Accelerating PDE Surrogates via RL-Guided Mesh Optimization

Yang Meng[1] **Ruoxi Jiang**[2] **Zhuokai Zhao**[1] **Chong Liu**[3] **Rebecca Willett**[1] **Yuxin Chen**[1]

[1] University of Chicago [2] Fudan University [3] University at Albany

Abstract

Deep surrogate models for parametric partial differential equations (PDEs) can deliver high-fidelity approximations but remain prohibitively data-hungry: training often requires thousands of fine-grid simulations, each incurring substantial computational cost. To address this challenge, we introduce RLMesh, an end-to-end framework for efficient surrogate training under limited simulation budget. The key idea is to use reinforcement learning (RL) to adaptively allocate mesh grid points non-uniformly within each simulation domain, focusing numerical resolution in regions most critical for accurate PDE solutions. A lightweight proxy model further accelerates RL training by providing efficient reward estimates without full surrogate retraining. Experiments on PDE benchmarks demonstrate that RLMesh achieves competitive accuracy to baselines but with substantially fewer simulation queries. These results show that solver-level spatial adaptivity can dramatically improve the efficiency of surrogate training pipelines, enabling practical deployment of learning-based PDE surrogates across a wide range of problems.

1 INTRODUCTION

Deep learning–based surrogates for parametric partial differential equations (PDEs) have emerged as powerful tools for accelerating scientific simulation [Brandstetter et al., 2022, Kovachki et al., 2023, Lippe et al., 2023, Yu and Wang, 2024, Hu et al., 2024, Lai et al., 2024, Tang et al., 2024, Jiang et al., 2025]. Popular approaches—such as Fourier neural operators (FNOs) [Li et al., 2020a, 2023], DeepONets [Lu et al., 2021], and more recent graph- and transformer-based operators [Li et al., 2020b, Guibas et al., 2021]—learn mappings from PDE inputs (e.g., coefficients, boundary conditions, or initial conditions) to solutions, and have achieved state-of-the-art accuracy across diverse physical systems [Kovachki et al., 2023]. However, they are inherently data-hungry: training typically requires thousands of high-fidelity numerical simulations on fine spatial grids, each of which carries substantial computational cost. Such reliance on expensive solver calls become a critical bottleneck for practical deployment.

A central inefficiency is that most surrogate training pipelines rely on expensive numerical simulations obtained on dense, structured meshes even when the underlying solutions are strongly inhomogeneous (e.g., shocks in Burgers' equation [Burgers, 1948], high-conductivity channels in Darcy flow [Homsy, 1987]). Some recent works begin to relax operator learning to simulations with irregular, non-uniform meshes—for instance, Geo-FNO [Li et al., 2023] learns a deformation to handle irregular geometries; MAgNet [Boussif et al., 2022] learns a continuous, mesh-agnostic PDE solver by combining coordinate-based implicit neural representations with graph neural networks—yet these approaches do not decide where to spend solver queries during training, and still rely on many high-fidelity samples. We therefore ask: Can we design a learning algorithm that adaptively allocates solver calls to a small set of mesh grid points per instance, so that we can train a surrogate PDE solver more efficiently, without sacrificing accuracy?

Designing such a system raises several challenges: (i) *Sparse and expensive feedback:* Naively measuring surrogate improvement after each newly queried point would entail frequent full retraining, which is computationally prohibitive. (ii) *Budgeted exploration:* Under limited budget for solver calls, the adaptive mesh allocation policy must balance exploring new regions with exploiting known informative structures. (iii) *Per-instance adaptivity:* the most informative mesh grid points depend on each instance's input field and local geometry, so a one-size-fits-all mesh is suboptimal.

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. Copyright 2026 by the author(s).

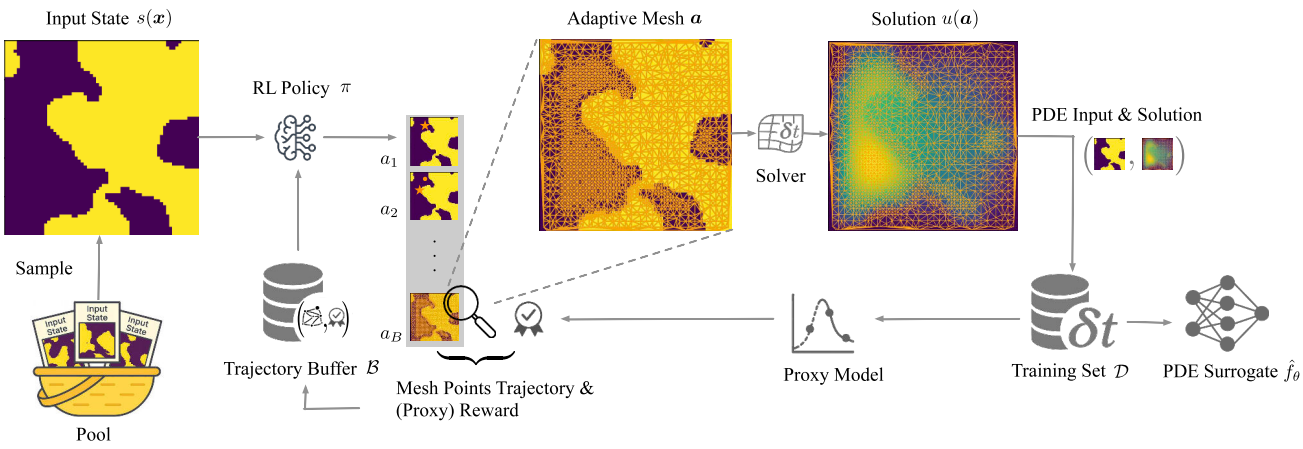


Figure 1: The RLMesh framework. An input state (e.g., PDE initial condition) is sampled from a prior distribution and passed to an RL policy that selects a small set of mesh grid points under a budget. The PDE solver is queried only at these points to produce high-fidelity but sparse observations, which are stored in an updated dataset for surrogate training. A lightweight proxy model provides fast estimates of surrogate improvement and provides reward signal for the RL agent, updating the policy and closing the loop.

Our approach. We introduce RLMesh, which formulates per-instance *mesh grid point selection* as a sequential decision process (Fig. 1). For each sampled PDE input state (e.g., an initial condition), a reinforcement learning (RL) policy adaptively selects a small set of mesh grid points; the solver is queried *only* at these locations, yielding sparse but informative observations on a non-uniform mesh that are added to the training set. To avoid prohibitively expensive full retraining as feedback, we introduce a lightweight *proxy model* that predicts the surrogate's improvement attributable to the newly acquired data and supplies a dense, well-aligned reward for RL. The surrogate (instantiated as an FNO) is retrained periodically on the accumulated adaptively sampled data, using a simple set-to-grid interface to handle irregular inputs. In effect, RLMesh alternates between two steps: (i) *policy improvement* via proxy-rewarded trajectories on new instances, and (ii) *surrogate improvement* via periodic retraining on the extended, non-uniform dataset.

Our empirical results on several PDE benchmarks demonstrate that RLMesh consistently outperforms uniform, random, and several heuristic-based mesh allocation baselines under equal query budgets, achieving significantly lower reconstruction error at substantially reduced simulation cost. With only a fraction of full-grid queries, our surrogate approaches the accuracy of models trained on dense data. The proxy model (e.g., kernel ridge regression) exhibits strong correlation with actual surrogate improvements, enabling stable RL training without excessive retraining of the (more computationally expensive) PDE surrogate. Ablations further confirm the benefits of per-instance adaptivity and the importance of proxy alignment.

In summary, our contributions are:

- We formulate adaptive, budget-constrained mesh-point selection for PDE surrogates as a sequential decision problem (§3).
- We introduce an end-to-end RL-guided active surrogate learning framework for this problem, where a lightweight proxy model is used to approximate downstream surrogate gains, enabling practical RL reward assignment without frequent full retrains (§4).
- We demonstrate empirically that our method achieves superior accuracy–cost trade-offs compared to strong baselines (§5).

2 RELATED WORK

Active learning for PDE surrogates. A growing body of work has explored active learning (AL) to accelerate PDE surrogate modeling by judiciously selecting training samples. Most existing approaches operate at the *instance* level—i.e. deciding which new PDE instances to simulate next for inclusion in the training set. For example, Wu et al. [2023] disentangle instance- and fidelity-level uncertainty via generative modeling, but their acquisition functions are designed for low-dimensional summary outputs and do not natively handle spatially-structured fields. More recently, MRA-FNO [Li et al., 2024] introduces a cost-aware strategy for selecting both instances and mesh resolutions in training an FNO. By maximizing a utility-cost ratio, MRA-FNO chooses at each step an instance and one of several predefined grid resolutions

to simulate, thereby saving data cost. While effective at the dataset level, this approach still confines resolution to a fixed set of global grids and does not perform fine-grained spatial adaptivity *within* each instance. Similarly, AL4PDE [Musekamp et al., 2025] evaluates uncertainty- and feature-based selection strategies (e.g. variance, mutual information), but these acquisitions choose whole instances to add and have no mechanism for adaptive mesh allocation inside an instance's domain. In short, existing AL frameworks for neural PDE solvers (e.g., MRA-FNO, AL4PDE) focus on instance-level acquisition—selecting which simulations or global resolutions to run under the assumption that each selected instance yields a full solution field—whereas RLMesh addresses a complementary and previously unmodeled axis of adaptivity: per-instance spatial point selection under a strict local query budget.

Adaptive mesh refinement. Adaptive mesh refinement (AMR) is a classical strategy in numerical PDE solvers [Melia et al., 2025]: the mesh is locally refined in regions of large error, steep gradients, or discontinuities, while keeping a coarse grid elsewhere to save cost; Early AMR methods introduced structured grid refinement with hierarchical data structures such as quadtrees/octrees [Berger and Oliger, 1984, Berger and Colella, 1989], and used error estimators or physics-based heuristics to decide where to refine. In the finite element community, *hp*-adaptivity provides an additional lever: solvers reduce element size (h) and/or increase polynomial degree (p) according to local regularity, yielding exponential convergence on suitably smooth subregions [Demkowicz, 2006]. Traditional methods use handcrafted error indicators or solve auxiliary equations to adapt the mesh during a single forward run for improved accuracy.

Recently, there is growing interest in using deep RL to *learn* mesh adaptation policies automatically. Foucart et al. [2023] formulate AMR as a sequential decision process in which an agent decides whether to refine or coarsen individual grid cells, and show that an RL policy can replicate classical refinement strategies without explicit error estimators. Yang et al. [2023] extend this to global mesh views with variable-size state/action spaces, using specialized architectures (including graph-based) to scale across meshes. Rowbottom et al. [2024] study mesh *relocation* (r-adaptivity) and use graph neural networks trained via differentiable finite element solvers to move mesh nodes and minimize the PDE solution error directly. Sarkar et al. [2024] present *SwarmRL*, a modular framework for multi-agent reinforcement learning in noisy physical environments, further illustrating the broader push toward RL-driven adaptivity in scientific systems. All of these methods tackle adaptivity for improving a solver's result *during a single simulation*. In contrast, our work uses adaptivity to improve *learning efficiency across many instances*: Instead of refining a mesh to solve one PDE instance accurately, we adaptively sample each instance's solution to collect the most informative training data for a surrogate model. This shifts the focus from solver-time to training-time efficiency and introduces a new feedback mechanism, using a proxy model to estimate surrogate improvement instead of relying on PDE residuals.

3 PROBLEM SETUP

Neural operator. Let $\Omega \subset \mathbb{R}^d$ denote the spatial coordinates (e.g., $d = 1$ represents the Burgers' equation; $d = 2$ for 2D Darcy flow) and $t \in [0, T]$ denote the physical time. We consider the PDE in its generic form written as:

$$G\big(u, \partial_x u, \partial_t u, \ldots, x, t, c(x)\big) = 0 \qquad x \in \Omega \tag{1}$$

subject to the governing equation G, initial condition $u_0(x)$, and spatially varying coefficients $c(x)$. For a time-discretized system, we view the solution at each time step t as a spatial field $u(x)$. The PDE solver then induces a solution operator f that, given the input functions (e.g., initial conditions, coefficients), returns the full solution:

$$f : \mathcal{S} \to \mathcal{U}, \qquad s(\cdot) \mapsto u(\cdot) \tag{2}$$

where $s \in \mathcal{S}$ is the input function $s : \Omega \to \mathbb{R}^{d_s}$ (e.g., $u_0(x)$, $c(x)$), and $u \in \mathcal{U}$ is the output function $u : \Omega \to \mathbb{R}^{d_u}$. In practice, Ω is discretized by n points $\Omega_n \equiv \boldsymbol{x} := \{x_1, x_2, \ldots, x_n\}$ with $x_i \in \mathbb{R}^d$. Then the functions are represented as finite-dimensional tensors $s(\boldsymbol{x}) \in \mathbb{R}^{n \times d_s}$ and $u(\boldsymbol{x}) \in \mathbb{R}^{n \times d_u}$.

The neural operator aims to train a network $\hat{f}(\cdot; \theta)$, with parameters θ, to approximate the solver f [Li et al., 2020a, Wen et al., 2022].

RL-guided mesh optimization. Our goal is to learn an adaptive policy for mesh point selection $\pi : \mathcal{S} \times 2^{\Omega_n} \to \Omega_n$ for the PDE surrogate training. The input of π includes the input function to the neural operator and the subset of mesh grid points already chosen; the output of π corresponds to an action $a \in \Omega_n$ as the next mesh grid point in spatial domain Ω_n to be selected.

The objective is to learn $\hat{\pi}$ that minimizes expected surrogate error:

$$\hat{\pi} = \arg\min_{\pi} \mathbb{E}\big[\ell(\hat{f}(s(\boldsymbol{x}); \theta_\pi), f(s(\boldsymbol{x})))\big]. \tag{3}$$

We assume there exists a numerical solver that can be queried online (see Fig. 1). In addition, we define a per-

instance budget B as the maximum number of mesh points allowed for each PDE instance. Querying the solver at chosen mesh points $\boldsymbol{a} := \{a_j\}_{j=1}^{B}$, we update the PDE surrogate model $\hat{f}(;\theta)$ using the training dataset containing historical samples and the newly arrived data pair $(s(\boldsymbol{x}), u(\boldsymbol{a}))$. Here $u(\boldsymbol{a}) := \tilde{f}(s(\boldsymbol{x}), \boldsymbol{a})$, with a solver $\tilde{f}$ taking non-structured meshes to focus on the solution at the mesh grid points $\boldsymbol{a}$.

The model is evaluated using RMSE loss on high-resolution validation instances

$$\ell(\hat{f}\big(s(\boldsymbol{x});\theta), f(s(\boldsymbol{x})\big) = \sqrt{\frac{1}{M}\sum_{j=1}^{M}\big(\hat{f}(s^{(j)};\theta) - f(s^{(j)})\big)^2}.$$

where $s^{(j)} := s^{(j)}(\boldsymbol{x})$ represents the input state at j-th sample, and M denotes the size of the validation dataset.

4 RL MESH OPTIMIZATION

In this section, we present RLMesh, an RL-guided active acquisition framework that learns. For each PDE instance, we select a small number of meaningful mesh points so that a downstream neural operator surrogate can be trained efficiently under a fixed budget.

4.1 High-Level Workflow

Following §3, let $s(\boldsymbol{x})$ denote the PDE input state and $\tilde{f}$ the numerical solver that returns the field $u(\cdot)$. For each sampled instance s, our goal is to sequentially choose a set of mesh points $\boldsymbol{a} \subseteq \Omega_n$ under a per-instance budget $|\boldsymbol{a}| \leq B$ and query the solver only at those locations; the resulting sparse measurements are added to a growing training set for the surrogate $\hat{f}(\cdot;\theta)$. To avoid full surrogate retraining after each mesh grid selection, we fit a proxy model that estimates the improvement of $\hat{f}(\cdot;\theta)$ due to the newly acquired data and uses that estimate as the reward for the RL policy. Periodically, we retrain $\hat{f}(\cdot;\theta)$ on all data collected so far.

Algorithm 1 shows the high-level workflow (as was illustrated in Fig. 1): RLMesh operates as an iterative loop where a new PDE instance is sampled, a set of mesh points is selected according to the current policy, the solver is queried at these locations to obtain sparse observations, a proxy reward is computed to guide policy optimization, and the surrogate is periodically retrained on the accumulated non-uniform data.

Algorithm 1: Learning Protocol

Input: Seed dataset $\mathcal{D}_0$; RL trajectory buffer storing transitions $\mathcal{B} := (s, \boldsymbol{a}, r, s')$ (s shows the current state, $\boldsymbol{a}$ is the action denoting the selected mesh grid points, r is the reward, s' is the next state); PDE solver $\tilde{f}$; Surrogate model $\hat{f}(\cdot;\theta)$ with initial parameters $\theta^{(0)}$; Initial RL policy $\pi^{(0)}$; Iteration number K; Per-instance Budget B

Initialize: $\mathcal{D} \leftarrow \mathcal{D}^{(0)}$, B, $\mathcal{B} \leftarrow \emptyset$;
for $k = 1, \ldots, K$ **do**
  Randomly sample an input state $s^{(k)}$;
  Select mesh points $\boldsymbol{a}^{(k)}$ using current policy $\pi^{(k-1)}$ under budget B;
  Query PDE solver: $(s^{(k)}, \boldsymbol{a}^{(k)}) \leftarrow \tilde{f}(s^{(k)}, \boldsymbol{a}^{(k)})$
  Update dataset: $\mathcal{D}^{(k)} \leftarrow \mathcal{D}_{k-1} \cup \{(s^{(k)}, \boldsymbol{a}^{(k)})\}$
  Update trajectory buffer: $\mathcal{B}^{(k)} \leftarrow \mathcal{B}_{k-1} \cup (s^{(k)}, a^{(k)}, r, s^{(k+1)})$
  Update RL policy $\pi^{(k)}$ with $\mathcal{B}^{(k)}$
  Retrain surrogate model: $\hat{f}(\cdot;\theta) \leftarrow \text{train}(\mathcal{D}^{(k)})$
Return: Final dataset $\mathcal{D}^{(K)}$, final surrogate parameters $\theta^{(K)}$

4.2 RL for Adaptive Mesh Grid Selection

Formulation: We formulate the adaptive mesh grid selection as a finite-horizon Markov Decision Process (MDP) in which an agent sequentially selects cells of the mesh grid under a fixed per-instance budget B. Each episode corresponds to one PDE instance and proceeds for B steps. At each step k, the environment is represented by a state $s^{(k)}$ that concatenates a binary mask indicating which cells have already been selected, and a normalized encoding of the PDE input.

The agent's action space at step k consists of all grid cells not yet selected. When the policy selects an action a_k, the corresponding cell is marked as selected and the state is updated to reflect the new mask and any contextual features derived from the partially chosen grid. This process continues until the budget B is exhausted, yielding a sequence of grid selections $(a_1, \ldots, a_B)$ for the given PDE instance.

Reward construction. Rather than providing step-wise feedback, the environment issues a *single terminal reward* once the full set of B mesh grid points has been chosen. This reward is designed to measure the expected benefit of the selected mesh for training a downstream surrogate solver $\hat{f}(\cdot;\theta)$. Because retraining $\hat{f}(\cdot;\theta)$ after each episode to obtain an exact reduction would be computationally prohibitive, we introduce a lightweight *proxy model* that predicts the surrogate's

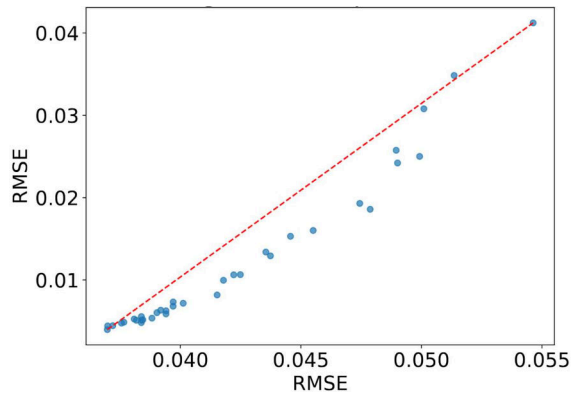


Figure 2: Correlation between proxy (RBF kernel ridge, x-axis) and FNO surrogate errors (y-axis) across subset sizes. Each point corresponds to a retrained model. Spearman correlation (0.9908) indicates strong rank consistency.

generalization error given a proposed mesh.

In practice, the proxy is implemented as kernel ridge regression with a closed-form solution

$$\hat{g}(s) = \sum_{j=1}^{N} \alpha_j \, k\big(s(\boldsymbol{x}), s^{(j)}(\boldsymbol{x})\big), \quad \boldsymbol{\alpha} = (\boldsymbol{K} + \lambda I)^{-1}\boldsymbol{u}, \tag{4}$$

where $\boldsymbol{K}$ is the kernel matrix evaluated on the training inputs $s^{(j)}$, and $\boldsymbol{u}$ denotes the vector of known function values. The proxy model allows fast updates and inference while maintaining a strong correlation with the true surrogate error. It is only used after an initial pretraining phase in which the FNO surrogate is trained on 100 fully observed instances, ensuring a stable warm-start regime for acquisition. In this regime, the RBF kernel ridge proxy exhibits near-perfect monotonic agreement with the FNO surrogate across subset sizes (Spearman $\rho = 0.9908$; see Fig. 2). This high rank correlation indicates that the proxy preserves the relative ordering of mesh selections in terms of downstream surrogate error, ensuring that reward signals remain well aligned with true generalization improvement. As a result, the RL agent can optimize placements using the proxy without frequent expensive retraining of the full surrogate. Additional details on proxy selection and evaluation are provided in Appendix A.3.

For each episode, we compute the proxy-predicted error of the surrogate before and after adding the instance with B newly selected mesh grid points, denoted as ϵ_{old} and ϵ_{new}. The terminal reward is then defined as

$$r = -\kappa(\epsilon_{\text{new}} - \epsilon_{\text{old}}), \tag{5}$$

with a scaling factor κ. It is then rescaled to the interval $[-1, 1]$ to stabilize the learning. Concentrating the reward at the end of the episode directly aligns the signal with the goal of improving surrogate accuracy, while avoiding hand-crafted local heuristics such as uncertainty sampling. Throughout the paper, the surrogate model refers to a Fourier Neural Operator trained to approximate the PDE solution operator, while the proxy model refers to a lightweight kernel ridge regression estimator used solely for reward computation during RL training.

Training. The policy is trained using a value-based RL algorithm, instantiated as a deep Q-network with experience replay and a target network under a greedy exploration schedule. Episodes thus generate trajectories of states and actions with a single terminal reward supplied by the proxy, enabling efficient policy optimization without retraining the surrogate at every step. To keep the proxy well aligned with the true surrogate error, the surrogate $\hat{f}(\cdot;\theta)$ is periodically retrained on the accumulated non-uniform data, amortizing its cost while ensuring that the proxy remains informative. The complete RLMesh is also detailed in Algorithm 2.

Algorithm 2: RLMesh Algorithm

Input: Initial FNO $\hat{f}(\cdot;\theta)$, initial proxy model $\hat{g}$, initial RL policy $\pi^{(0)}$, iteration number K, per-instance budget B, retraining interval L

Initialize: $\mathcal{D} \leftarrow \mathcal{D}^{(0)}$, trajectory buffer $\mathcal{B} \leftarrow \emptyset$.
for $k = 1, \ldots, K$ **do**
  Select input states $s^{(k)}$ randomly
  Select B mesh grid points $\boldsymbol{a}^{(k)} = (a_1^{(k)}, \ldots, a_B^{(k)})$ using current RL policy $\pi^{(k-1)}$
  Query solver to obtain solution on a non-uniform grid: $u(\boldsymbol{a}^{(k)}) \leftarrow \tilde{f}(s^{(k)}, \boldsymbol{a}^{(k)})$
  Update dataset: $\mathcal{D}^{(k)} \leftarrow \mathcal{D}^{(k-1)} \cup (s^{(k)}, u(\boldsymbol{a}^{(k)}))$
  Compute proxy-based terminal reward R_{batch} using $\hat{g}$.
  Update RL dataset: $\mathcal{B} \leftarrow \mathcal{B} \cup \big(s_b^{(k)},\, a_b^{(k)},\, 0,\, s_{b+1}^{(k)}\big), \forall b \le B-1,$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \big(s_B^{(k)},\, a_B^{(k)},\, R_{\text{batch}}\big)$
  Update RL policy $\pi^{(k)}$ (e.g. via deep Q-learning) using batch trajectories $\mathcal{B}$
  Retrain proxy model using current dataset: $\hat{g} \leftarrow \text{train}(\mathcal{D}^{(k)})$
  if $k \mod L = 0$ **then**
    Retrain FNO model using current dataset: $\hat{f}(\cdot;\theta) \leftarrow \text{train}(\mathcal{D}^{(k)})$

Return: Final dataset $\mathcal{D}^{(K)}$, trained RL policy $\pi^{(K)}$, trained FNO $\hat{f}(\cdot;\theta)$.

5 EXPERIMENTS

In this section, we show our empirical verification on three PDE families, 1D Burgers (terminal-time prediction), 2D Darcy (steady mapping), and Lorenz-96 system (chaotic lattice system).

5.1 Experimental Setup

For each dataset, we maintained 1,000 training instances and 200 held-out test instances. From the training set, 100 are reserved for surrogate pretraining and the 900 for active acquisition (18 iterations with 50 instances per iteration). Results are averaged over five independent runs with different seeds and constant train/test splits. Our primary metric is RMSE on a dense evaluation grid, and we report mean $\pm$ std across runs. We compare our RLMesh against standard heuristic baselines (uniform, random, gradient, variance, intensity), with all methods sharing the same budget, solver query count per iteration, and surrogate retraining cadence. Hyperparameters (networks, optimizers, replay/config for RL, proxy settings, and surrogate schedules) are fixed once chosen and summarized in Appendix A.4.

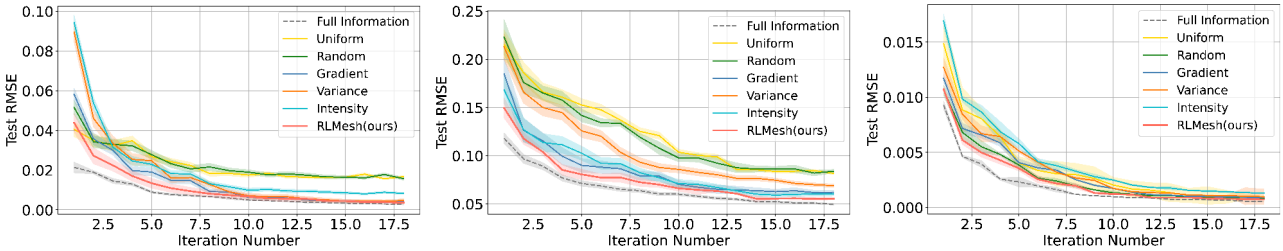
To ensure the feasibility of our pipeline, we employ a custom finite-volume solver designed to provide accurate solutions on non-uniform meshes. Specifically, we employ a custom finite-volume solver with an additional geometry augmentation step that inserts virtual midpoints to cap maximum gaps under sparse grid selections (Appendix A.5). This solver preserves good solution quality even under an irregular mesh grid, while maintaining a monotone cost–complexity relationship (i.e., solve time increases reasonably with the number of queried points). Such a property makes it well-suited for adaptive acquisition: the policy can flexibly select arbitrary grid points without sacrificing correctness or tractability. In practice, we interpolate as needed to guarantee accurate values at queried locations, ensuring that sample complexity is well-controlled and results are comparable across baselines. This differs from classical hierarchical adaptive mesh refinement, where refinements occur on structured strata; instead, our solver supports localized, irregular sampling that integrates naturally with the learning framework. To further strengthen the surrogate, we apply two modifications for training FNOs on non-uniform grids: (1) a masked training strategy, where gradient updates are computed only on selected mesh grid points; and (2) positional embeddings to improve the network's ability to process geometric information. For fairness, all baselines are adapted to ensure stable interpolation under sparse selections. Full implementation details are provided in Appendix A.1 and Appendix A.5.

5.2 Main Results

Before presenting the main results, we emphasize that all comparisons in this subsection are conducted using an *oracle uniform solver* rather than our non-uniform PDE solver. The rationale is that on highly irregular meshes, interpolation errors can dominate at queried points, making it difficult to disentangle algorithmic differences from solver artifacts. By employing the oracle solver, we ensure that all heuristics and our RL policy are evaluated under identical, noise-free conditions.

Figures 3a, 3b, and 3c present the active learning performance of RLMesh on Burgers, Darcy, and Lorenz–96, respectively, compared with several heuristic baselines. We also plot the full-information curve, which serves as an information-theoretic lower bound on test error. Across all three systems, the RL policy improves faster and attains lower error than heuristic baselines under identical per-instance budgets and retraining cadence, and consistently approaches the full-information curve. The largest gains appear in the early to mid acquisition stages, where the RL error decreases rapidly while heuristic methods plateau. For example, on Burgers, RLMesh reaches an RMSE of 0.02 by approximately iteration 6, whereas variance, intensity, and gradient-based heuristics require 9–10 iterations and uniform or random selection require 12 or more, corresponding to a 33% $\sim$ 50% reduction in total labeling cost. Shaded bands indicate that the results are stable: the variability of RLMesh is comparable to or smaller than that of the stochastic baselines across acquisition iterations. The same qualitative behavior is observed on Darcy and Lorenz–96, despite their very different dynamics, with Lorenz–96 representing a chaotic lattice system rather than a spatially inhomogeneous PDE. Overall, these results support the core claim that treating point selection as a sequential decision problem yields superior accuracy–vs–budget trade-offs compared to fixed heuristics, both for PDEs with spatial structure and for chaotic dynamical systems.

Fig. 4 shows how performance varies with different per-instance budget B. With only 20 or 40 grid points, the RL policy is unable to drive error down substantially—both curves plateau at relatively high RMSE. To understand where the limitation originates, we ran experiments with alternative heuristics and found it is primarily due to FNO capacity rather than the RL policy, as shown in Appendix B.1. In contrast, once the budget reaches 60 grid points, the loss drops quickly and ultimately converges to the same level as the 80- and 100-grid point settings. This indicates that while additional mesh points (80, 100) yield slightly faster early progress, the marginal returns diminish: all three even-



(a) Burgers, RMSE vs. Iterations (b) Darcy, RMSE vs. Iterations (c) Lorenz–96, RMSE vs. Iterations

Figure 3: Active learning performance across three benchmark systems. RLMesh consistently outperforms heuristic baselines under identical query budgets.

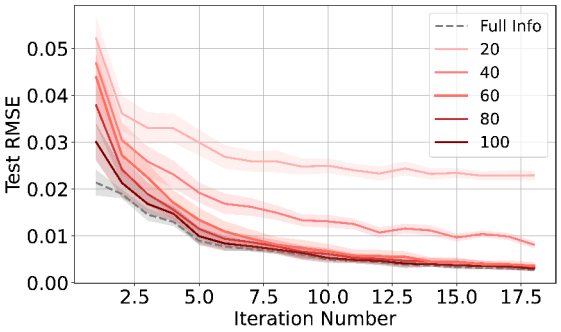


Figure 4: Burgers: effect of varying per-instance query budget B on active learning performance.

tually overlap near the full-information lower bound. The variance bands also become narrower with increasing budget, suggesting more stable learning. Overall, 60 appears to be the most cost-efficient setting, achieving the same accuracy as larger budgets while requiring lower labeling cost.

6 ANALYSIS

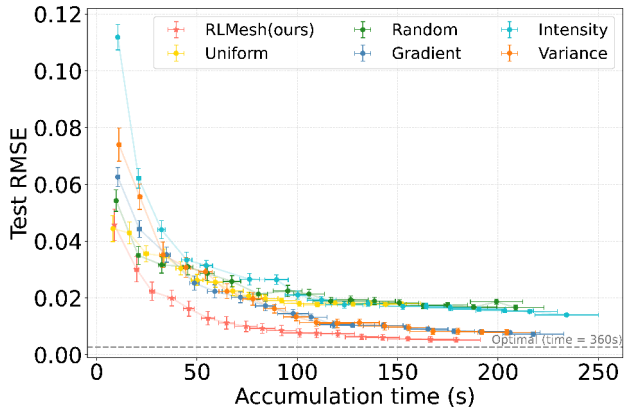
Time-error tradeoff. In contrast to Section 5.2, where we isolated algorithmic effects using an oracle solver, here we explicitly account for *simulation time*. Because solver runtime depends strongly on mesh regularity and numerical schemes, we report results using our custom finite-difference solver, which reflects the wall-clock cost of non-uniform selections. We also validate that our custom solver preserves these conclusions under non-uniform selections (Appendix B.2). To avoid ambiguity, we emphasize that each figure uses a single, shared solver across all methods: Fig. 3 evaluates all algorithms under the same oracle uniform-grid solver to isolate accuracy effects, while Fig. 5 evaluates all algorithms under the same non-uniform solver to assess time–error tradeoffs.

Fig. 5a presents the tradeoff between accumulation time and test error. Across acquisition iterations, RLMesh consistently reduces error more quickly than heuristic baselines, approaching the full-information lower bound in substantially less time. In particular, to reach the 0.02 error threshold, RLMesh requires only about 40 seconds of accumulated simulation, whereas variance, intensity, and gradient heuristics require roughly 80–120 seconds, and uniform or random baselines require well over 150 seconds. This gap highlights that RLMesh not only achieves superior predictive accuracy but also makes far more efficient use of the simulation budget, converting the same amount of wall-clock computation into significantly larger accuracy gains.

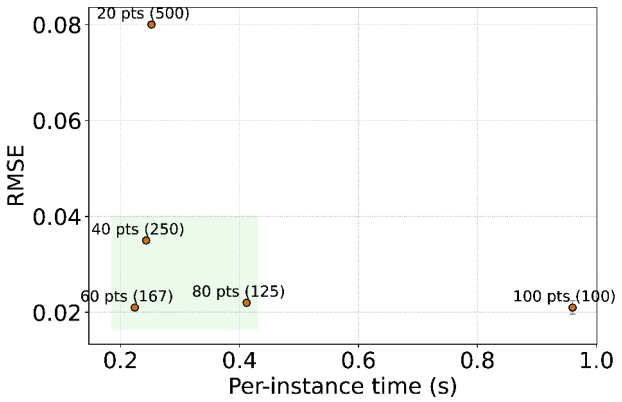
In Fig. 5b we fix the total label budget at 10,000 points and vary the per-instance grid point budget which also sets the number of instances (10,000/budget) The scatter reveals a clear Pareto region (lower error at lower per-instance time). Using 20 grid points is faster but underfits (higher error), whereas 100 grid points yields only modest error gains at substantially higher time per initial-condition. We therefore adopt 60 grid points as a practical operating regime for simulator-assisted acquisition.

Qualitative analysis. We next turn to analyzing the grid point selections learned by our policy. For Burgers in Fig. 6a, across the four instances the learned policy concentrates selection around steep fronts and turning points, while keeping scaffold in smooth ramp regions. For Darcy in Fig. 6b, grid points cluster along high-contrast channels and near boundary layers, with coverage in homogeneous areas. The patterns shift across instances, confirming per-instance adaptivity rather than a static mask, and showing that the learned policy adapts naturally to the characteristic features of each PDE family. We do not include a corresponding visualization for Lorenz–96, as it lacks a continuous spatial geometry and selected indices do not admit the same geometric interpretation as PDE mesh points.

The benefits of such adaptive selection are reflected in the learning curves. For Burgers in Fig. 7a, from pretrain to 100 samples the model quickly locks onto the

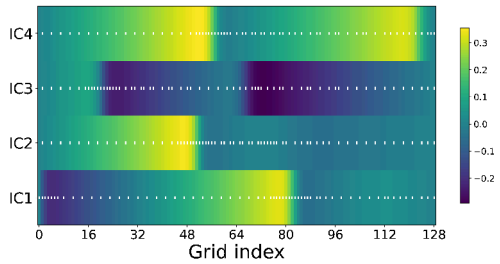


(a) Time–error illustration on Burgers under different selection strategies. Each dot corresponds to an acquisition iteration. The dashed grey line marks the optimal RMSE and solving time under full information. RLMesh consistently achieves lower error with shorter solving time compared to heuristic baselines.

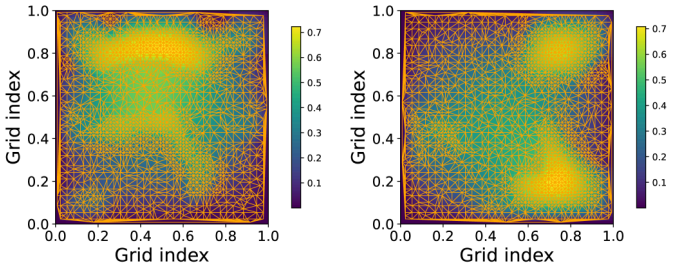


(b) Time–error tradeoff using RLMesh on Burgers under a fixed total budget. Each point corresponds to a different per-instance grid point budget, annotated as "number of grid points (number of instances)." The shaded green region indicates the preferable regime, balancing low error with shorter solve times.

Figure 5: Time–error analysis on Burgers. (a) Comparison across selection strategies. (b) Tradeoff across per-instance budgets using RLMesh.



(a) Burgers: grid point selection on four different ICs.

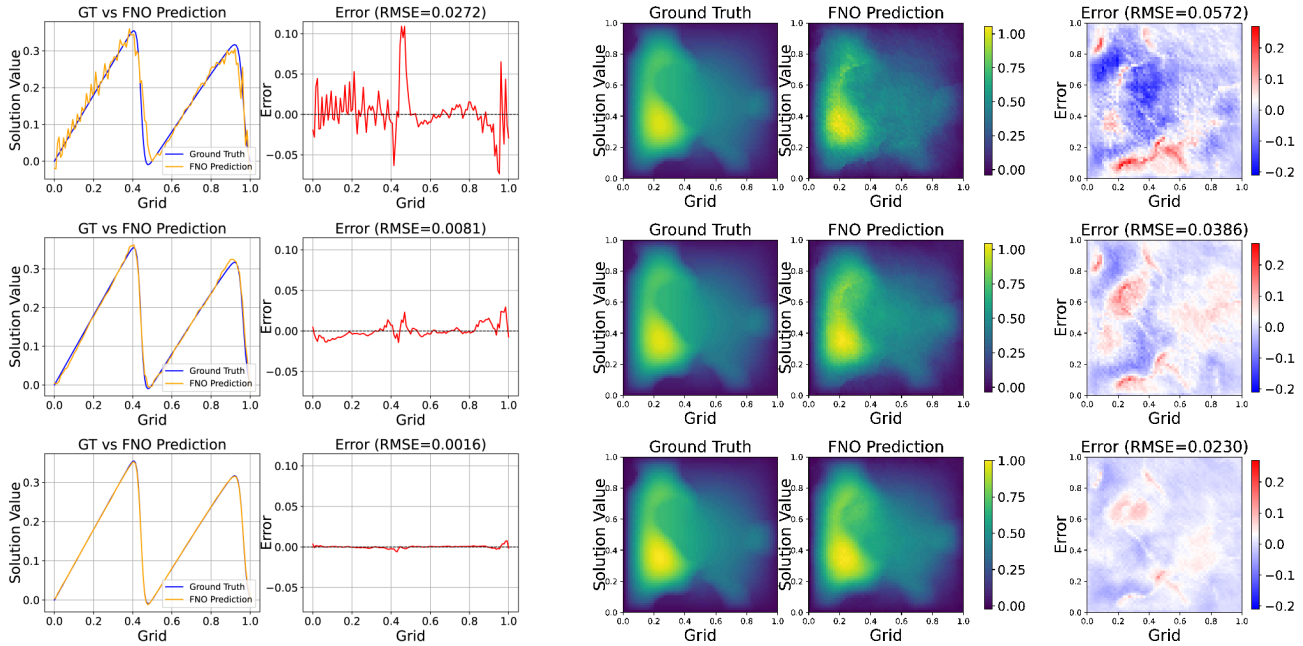


(b) Darcy: grid point selections on two representative input states.

Figure 6: Visualization of grid point selections by RLMesh on Burgers and Darcy. (a) Burgers: selections adapt to steep shock fronts and turning points while maintaining scaffolding in smoother regions. (b) Darcy: selections concentrate along high-contrast channels and near boundary layers, with adaptive coverage in homogeneous regions.

global waveform, eliminating large bias and smoothing the oscillatory error. The remaining error is confined to a narrow neighborhood around the shock/turning points. By 900 samples the curve is visually indistinguishable from ground truth except for tiny boundary artifacts. For Darcy in Fig. 7b, pretrain reconstructions capture coarse structure but miss high-contrast channels; with 100 samples the bulk geometry is recovered while error concentrates along sharp interfaces. By 900 samples the error ridges thin and weaken, especially around permeability contrasts and boundary layers, indicating that later acquisitions act as local refinements rather than global corrections. The spatial patterns of error reduction closely align with our selected mesh grid points, underscoring the informativeness of the chosen grid points.

Mesh-level vs. instance-level adaptivity. Note that RLMesh was designed for per-instance spatial point selection under a strict local query budget. Meanwhile, prior frameworks such as MRA-FNO [Li et al., 2024] and AL4PDE [Musekamp et al., 2025] focus on instance-level acquisition (choosing which simulations or global resolutions to run), assuming that each selected instance returns a full field solution. To contextualize RLMesh relative to such instance-level active learning methods, we adapt acquisition components from MRA-FNO and AL4PDE to our single-fidelity setting. Specifically, we implement a mutual-information–based selector (Self-MI, derived from MRA-FNO) and LCMD (the strongest AL4PDE heuristic for Burgers), while holding the remainder of the pipeline fixed. Since these baselines select entire



(a) Learning progress on Burgers: pretrained, 100 samples, and 900 samples. Each row shows the ground truth, predictions, and the error map.

(b) Learning progress on Darcy: pretrained, 100 samples, and 900 samples. Each row shows the ground truth, predictions, and the error map.

Figure 7: Learning progression under active acquisition. (a) Burgers: the model rapidly reduces bias and oscillatory error, with residual discrepancies confined to shock neighborhoods. (b) Darcy: coarse structure is captured early, while later acquisitions refine high-contrast channels and boundary layers.

Table 1: Burgers equation: RLMesh with instance-level baselines (mean ± std).

Iteration	RLMesh	Self-MI	LCMD
1	**0.0439 ± 0.0047**	0.0489 ± 0.0108	0.0464 ± 0.0169
6	**0.0109 ± 0.0021**	0.0146 ± 0.0014	0.0126 ± 0.0007
12	**0.00566 ± 0.00128**	0.00697 ± 0.00084	0.00618 ± 0.00041
18	**0.00364 ± 0.00067**	0.00418 ± 0.00038	0.00384 ± 0.00010

instances and return full-field solutions, we equalize total query budgets by assigning each selected instance a fixed 60-point uniform grid. Table 1 reports RMSE under identical total query budgets on Burgers. RLMesh consistently achieves lower error across acquisition iterations, indicating that instance-only selection cannot exploit fine-grained spatial structure that per-instance spatial adaptivity captures.

7 CONCLUSION

We proposed RLMesh, a co-evolving active learning framework that casts per-instance, budgeted mesh-point selection as a sequential decision problem. By coupling a non-uniform–capable neural operator surrogate with an RL policy and leveraging a lightweight proxy for aligned terminal rewards, our method enables practical online policy improvement under strict labeling budgets. Across Burgers (1D) and Darcy (2D), our method consistently outperforms heuristics in active learning performance. On Burgers, detailed experiments on error–versus–budget and time–error trade-offs further show that targeted acquisition achieves comparable or better surrogate accuracy at substantially lower cost. Future directions include extending the framework to space–time sensing and multi-fidelity costs, scaling to higher-dimensional or irregular geometries with geometry-aware operators, and exploring joint instance-plus-grid-points selection with theoretical guarantees on sample efficiency. Overall, our results demonstrate that reinforcement learning–driven adaptivity provides a principled and effective path toward cost-efficient PDE surrogate modeling.

Acknowledgment

This work was partially done when CL was at the University of Chicago. We gratefully acknowledge the

support of DOE 0J-60040-0023A, AFOSR FA9550-24-1-0327, NSF IIS 2313131, NSF IIS 2332475, the NSF-Simons AI-Institute for the Sky (SkAI) via grants NSF AST-2421845 and Simons Foundation MPS-AI-00010513, and the University of Chicago's Research Computing Center for their support of this work.

References

Marsha J. Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, 1989.

Marsha J. Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484–512, 1984.

Oussama Boussif, Yoshua Bengio, Loubna Benabbou, and Dan Assouline. Magnet: Mesh agnostic neural pde solver. *Advances in Neural Information Processing Systems*, 35:31972–31985, 2022.

Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

Johannes Martinus Burgers. A mathematical model illustrating the theory of turbulence. *Advances in applied mechanics*, 1:171–199, 1948.

Leszek Demkowicz. *Computing with hp-Adaptive Finite Elements: Volume 1. One and Two Dimensional Elliptic and Maxwell Problems*. Chapman and Hall/CRC, 2006.

Corbin Foucart, Aaron Charous, and Pierre FJ Lermusiaux. Deep reinforcement learning for adaptive mesh refinement. *Journal of Computational Physics*, 491:112381, 2023.

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Efficient token mixing for transformers via adaptive fourier neural operators. In *International Conference on Learning Representations*, 2021.

George M Homsy. Viscous fingering in porous media. *Annu. Rev. Fluid Mech*, 19(1):271–311, 1987.

Peiyan Hu, Rui Wang, Xiang Zheng, Tao Zhang, Haodong Feng, Ruiqi Feng, Long Wei, Yue Wang, Zhi-Ming Ma, and Tailin Wu. Wavelet diffusion neural operator. *arXiv preprint arXiv:2412.04833*, 2024.

Ruoxi Jiang, Xiao Zhang, Karan Jakhar, Peter Y Lu, Pedram Hassanzadeh, Michael Maire, and Rebecca Willett. Hierarchical implicit neural emulators. *arXiv preprint arXiv:2506.04528*, 2025.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Ching-Yao Lai, Pedram Hassanzadeh, Aditi Sheshadri, Maike Sonnewald, Raffaele Ferrari, and Venkatramani Balaji. Machine learning for climate physics and simulations. *Annual Review of Condensed Matter Physics*, 16, 2024.

Shibo Li, Xin Yu, Wei Xing, Robert Kirby, Akil Narayan, and Shandian Zhe. Multi-resolution active learning of fourier neural operators. In *International Conference on Artificial Intelligence and Statistics*, pages 2440–2448. PMLR, 2024.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:9702–9713, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020b.

Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023.

Phillip Lippe, Bas Veeling, Paris Perdikaris, Richard Turner, and Johannes Brandstetter. Pde-refiner: Achieving accurate long rollouts with neural pde solvers. *Advances in Neural Information Processing Systems*, 36:67398–67433, 2023.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

Owen Melia, Daniel Fortunato, Jeremy Hoskins, and Rebecca Willett. Hardware acceleration for hps algorithms in two and three dimensions. *arXiv preprint arXiv:2503.17535*, 2025.

Daniel Musekamp, Marimuthu Kalimuthu, David Holzmüller, Makoto Takamoto, and Mathias Niepert. Active learning for neural PDE solvers. In *International Conference on Learning Representations*, 2025.

James Rowbottom, Georg Maierhofer, Teo Deveney, Eike Mueller, Alberto Paganini, Katharina Schratz, Pietro Lio, Carola-Bibiane Schönlieb, and Chris Budd. G-adaptivity: optimised graph-based mesh relocation for finite element methods. *arXiv preprint arXiv:2407.04516*, 2024.

Anupam Sarkar, Luigi Cipriani, Avraham Be'er, Liang Chen, Dipto Debnath, Amin Dehkharghani, Lior Farkash, Lukas Fischer, Anupam Ghosh, Els Heinsalu, Hartmut Löwen, Andreas M. Menzel, Anton Morozov, Shayak Mukherjee, Orit Peleg, Primandhika Putri, Subham Sarkar, Ralf Seemann, Ananyo Sengupta, Rastko Sknepnek, Holger Stark, Bartosz ten Hagen, Julia M. Yeomans, and Giorgio Volpe. Swarmrl: Building the future of smart active systems. *arXiv preprint arXiv:2404.16388*, 2024.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

Sunbochen Tang, Themistoklis Sapsis, and Navid Azizan. Learning chaotic dynamics with embedded dissipativity. *arXiv preprint arXiv:2410.00976*, 2024.

Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

Dongxia Wu, Ruijia Niu, Matteo Chinazzi, Yian Ma, and Rose Yu. Disentangled multi-fidelity deep bayesian active learning. In *International Conference on Machine Learning*, pages 37624–37634. PMLR, 2023.

Jiachen Yang, Tarik Dzanic, Brenden Petersen, Jun Kudo, Ketan Mittal, Vladimir Tomov, Jean-Sylvain Camier, Tuo Zhao, Hongyuan Zha, Tzanio Kolev, et al. Reinforcement learning for adaptive mesh refinement. In *International conference on artificial intelligence and statistics*, pages 5997–6014. PMLR, 2023.

Rose Yu and Rui Wang. Learning dynamical systems from data: An introduction to physics-guided deep learning. *Proc. Natl. Acad. Sci.*, 121(27): e2311808121, 2024.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes, provided in problem formulation section.]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [No, we do not provide a formal complexity analysis. We include only empirical results on time and sample size.]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [No, not released at submission time, may release after acceptance.]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [No, results reproducibility is described through training details in appendix, but code is not released at submission time.]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes, main data splits and hyperparameters are included in appendix.]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes, evaluation metrics defined and error bars plotted.]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes, GPU and cluster are included in appendix.]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes, PDEBench dataset is under CC-BY-4.0.]
   (c) New assets either in the supplemental material or as a URL, if applicable. [No]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Accelerating PDE Surrogates via RL-Guided Mesh Optimization: Supplementary Materials

A DETAILS ON EXPERIMENTAL SETUP

A.1 Fourier Neural Operator

In this section, we describe the Fourier Neural Operator (FNO) architecture used as our surrogate model, detailing the architectural modifications made to handle non-uniform grids and the training procedures employed in both pretraining and active learning phases.

A.1.1 Modification

The standard FNO operates efficiently on uniform grids by learning global convolutions in the frequency domain. However, our framework requires flexible modeling on non-uniform grid points placements. To this end, we propose architectural modifications and extensions to enable robust learning from non-uniformly distributed grid points.

- **Fourier feature encoding:** We augment raw spatial coordinates with a learnable Fourier feature embedding. This encoding transforms each coordinate into a higher-dimensional space using sinusoidal functions with Gaussian-sampled frequencies, enriching positional information for better generalization on irregular layouts.
- **Batch normalization on Fourier features:** To stabilize training and improve convergence, batch normalization is applied after the Fourier feature encoding. This normalization operates across feature channels and helps regularize the model when processing diverse input patterns.
- **Increased number of Fourier modes:** We increase the number of Fourier modes from typical values (e.g., 16) to 49 to capture finer spectral details relevant to the 129-point grid resolution, enhancing the model's capacity to learn complex solution structures.
- **Multi-head self-attention with residual connection:** A multi-head self-attention layer is integrated into the spectral convolution stack to further model long-range dependencies across locations. Combined with residual connections, this addition strengthens global feature interactions, which are critical for capturing the complex spatial correlations present in non-uniform mesh grid points data.

Together, these modifications enable the surrogate $\hat{f}(\cdot;\theta)$ to learn PDE solution operators from irregular inputs, improving predictive robustness in the active learning loop.

A.1.2 Training Details

We pretrained an ensemble of five FNOs on a subset of 100 samples from the chosen-fidelity (60-point) training data for stability and robustness. Each FNO was configured with 49 Fourier modes and a width of 64 channels, incorporating our proposed modifications including Fourier feature encoding with batch normalization and a multi-head self-attention layer with residual connections. Models were trained with a negative log-likelihood (NLL) loss over 250 epochs (batch size 32) using Adam (learning rate 1×10^{-3}). The NLL objective allows each model to capture predictive uncertainty by learning both the mean and variance; at inference we aggregate ensemble predictions by averaging means and, when needed, combining variances.

In the active learning phase, grid-point placements are iteratively selected by a reinforcement learning policy, with batches of 50 newly selected samples per iteration across 18 iterations (total 900 samples). After each batch, all five FNOs are retrained on the union of the pretraining set and the newly collected samples using the same

hyperparameters. This retraining strategy lets the surrogate ensemble progressively refine accuracy in regions deemed most informative by the RL policy. Concurrently, the RL agent's policy network is updated from rollout data, improving grid-point placement over time to minimize reconstruction error.

In both phases, we train the FNO surrogate on non-uniform samples uses a mask-based set-to-grid interface. Sparse observations queried at non-uniform mesh points are mapped to the nearest grid locations; a binary mask records which grid points are observed. During training, the FNO predicts the full field on the standard grid, but the loss is computed only at masked locations. This preserves the standard FNO architecture and spectral convolutions, while enabling learning from irregularly sampled data.

A.2 Reinforcement Learning Agent

In this section, we present the RL component of our framework, detailing the Deep Q-Network (DQN) architecture, the offline pretraining procedure using oracle demonstrations, and the online training process through environment interaction and solver feedback.

A.2.1 Deep Q-Network

We implement a DQN agent, which uses a neural network function approximator to estimate the Q-values for each state-action pair. The network architecture includes two hidden layers with ReLU activations and a final Tanh layer bounding Q-values. To stabilize training, we employ an experience replay buffer that stores transitions, enabling off-policy learning through mini-batch updates, and maintain a separate target network updated periodically to reduce oscillations and divergence. Exploration is governed via an epsilon-greedy policy, with epsilon annealed from an initial value to a minimum threshold to balance exploration and exploitation. The DQN agent learns from both intermediate discounted rewards and terminal rewards, updating its Q-values through gradient descent minimizing temporal difference errors.

A.2.2 Offline Pretraining

To initialize the reinforcement learning agent effectively, we perform offline pretraining using expert demonstrations derived from a heuristic policy informed by domain knowledge of the PDE. This expert policy leverages oracle information about the physical characteristics of sharp fronts and informative points, ensuring selection of grid points with high utility. The oracle follows a simple rule that combines local variation and spatial coverage: it prioritizes regions with strong gradients or curvature, while ensuring that the selected mesh grid points are spread evenly across the domain. The same principle is applied consistently to all problems. We collect transitions, specifically state-action pairs, over 100 training instances using this policy. The transitions are stored as tuples of the environment state and the chosen action, capturing the expert's behavior without requiring rewards or next states. Subsequently, imitation learning is applied to pretrain the RL agent by treating this offline dataset as supervised learning examples, where the agent's Q-function approximator is trained to predict expert actions given states. For the DQN agent, this involves training the neural network policy to minimize cross-entropy loss between predicted action logits and expert actions. This approach accelerates convergence during online learning by providing a strong initial policy derived from domain expertise. The imitation dataset and pretraining process are logged extensively to ensure reproducibility and monitor training progress.

A.2.3 Online Training

During the online phase, the RL agent interacts with the environment to sequentially select grid points within each episode. After an episode concludes, the selected grid points define a nonuniform mesh that is passed to the numerical solver to obtain the PDE solution restricted to those chosen locations. This newly generated data is then used to retrain the FNO surrogate model. The terminal reward for the RL agent is computed based on the improvement in the proxy model's predictive performance after retraining (the exact reward formulation is described in the next section). We train the DQN with Adam (learning rate 1×10^{-4}, weight decay 1×10^{-4}), discount $\gamma = 0.99$, a replay buffer of 10,000, batch size 64, ϵ-greedy exploration decaying from 1.0 to 0.1 at 0.995 per step, target-network synchronization every 100 updates, and gradient clipping at 1.0. Meanwhile, the RL policy network is updated using Monte Carlo backups of discounted rewards, allowing the agent to gradually learn more effective and informative mesh grid points placement strategies over time.

A.3 Proxy Model

In this section, we motivate and select a lightweight proxy to stand in for the FNO, describe how the proxy is trained and updated online, and formalize the terminal-reward computation used by the RL agent.

A.3.1 Alignment between FNO and Proxy Models

Intuitively, using the surrogate's downstream error as the RL signal aligns the agent's objective with the end task—accurate PDE prediction—so each placement is rewarded only insofar as it improves real generalization. However, evaluating an FNO ensemble per episode is prohibitively slow, so we introduce a lightweight proxy to stand in for the FNO when computing terminal rewards. Figure 8 compares candidate proxies trained across varying subset sizes and assesses their agreement with the FNO: panel (a) shows RMSE trends as data grow; panel (b) plots proxy versus FNO errors with monotonicity (Spearman) scores. Kernel Ridge Regression (RBF) exhibits the strongest and most stable alignment with the FNO (near-perfect monotonic correlation) while remaining orders of magnitude faster than retraining the ensemble. We therefore adopt Kernel Ridge as the proxy for reward computation: after each episode's solver call, we append the new sample, retrain the proxy in milliseconds, and compute the terminal reward from the proxy's test-RMSE improvement (exact reward form in the next section).

A.3.2 Training Details

To accelerate reward calculation, we trained the kernel ridge regression proxy on an initial subset of 100 labeled training instances sampled from the full 129-point fidelity data. The kernel ridge model uses a Gaussian radial basis function kernel with parameters $\alpha = 0.1$ and $\gamma = 1.0$ learned via cross-validation. The model is retrained online at each active learning episode, incorporating newly collected samples. We resample the non-uniform grid point observations onto a uniform grid via linear interpolation, since the kernel-ridge proxy assumes fixed-length inputs on a uniform lattice. This retraining typically completes in less than 0.8 seconds on an AWS EC2 g6e.xlarge instance equipped with an NVIDIA L40S GPU, enabling rapid updates to the proxy's predictive accuracy.

A.3.3 Reward Computation for RL

The RL reward r is computed solely from the terminal improvement in ϕ_γ after retraining. Specifically, the raw reward for an episode is

$$r_{\text{raw}} = -\kappa\big(\varepsilon_{\text{new}} - \varepsilon_{\text{old}}\big),$$

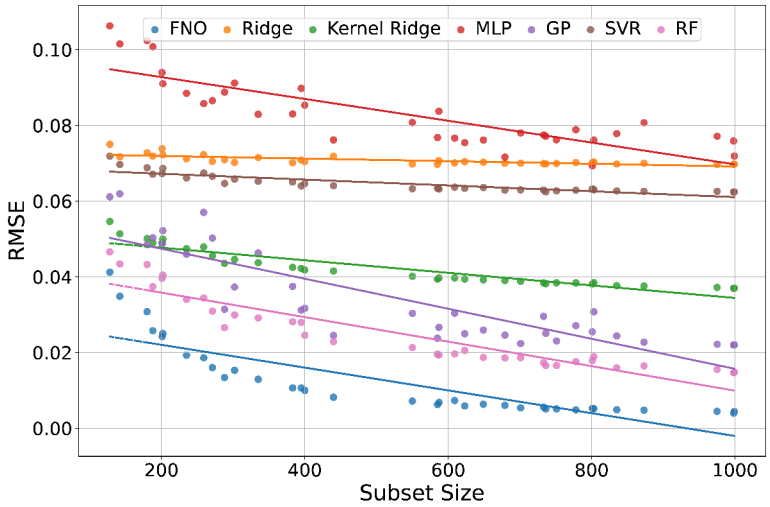
where ε_{old} and ε_{new} are proxy RMSE values on a held-out test set before and after adding the newly acquired samples. κ is a scaling constant and was set to $\kappa = 10^4$ for all experiments considered in this paper. To stabilize learning, we apply a piecewise scaling function to map r_{raw} into $[-1, 1]$:

$$\text{scale}(r_{\text{raw}}) = \begin{cases} 0.8\,\text{sgn}(r_{\text{raw}}), & |r_{\text{raw}}| < 0.01, \\ \left(0.8 + 0.2\,\frac{|r_{\text{raw}}|}{0.1}\right)\text{sgn}(x), & 0.01 \le |r_{\text{raw}}| < 0.1, \\ r_{\text{raw}}, & 0.1 \le |r_{\text{raw}}| < 1, \\ \left(1.0 - 0.01\,\frac{|r_{\text{raw}}|-1}{9}\right)\text{sgn}(r_{\text{raw}}), & 1 \le |r_{\text{raw}}| < 10, \\ \text{sgn}(r_{\text{raw}})\min\left\{1.0,\ 0.99 + 0.01\ln\frac{|r_{\text{raw}}|}{10}\right\}, & |r_{\text{raw}}| \ge 10. \end{cases}$$

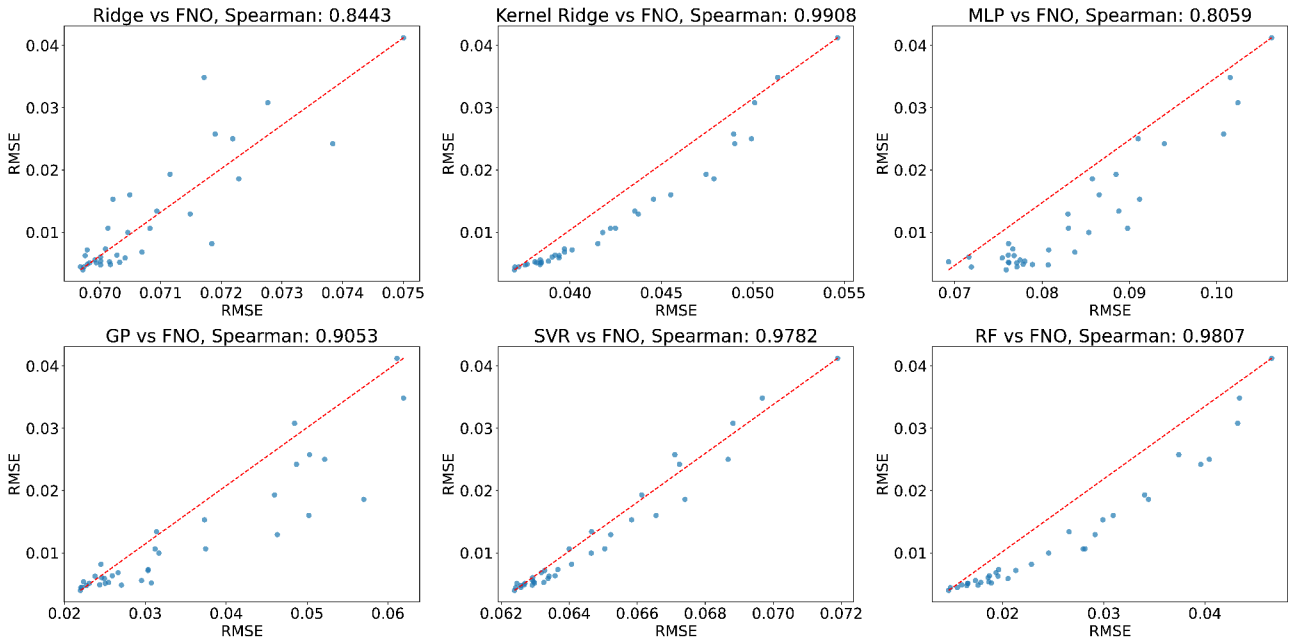
No rewards are provided at intermediate steps; the entire episode's credit is assigned via this terminal proxy-improvement signal. We avoid intermediate heuristics because we find no reliable per-step surrogate that aligns with downstream generalization, and such signals can introduce bias toward myopic or miscalibrated placements.

A.4 Experiment Details

We evaluate on three PDE families—1D Burgers (terminal-time prediction), 2D Darcy (steady mapping), and Lorenz-96 system (chaotic spatiotemporal dynamics)—with 1,000 training and 200 held-out test instances per problem. Of the 1,000 training instances, 100 are used for surrogate pretraining and the remainder for active acquisition. We sweep per-instance budgets and report test RMSE on a fixed dense grid (129 points for Burgers; 128×128 for Darcy) as a function of labeled samples and wall-clock cost. Unless noted, all hyperparameters (RL



(a) Model correlation



(b) Model RMSE

Figure 8: Model choice assessment for the proxy. (a) Correlation between the proxy and FNO across training subsets. (b) RMSE for candidate models across subset sizes.

agent, proxy, surrogate training schedule) are fixed across runs, and all methods (RL and baselines: uniform, random, gradient, variance, intensity) share the same sbudget, number of solver queries per iteration, and surrogate retraining cadence for fairness. Results are averaged over five independent seeds (mean ± std) with a fixed train/test split per seed. Experiments were run on Amazon EC2 instances and an HPC cluster with identical code, data, and hyperparameters using L40S GPU. We get our data from the PDEBench dataset under CC-BY-4.0 [Takamoto et al., 2022]. We additionally evaluate RLMesh on the Lorenz–96 system, a 1D lattice dynamical model with $N = 60$ state variables and periodic coupling. Each instance is generated by integrating the system from $t = 0$ to $t = 1$ with step size $\Delta t = 0.01$, and forming a supervised pair $(x(0), x(1))$. We impose a strict per-instance budget of 15 queried coordinates (25% of the state). Results are averaged over five independent seeds, and all baselines share the same query budget and acquisition protocol.

A.5 PDE Simulator

For Burgers we use a custom finite-volume solver with Dirichlet walls at the domain endpoints, integrated in time with SciPy's odeint (LSODA) under tight tolerances. Spatially, we employ MUSCL (Van Leer) reconstruction and a blended convective flux that interpolates between Godunov (sharper, less diffusive) and Rusanov (more robust); viscosity is handled via a face-based diffusive flux. The solver supports uniform and non-uniform grids: on non-uniform meshes we reconstruct left/right face states using node-to-face distances and compute face spacings explicitly in both convection and diffusion; boundary values are pinned only if boundary nodes are present, so interior-only sets are respected. To stabilize pathological layouts from sparse mesh grid points, we add a geometry augmentation step that inserts virtual midpoints to cap the maximum gap (relative to the nominal spacing) and the adjacent-gap ratio, with optional wall anchoring; the initial condition is re-interpolated onto the augmented grid before integration. We use a simple CFL-based heuristic for the integrator's initial step and provide an option to restrict outputs back to the original set for consistent evaluation. Consequently, runtime is not monotone in the number of mesh grid points: very sparse, irregular 20-point layouts can trigger more geometry augmentation and force smaller stable time steps than a better-distributed 40-point layout, so the 20-point solve can actually take longer (see Fig. 5b). Likewise, the more-point case can benefit from vectorized operations and larger stable steps.

We quantitatively validate our non-uniform finite-volume solver against dense uniform-grid oracle solutions, obtaining MAE $7.42 \times 10^{-3} \pm 2.91 \times 10^{-3}$ and RMSE 1.60×10^{-2} under the sparse regimes used by RLMesh. While developing a production-grade, fully verified adaptive solver integrated with industrial PDE frameworks is beyond the scope of this work, these checks and the use of a common oracle across all comparisons ensure that solver accuracy is sufficient and does not affect soundness or fairness.

Extending the time–error tradeoff analysis to 2D Darcy is an interesting direction for future work. However, constructing a robust non-uniform 2D Darcy solver involves substantial numerical complexity, including geometry-aware meshing, stability constraints, and topology-dependent stencil construction, which are typically handled by specialized AMR frameworks. Developing such solvers is beyond the scope of this proof-of-concept study, which focuses on demonstrating the feasibility of the proposed approach.

B ADDITIONAL EXPERIMENTAL RESULTS

B.1 Fewer Grid Points Performance

In Fig. 9, all baselines degrade under the smaller 20-point budget: their curves fall more slowly and plateau higher than with larger budgets. RLMesh (ours) still outperforms the heuristics across iterations, but the gap to the full-information oracle widens (same for all other baselines), indicating that limited sensing amplifies the difficulty. This pattern is consistent with a capacity/conditioning bottleneck in the surrogate (FNO) under sparse measurements rather than a failure of the placement policy.

B.2 Simulator Performance

Fig. 10 shows that with the simulator enabled, the results provide the same insight as before: RLMesh remains the best non-oracle method across iterations.

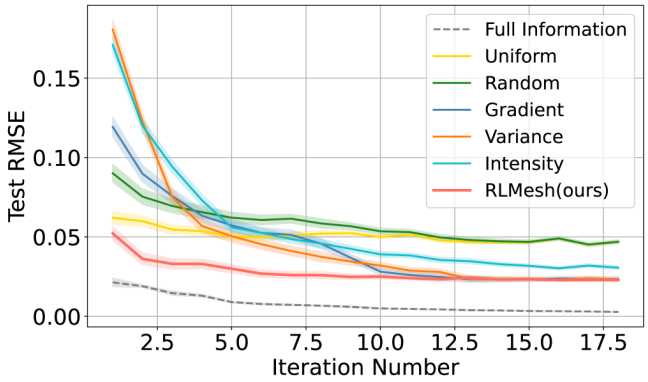


Figure 9: Active learning performance on Burgers with 20 per-instance budget

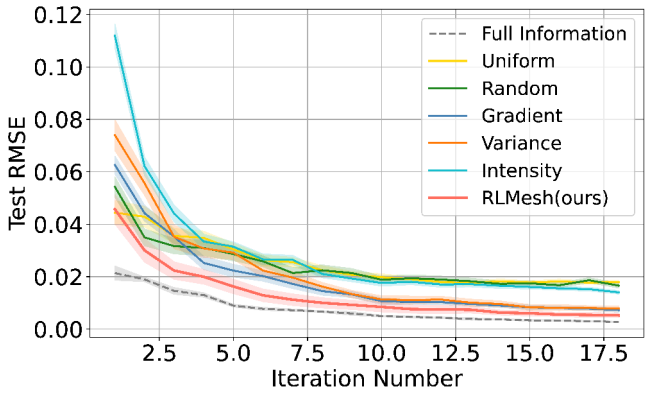


Figure 10: Active learning performance on Burgers with PDE simulator. RLMesh consistently outperforms baselines under identical budgets.

B.3 Mesh Grid Illustrations

Fig. 11 is a joint visualization designed to reveal the relationship between our selected grid points and model errors. It shows four Burgers instances (rows) across four views (columns): the initial condition, the ground-truth solution, the FNO prediction, and the error (prediction minus ground truth). The error panel uses a symmetric blue–white–red scale, with blue indicating under-prediction and red indicating over-prediction. The horizontal axis in every panel is the normalized spatial grid. Vertical orange tick marks denote the chosen mesh points; the same set is overlaid on all four views for each row to show how measurements align with field features. Per-instance RMSE values are listed on the right side of the error column. Overall, the placements concentrate near steep transitions and high-error bands, indicating that our selection strategy effectively targets the regions where the model struggles (the same pattern appears in the Darcy counterpart).

Fig. 12 is a joint visualization built to show how our selected grid points relate to model errors. Each Darcy instance appears twice—first without the overlay and then with the same overlay—to compare the fields and the sampling pattern. Within each row, the four panels (left to right) show the initial condition, the ground-truth solution, the FNO prediction, and the error (prediction minus ground truth) rendered with a symmetric blue–white–red scale; per-instance RMSE is annotated on the error panel. The first three panels share a common color range so magnitudes are directly comparable. In the "with mesh overlay" rows, an orange triangulated overlay marks the chosen grid points and their connectivity across all three left panels. Across instances, these placements align with steep transitions and high-error bands, indicating that the selection strategy effectively targets the regions where the model is most challenged.

Fig. 13 tracks the active-learning trajectory on Burgers using a fixed test instance: each subfigure overlays ground truth and the model prediction (left) and plots the pointwise error (right) with the RMSE noted in the title. Starting from the pretrained model the error is relatively large and oscillatory (RMSE ≈ 0.027), but after selecting only 100 samples the mismatch drops substantially (≈ 0.008). Subsequent rounds continue to refine the

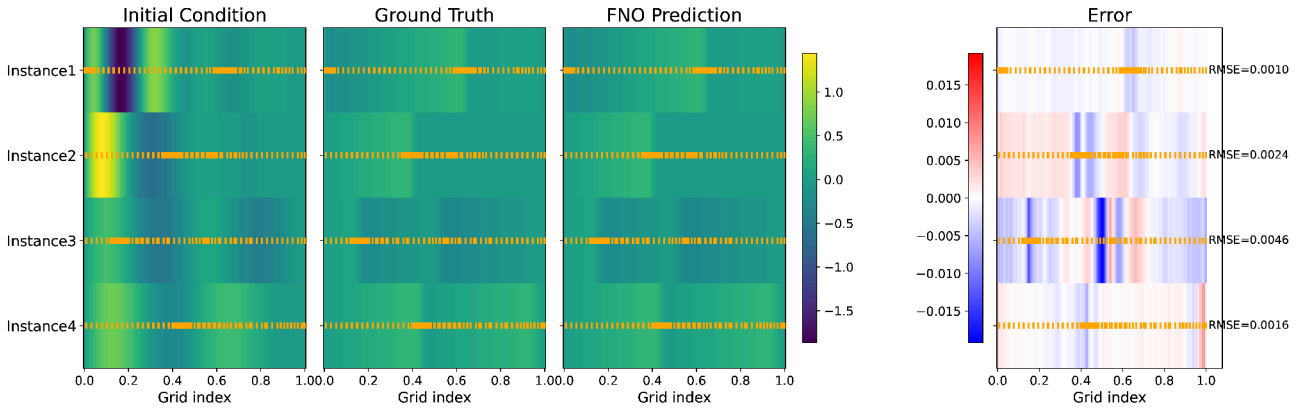


Figure 11: Burgers prediction with and without mesh overlay

solution—about 300 samples (≈ 0.0035), 500 samples (≈ 0.0025), 700 samples (≈ 0.0018), and 900 samples (≈ 0.0016)—with the error curve flattening toward zero across most of the domain and remaining concentrated near steep transitions. The sequence illustrates fast early gains from informative sampling followed by diminishing returns, while consistently preserving the overall shape of the solution.

Fig. 14 illustrates active learning on a Darcy test case over six training stages: pretrained, 100, 300, 500, 700, and 900 samples. Each subfigure contains three panels—ground truth on the left, model prediction in the middle (both on a shared color scale), and a signed error map on the right using a symmetric blue-white-red scale. As sampling increases, the prediction progressively matches the ground truth, and the error maps fade from broad, structured errors to mostly low-magnitude residuals. The reported RMSE drops steadily from about 0.057 at initialization to about 0.023 by 900 samples, with the remaining errors concentrated near sharp features and boundary-aligned regions. Overall, the sequence shows rapid early gains from informative data and smaller, consistent improvements at later stages.

C PDE BENCHMARK FORMULATIONS

We provide the precise mathematical formulations of the three benchmark systems used in our experiments: 1D Burgers, 2D Darcy flow, and the Lorenz–96 dynamical system.

C.1 Burgers' Equation

We consider the one-dimensional viscous Burgers equation

$$u_t + uu_x = \nu u_{xx}, \quad (x,t) \in [0,1] \times [0,1], \tag{6}$$

with periodic boundary conditions and viscosity $\nu = 0.002$. The learning objective is to approximate the solution operator

$$u_0(x) \longmapsto u(x,1),$$

i.e., predicting the terminal-time solution from the initial condition. The initial condition is parameterized as

$$u_0(x) = ae^{-ax}\sin(2\pi x)\cos(b\pi x), \tag{7}$$

where $a, b \sim \mathcal{U}[1,6]$ are sampled independently.

C.2 Darcy Flow

We consider the two-dimensional steady-state Darcy flow equation

$$-\nabla \cdot (c(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x}), \quad \mathbf{x} \in [0,1]^2, \tag{8}$$

with homogeneous Dirichlet boundary condition $u(\mathbf{x}) = 0$ on $\partial[0,1]^2$. The forcing term is fixed to $f(\mathbf{x}) = 1$, and the diffusion coefficient $c(\mathbf{x}) > 0$ is treated as the input function. The learning objective is to approximate the mapping

$$c(\mathbf{x}) \longmapsto u(\mathbf{x}),$$

i.e., from coefficient field to steady-state solution. Following Li et al. [2020a], coefficient fields are generated by sampling a Gaussian random field and thresholding to obtain a binary permeability field taking values in $\{4, 12\}$.

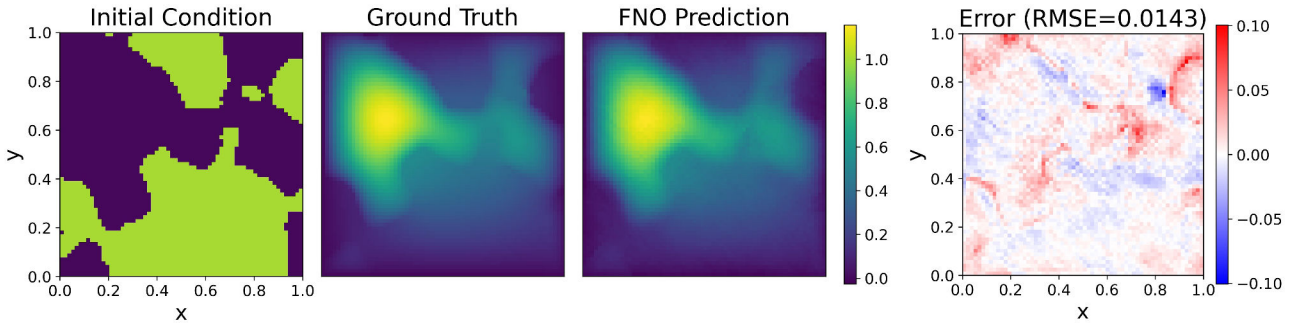
C.3 Lorenz–96 System

We consider the Lorenz–96 dynamical system on a one-dimensional periodic lattice of size $N = 60$:

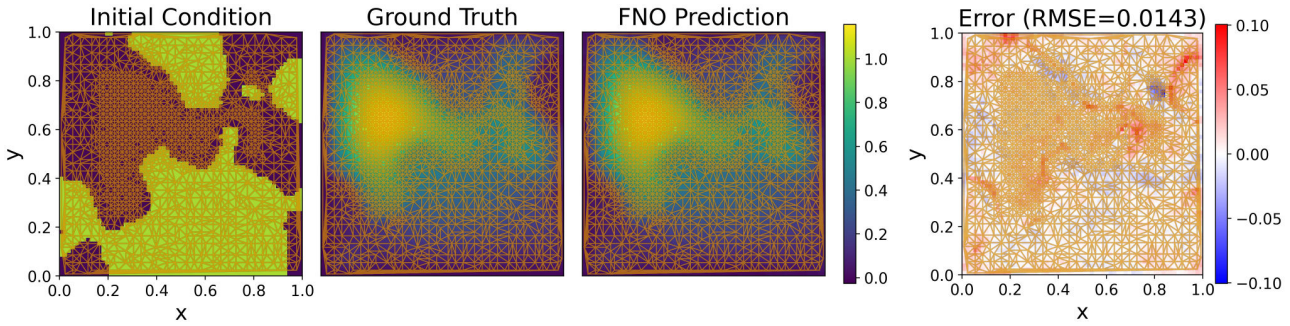
$$\frac{dx_i}{dt} = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F, \quad i = 1, \ldots, N, \tag{9}$$

with periodic boundary conditions $x_{i+N} = x_i$. The forcing parameter is fixed to $F = 4.0$. The system is integrated over time horizon $T = 1.0$ with time step $\Delta t = 0.01$. For each instance, we retain only the initial and terminal states $\mathbf{x}(0)$ and $\mathbf{x}(1)$, and the learning objective is to approximate the mapping

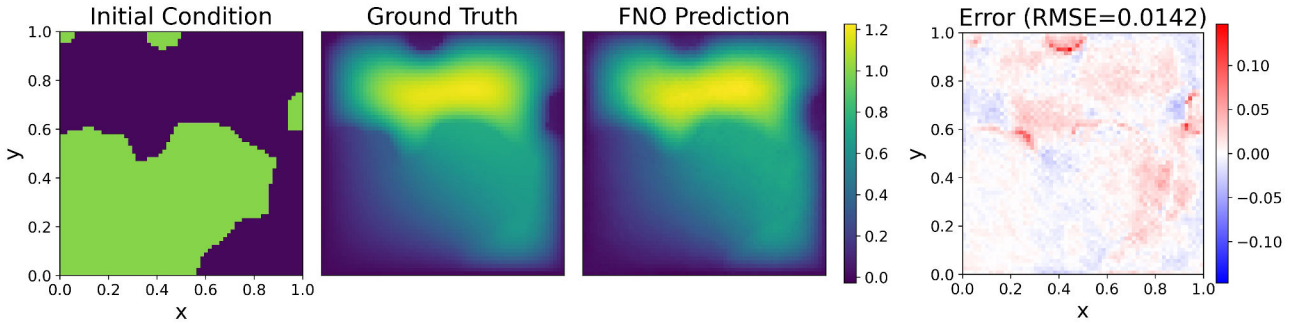
$$\mathbf{x}(0) \longmapsto \mathbf{x}(1).$$



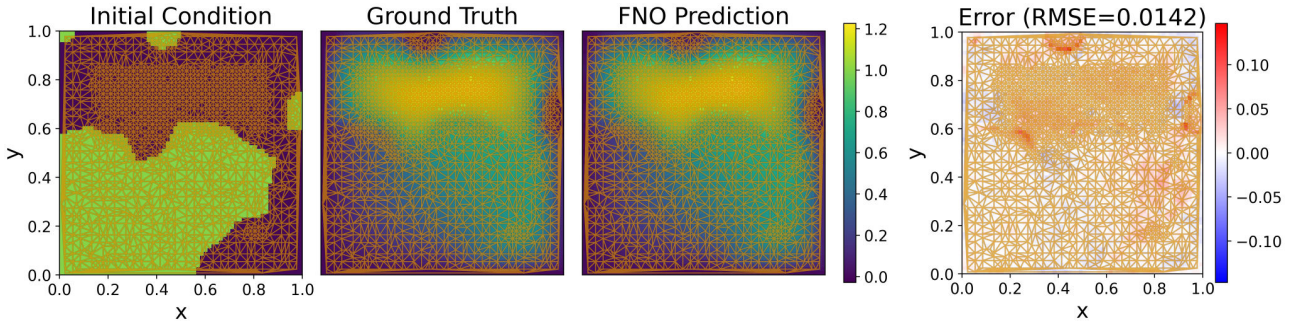
(a) Instance 1 — no mesh



(b) Instance 1 — with mesh overlay

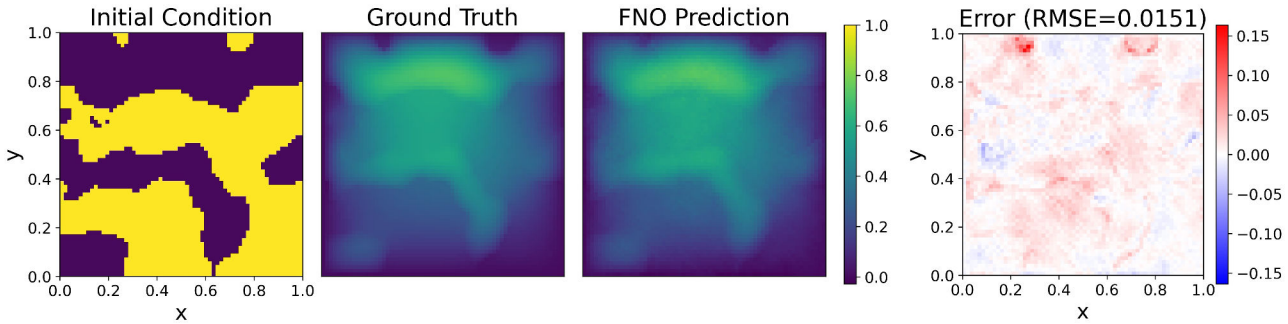


(c) Instance 2 — no mesh

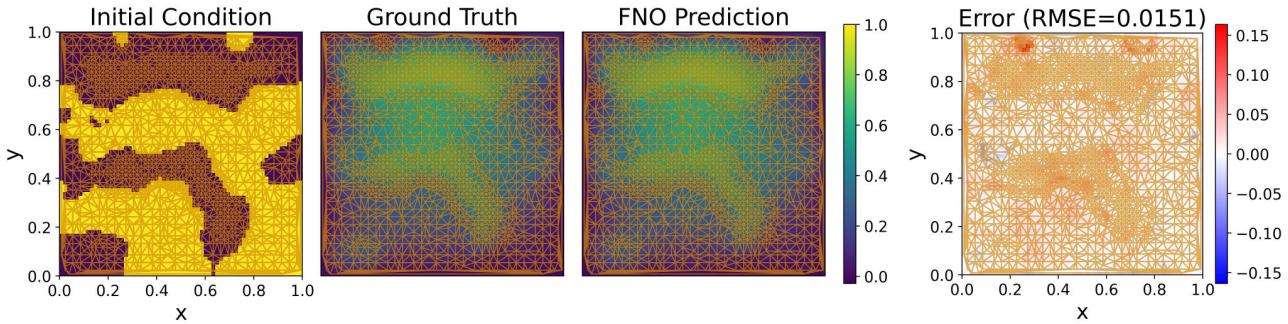


(d) Instance 2 — with mesh overlay

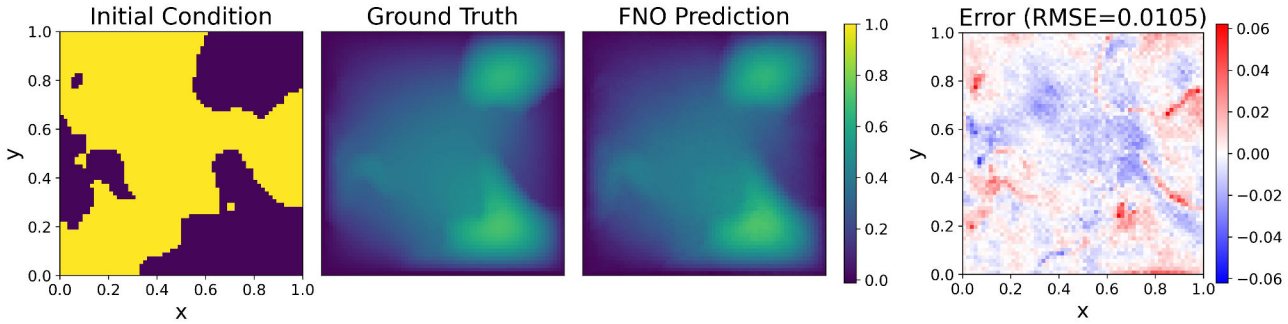
Figure 12: Darcy prediction with and without mesh overlay (instances 1–2).



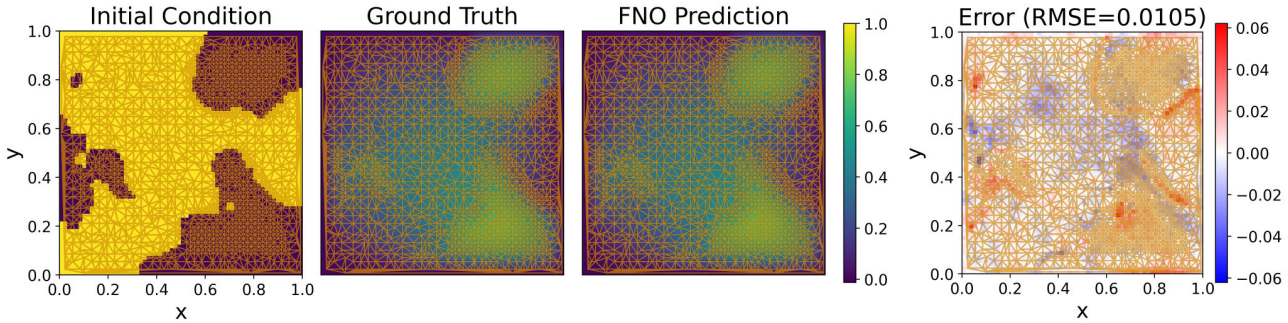
(e) Instance 3 — no mesh



(f) Instance 3 — with mesh overlay

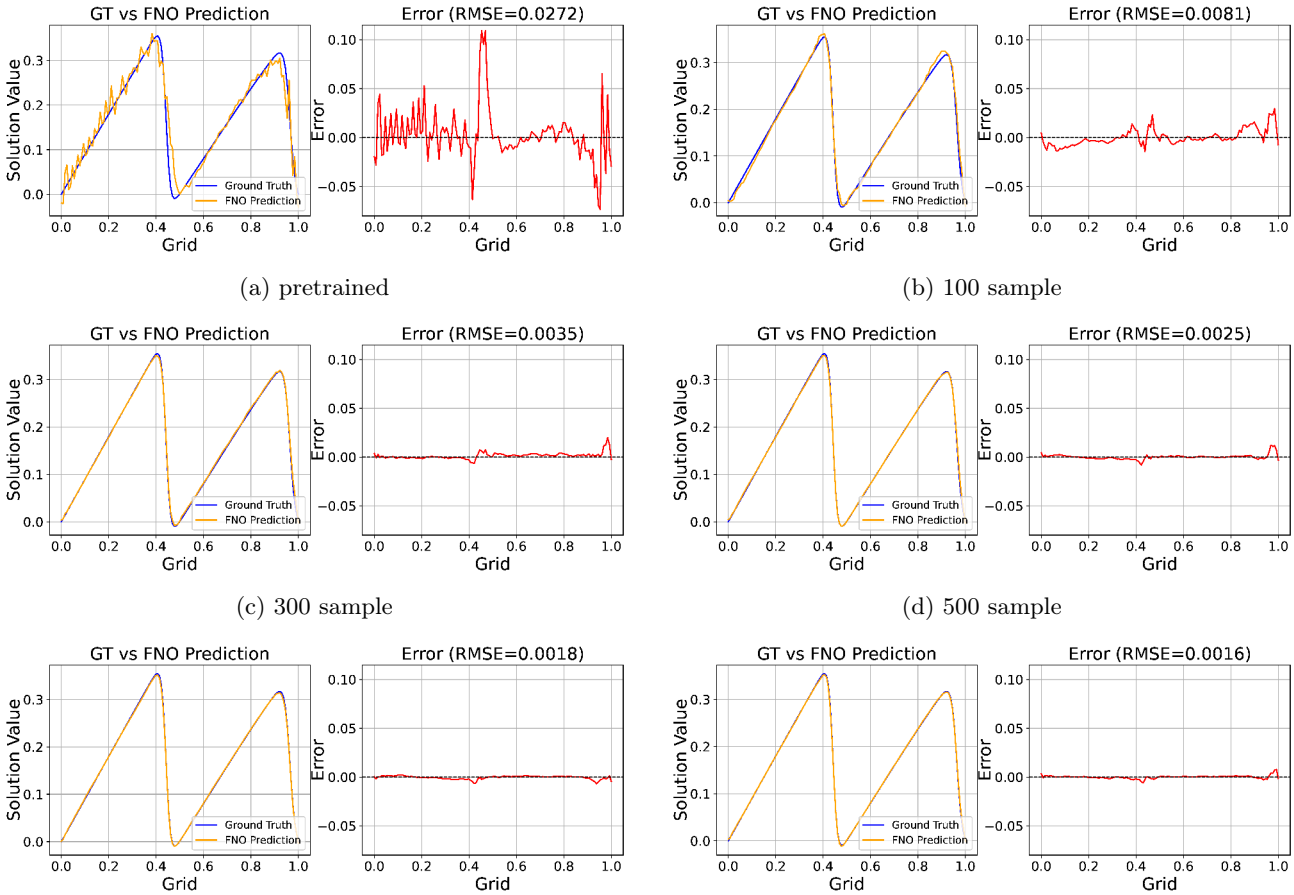


(g) Instance 4 — no mesh



(h) Instance 4 — with mesh overlay

Figure 12: Darcy predictions with and without mesh overlay (instances 3–4, continued).



(a) pretrained

(b) 100 sample

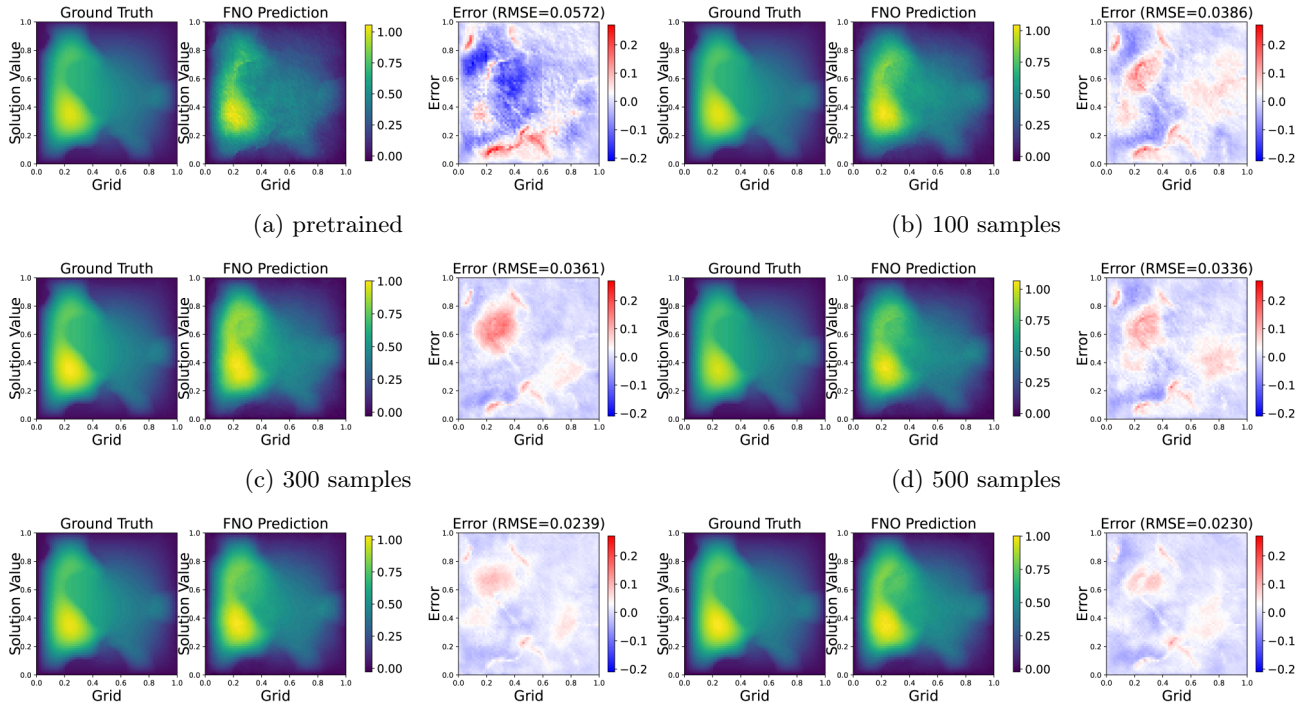
(c) 300 sample

(d) 500 sample

(e) 700 sample

(f) 900 sample

Figure 13: Active learning progression: each subfigure shows *(left)* ground truth and model prediction, and *(right)* error map as iterations proceed.



(a) pretrained (b) 100 samples

(c) 300 samples (d) 500 samples

(e) 700 samples (f) 900 samples

Figure 14: Active learning progression: each subfigure shows *(left)* ground truth, *(middle)* model prediction, and *(right)* error map as the active learning iterations proceed.