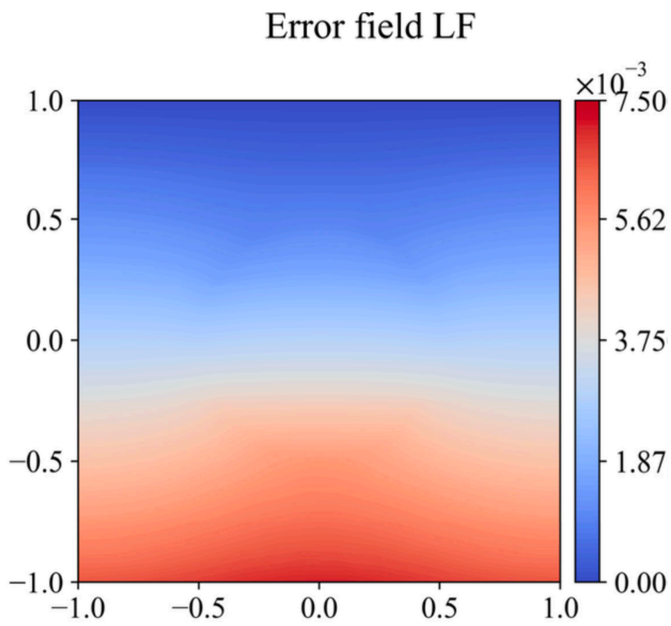
Thermal block problem

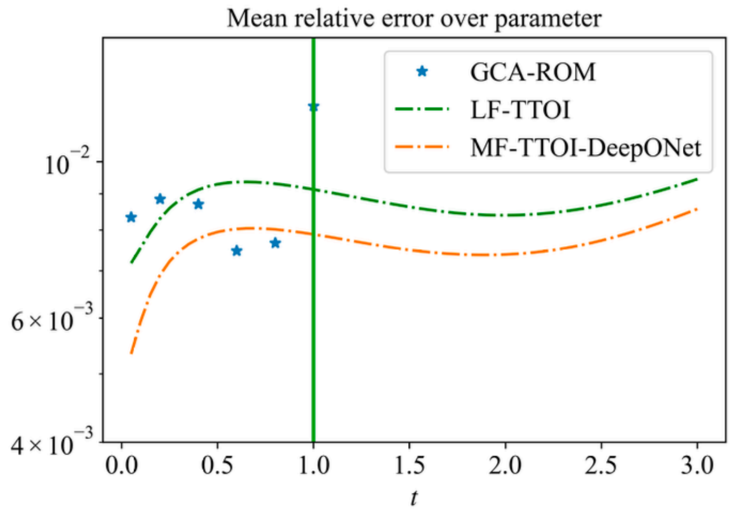
Hyperparameters:

- $M = 100$, train rate $r_t = 30\%$
- latent $n = 11$, $N_{GCA} = 6$

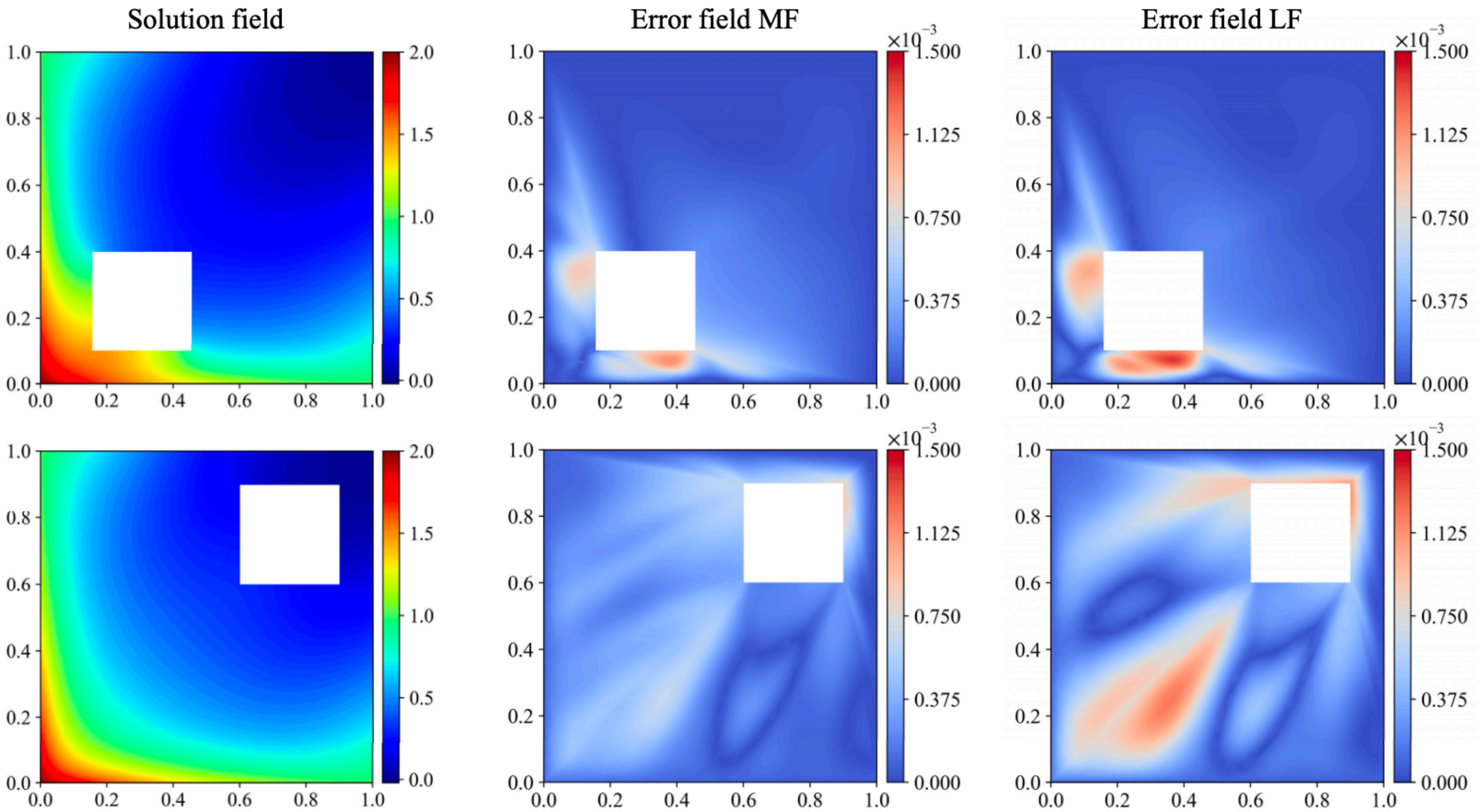




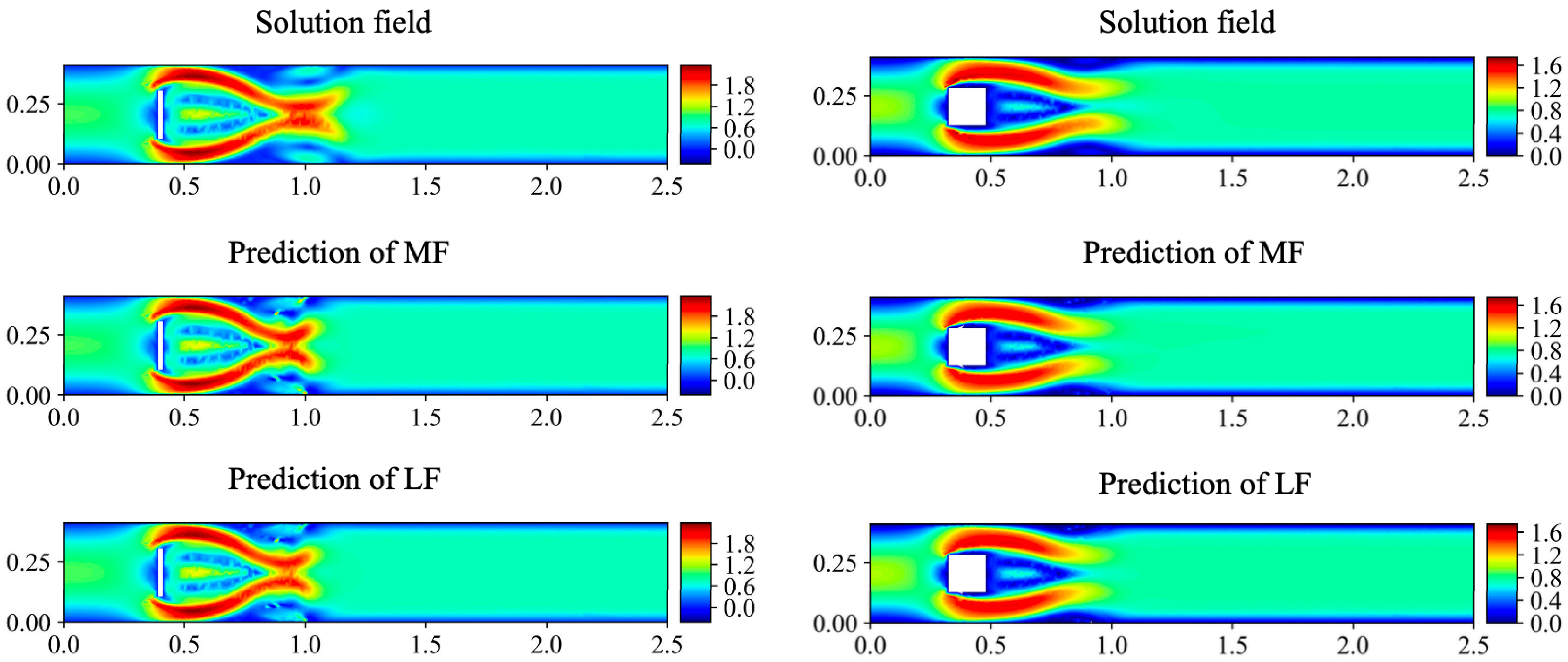




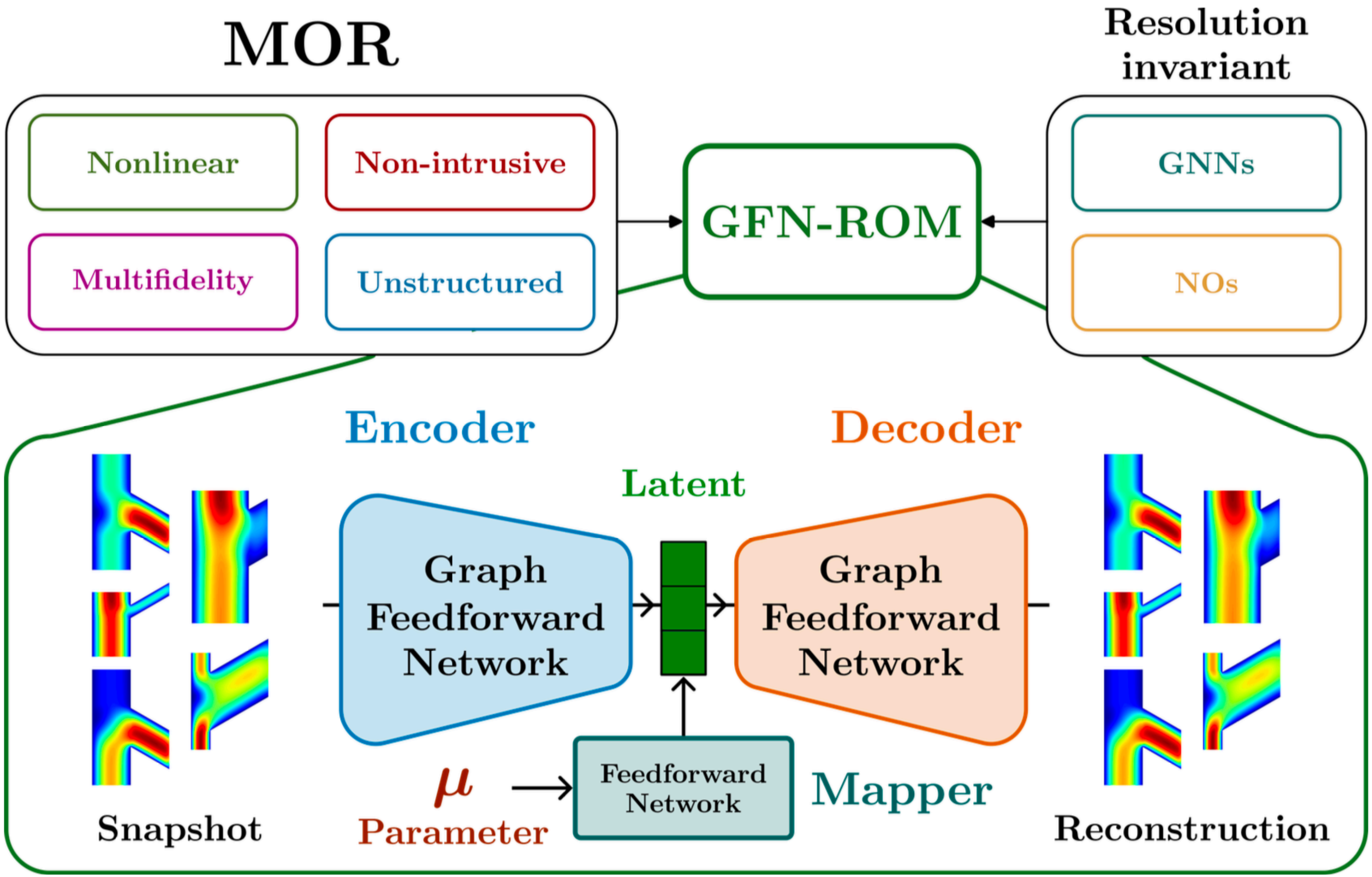
Advection with a square hole



Navier-Stokes: flow past a parametrized rectangle



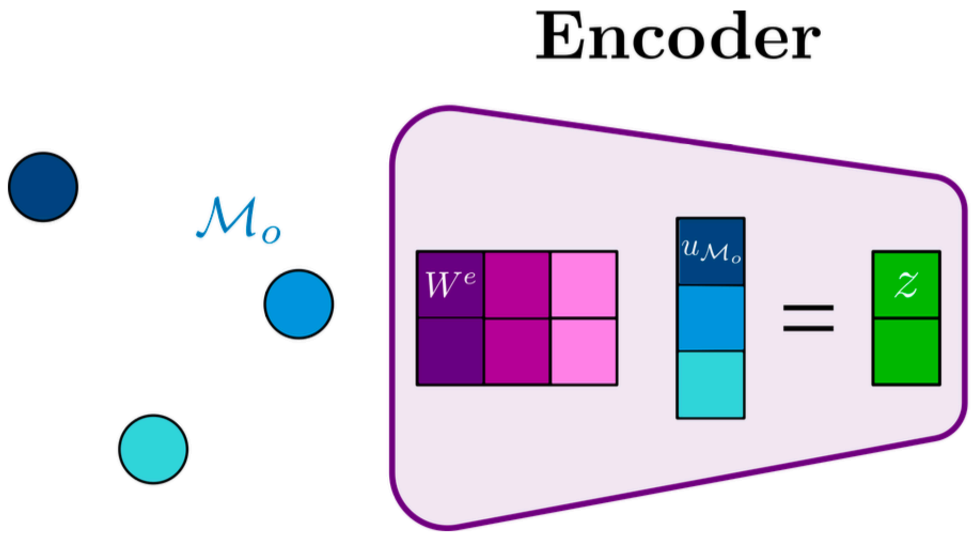
Graph Neural Networks

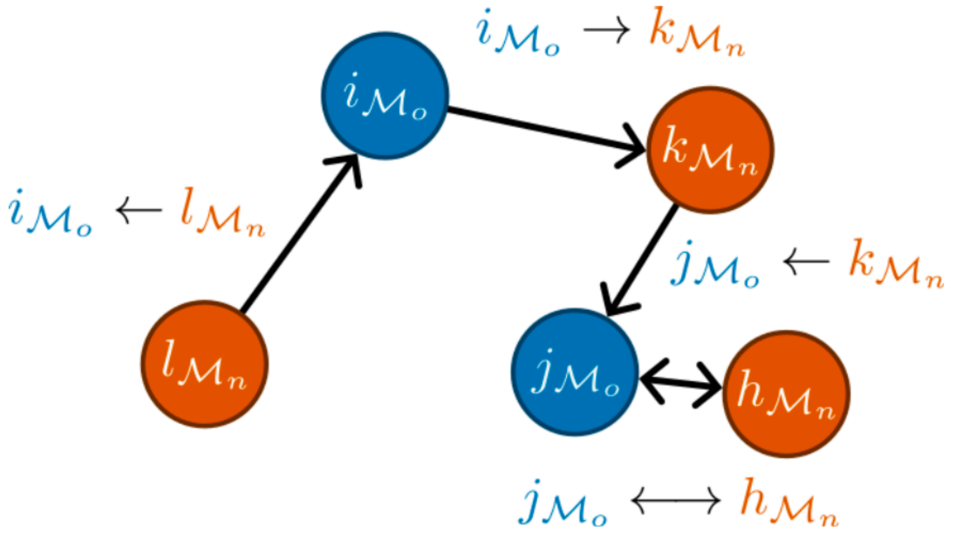


Graph Neural Networks

Feedforward autoencoder for a mesh $\mathcal{M}_o$ with N_o nodes and L latent coordinates.

$$\mathrm{enc}(\boldsymbol{u}_{\mathcal{M}_o})_i = \sigma\left(\sum_{j_{\mathcal{M}_o}=1}^{N_o} W^e_{ij_{\mathcal{M}_o}} u_{j_{\mathcal{M}_o}} + b^e_i\right)$$



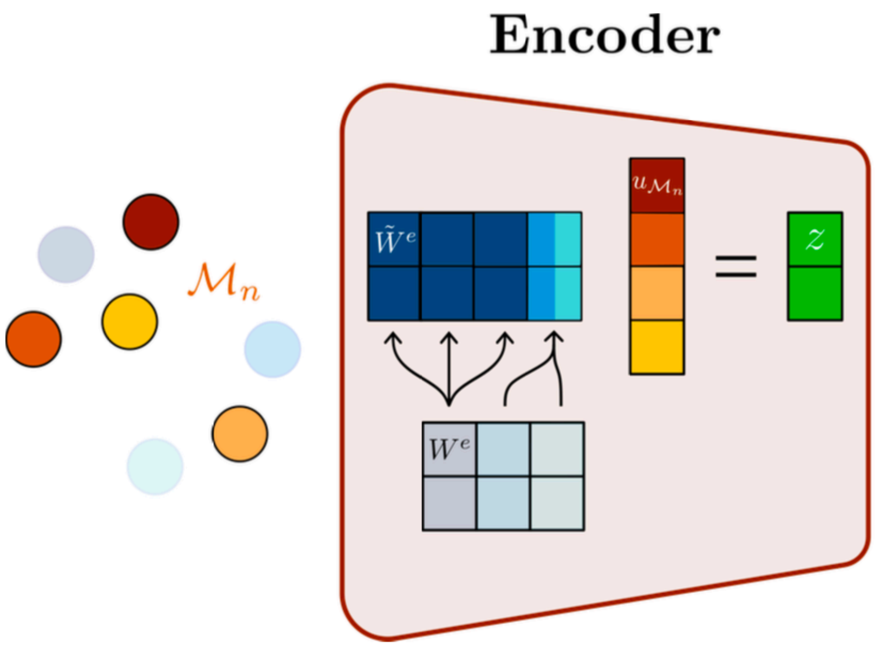


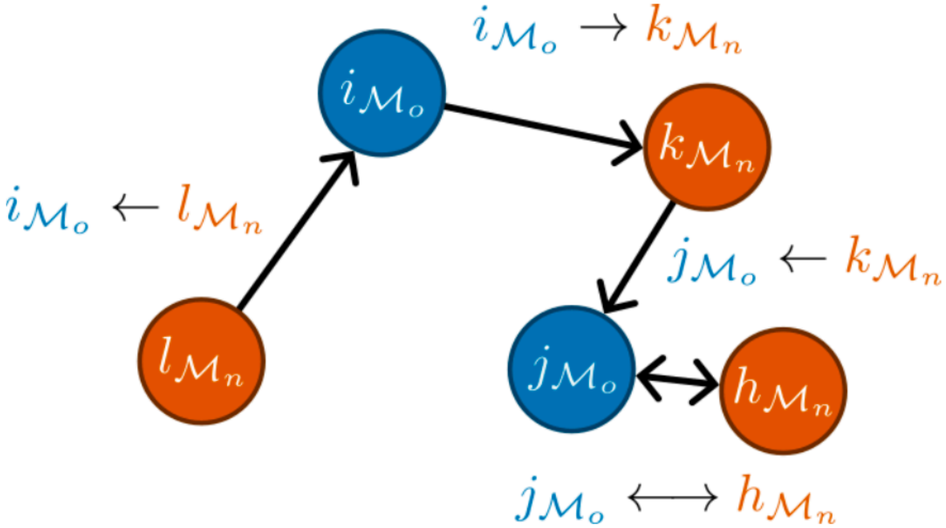
$$\tilde{\boldsymbol{W}}^e = \mathrm{GFN}^{\mathcal{M}_o \to \mathcal{M}_n}(\boldsymbol{W}^e)$$

Graph Neural Networks

Adapt weights for a new mesh $\mathcal{M}_n$ with N_n nodes and L latent coordinates.

$$\mathrm{enc}(\boldsymbol{u}_{\mathcal{M}_n})_i = \sigma\left(\sum_{j_{\mathcal{M}_n}=1}^{N_n} \tilde{W}^e_{ij_{\mathcal{M}_n}} u_{j_{\mathcal{M}_n}} + b^e_i\right)$$

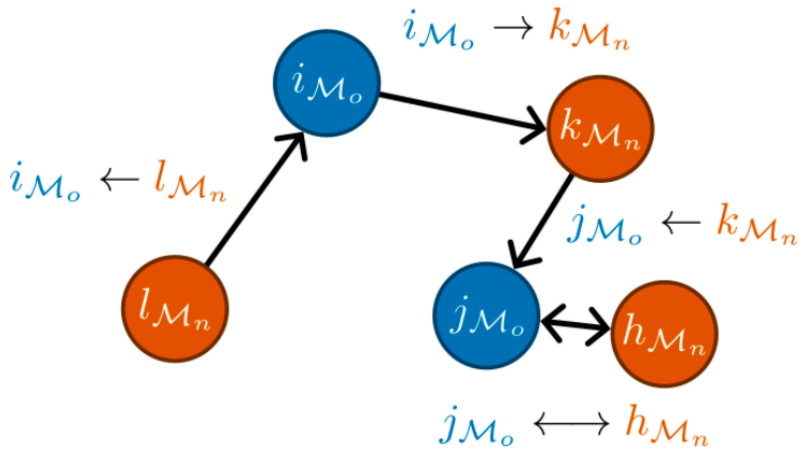




$$\tilde{W}^e = \mathrm{GFN}^{\mathcal{M}_o \to \mathcal{M}_n}(W^e)$$

Graph Neural Networks

$$\tilde{\boldsymbol{W}}^e, \tilde{\boldsymbol{W}}^d, \tilde{\boldsymbol{b}}^d = \mathrm{GFN}^{\mathcal{M}_o \to \mathcal{M}_n}(\boldsymbol{W}^e, \boldsymbol{W}^d, \boldsymbol{b}^d)$$



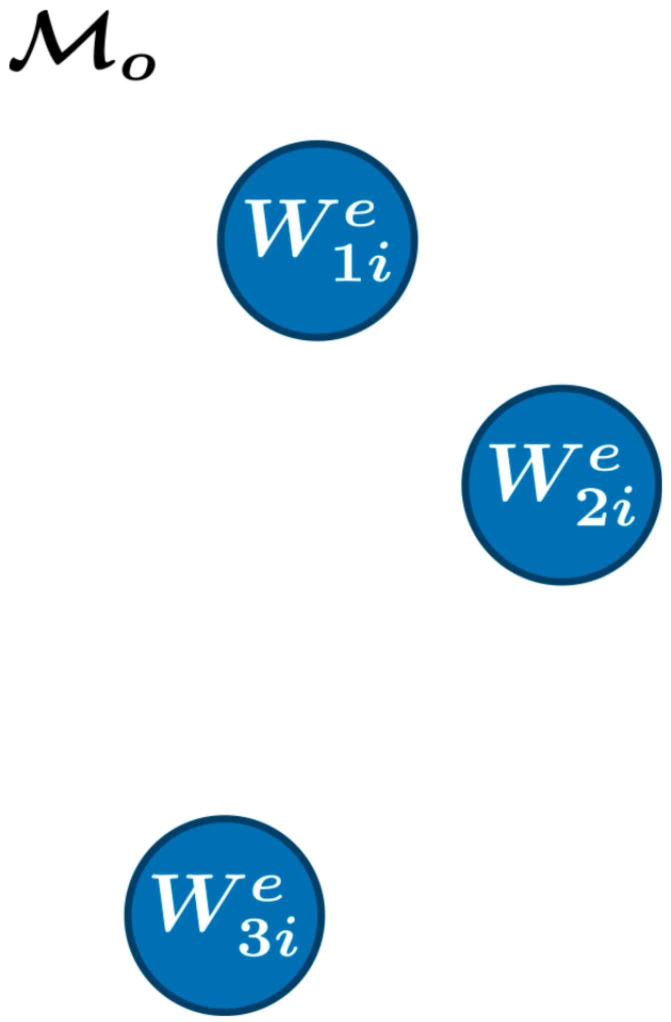
$$\tilde{W}^e_{ij_{\mathcal{M}_n}} = \sum_{\substack{\forall k_{\mathcal{M}_o} \text{ s.t} \\ k_{\mathcal{M}_o} \leftarrow\!/\!\rightarrow j_{\mathcal{M}_n}}} \frac{W^e_{ik_{\mathcal{M}_o}}}{|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow\!/\!\rightarrow h_{\mathcal{M}_n}\}|},$$

$$\tilde{b}^e_i = b^e_i,$$

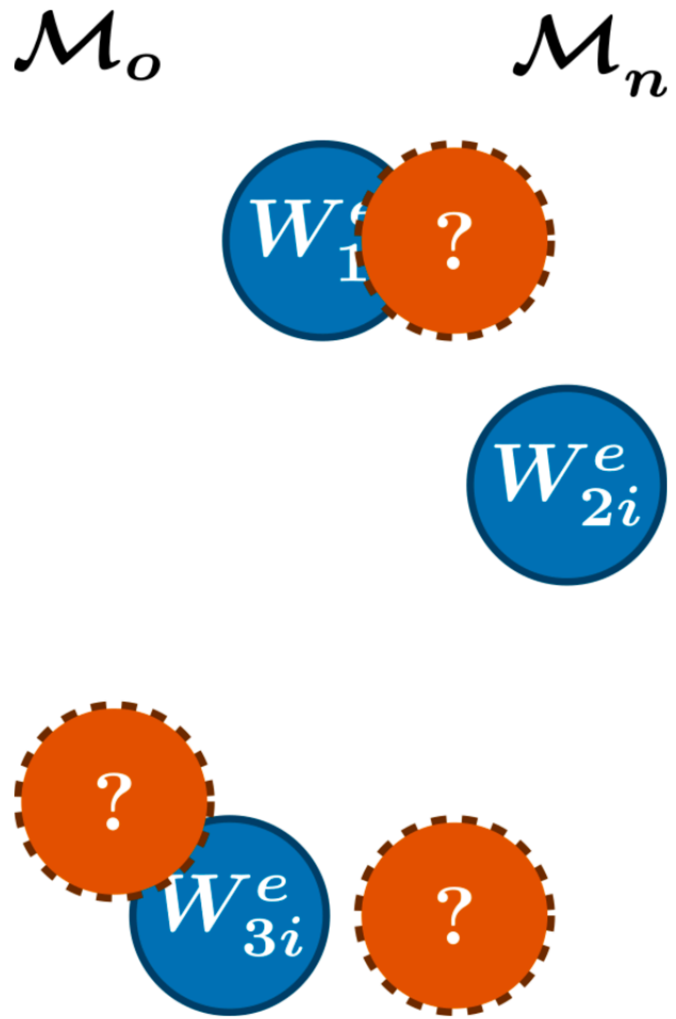
$$\tilde{W}^d_{i_{\mathcal{M}_n}j} = \underset{\substack{\forall k_{\mathcal{M}_o} \text{ s.t} \\ k_{\mathcal{M}_o} \leftarrow\!/\!\rightarrow i_{\mathcal{M}_n}}}{\mathrm{mean}} W^d_{k_{\mathcal{M}_o}j},$$

$$\tilde{b}^d_{i_{\mathcal{M}_n}} = \underset{\substack{\forall k_{\mathcal{M}_o} \text{ s.t} \\ k_{\mathcal{M}_o} \leftarrow\!/\!\rightarrow i_{\mathcal{M}_n}}}{\mathrm{mean}} b^d_{k_{\mathcal{M}_o}}.$$

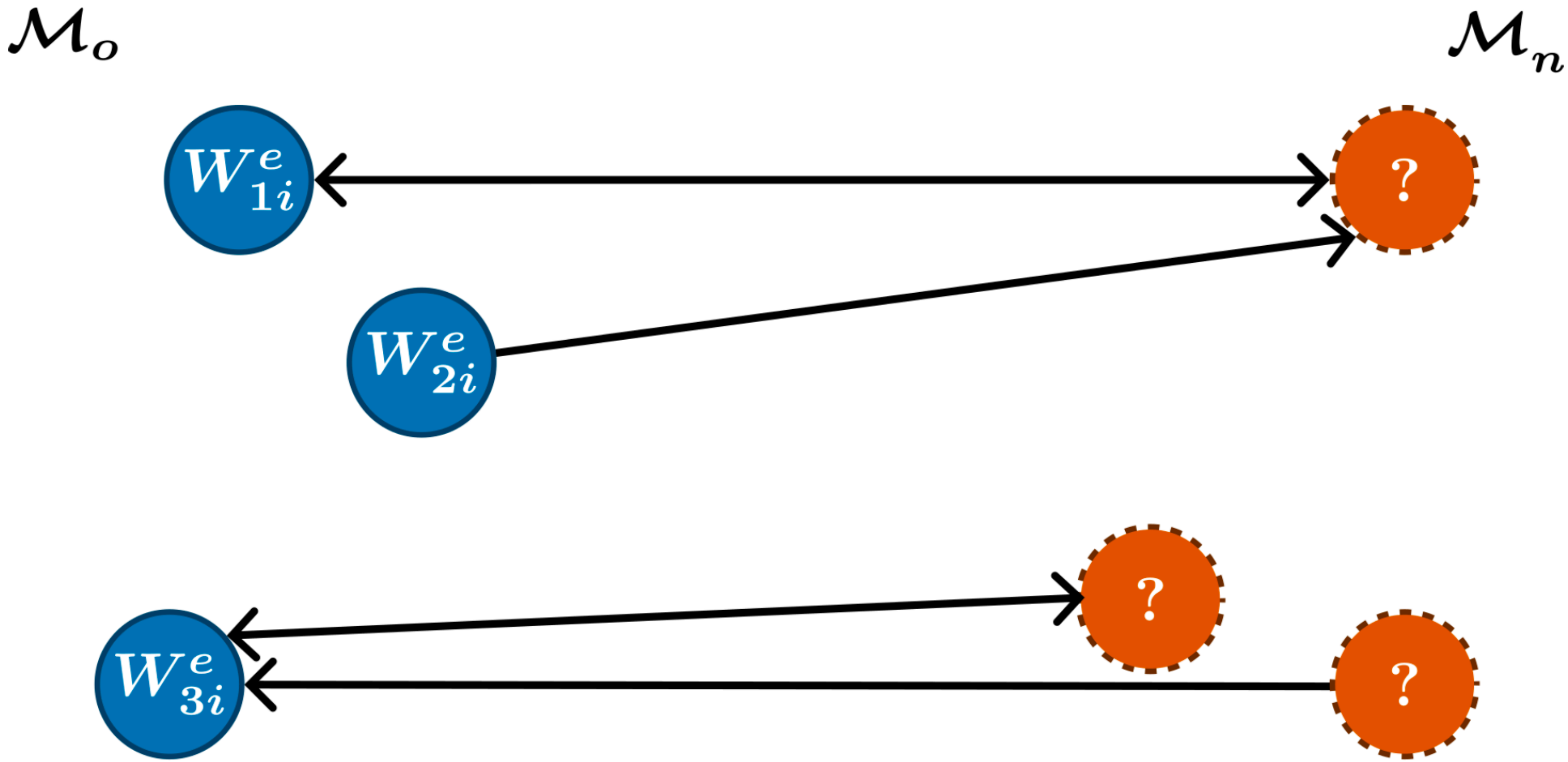
Graph Neural Networks



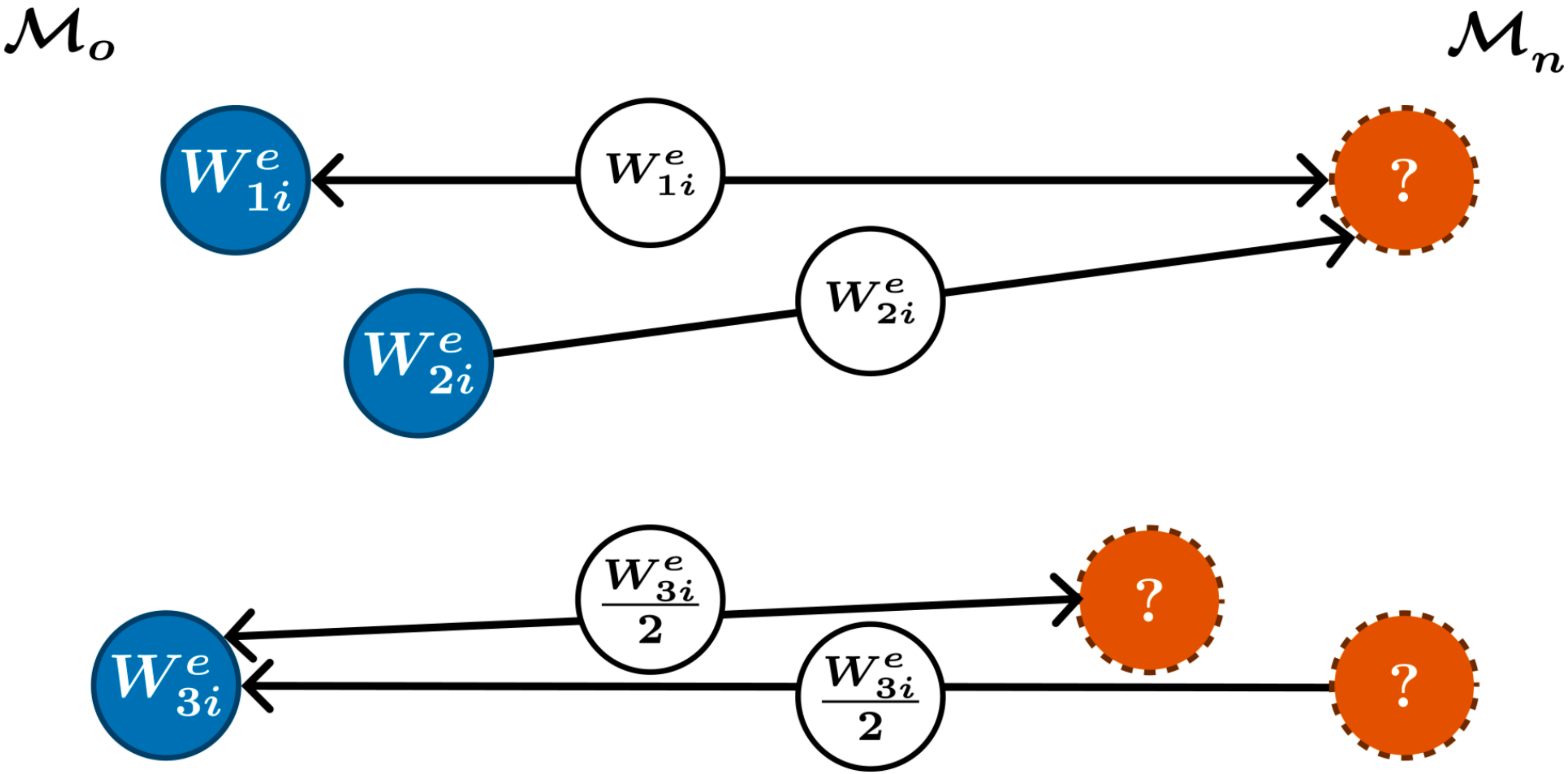
Graph Neural Networks



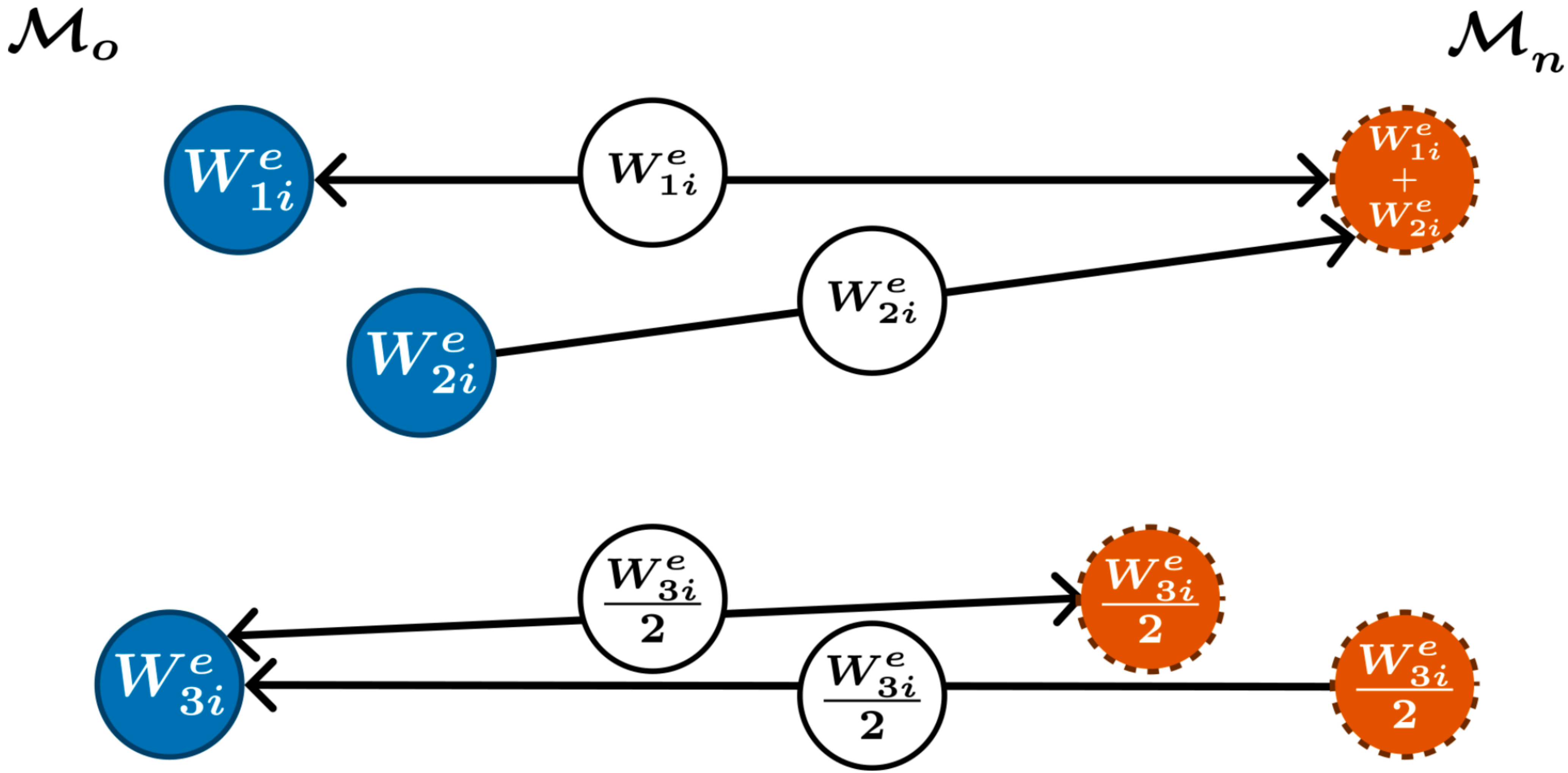
Graph Neural Networks



Graph Neural Networks



Graph Neural Networks



Graph Neural Networks

Consider a model trained on $\mathcal{M}_o$ that we want to evaluate on $\mathcal{M}_n$

$$\forall i_{\mathcal{M}_o},\ j_{\mathcal{M}_n} \text{ s.t. } i_{\mathcal{M}_o} \leftarrow\!/\!\!\rightarrow j_{\mathcal{M}_n}, \quad \left| u(\boldsymbol{x}_{i_{\mathcal{M}_o}}, \boldsymbol{\mu}) - u(\boldsymbol{x}_{j_{\mathcal{M}_n}}, \boldsymbol{\mu}) \right| \leq \underbrace{\delta(\boldsymbol{\mu})}_{\text{Mesh similarity}}$$

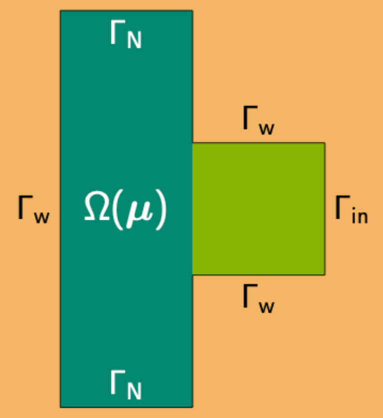
$$\forall i_{\mathcal{M}_o}, \quad \left| u(\boldsymbol{x}_{i_{\mathcal{M}_o}}, \boldsymbol{\mu}) - \mathrm{dec}_{\mathcal{M}_o}(\mathrm{map}(\boldsymbol{\mu}))_{i_{\mathcal{M}_o}} \right| \leq \underbrace{\tau(\boldsymbol{\mu})}_{\text{Performance}}$$

Error bounds on super- and sub-resolution:

$$\forall i_{\mathcal{M}_n}, \quad \left| u(\boldsymbol{x}_{i_{\mathcal{M}_n}}, \boldsymbol{\mu}) - \mathrm{dec}_{\mathcal{M}_n}(\mathrm{map}(\boldsymbol{\mu}))_{i_{\mathcal{M}_n}} \right| \leq \tau(\boldsymbol{\mu}) + \delta(\boldsymbol{\mu})$$

Graph Neural Networks

$$\begin{cases} -\Delta \boldsymbol{u} + \nabla p = (0, \mu_6) & \text{in } \Omega \\ \nabla \cdot \boldsymbol{u} = 0 & \text{in } \Omega \\ \boldsymbol{u} = \boldsymbol{0} & \text{on } \Gamma_{\mathrm{w}} \\ \boldsymbol{u} = (4(y-1)(2-y), 0) & \text{on } \Gamma_{\mathrm{in}} \\ \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}} = \boldsymbol{0} & \text{on } \Gamma_{\mathrm{N}} \end{cases}$$



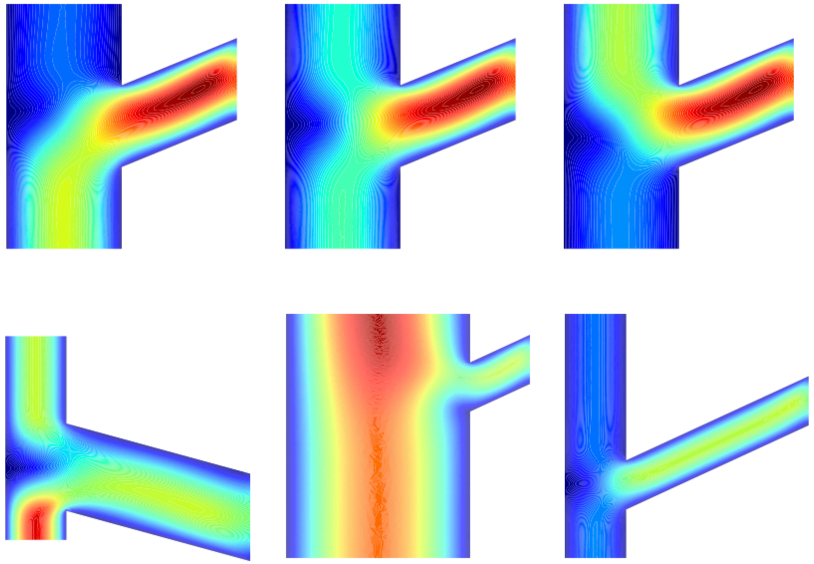
Parameters:

- $\{\mu_i\}_{i=0}^4 \in [0.5, 1.5]$ lengths
- $\mu_5 \in [-\frac{\pi}{6}, \frac{\pi}{6}]$ angle
- $\mu_6 \in [-10, 10]$ forcing term

Keywords:

- h-dim param. space
- complex geometry
- small-data

Graph Neural Networks



Hyperparameters:

- $M = 704$, train rate $r_t = 30\%$
- latent $L = 10$, epochs $N_{ep} = 5000$

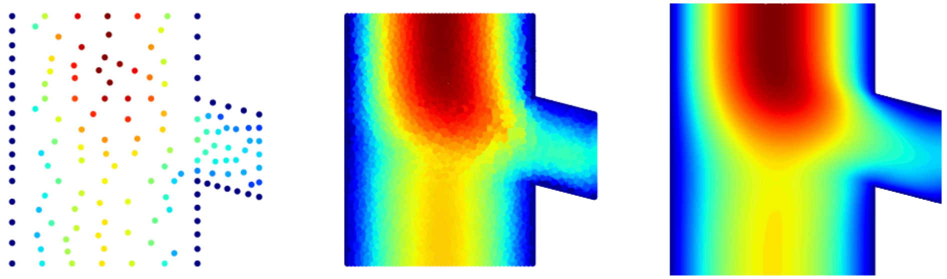
Relative errors:

		Testing			
		Tiny (262)	**Small (756)**	**Medium (2226)**	**Large (7019)**
Training	**Tiny**	**2.57 %**	5.56 %	5.49 %	5.55 %
	Small	4.41 %	**3.44 %**	4.45 %	4.44 %
	Medium	4.58 %	4.12 %	**3.76 %**	**4.12 %**
	Large	5.02 %	4.51 %	4.32 %	4.24 %

Graph Neural Networks

Hyperparameters:

- $M = 704$, train rate $r_t = 30\%$
- latent $L = 10$, epochs $N_{ep} = 5000$



Relative errors:

		Testing			
		Tiny (262)	**Small (756)**	**Medium (2226)**	**Large (7019)**
Training	**Tiny**	**2.57 %**	5.56 %	5.49 %	5.55 %
	Small	4.41 %	**3.44 %**	4.45 %	4.44 %
	Medium	4.58 %	4.12 %	**3.76 %**	**4.12 %**
	Large	5.02 %	4.51 %	4.32 %	4.24 %

Graph Neural Networks

Errors	POD-G	POD-Proj	POD-NN	GCA-ROM	GFN-ROM			
	Large	Large	Large	Large	Large	Medium	Small	Tiny
Graetz	3.44	3.44	3.54	0.74	1.02	0.88	0.98	1.28
Advection	27.09	25.66	30.58	4.73	4.73	4.48	7.22	12.35
Stokes	2.45	2.45	2.94	4.63	4.24	4.12	4.44	5.55

Architecture		GFN-ROM				GCA-ROM
		Large	Medium	Small	Tiny	Large
Graetz	Training time (s)	119	46	31	26	600
	Trainable parameters	2 898 761	911 004	311 910	115 821	2 898 795
	Mesh nodes	7205	2248 (31%)	754 (10%)	265 (4%)	7205
Advection	Training time (s)	124	53	34	23	408.9
	Trainable parameters	3 538 757	1 110 702	387 298	140 282	3 538 791
	Mesh nodes	8801	2746 (31%)	942 (11%)	326 (4%)	8801
Stokes	Training time (s)	248	90	45	31	1949
	Trainable parameters	2 827 589	905 596	316 126	118 032	2 827 623
	Mesh nodes	7019	2226 (32%)	756 (11%)	262 (4%)	7019

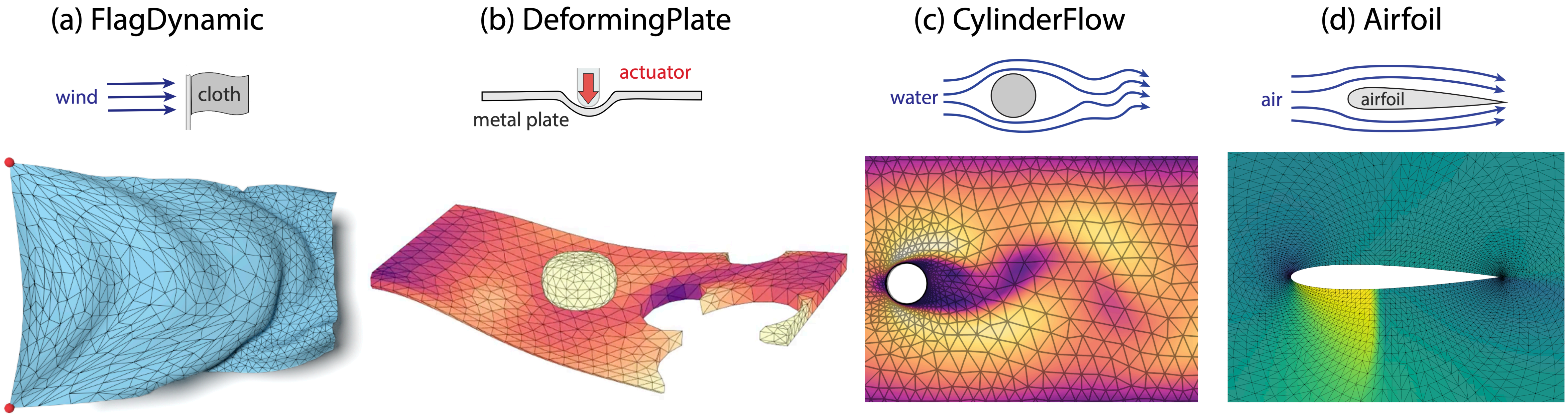
Graph Neural Networks

Errors	**POD-G**	**POD-Proj**	**POD-NN**	**GCA-ROM**	**GFN-ROM**			
	Large	Large	Large	Large	Large	Medium	Small	Tiny
Graetz	3.44	3.44	3.54	0.74	1.02	0.88	0.98	1.28
Advection	27.09	25.66	30.58	4.73	4.73	4.48	7.22	12.35
Stokes	2.45	2.45	2.94	4.63	4.24	4.12	4.44	5.55

	GFN-ROM					
	Large & Medium	Large & Small	Large & Tiny	Medium & Small	Medium & Tiny	Small & Tiny
Graetz	0.96 (+0.06)	0.98 (+0.04)	1.40 (-0.37)	4.44 (-3.56)	1.03 (-0.15)	1.09 (-0.11)
Advection	5.02 (-0.30)	5.35 (-0.62)	5.63 (-0.91)	5.22 (-0.73)	6.27 (-1.79)	8.77 (-1.55)
Stokes	3.63 (+0.61)	4.94 (-0.70)	4.44 (-0.19)	4.51 (-0.39)	5.56 (-1.45)	5.65 (-1.21)

MeshGraphNets

Goal: a framework for learning *mesh-based* simulations using graph neural networks.
Task: learn a forward model of the *dynamic* quantities of the mesh at time $t{+}1$ given the current mesh M^t with an *Encode-Process-Decode* architecture followed by an integrator.



[1] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., Battaglia, P.W., 2020. Learning to Simulate Complex Physics with Graph Networks. arXiv:2002.09405

MeshGraphNets

Setting: the governing equations depend on a vector of geometrical parameters $\boldsymbol{\mu}$ so that $\Omega = \Omega_{\boldsymbol{\mu}}$.
Issue: for $\overline{\boldsymbol{\mu}} \neq \boldsymbol{\mu}$, the total number of dofs might change $\mathcal{N}_{\overline{\boldsymbol{\mu}}} \neq \mathcal{N}_{\boldsymbol{\mu}}$ (also mesh-adaptive strategies).
Local clustering techniques can help but unsuitable for new parametric instance $\boldsymbol{\mu} \notin \{\boldsymbol{\mu}_i\}_{i=1}^{N_s}$.

GNN: Given $G = (V, E)$ with vertices V and edges $E \subseteq V \times V$, a GNN receive *input* node features $\mathbf{u} : V \to \mathbb{R}^l$, and return *output* node features $\mathbf{u}' : V \to \mathbb{R}^{l'}$.
Remarks: (i) process data coming from *different* graphs, (ii) same *size* of the input and output features, q and q', (iii) each input-output pair must be defined over the same *graph*.

Problem: Time-dependent and parametrized PDE in $\Omega(\boldsymbol{\mu}) \times (0, T)$ with N_t equally spaced subintervals

$$\begin{cases} \dfrac{\mathbf{u}_{\boldsymbol{\mu}}^{n+1} - \mathbf{u}_{\boldsymbol{\mu}}^{n}}{\Delta t} = \mathbf{f}\left(t^{n+1}, \mathbf{u}_{\boldsymbol{\mu}}^{n+1}, \boldsymbol{\mu}\right), \qquad n \geq 0, \\ \mathbf{u}_{\boldsymbol{\mu}}^{0} = \mathbf{g}_{\boldsymbol{\mu}}. \end{cases}$$

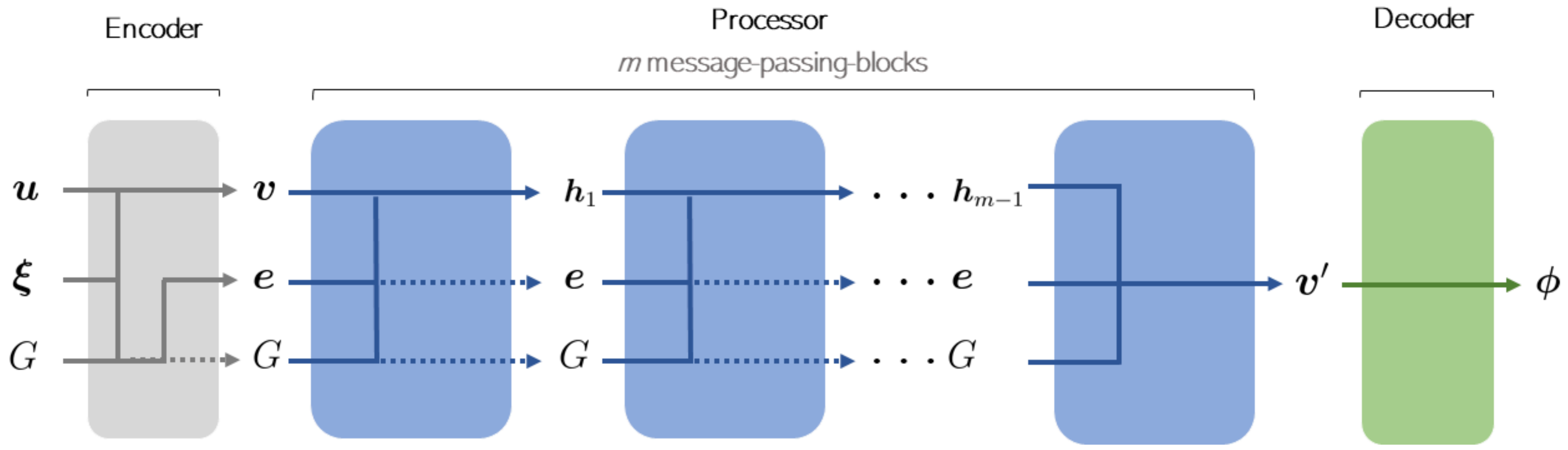
Strategy: Construct a GNN $\boldsymbol{\Phi}$ such that the solution is given by $\mathbf{u}_{\boldsymbol{\mu}}^{n+1} \approx \boldsymbol{\Phi}(\mathbf{u}_{\boldsymbol{\mu}}^{n}, t^n, \boldsymbol{\mu})$.

MeshGraphNets architecture

The **Encoder-Processor-Decoder** model is a GNN architecture that can process mesh-based data

- a directed graph $G = (V, E)$ associated to some mesh $\Omega_{\boldsymbol{\mu}}$ with $V \subset \mathbb{R}^d$
- an input signal defined over the mesh vertices, namely $\boldsymbol{u} : V \to \mathbb{R}^q$
- a global feature vector, $\boldsymbol{\xi} \in \mathbb{R}^s$, describing a given nonspatial property of the system (time, param.)

$\rightsquigarrow$ the output of such a model is a new signal $\boldsymbol{u}' : V \to \mathbb{R}^{q'}$ defined over the given mesh.



[1] Franco, N.R., Fresca, S., Tombari, F., Manzoni, A., 2023. Deep learning-based surrogate models for parametrized PDEs: Handling geometric variability through graph neural networks. Chaos. https://doi.org/10.1063/5.0170101

MeshGraphNets architecture

1. **Encoder Module**: It preprocess the input data and return a collection of hidden nodes/edges features, expanding them with information coming from the global variables $\boldsymbol{\xi}$ and G, and two MLP units $\Psi_{\mathcal{E}}^{v} : \mathbb{R}^{q+s+1} \to \mathbb{R}^{l}$ and $\Psi_{\mathcal{E}}^{e} : \mathbb{R}^{d+1} \to \mathbb{R}^{l}$, map onto the hidden-state space as $\mathcal{E}_v(\mathbf{u}, \boldsymbol{\xi}, G) := \Psi_{\mathcal{E}}^{v} \circ (\mathbf{u} \oplus \boldsymbol{\xi} \oplus \boldsymbol{b}_G)$, and $\mathcal{E}_e(\mathbf{u}, \boldsymbol{\xi}, G) := \Psi_{\mathcal{E}}^{e} \circ e_G$.
2. **Processor Module**: The encoded features, $\mathbf{v} := \mathcal{E}_v(\mathbf{u}, \boldsymbol{\xi}, G)$ and $\mathbf{e} := \mathcal{E}_e(G)$, are then elaborated by a GNN processor $\mathcal{P}$ with m message-passing-blocks $F_1, \ldots, F_m$ for each $j = 1, \ldots, m-1$, as
$$\mathcal{P}(\mathbf{v}, \mathbf{e}, G) := F_m(\boldsymbol{h}_m, \mathbf{e}, G), \quad \text{with} \quad \boldsymbol{h}_1 = \mathbf{v}, \text{ and } \boldsymbol{h}_{j+1} = F_j(\boldsymbol{h}_j, \mathbf{e}, G),$$
3. **Decoder Module**: The processed node fearures $\mathbf{v}' := \mathcal{P}(\mathbf{v}, \mathbf{e}, G)$, with $\mathbf{v}' : V \to \mathbb{R}^{l}$, are decoded to recover the desired output in $\mathbb{R}^{q}$, via a suitable MLP unit $\Psi_{\mathcal{D}} : \mathbb{R}^{l} \to \mathbb{R}^{q}$, transforming the original l features onto the q desired ones, operating nodewise as $\mathcal{D}(\mathbf{v}') := \Psi_{\mathcal{D}} \circ \mathbf{v}'$.

$\rightsquigarrow$ The **Encoder-Processor-Decoder** model reads $\Phi(\mathbf{u}, \boldsymbol{\xi}, G) := \mathcal{D}(\mathcal{P}(\mathcal{E}_v(\mathbf{u}, \boldsymbol{\xi}, G), \mathcal{E}_e(G), G))$.

Surrogate modeling of parametrized PDEs

Goal: predict an approximate solution $\widetilde{\mathbf{u}}$ at time t^{n+1}, given the state of the system at time t^n, for each node $i = 1, \dots, \mathcal{N}_{\boldsymbol{\mu}}$ of the computational mesh $\Omega_{\boldsymbol{\mu}}$, as $\Phi(\mathbf{u}_{\boldsymbol{\mu}}^n, t^n, \boldsymbol{\mu}) \approx \mathbf{u}_{\boldsymbol{\mu}}^{n+1}$.

Idea: explicit Runge-Kutta methods, modelling the time stepping scheme defined by the GNN $\tilde{\Phi}$

$$\Phi(\mathbf{v}, t^n, \boldsymbol{\mu}) := \mathbf{v} + \Delta t \tilde{\Phi}(\mathbf{v}, t^n, \Omega_{\boldsymbol{\mu}}),$$

Remark:the global feature is defined as $\xi = t^n$, while the dof vector as $\mathbf{u}_{\boldsymbol{\mu}}^n \in \mathbb{R}^{\mathcal{N}_{\boldsymbol{\mu}}}$.

Training: The dataset $\{\boldsymbol{\mu}_i, \mathbf{u}_{\boldsymbol{\mu}_i}^0, \dots, \mathbf{u}_{\boldsymbol{\mu}_i}^{N_t}\}_{i=1}^{N_{\text{train}}}$, contains a total of N_{train} different trajectories, corresponding to different geometrical configurations $\boldsymbol{\mu}$. The loss function to minimize is given as

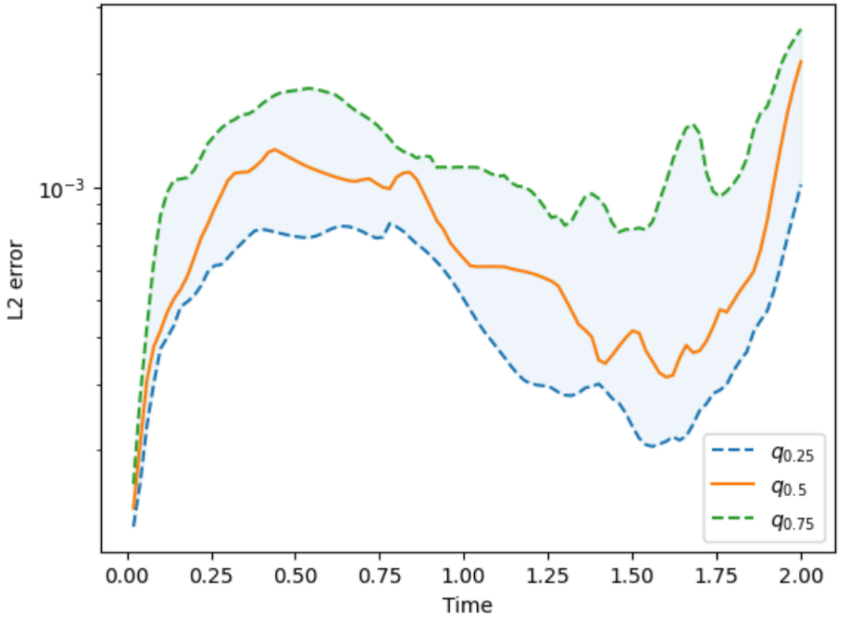
$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}} N_t} w_1 \sum_{i=1}^{N_{\text{train}}} \sum_{n=0}^{N_t - 1} |\mathbf{u}_{\boldsymbol{\mu}}^{n+1} - \mathbf{u}_{\boldsymbol{\mu}}^n - \Delta t \tilde{\Phi}_{\boldsymbol{\theta}}(\mathbf{u}_{\mu_i}^n, t^n, \Omega_{\boldsymbol{\mu}_i})|^2 + \frac{1}{N_{\text{train}} N_t} w_2 \sum_{i=1}^{N_{\text{train}}} \sum_{n=0}^{N_t - 1} |\dot{\mathbf{u}}_{\boldsymbol{\mu}}^n - \tilde{\Phi}_{\boldsymbol{\theta}}(\mathbf{u}_{\mu_i}^n, t^n, \Omega_{\boldsymbol{\mu}_i})|^2,$$

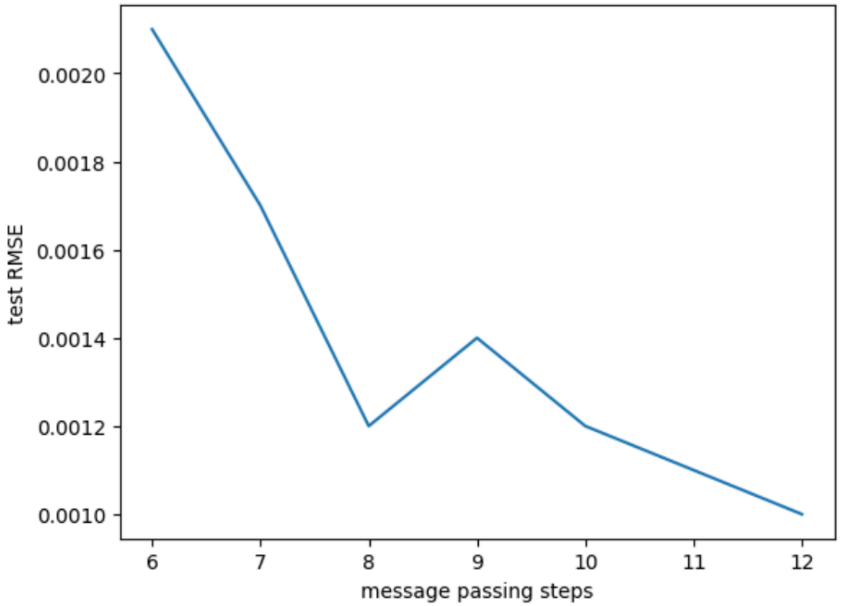
MeshGraphNet results (advection-diffusion problem)

Advection-diffusion problem in $\Omega = (0,1)^2 \setminus B_{0.15}(c_x, c_y)$, $T = 2$, $D = 0.1$ and $\mathbf{b}(t) = [1-t, 1-t]$

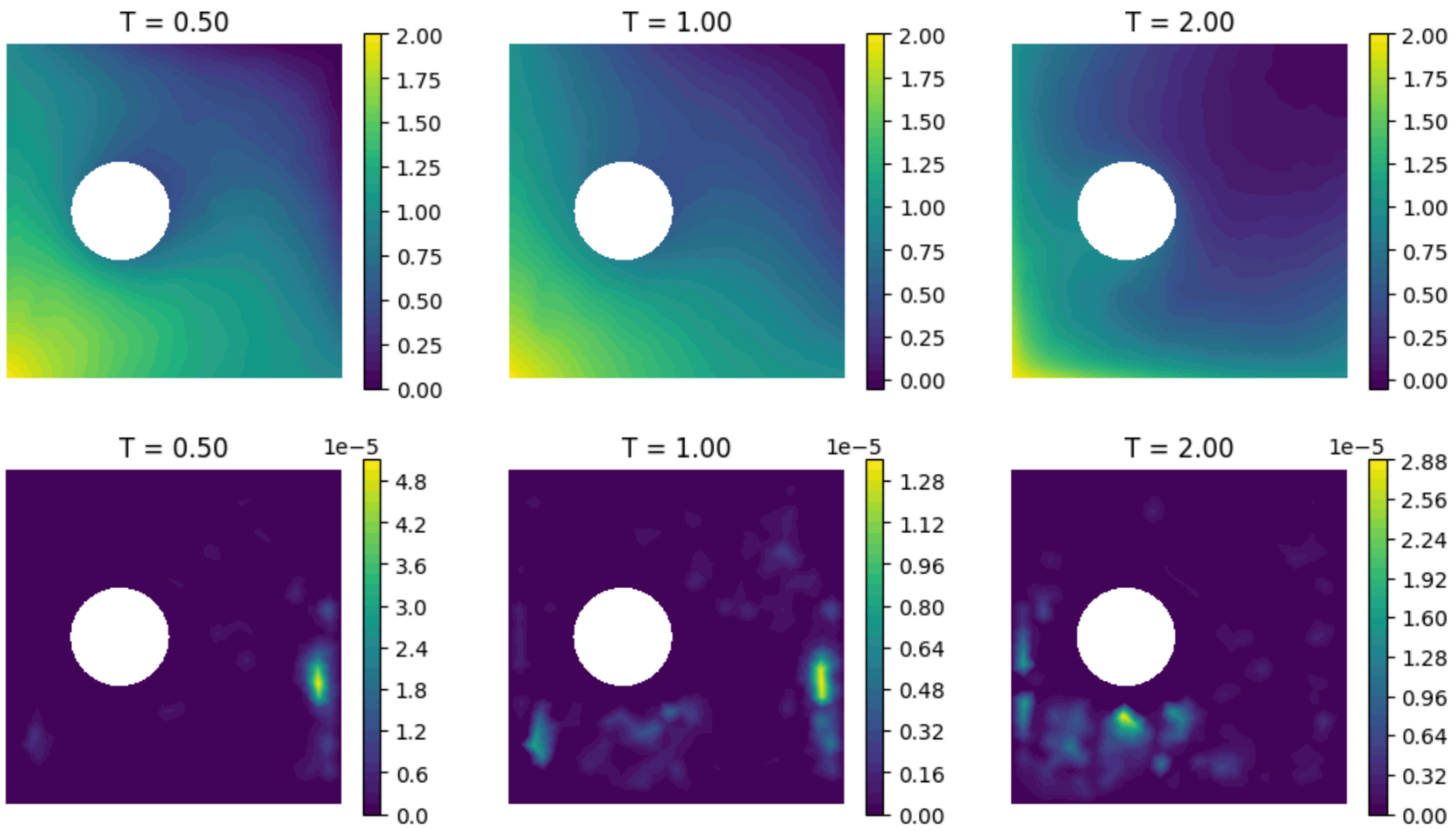
$$\begin{cases} \dfrac{\partial u}{\partial t} - D\Delta u + \mathbf{b}(t) \cdot \nabla u = 0 & \text{in } \Omega \times (0,T] \\ u(x,y) = (x-1)^2 + (y-1)^2 & \text{on } \partial\Omega \times (0,T] \\ u_0(x,y) = (x-1)^2 + (y-1)^2 & \text{in } \Omega, \end{cases}$$

Remark $N_t = 101$, $N_{\text{train}} = 100$, $q = 1$, $N_{\boldsymbol{\mu}} \in [770, 790]$, $r_t = 80$

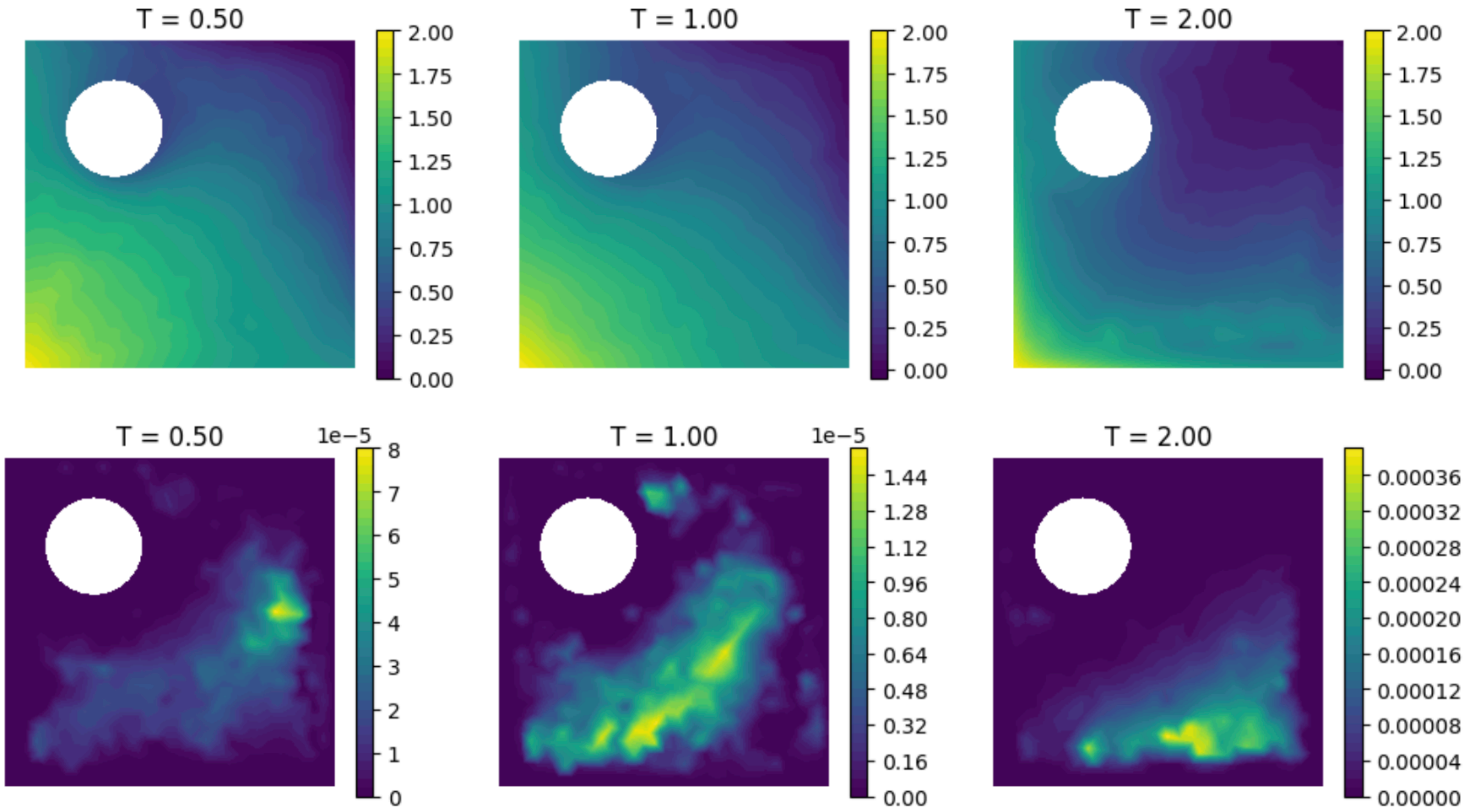




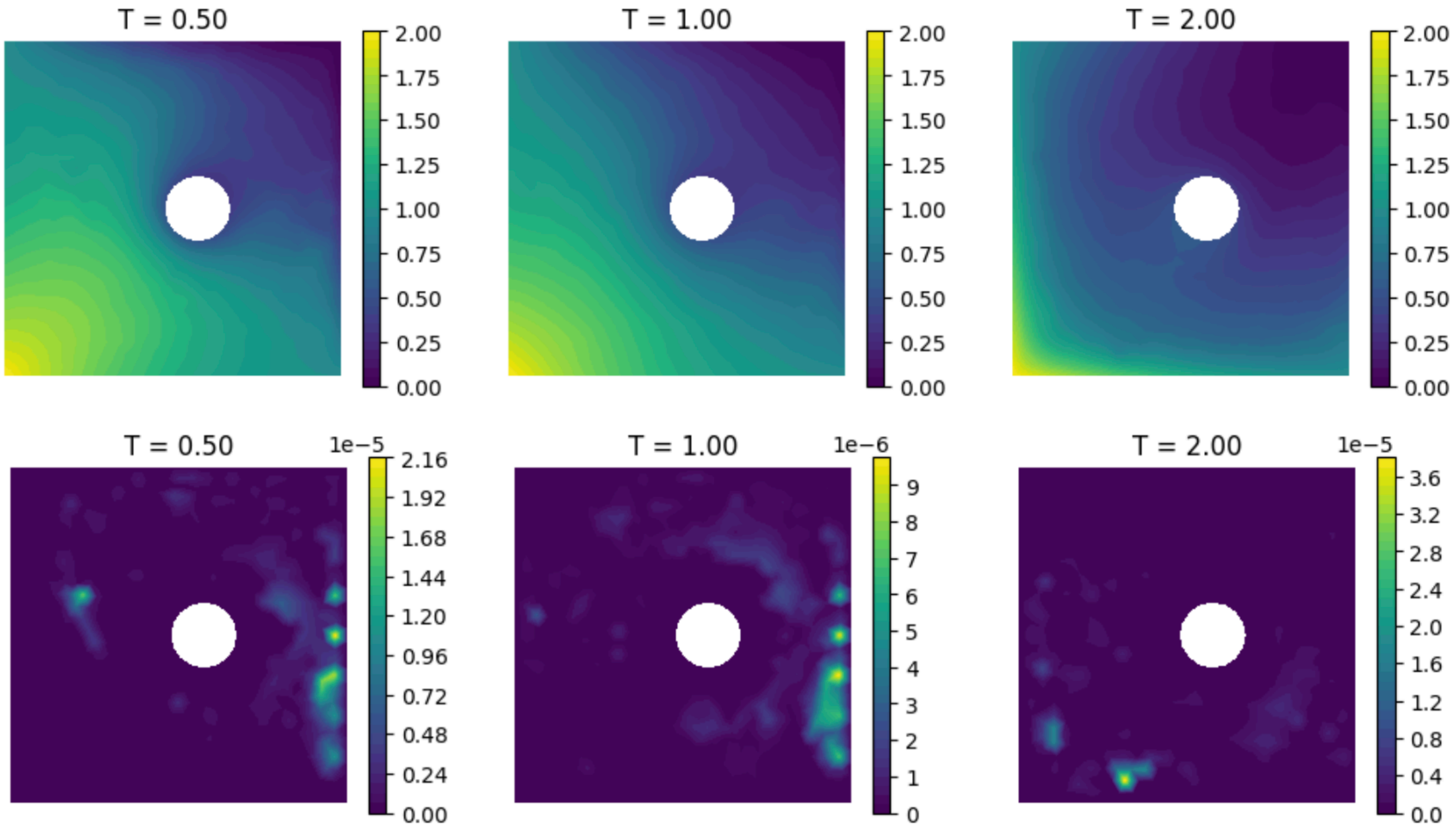
MeshGraphNet results (advection-diffusion problem)



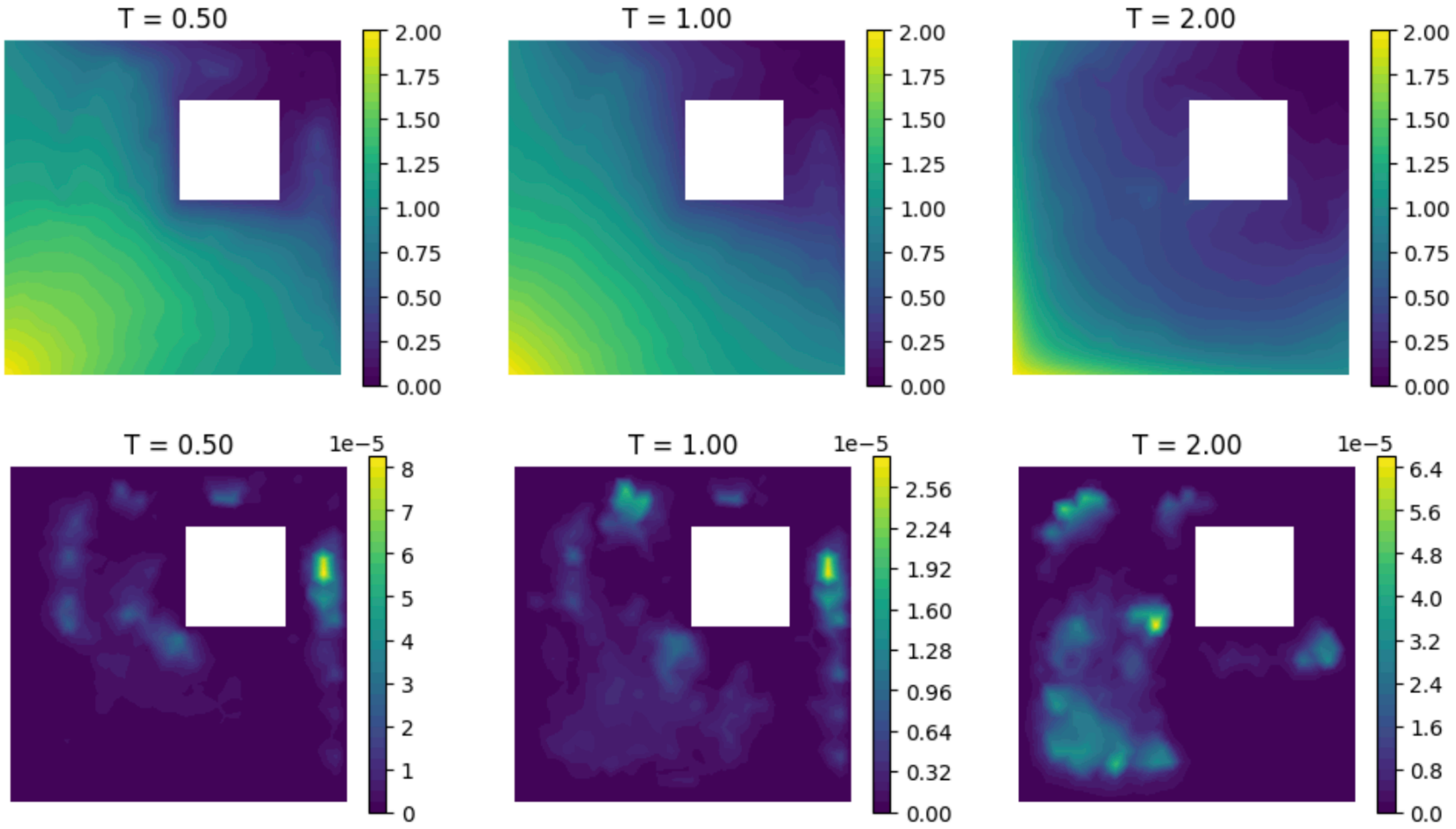
MeshGraphNet results (advection-diffusion problem)



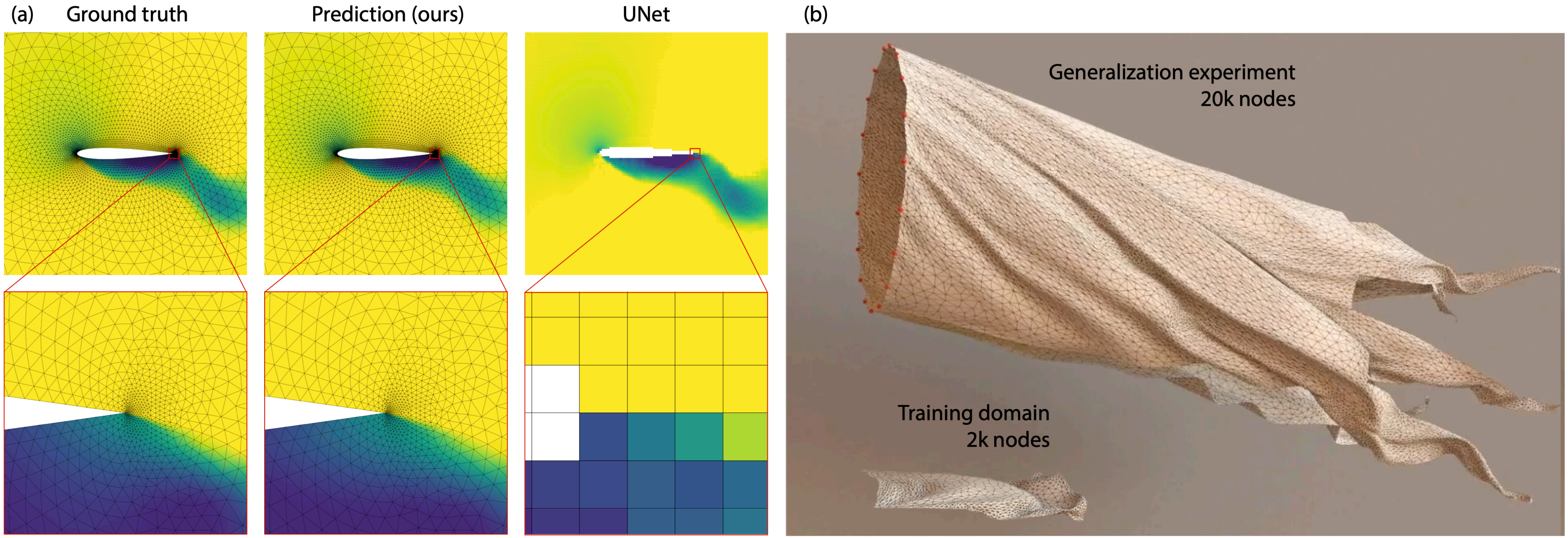
MeshGraphNet results (advection-diffusion problem)



MeshGraphNet results (advection-diffusion problem)



MeshGraphNet original results

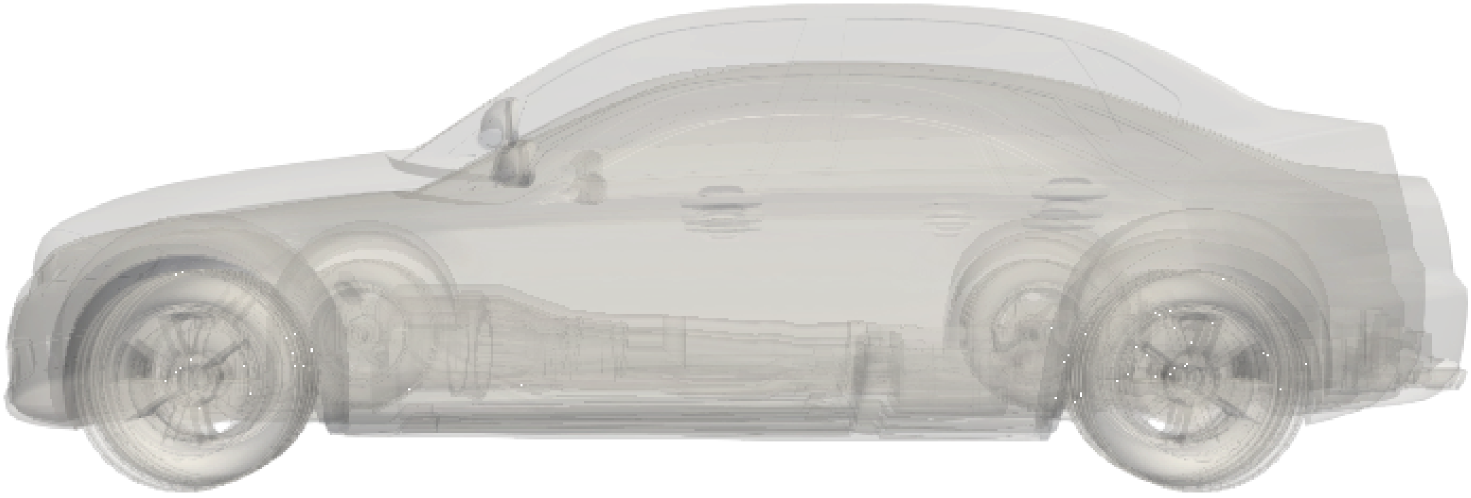


[1] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., Battaglia, P.W., 2020. Learning to Simulate Complex Physics with Graph Networks. arXiv:2002.09405

X-MeshGraphNet

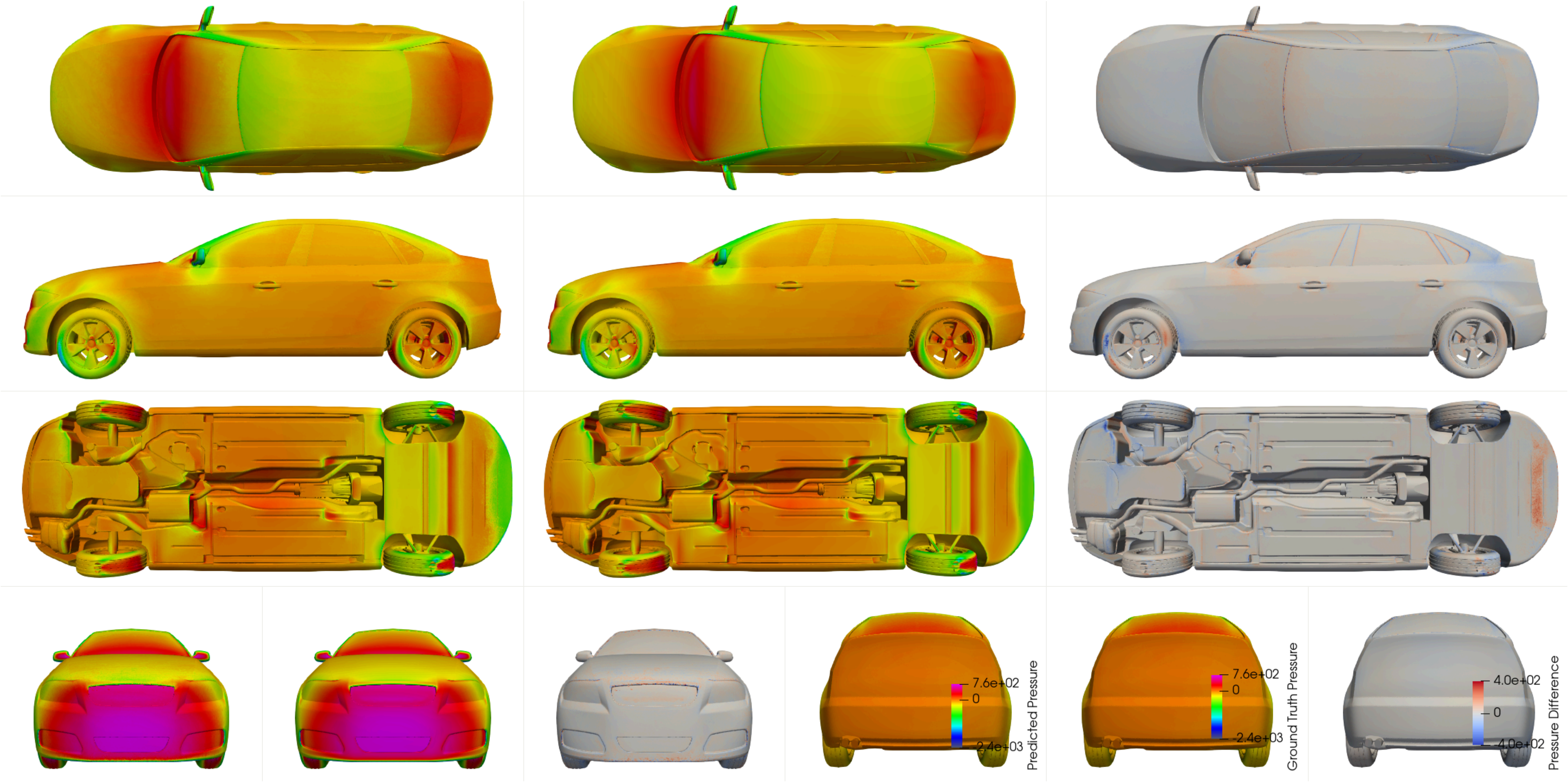
DrivAerML dataset of high-fidelity surface pressure, wall shear stress, and flow-field quantities for 500 parametrically morphed geometries of the DrivAer notchback vehicle.

Idea: overcoming the scalability bottleneck by partitioning large graphs and incorporating regions that enable seamless message passing across partitions.

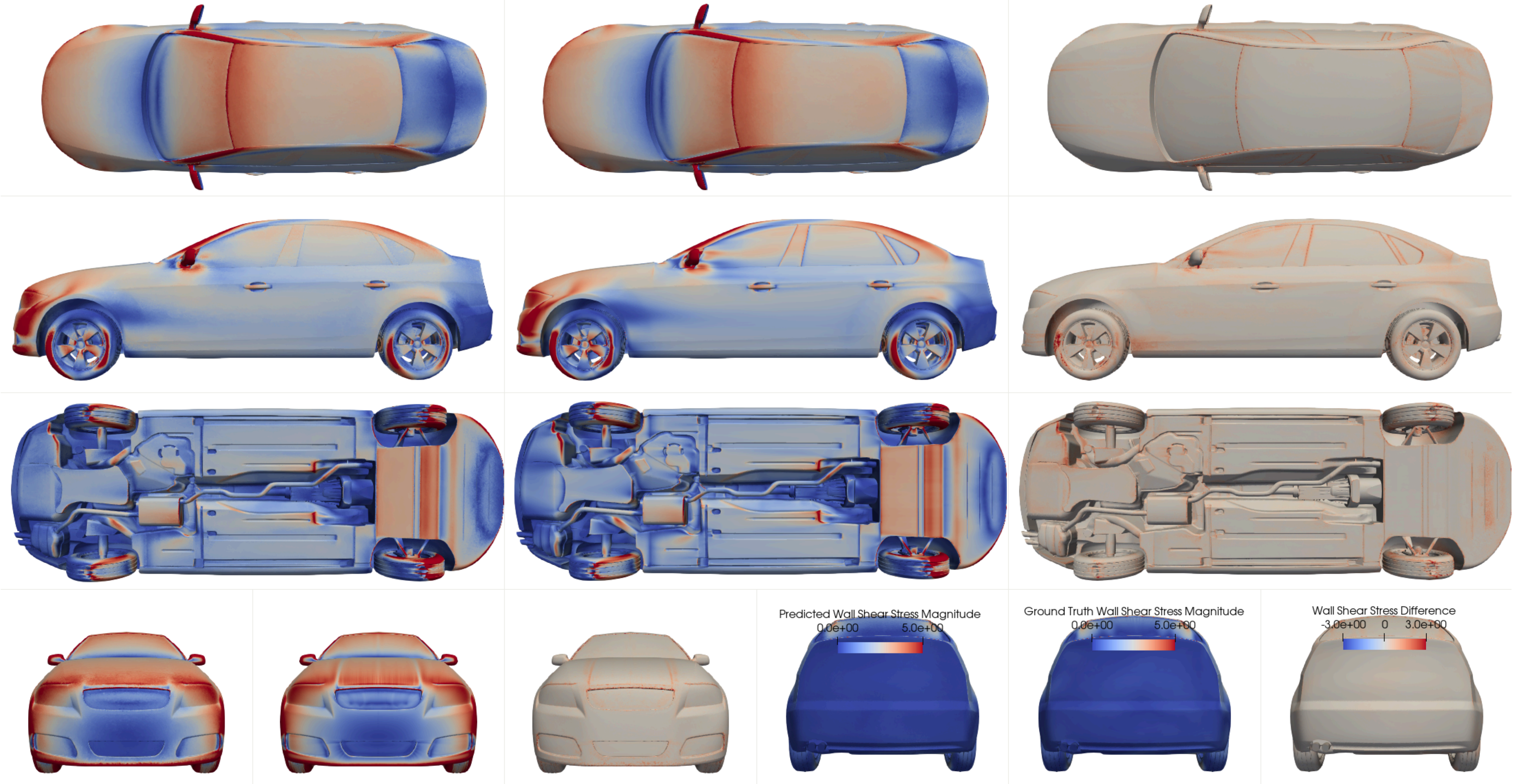


[1] Nabian, M.A., Liu, C., Ranade, R., Choudhry, S., 2020. X-MeshGraphNet: Scalable Multi-Scale Graph Neural Networks for Physics Simulation. arXiv:2411.17164

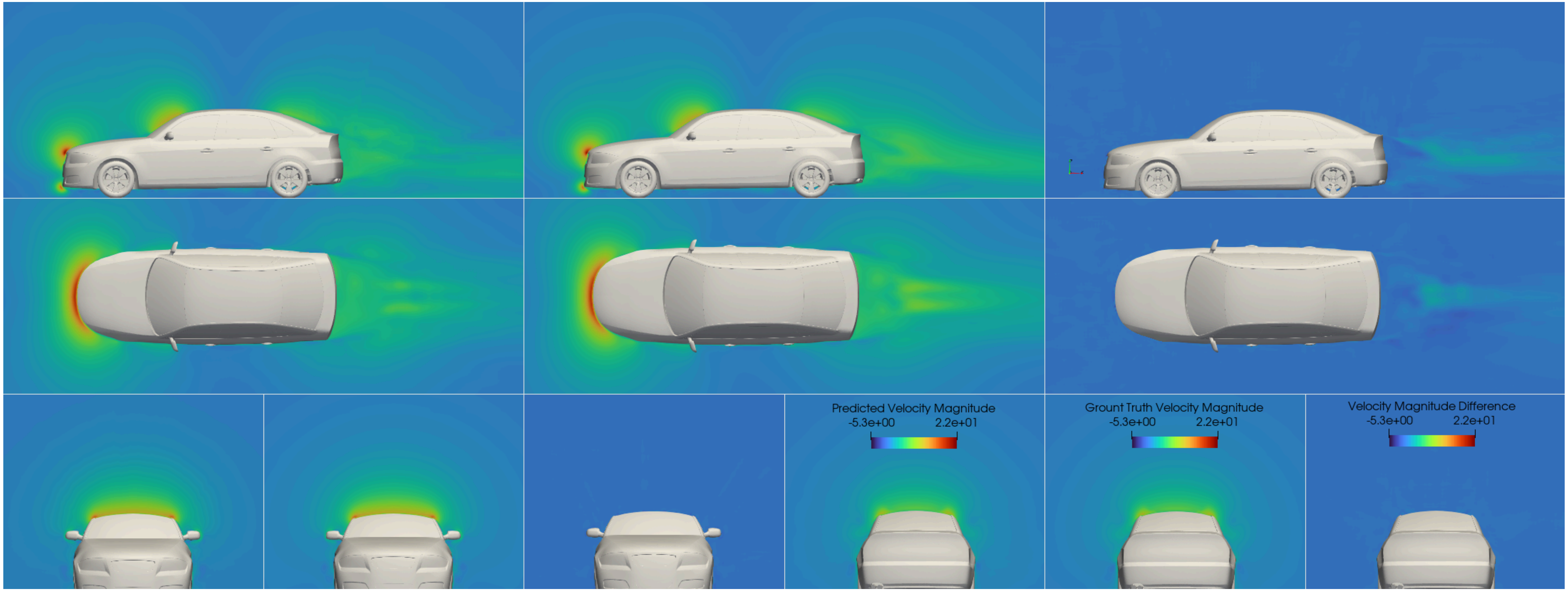
X-MeshGraphNet results (DrivAerML)



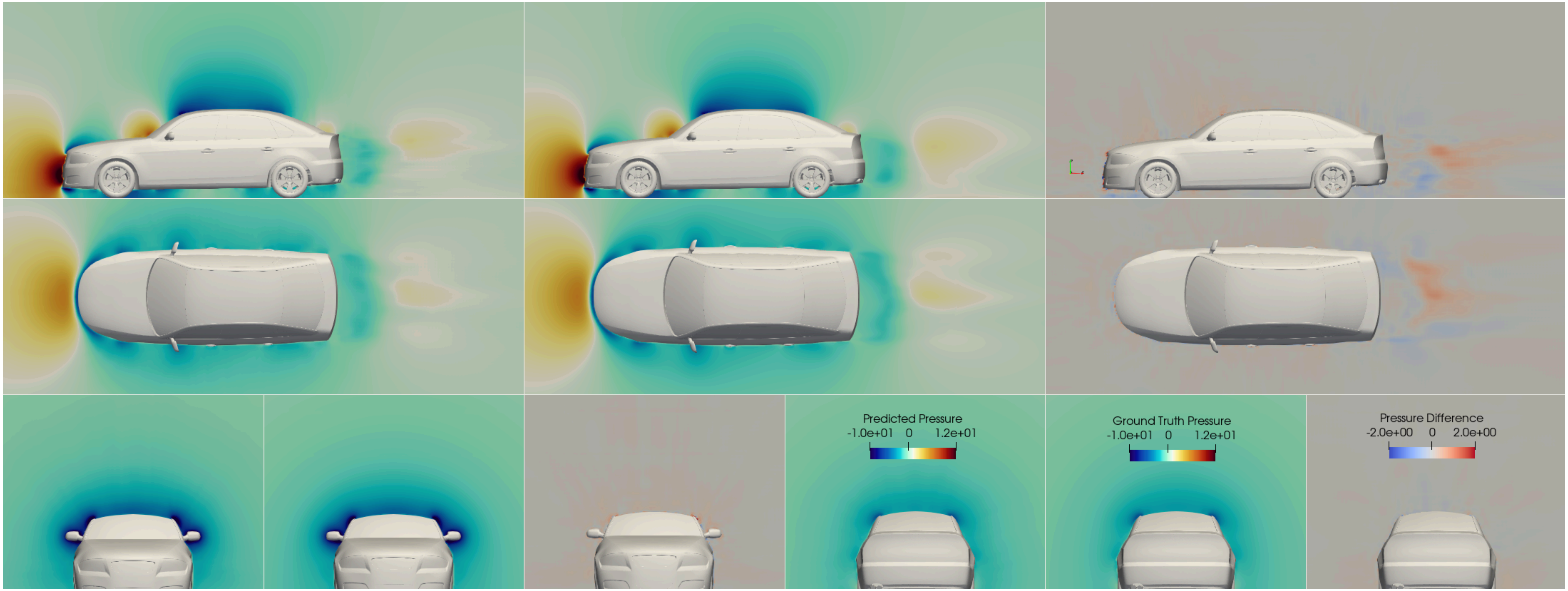
X-MeshGraphNet results (DrivAerML)



X-MeshGraphNet results (DrivAerML)



X-MeshGraphNet results (DrivAerML)



References

- Fresca, S., Dede', L., Manzoni, A., 2021. A Comprehensive Deep Learning-Based Approach to Reduced Order Modeling of Nonlinear Time-Dependent Parametrized PDEs. J Sci Comput 87, 61. https://doi.org/10.1007/s10915-021-01462-7
- Fresca, S., Manzoni, A., 2022. POD-DL-ROM: Enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition. Computer Methods in Applied Mechanics and Engineering 388, 114181. https://doi.org/10.1016/j.cma.2021.114181
- Pichi, F., Moya, B., Hesthaven, J.S., 2024. A graph convolutional autoencoder approach to model order reduction for parametrized PDEs. Journal of Computational Physics 501, 112762. https://doi.org/10.1016/j.jcp.2024.112762
- Morrison, O.M., Pichi, F., Hesthaven, J.S., 2024. GFN: A graph feedforward network for resolution-invariant reduced operator learning in multifidelity applications. Computer Methods in Applied Mechanics and Engineering 432, 117458. https://doi.org/10.1016/j.cma.2024.117458
- Romor, F., Stabile, G., Rozza, G., 2023. Non-linear Manifold Reduced-Order Models with Convolutional Autoencoders and Reduced Over-Collocation Method. J Sci Comput 94, 74. https://doi.org/10.1007/s10915-023-02128-2
- Lee, K., Carlberg, K.T., 2020. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. Journal of Computational Physics 404, 108973. https://doi.org/10.1016/j.jcp.2019.108973
- Reiss, J., Schulze, P., Sesterhenn, J., Mehrmann, V., 2018. The Shifted Proper Orthogonal Decomposition: A Mode Decomposition for Multiple Transport Phenomena. SIAM J. Sci. Comput. 40, A1322–A1344. https://doi.org/10.1137/17M1140571
- Taddei, T., 2020. A Registration Method for Model Order Reduction: Data Compression and Geometry Reduction. SIAM J. Sci. Comput. 42, A997–A1027. https://doi.org/10.1137/19M1271270

References

- Romor, F., Torlo, D., Rozza, G., 2023. Friedrichs' systems discretized with the Discontinuous Galerkin method: domain decomposable model order reduction and Graph Neural Networks approximating vanishing viscosity solutions.
- Romor, F., Galarce, F., Brüning, J., Goubergrits, L., Caiazzo, A., 2025. Data assimilation performed with robust shape registration and graph neural networks: application to aortic coarctation. https://doi.org/10.48550/arXiv.2502.12097
- Franco, N.R., Fresca, S., Tombari, F., Manzoni, A., 2023. Deep learning-based surrogate models for parametrized PDEs: Handling geometric variability through graph neural networks. Chaos: An Interdisciplinary Journal of Nonlinear Science 33, 123121. https://doi.org/10.1063/5.0170101
- Franco, N.R., Manzoni, A., Zunino, P., 2022. Learning Operators with Mesh-Informed Neural Networks.
- Vitullo, P., Colombo, A., Franco, N.R., Manzoni, A., Zunino, P., 2023. Nonlinear model order reduction for problems with microstructure using mesh informed neural networks.
- Barwey, S., Pal, P., Patel, S., Balin, R., Lusch, B., Vishwanath, V., Maulik, R., Balakrishnan, R., 2024. Mesh-based Super-Resolution of Fluid Flows with Multiscale Graph Neural Networks. https://doi.org/10.48550/arXiv.2409.07769
- Barwey, S., Shankar, V., Viswanathan, V., Maulik, R., 2023. Multiscale graph neural network autoencoders for interpretable scientific machine learning. Journal of Computational Physics 495, 112537. https://doi.org/10.1016/j.jcp.2023.112537
- Pegolotti, L., Pfaller, M.R., Rubio, N.L., Ding, K., Brufau, R.B., Darve, E., Marsden, A.L., 2023. Learning Reduced-Order Models for Cardiovascular Simulations with Graph Neural Networks.

References

- Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., Battaglia, P.W., 2021. Learning Mesh-Based Simulation with Graph Networks.
- Rowbottom, J., Maierhofer, G., Deveney, T., Schratz, K., Liò, P., Schönlieb, C.-B., Budd, C., 2024. G-Adaptive mesh refinement -- leveraging graph neural networks and differentiable finite element solvers.
- D'Inverno, G.A., Bianchini, M., Sampoli, M.L., Scarselli, F., 2023. On the approximation capability of GNNs in node classification/regression tasks. https://doi.org/10.48550/arXiv.2106.08992
- Matray, V., Amlani, F., Feyel, F., Néron, D., 2024. A hybrid numerical methodology coupling Reduced Order Modeling and Graph Neural Networks for non-parametric geometries: applications to structural dynamics problems. https://doi.org/10.48550/arXiv.2406.02615
- Brivio, S., Fresca, S., Franco, N.R., Manzoni, A., 2023. Error estimates for POD-DL-ROMs: a deep learning framework for reduced order modeling of nonlinear parametrized PDEs enhanced by proper orthogonal decomposition. https://doi.org/10.48550/arXiv.2305.04680
- Mousavi, S., Wen, S., Lingsch, L., Herde, M., Raonić, B., Mishra, S., 2025. RIGNO: A Graph-based framework for robust and accurate operator learning for PDEs on arbitrary domains. https://doi.org/10.48550/arXiv.2501.19205
- Brandstetter, J., Worrall, D., Welling, M., 2023. Message Passing Neural PDE Solvers.
- Bronstein, M.M., Bruna, J., Cohen, T., Veličković, P., 2021. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. https://doi.org/10.48550/arXiv.2104.13478

References

- Hernández, Q., Badías, A., Chinesta, F., Cueto, E., 2024. Thermodynamics-Informed Graph Neural Networks. IEEE Transactions on Artificial Intelligence 5, 967–976. https://doi.org/10.1109/TAI.2022.3179681
- Giovanni, F.D., Rusch, T.K., Bronstein, M.M., Deac, A., Lackenby, M., Mishra, S., Veličković, P., 2024. How does over-squashing affect the power of GNNs? https://doi.org/10.48550/arXiv.2306.03589
- Gladstone, R.J., Rahmani, H., Suryakumar, V., Meidani, H., D'Elia, M., Zareei, A., 2023. GNN-based physics solver for time-independent PDEs.
- Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., n.d. Multipole Graph Neural Operator for Parametric Partial Differential Equations.
- Li, Z., Kovachki, N.B., Choy, C., Li, B., Kossaifi, J., Otta, S.P., Nabian, M.A., Stadler, M., Hundt, C., Azizzadenesheli, K., Anandkumar, A., 2023. Geometry-Informed Neural Operator for Large-Scale 3D PDEs.
- Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A., 2020. Neural Operator: Graph Kernel Network for Partial Differential Equations.
- Lin, R.Y., Berner, J., Duruisseaux, V., Pitt, D., Leibovici, D., Kossaifi, J., Azizzadenesheli, K., Anandkumar, A., 2025. Enabling Automatic Differentiation with Mollified Graph Neural Operators. https://doi.org/10.48550/arXiv.2504.08277
- Magargal, L.K., Khodabakhshi, P., Rodriguez, S.N., Jaworski, J.W., Michopoulos, J.G., 2024. Projection-based model-order reduction for unstructured meshes with graph autoencoders. https://doi.org/10.48550/arXiv.2407.13669