# AI in the Sciences and Engineering HS25

# Final Project Report

Marco De Negri

ETH Zurich

mdenegri@ethz.ch

February 1, 2026

# 1 Task 1: Visualizing Loss Landscapes - PINNs vs. Data-Driven

## 1.1 Problem and Methodology

Spectral bias is investigated in Physics-Informed Neural Networks (PINNs) by solving the 2D Poisson equation $-\Delta u = f$ on the unit square $D = [0,1]^2$ with homogeneous Dirichlet boundary conditions. The source term $f$ and exact solution $u$ are constructed as Fourier series with random coefficients $a_{ij} \sim \mathcal{N}(0,1)$, where the parameter $K \in \mathbb{N}$ controls the maximum frequency content. As $K$ increases, the solution shows increasingly high-frequency spatial variations, challenging the network's ability to capture oscillatory behavior. The network is trained to fit a single PDE instance with fixed coefficients (seed=0), focusing on optimization dynamics rather than generalization across distributions.

**Neural Network Architecture.** Both approaches use an identical MLP with 4 hidden layers of 256 neurons and Tanh activations, mapping spatial coordinates $(x, y)$ to solution values $\hat{u}_\theta(x, y)$. The *data-driven loss* uses mean squared error between network predictions and exact solution values on $N_d = 4{,}096$ grid points ($64 \times 64$) with output normalization. The *PINN loss* enforces PDE residuals: the Laplacian $\Delta\hat{u}_\theta$ is compared against the source term $f$ at $N_f = 20{,}000$ Sobol-sampled collocation points, plus a boundary condition penalty ($\lambda = 400$) enforcing zero values at domain edges. Representative examples of source terms and exact solutions at different frequency levels are shown in Figure 5.

**Training and Visualization.** Both models use two-stage optimization: Adam (PINN: 5,000 iterations at lr=$10^{-3}$; Data-Driven: 2,000 iterations at lr=$10^{-4}$) followed by L-BFGS fine-tuning (max 10,000 iterations). Loss landscapes are visualized following Li et al. [2018] by evaluating $g(\alpha, \beta) = \mathcal{L}(\theta^* + \alpha\delta + \beta\eta)$ on a $64 \times 64$ grid with $\alpha, \beta \in [-1, 1]$, where $\delta, \eta$ are filter-normalized random directions from the converged parameters $\theta^*$.

**Implementation Details.** To ensure an identical network capacity and training procedure, the implementation uses a `BaseSolver` abstract class containing shared logic (network initialization, optimizer loops, error computation), with `PoissonPINN` and `DataDrivenSolver` subclasses overriding only `prepare_training_data()` and `compute_loss()`. Collocation points for PINN are sampled using Sobol quasi-random sequences for improved space-filling coverage over uniform random sampling. The PINN Laplacian $\nabla^2 u$ is computed via automatic differentiation following the course tutorial implementation: first-order derivatives are obtained using `torch.autograd.grad` with `create_graph=True`, then differentiated again to get second-order terms. Output normalization (zero mean, unit variance) is applied only in the Data-Driven approach to address gradient scaling issues from small solution magnitudes ($\sim 10^{-2}$); the PINN loss naturally constrains solution scale through physics and does not require normalization. For Task 1.3, the filter normalization code was adapted from the Li et al. [2018]'s official GitHub repository to generate direction vectors $\delta, \eta$ with layer-wise scaling.

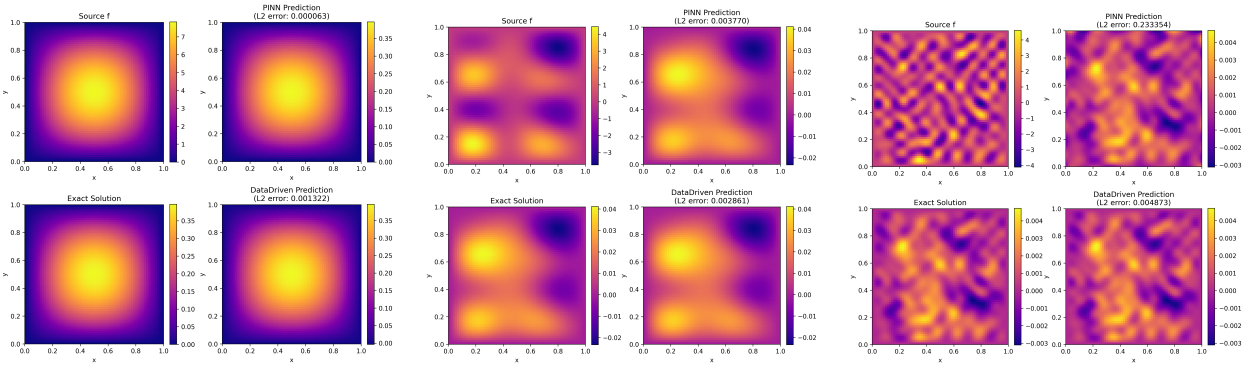



Figure 1: Predictions for increasing complexity $K \in \{1, 4, 16\}$ (left to right). Each 2×2 panel displays (clockwise): source term $f$, PINN prediction, Data-Driven prediction, and exact solution $u$.

## 1.2 Results and Analysis

Table 1: Performance comparison.

| K | $L^2$ Error | | Iter. | |
|---|---|---|---|---|
| | PINN | DD | PINN | DD |
| 1 | **0.0063%** | 0.1322% | 5,745 | 2,401 |
| 4 | 0.3770% | **0.2861%** | 7,067 | 4,059 |
| 16 | 23.3354% | **0.4873%** | 15,379 | 9,424 |

**Quantitative Performance.** Table 1 summarizes the relative $L^2$ errors and training iterations for both approaches. At low frequency ($K = 1$), PINN outperforms with 21× lower error. At medium frequency ($K = 4$), both methods achieve comparable accuracy. At high frequency ($K = 16$), PINN suffers a major degradation (23.3% error) while Data-Driven maintains 0.49% error despite using identical network architecture. The iteration counts reveal that PINN at $K = 16$ requires 2.7× more training than $K = 1$, but still fails, indicating optimization problems rather than capacity limitations. Figure 6 shows training loss evolution for all $K$ values, confirming convergence for both approaches, however, PINN's accuracy degrades at high $K$ despite successful loss minimization.

**Loss Landscape Geometry.** Figure 2 reveals the underlying cause through surface topology. At $K = 1$, both landscapes are well-shaped (PINN peak: 108 vs DD: 10). At $K = 4$, PINN develops sharper valleys. At $K = 16$, Data-Driven remains smooth (max ∼216) while PINN transforms into a big spike exceeding 5,000 (plot limit for better visualization), creating difficult optimization barriers.

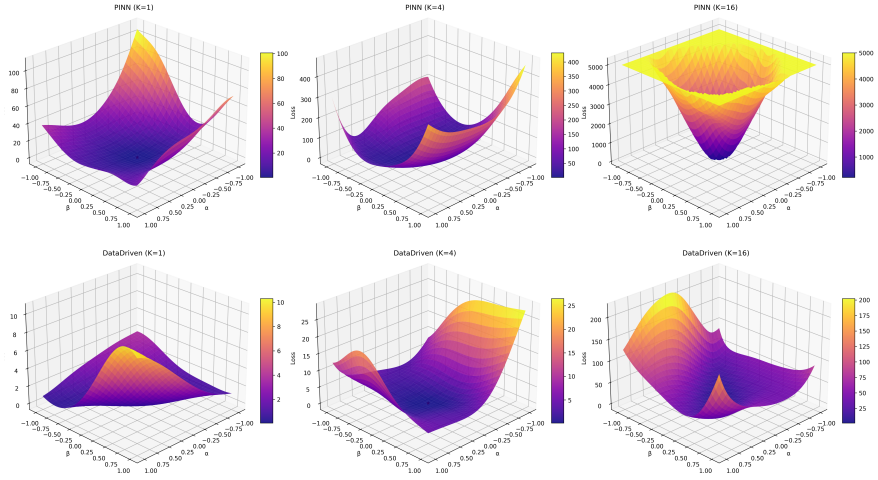



Figure 2: 3D loss landscapes: PINN (top row) vs Data-Driven (bottom row) for $K \in \{1, 4, 16\}$ (left to right).

**Spectral Bias and Failure Mechanisms.** The observed failure patterns align with established spectral bias phenomena [Krishnapriyan et al., 2021]: (1) ill-conditioned gradients emerge because the Laplacian operator scales quadratically with frequency ($\nabla^2$ scales as $k^2$), making high-frequency components exponentially harder to train; (2) neural networks intrinsically prefer low-frequency functions due to their smooth activation functions and the way gradients propagate through the network during backpropagation—high-frequency features receive weaker gradient signals, causing the optimizer to converge toward smoother, low-frequency solutions; and (3) the PDE residual loss becomes increasingly difficult to minimize at high frequencies, as computing second derivatives of the network output introduces additional sensitivity to parameter changes, creating the sharp optimization barriers visible in the loss landscape geometry. Both approaches use identical network architectures (4 layers, 256 neurons), but Data-Driven achieves 0.49% error while PINN reaches 23.3% at $K = 16$, this difference shows that PINNs fail because they are hard to optimize, not because the network has insufficient representational capacity. The network can represent the correct solution, as shown by the data-driven case, however, when physics constraints are added, the loss landscape becomes very difficult to navigate at high frequencies, which is directly seen in Figure 2.

# 2 Task 2: Fourier Neural Operators for Dynamical Systems

## 2.1 Problem and Methodology

We approximate the solution operator of an unknown 1D dynamical system $\partial_t u = D(u(x,t))$ defined on $x \in [0,1]$ and $t \in (0,1]$, with boundary conditions $u(0,t) = u(1,t) = 0$ and initial condition $u(x,0) = u_0(x)$. Both the governing operator $D$ and the distribution of initial conditions are unknown and must be inferred purely from data. Training data consists of 1024 trajectories at resolution 128 with temporal snapshots at $t \in \{0, 0.25, 0.5, 0.75, 1.0\}$.

**FNO Architecture.** All models employ a Fourier Neural Operator architecture, which learns operators in spectral space rather than directly in physical space. The core component is the spectral convolution layer: after applying FFT to the input, the network performs learnable multiplication on the first 16 Fourier modes in frequency domain, then transforms back to physical space via inverse FFT. Each model stacks 3 such spectral layers interleaved with 1D convolutions and Tanh activations.

The *one2one* variant maps $[x, u_0] \to u(t=1)$ with input dimension 2 and 211,393 parameters, learning a direct endpoint mapping without temporal conditioning. The *all2all* variant uses input $[x, u(x,t_i), \Delta t]$ with dimension 3 and 212,609 parameters, incorporating Feature-wise Linear Modulation (FiLM) layers after each spectral block. FiLM applies affine transformations conditioned on time: $\text{FiLM}(x,t) = x \cdot (1 + \gamma(t)) + \beta(t)$, where learnable functions $\gamma(\Delta t)$ and $\beta(\Delta t)$ scale and shift feature channels to adapt the network's behavior to different temporal evolution intervals.

**Training Strategies.** All models are trained using mean squared error (MSE) loss. We investigate four complementary aspects of FNO-based operator learning:

*(1) One-to-One*: Direct mapping from $u_0$ to $u(t=1)$ trained on 1024 training samples (32 validation) for 100 epochs using AdamW optimizer (lr=$10^{-3}$, weight decay=$10^{-5}$, batch size 32) with StepLR scheduler (step size 50, $\gamma = 0.5$).

*(2) Multi-Resolution Testing*: Evaluating the trained one2one model at spatial resolutions $\{32, 64, 96, 128\}$ without retraining to assess resolution invariance.

*(3) All-to-All*: Training on all forward-time pairs $(t_i, t_j)$ with $t_i < t_j$ generates 10,240 samples (10 pairs per trajectory), trained for 250 epochs using AdamW optimizer (lr=$5 \times 10^{-4}$, weight decay=$10^{-5}$, batch size 32) with StepLR scheduler (step size 50, $\gamma = 0.5$).

*(4) Transfer Learning*: We evaluate transfer learning on an unknown initial condition distribution. The pretrained all2all model is first evaluated zero-shot (without retraining), then finetuned using 32 training trajectories (lr=$5 \times 10^{-5}$, weight decay=$10^{-5}$, 200 epochs, batch size 8), and compared against training from scratch on the same 32 trajectories (lr=$10^{-3}$, weight decay=$2 \times 10^{-5}$, 200 epochs, batch size 8). Both finetuning and from-scratch training use StepLR scheduler (step size 30, $\gamma = 0.5$).

**Implementation Details.** The spectral convolution layers perform learnable complex multiplication on the first 16 Fourier modes using `torch.einsum("bix,iox->box")`, with higher modes set to zero in the output. Complex weights are initialized with uniform random values scaled by $1/(c_{\text{in}} \cdot c_{\text{out}})$, similar to Xavier initialization. Each spectral layer includes parallel skip connections through 1D convolutions, allowing the network to learn both global patterns in frequency space and local features in physical space.

The time-conditional architecture adds FiLM layers after each spectral block. The affine transformation parameters $\gamma(\Delta t)$ and $\beta(\Delta t)$ are initialized to zero, so the network starts as an identity mapping ($x \cdot (1 + 0) + 0 = x$) and gradually learns temporal modulation during training, this prevents gradient instability in early training epochs. Data augmentation for all2all training creates all 10 forward-time pairs from the 5 timesteps: $(t_0, t_1), (t_0, t_2), \ldots, (t_0, t_4), (t_1, t_2), \ldots, (t_3, t_4)$. This generates 10,240 training samples from 1,024 trajectories without duplicating data.

Transfer learning updates all model parameters, spectral convolutions, FiLM modules, and skip connections, instead of freezing some layers, this allows the entire network to adapt to the new distribution.

## 2.2 Results and Analysis

**One-to-One Training and Resolution Invariance.** The one2one model achieves 1.71% error at $t = 1.0$, demonstrating effective learning of the $u_0 \to u(t=1)$ mapping from 1024 trajectories. Table 2 evaluates this model across different spatial resolutions without retraining. Errors decrease monotonically with resolution: from 24.0% at resolution 32 to 1.71% at native resolution 128. Despite training exclusively at resolution

128, the model generalizes across resolutions, leveraging the FNO's spectral representation in Fourier space. The performance degradation at resolution 32 is particularly pronounced because it represents the Nyquist sampling limit for 16 Fourier modes ($2 \times 16 = 32$ points): below this critical threshold, aliasing prevents accurate reconstruction of high-frequency components. Higher resolutions provide oversampling, enabling progressively better representation of the solution's spectral content.

Table 2: One2One model: resolution invariance testing at different spatial discretizations.

| **Resolution** | **Error (%)** |
|---|---|
| 32 | 24.00 |
| 64 | 7.53 |
| 96 | 4.19 |
| 128 | 1.71 |

**All-to-All Temporal Dynamics.** The time-conditional all2all model uses FiLM layers to incorporate temporal information, training on all forward-time pairs $(t_i, t_j)$ with $t_i < t_j$ (10,240 samples total). Table 3 shows all2all predictions at multiple timesteps. The model achieves 0.72% error at $t = 1.0$, representing 2.4× improvement over one2one's 1.71%. Training on 10 pairs per trajectory provides 10× more supervision than one2one's single endpoint. Remarkably, errors remain stable across all timesteps (∼0.7%): 0.72% at $t = 0.25$, 0.70% at $t = 0.5$, 0.69% at $t = 0.75$, and 0.72% at $t = 1.0$. This stability, unlike autoregressive methods where errors compound, arises because all predictions originate from the initial condition $u_0$ rather than recursively from intermediate states. The FiLM conditioning mechanism treats $\Delta t$ as a continuous input variable, scaling and shifting feature representations through learned affine transformations, enabling a single unified architecture to handle all time evolution tasks without requiring separate models for each target timestep.

Table 3: All2All model: error at multiple timesteps from initial condition.

| **Time** | **Error (%)** |
|---|---|
| $t = 0.25$ | 0.72 |
| $t = 0.50$ | 0.70 |
| $t = 0.75$ | 0.69 |
| $t = 1.00$ | 0.72 |

**Transfer Learning Effectiveness.** Table 4 evaluates transfer learning on an unknown initial condition distribution at $t = 1.0$. Zero-shot evaluation (using the pretrained all2all model without finetuning) gets 3.29% error, indicating reasonable generalization despite distributional shift. Finetuning with only 32 trajectories improves to 3.07%, while training from scratch on the same limited data severely underfits at 9.93% error. The finetuned model achieves 3.2× lower error than from-scratch training, conclusively demonstrating transfer learning success. Pretrained representations encode temporal evolution dynamics that transfer across distributions, providing inductive bias in low-data regimes where random initialization leads to poor convergence.

Table 4: Transfer learning on unknown distribution: zero-shot, finetuning, and from-scratch comparison.

| **Approach** | **Training Data** | **Error (%)** |
|---|---|---|
| Zero-Shot | 0 traj (pretrained) | 3.29 |
| Finetuned | 32 traj | 3.07 |
| From Scratch | 32 traj | 9.93 |

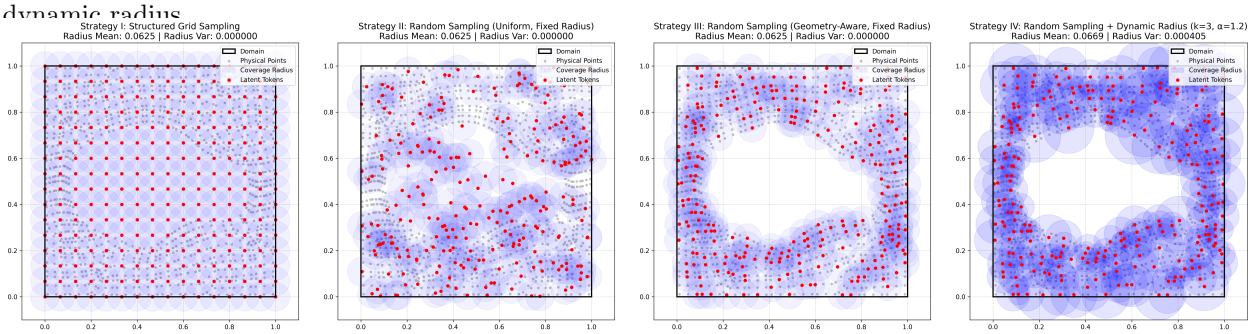
# 3 Task 3: GAOT with Random Sampling Tokenization

## 3.1 Problem and Methodology

The Geometry-Aware Operator Transformer (GAOT) [Wen et al., 2025] uses latent tokens placed on a regular grid (*Strategy I*) to represent spatial information. While effective for regular geometries, this becomes inefficient for irregular meshes: the Elasticity dataset has only 972 mesh points, yet baseline GAOT uses 4,096 tokens ($64 \times 64$ grid). This task extends GAOT to *Strategy II*: random sampling with dynamic radius, where $N = 256$ tokens are sampled directly from mesh points, reducing token count by $16\times$ while respecting problem geometry.

With random token placement, fixed radius $r$ causes gaps in sparse regions or redundant overlap in dense regions. Each token is assigned an adaptive radius following RIGNO [Mousavi et al., 2025]: $r_l = \alpha \cdot d_k(y_l)$ where $d_k(y_l)$ is the distance to the $k$-th nearest token neighbor. With $\alpha = 1.2$ and $k = 3$, tokens in sparse regions get larger radii while dense regions get smaller radii, ensuring uniform coverage.

Evaluation uses the Elasticity dataset (2D linear elasticity, 972 irregular mesh points, predicting displacement fields). *Task 3.1* establishes the baseline with grid tokenization; *Task 3.2* tests random sampling with dynamic radius.




(a) Grid (baseline) (b) Random uniform (c) Random geo-aware (d) Dynamic radius

Figure 3: Tokenization strategies comparison. (a) Baseline: 4,096 tokens on $64 \times 64$ grid with fixed radius. (b-c) Random sampling: 256 tokens with uniform vs geometry-aware placement. (d) Dynamic radius: circle sizes vary from 0.14 to 0.49 based on local token density, with larger radii in sparse regions.

## 3.2 Implementation Details

**Architecture Modifications.** The implementation required changes to 10 source files in the GAOT codebase. The main modifications are in: (1) `data_processor.py` for random token generation, (2) `graph_builder.py` for dynamic radius computation and graph construction, and (3) `neighbor_search.py` to support per-token radius values.

**Token Generation.** Given the 972 physical mesh points, 256 indices are sampled using `torch.randperm` with a fixed seed (42) for reproducibility. Token coordinates are normalized to $[-1, 1]$ using the same scaling as the baseline. This ensures tokens lie on actual mesh positions rather than arbitrary points in the bounding box.

**Dynamic Radius Computation.** For each token $y_l$, pairwise distances to all other tokens are computed using `torch.cdist`. The $k$-th smallest distance (excluding self-distance) is extracted and multiplied by $\alpha$. This computation runs once during initialization and is reused throughout training.

**Graph Construction.** The encoder graph connects physical points to tokens: for each token $y_l$, physical points $x$ within distance $r_l$ are identified. The decoder graph is obtained by transposing this relation using a CSR (Compressed Sparse Row) operation, ensuring symmetric information flow without recomputation.

**Configuration.** Both baseline and random sampling use the same GAOT architecture: hidden size 64, 3 MLP layers, 256-dimensional transformer, and 3,396,033 parameters. Training uses AdamW (lr=$8 \times 10^{-4}$, weight decay=$10^{-5}$), 300 epochs, and batch size 64.

## 3.3 Results and Analysis

**Task 3.1: Baseline Results.** The grid $64 \times 64$ baseline (4,096 tokens, $r = 0.033$) achieves 5.29% error after 300 epochs. With token count reduced to 256 (grid $16 \times 16$, $r = 0.125$), error increases to 8.83%

Table 5: Task 3 results: comparison of tokenization strategies on the Elasticity dataset.

| Strategy | Tokens | Radius | Rel. Error | Time |
|---|---|---|---|---|
| Grid $64 \times 64$ (baseline) | 4,096 | 0.033 (fixed) | **5.29%** | ∼8h |
| Grid $16 \times 16$ | 256 | 0.125 (fixed) | 8.83% | ∼1h |
| Random + Dynamic ($k$=3) | 256 | 0.07–0.25 (mean 0.13) | 24.46% | ∼1h |

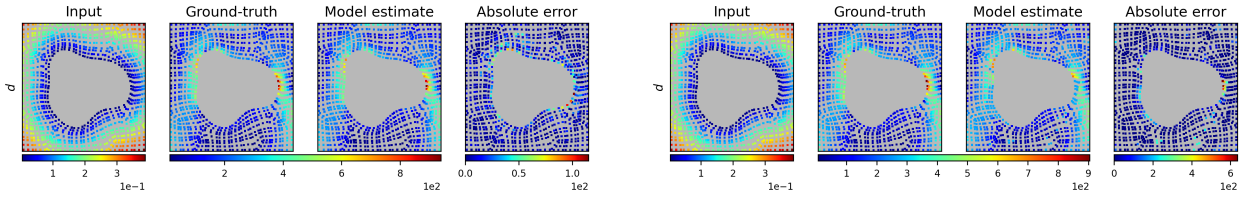
(1.7× degradation). An initial experiment with 256 tokens but the original small radius ($r$=0.033) produced catastrophic failure (81.9% error), demonstrating that radius must scale with token density to maintain coverage.

**Task 3.2: Random Sampling Results.** Random sampling with dynamic radius ($k$=3, $\alpha$=1.2) achieves 24.46% error, 2.8× worse than grid $16 \times 16$ (8.83%) at equal token count (256), indicating the gap stems from tokenization strategy rather than token reduction. This reveals that structured grid placement provides a fundamental advantage: regular spatial structure benefits the transformer's attention mechanism, which relies on learned positional relationships.

Table 6: Dynamic radius statistics ($k$=3, $\alpha$=1.2).

| Metric | Value |
|---|---|
| Minimum | 0.065 |
| Maximum | 0.254 |
| Mean | 0.134 |

**Dynamic Radius Behavior.** Table 6 shows radii statistics: range 0.065 - 0.254, mean 0.134. Crucially, all 972 physical mesh points are covered by at least one token (100% coverage verified for both strategies, see Figure 3), confirming the performance gap comes from how tokens are placed, not from how much is covered.




(a) Baseline (5.29% error)

(b) Random + Dynamic (24.45% error)

Figure 4: Qualitative comparison of displacement field predictions. The baseline captures fine-scale deformation patterns, while random sampling produces smoother predictions with less detail.

## 3.4 Discussion: Performance Gap Analysis

The experiments demonstrate that random token placement fundamentally underperforms structured grids even with full mesh coverage. At equal token count (256) and comparable mean radius (∼0.13), random sampling achieves 24.46% error versus the grid's 8.83%, proving that *token placement pattern itself* highly influence performance.

This performance gap comes from GAOT's transformer architecture, which uses positional encodings designed for regular grids where spatial and sequence adjacency align. Random tokenization breaks this correspondence: sequentially adjacent tokens may be spatially distant, disrupting learned positional priors essential for accurate operator learning. The Bonus Task's continuous coordinate-based encodings could address this limitation but were not implemented here.

While GAOT [Wen et al., 2025] focuses on grid tokenization, RIGNO [Mousavi et al., 2025], which inspired the dynamic radius formula, achieves competitive results with random sampling using graph neural networks instead of transformers. This might suggests that successful random tokenization with attention-based models requires fundamental architectural changes beyond tokenization strategy alone.

While the dynamic radius mechanism successfully adapts to random placement (mean radius 0.134 ≈ grid's 0.125), it cannot compensate for lost spatial structure. Potential improvements like increasing training samples/epochs or tuning $\alpha$ and $k$ would likely yield only incremental gains without addressing the core architectural mismatch between random tokenization and transformer-based positional encodings.

# Appendix: Additional Figures

## Task 1: Visualizing Loss Landscapes

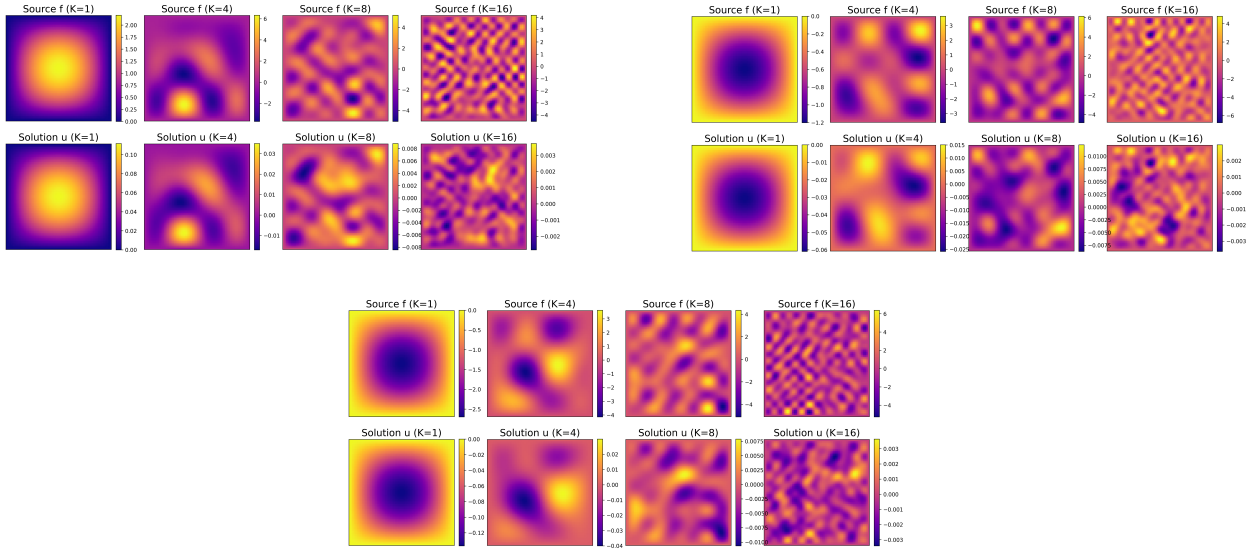



Figure 5: Representative examples of source terms $f$ and exact solutions $u$ for $K \in \{1, 4, 8, 16\}$. Each panel shows a 2×2 grid with source term $f$ (top-left), exact solution $u$ (top-right), and their spatial variations.

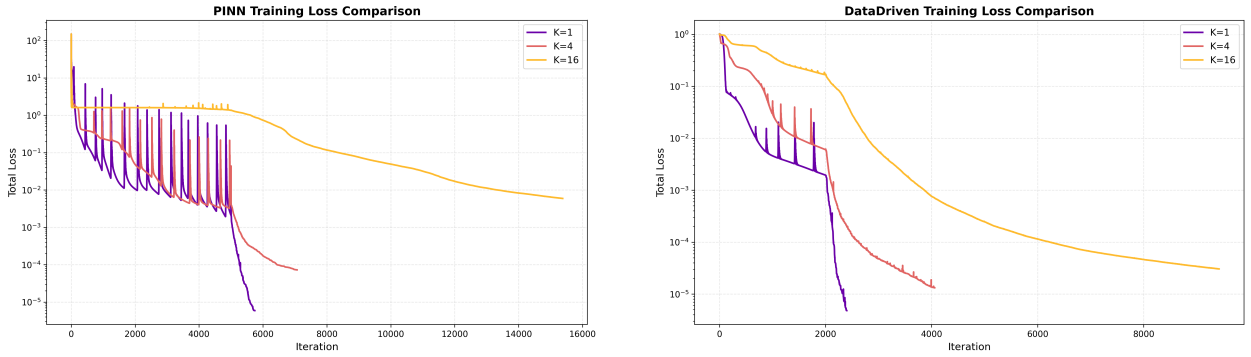



Figure 6: Training loss evolution for all $K$ values, showing convergence for both approaches.

# References


Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks, 2021. URL https://arxiv.org/abs/2109.01050.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets, 2018. URL https://arxiv.org/abs/1712.09913.

Sepehr Mousavi, Shizheng Wen, Levi Lingsch, and Siddhartha Mishra. A graph-based framework for robust and accurate operator learning for pdes on arbitrary domains, 2025. URL https://arxiv.org/abs/2501.19205.

Shizheng Wen, Arsh Kumbhat, Levi Lingsch, Sepehr Mousavi, Yizhou Zhao, Praveen Chandrashekar, and Siddhartha Mishra. Geometry aware operator transformer as an efficient and accurate neural surrogate for pdes on arbitrary domains, 2025. URL https://arxiv.org/abs/2505.18781.