Final Project

ETH zürich

AI in the Sciences and Engineering
Due date: January 26th, 2026

IMPORTANT INFORMATION

This document describes the final project. The due date for student submissions is set for **26 January 2026**.

This project consists of three main tasks. The total number of points is 140 + 50 bonus points. **The final submission consists of the the code and a report of maximum 2 pages per task**, where you should succinctly describe the procedure followed in each task. You may also include figures in your report. The submissions have to be collected in a zip folder named as *yourfirstname_yoursecondname_yourleginumber.zip*.

Grading System

- A perfect grade of 6.0 requires accumulating at least 120 points.
- Note that we aim to evaluate your understanding and effort. We will not enforce strict performance metrics in any task.

Bonus System

- The project includes multiple **bonus questions**.
- These questions account for additional 50 points.
- The questions are more challenging, and require deeper understanding of the topics.

1. Visualizing Loss Landscapes: PINNs vs. Data-Driven (40 points + 20 points)

Physics-Informed Neural Networks (PINNs) have emerged as a powerful tool for solving partial differential equations (PDEs). However, unlike traditional supervised learning (Data-Driven), PINNs often suffer from "pathological" optimization behaviors. The loss landscape of a PINN is determined by high-order derivatives of the network, which can result in highly non-convex, rough, and ill-conditioned landscapes, especially as the frequency of the solution increases.

In this exercise, you will investigate the spectral bias and optimization complexity of neural PDE solvers. You will solve a multiscale Poisson equation using both a **PINN** (residual-based) and a **Data-Driven** (supervised) approach. Your goal is to visualize and compare how the loss landscape degrades as the frequency of the target solution increases.

Problem Statement

We focus on a prototypical linear elliptic PDE, the **Poisson Equation**, defined on a 2D square domain $D = [0, 1]^2$:

$$\begin{aligned} -\Delta u &= f, \quad \text{in } D, \\ u &= 0, \quad \text{on } \partial D. \end{aligned}$$

To control the complexity of the problem, we define a source term f comprised of K spatial scales. The source term and its corresponding analytical solution $u(x, y)$ are given by:

$$f(x, y) = \frac{\pi}{K^2} \sum_{i,j=1}^{K} a_{ij} \cdot (i^2 + j^2)^r \sin(\pi i x) \sin(\pi j y)$$

$$u(x, y) = \frac{1}{\pi K^2} \sum_{i,j=1}^{K} a_{ij} \cdot (i^2 + j^2)^{r-1} \sin(\pi i x) \sin(\pi j y)$$

where $r = 0.5$ and coefficients $a_{ij} \sim \mathcal{N}(0, 1)$. The parameter K acts as an indicator for the problem's complexity: as K increases, the solution contains higher frequency components.

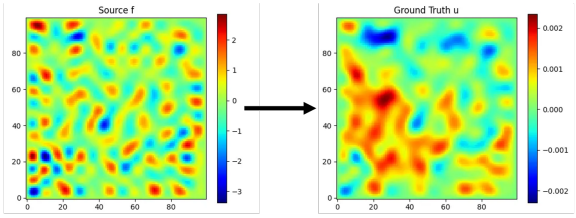

Figure 1: Visualization of one data sample with $K = 16$.

Tasks

Task 1: Data Generation (10 points)

Write a script to generate a dataset of input-output pairs $(f^{(i)}, u^{(i)})$. Discretize the domain D into an $N \times N$ structural grid (e.g. $N = 64$). Generate different samples by randomizing the coefficients

a_{ij} for each sample. Plot 3 examples of the source field f and the corresponding solution field u for $K = 1, 4, 8, 16$.

Task 2: Implementation and Training (30 points)

Implement a standard Multi-Layer Perceptron (MLP) with 3-4 hidden layers. The MLP here acts as a direct function approximator $u_\theta : \mathbb{R}^2 \to \mathbb{R}$. The input to the network is the spatial coordinates (x, y) of a point, and the output is the predicted scalar value $\hat{u}$ at that location. For this task, you will optimize the network to solve the PDE for one specific source term f only. The model is expected to fit the solution for this single case; no generalization to other source terms or distributions is required in this exercise. You will perform this optimization twice: once using the Data-Driven loss and once using the PINN loss.

- **(20 points)** For PINN loss, you need to define the loss calculator similar to the following form: $\mathcal{L}_{\text{PINN}}(\theta) = \frac{1}{N_f} \sum |-\Delta\hat{u}_\theta - f|^2 + \lambda\mathcal{L}_{\text{BC}}$. Then use the automatic differentiation to calculate the gradient.
- **(10 points)** For Data-Driven loss, the loss calculator: $\mathcal{L}_{\text{Data}}(\theta) = \frac{1}{N_d} \sum |\hat{u}_\theta - u_{\text{exact}}|^2$.

You need to train both models for different complexity levels: **Low** ($K = 1$), **Medium** ($K = 4$), and **High** ($K = 16$). The specific source term f to be solved can be defined by yourself. We recommend selecting one representative instance from the samples generated in the Task 1 to ensure consistency in your comparisons. Please record your final L_2 relative error, training loss curve, and model prediction in your report.

Note: To accelerate convergence, adopt a combined optimizer strategy: start with **Adam** and switch to **L-BFGS** for fine-tuning.

Task 3: Loss Landscape Visualization (BONUS - 20 points)

Using the techniques from *Li et al. (2018)*, visualize the geometry of the loss function around the converged (or stuck) solution parameters θ^*. Compute the loss on a 2D plane:

$$g(\alpha, \beta) = \mathcal{L}(\theta^* + \alpha\delta + \beta\eta) \tag{1}$$

where δ and η are two principal direction vectors.

- **(15 points)**: Provide 2D contour plots or 3D surface plots of the landscapes for both PINN and Data-Driven models across the three K levels.
- **(5 points)**: Qualitatively describe the differences. Is the PINN landscape sharper? Does it exhibit more local minima as K increases compared to the Data-Driven approach? Reference the "Failure modes" described by *Krishnapriyan et al. (2021)* in your discussion.

References

- Krishnapriyan et al. (2021). Characterizing possible failure modes in physics-informed neural networks.
- Li et al. (2018). Visualizing the loss landscape of neural nets.

2. Training an FNO to approximate a dynamical system (50 + 10 points)

In this exercise, you will train a **Fourier Neural Operator** to approximate an unknown dynamical system described by

$$\frac{\partial u}{\partial t} = \mathcal{D}(u(x,t)), \quad t \in (0,1], \quad x \in [0,1],$$

with boundary conditions

$$u(0,t) = u(1,t) = 0$$

and initial conditions

$$u(x,0) = u_0(x).$$

The initial conditions are sampled from an unknown distribution. As the exact physics of the problem is unknown, you will have to approximate it from the available data.

Note: You should use a Fourier Neural Operator (FNO) for all the tasks.

Dataset Details

You are provided with the following datasets in this **folder**:

1. **Training Dataset:** `data_train_128.npy`
   - Shape: $(1024, 5, 128)$
   - Description:
     - 1024: Number of trajectories.
     - 5: Time snapshots of the solution. For a given trajectory u, the time snapshots are:

       $u[0]$: Initial condition u_0 at $t = 0.0$,
       $u[1]$: Solution at $t = 0.25$,
       $u[2]$: Solution at $t = 0.50$,
       $u[3]$: Solution at $t = 0.75$,
       $u[4]$: Solution at $t = 1.0$.

     - 128: Spatial resolution of the data.
     - Please see Figure 2 for visualization.
2. **Validation Dataset:** `data_val_128.npy`
   - Shape: $(32, 5, 128)$
   - This dataset should be used as a validation dataset during the training.
   - The initial conditions $u[0]$ are sampled from the same distribution as in `data_train_128.npy`.
3. **Testing Datasets:** `data_test_{s}.npy`:
   - Testing datasets at varying spatial resolutions $s \in \{32, 64, 96, 128\}$ of the shape $(128, 5, s)$

- The initial conditions $u[0]$ are sampled from the same distribution as in `data_train_128.npy`.

4. **Different Initial Distribution Datasets:**

   (a) `data_finetune_train_unknown_128.npy`:

   - Shape: $(32, 5, 128)$
   - Dataset to be used for **finetunning**.
   - The initial conditions $u[0]$ are sampled from **an unknown distribution, different from** `data_train_128.npy`.

   (b) `data_finetune_val_unknown_128.npy`:

   - Shape: $(8, 5, 128)$
   - Dataset to be used for validation during finetunning.
   - The initial conditions $u[0]$ are sampled from the same distribution as in `data_finetune_train_unknown_128.npy`.

   (c) `data_test_unknown_128.npy`:

   - Shape: $(128, 5, 128)$
   - Dataset to be used for **testing the models on the unknown distribution** of inputs.
   - The initial conditions $u[0]$ are sampled from the same distribution as in `data_finetune_train_unknown_128.npy`.

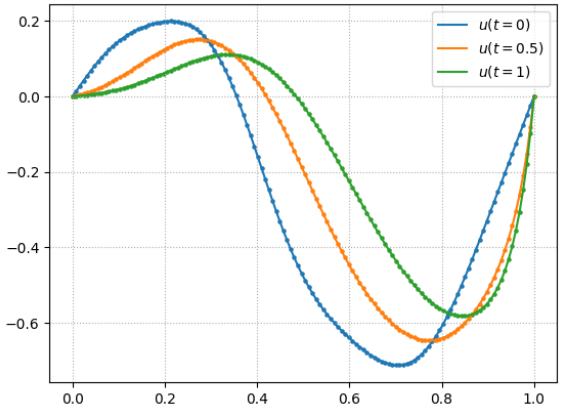

Figure 2: Training Dataset - One trajectory

Tasks

Task 1: One-to-One Training (10 points)

From the **all 1024 trajectories** in the training dataset, keep only the first and last time snapshots, corresponding to $t = 0.0$ and $t = 1.0$. Train an FNO model (**one2one** training) to learn the mapping

$$G : u_0 \mapsto u(t = 1.0).$$

After training, evaluate the model on the `data_test_128.npy` dataset, considering *only* predictions at the final time $t = 1.0$. Report the average relative L^2 error over the 128 test trajectories, defined as

$$\text{err} = \frac{1}{128} \sum_{n=1}^{128} \frac{\left\| u_{\text{pred}}^{(n)}(t = 1.0) - u_{\text{true}}^{(n)}(t = 1.0) \right\|_2}{\left\| u_{\text{true}}^{(n)}(t = 1.0) \right\|_2}.$$

Task 2: Testing on Different Resolutions (10 points)

Test the trained model **from Task 1** on the datasets `data_test_{s}.npy` for $s \in \{32, 64, 96, 128\}$. Compute and report the average relative L2 error for each dataset (at $t = 1.0$). What do you observe about the model's performance across different resolutions?

Task 3: All2All Training (15 points)

Now, use **all provided time snapshot**s ($t = 0.0, 0.25, 0.50, 0.75, 1.0$) of the 1024 training trajectories to **train a time-dependent** FNO model (**all2all** training). Note that this is similar to the task that we had in time-dependent CNO tutorial. *Hint: Use time-conditional normalizations and include time as one of the input channels.*

- **(10 points)** Test the trained model on the `data_test_128.npy` dataset, focusing **only** on predictions at $t = 1.0$. Report the average relative L2 error. Compare the error to the one obtained in Task 1. What do you observe?
- **(5 points)** Now use the model to make predictions at multiple time steps: $t = 0.25, t = 0.50, t = 0.75, t = 1.0$. Compute the average relative L2 error **for each time step**. What do you observe about the model's performance over time?

Task 4: Finetunning (15 points + 10 points)

- **(5 points)** Test the model model trained in Task 3 at $t = 1.0$ of the dataset drawn from an unknown distribution `data_test_unknown_128.npy` in a zero-shor manner. Report the average relative L2 error. How does the error compare to the one obtained in Task 1?
- **(10 points)** Use 32 trajectories from `data_finetune_train_unknown_128.npy` to **finetune the model trained in Task 3** in all2all fashion on the unknown data. Test the finetuned model on `data_test_unknown_128.npy`. Report the average relative L2 error. What is the influence of the finetunning?
- **(BONUS - 10 points)** Use 32 trajectories from `data_finetune_train_unknown_128.npy` to **train a new model from scratch** in all2all fashion on the unknown data. Test the finetuned model on `data_test_unknown_128.npy`. How does the error of the model trained from scratch compare to the error of the finetuned model? Is transfer learning successful?

3. Geometry-Aware Operator Transformer (GAOT) (50 points + 20 points)

In this project, you will work with the state-of-the-art Geometry-Aware Operator Transformer **(GAOT)**. The standard implementation primarily utilizes **Strategy I (Structured Stencil Grid)** for tokenization (see **S.M.** B.1), effectively treating the latent physics space as an image processed by a Vision Transformer (ViT).

However, real-world engineering problems often involve highly irregular geometries where structured grids are inefficient. Your goal is to extend GAOT to support **Strategy II: Random Sampling Tokenization**. This involves designing a dynamic radius for information aggregation and rethinking positional encodings (PE) for continuous coordinates.

Tasks

Task 1: Establishing a Baseline (10 points)

Before modifying the architecture, you must establish a performance baseline using the official GAOT implementation.

1. Download the **Elasticity** dataset (Link) as described in the GAOT paper.
2. Train the default GAOT model (using Strategy I: Stencil Grid) on this dataset.

Record the Test Relative L^1 Error, the number of tokens used, and the total training time. This will serve as your baseline for subsequent comparisons.

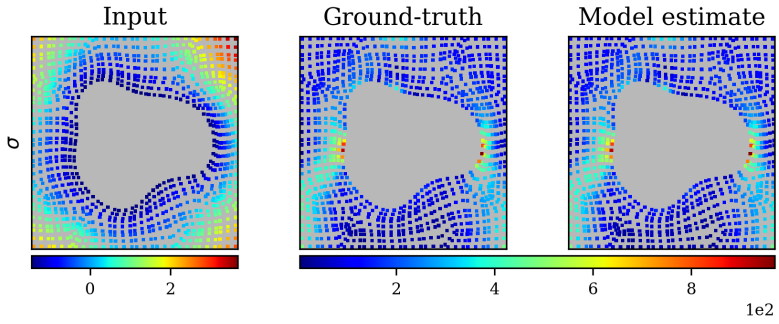

Figure 3: Model input, ground-truth solution, and model estimate of a test sample of the Elasticity dataset.

Task 2: Random Sampling & Dynamic Radius Strategy (40 points)

- **(20 points)** Modify the tokenization logic. Instead of a fixed meshgrid, you will randomly sample N points (e.g., $N = 128$ or 256) from the domain $\mathcal{D}$ to serve as latent tokens $\{y_l\}_{l=1}^{N}$. You may choose to use a *fixed set* of sampled tokens (support precompute graph structures) or *resample* every epoch (improves resolution invariance but requires rebuilding graphs).

- **(20 points)** With random points, a fixed radius r may give rise to "holes" (too small) or "large overlap" (too large) in the domain. You may implement a **Dynamic Radius** strategy inspired by *RIGNO*. Compute a local radius r_l for each token y_l:

$$r_l = \alpha \cdot d_k(y_l) \tag{2}$$

  where $d_k(y_l)$ is the distance determined by KNN or Delaunay triangulation, and α is a scaling factor. Your Goal is to ensure the graph connectivity covers the entire physical domain despite the randomness of tokens.

Task 3: Re-thinking Positional Encoding (BONUS - 20 points)

Transformers are permutation invariant and require explicit position information. In Strategy II, grid indices no longer exist, so you must implement:

- **Absolute PE**: Create an MLP that maps coordinate vectors $\mathbf{x} \in \mathbb{R}^2$ to the hidden dimension D.
- **Continuous Relative Bias**: Standard RoPE relies on integer gaps. Investigate applying RoPE using continuous coordinates or replacing it with a relative bias term based on Euclidean distance $\|\mathbf{x}_i - \mathbf{x}_j\|$ in the Attention matrix.

Since we are not using patching, the sequence length N can be large. Implement a **Cross-Attention** layer (similar to PerceiverIO) to project N random tokens into a smaller set of "seed" tokens, or propose a method to patch and merge irregular tokens (see *SpiderSolver*).

References

- Codebase: https://github.com/camlab-ethz/GAOT
- **GAOT**: Wen et al. (2025). *Geometry aware operator transformer as an efficient and accurate neural surrogate for PDEs on arbitrary domains.*
- **RIGNO**: Mousavi et al. (2025). *A Graph-based framework for robust and accurate operator learning for PDEs on arbitrary domains.*
- **SpiderSolver**: Qi et al. *A Geometry-Aware Transformer for Solving PDEs on Complex Geometries.* (NeurIPS).