Technical University Munich

School of Engineering and Design

Report for Deep Learning for PDEs in Engineering Physics

Jonas Kobel

Student ID: 03797405

Supervisors: Dr. Yaohua Zang & Vincent Scholz

July 23, 2025

Contents

Chapter 1

Problem A

1.1 Task 1

The first problem focuses on a one-dimensional elastostatic response of a linearly elastic rod subjected to a constant load. The analysis considers a static case, where dynamic effects such as acceleration and inertia are effects that are insignificant. This assumption is valid for quasi-static loading where either the load has been applied sufficiently slowly or, the system has reached an equilibrium state such that all particle movements have decayed and the configuration has reached a time-independent state.

1.1.1 Problem Description

The problem can be described with the governing equation for the displacement field $u(x) \in \mathbb{R}$ of the rod, defined over the spatial domain $x \in (0, L)$. The equation can be considered as a second-order-linear elliptic partial differential equation (PDE) given by:

$$-\frac{d}{dx}\left(E(x)\frac{du}{dx}\right) = f, \quad x \in (0, L) \tag{1.1}$$

where $E(x)$ represents the spatial distribution of the Young's modulus and f the constant body force per unit length, such as gravity. The rod can be assumed to be fixed at both ends, resulting in a homogeneous Dirichlet boundary conditions:

$$u(0) = u(L) = 0 \tag{1.2}$$

The material heterogeneity of the rod is modeld by a piecewiese-constant function of the Young's modulus, defined as:

$$E(x) = \begin{cases} 5. & 0.15 < |x - 0.5| < 0.35 \\ 2. & otherwise \end{cases} \tag{1.3}$$

This setup introduces a stiff inclusion in the central region of the rod, surrounded by a softer material, thus creating a discontinuous but well-defined stiffness profile The objective of this study is to approximate the displacement field $u(x)$ using a modern deep learning approach, given the known profile of $E(x)$ and the external loading f.
Traditional numerical methods, such as the finite element method like FEM, are well established for such problems. However, in this work, the use of self-superwised deep learning methods is used, which integrate physical laws into the learning process by embedding the PDE and boundary conditions directly into the loss function.

In particular, Task 1 focuses on using a deep learning model to approximate $u(x)$ and evaluate its accuracy compared to the reference solution provided in the dataset `LinearElasticity1d.h5`. The prediction and the true solutions are visualized along with the pointwise error. The accuracy will be measured using the L^2 relative error.

1.1.2 Methodology

The method selection

Since the task focuses on a problem with unknown targets, we have to consider self-supervised methods like Physics-Informed Neural Networks (PINNs) or the DeepRitz method. Both approaches use physical knowledge in the training process by directly minimizing a physics-based loss function rather than relying on labeled data.
To approximate the displacement field $u(x)$ governed by the elliptic PDE, the DeepRitz method is used. The DeepRitz method works by minimizing a scalar energy functional derived from the variational formulation of the PDE. Unlike residual-based methods such as PINNs, which enforce the governing equation pointwise. DeepRitz directly targets the global energy minimization, resulting in improved numerical stability and better convergence behavior.[4]
This variational formulation is well-suited for elliptic PDEs, especially when the solution may show a non-smooth behavior, such as kinks or discontinuities in the derivative caused by piecewise material properties like in this task.[3]
Although DeepRitz requires denser sampling and higher computational cost, these factors are negligible in this 1D setting.

The loss function

The energy loss function for the DeepRitz method can be defined as:

$$L_{PDE}(\theta) = I[u_\theta] \approx \sum_{i=1}^{N_{int}} w_i \left(\frac{1}{2} |\nabla u_\theta(x_i)|^2 - f(x_i) u_\theta(x_i) \right). \tag{1.4}$$

and is based on the minimization of the energy formulation:

$$\mathcal{E}[u] = \frac{1}{2} \int_0^1 E(x) \left(\frac{du}{dx} \right)^2 dx - \int_0^1 f(x) \cdot u(x)\, dx \tag{1.5}$$

To implement the boundary conditions, a mollifier function is used, which enforces the boundary conditions to be for $u(0) = u(1) = 0$. The mollifier function can be defined as:

$$u(x) = u * sin(\pi * x) \tag{1.6}$$

A total of 10,000 collocation points are used for the interior domain x_{in} to enforce the physics-informed constraints during training. For the activation function the function $\sin \pi * \tanh x$ from the lecture is used. This function performed best in various counducted tests.

The network structure

The network consists of four fully connected layers and one input and one output layer. For the activation function a self defined activation is used, which enforces non-linearity and smoothness and performed best in various conducted tests.

Table 1.1: Architecture of Neural Network Model

Layer	Size / Neurons	Activation Function	Notes
Input Layer	1	–	1 Input
Hidden Layer 1	100	$-sin(\pi tanh(x))$	Fully connected
Hidden Layer 2	100	$-sin(\pi tanh(x))$	Fully connected
Hidden Layer 3	100	$-sin(\pi tanh(x))$	Fully connected
Hidden Layer 4	100	$-sin(\pi tanh(x))$	Fully connected
Output Layer	1	$-sin(\pi tanh(x))$	1 Ouput

The code link

https://github.com/jonaskobel/Projects—Deep-Learning-for-Partial-Differential-Equations-in-Engineering-Physics

1.1.3 Numerical Experiments

Experiment Setups

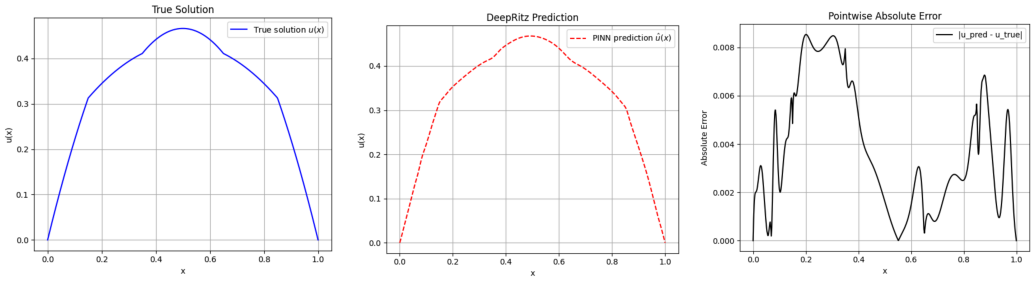
Table 1.2: Numerical Setup for Training the DNN in PyTorch

Parameter	Value
Optimizer	AdamW
Learning Rate (initial)	1×10^{-3}
Batch Size (training)	200
Batch Size (validation)	100
weight decay	1×10^{-3}
Number of Epochs	1000
Learning Rate Scheduler	StepLR
Decay Rate (α/N epochs)	0.5/250
Hardware	T4 GPU (Google Colab)
Precision	Float32
Early Stopping	Yes (patience = 200)

To implement a validation loop and set a early stop criteria, the test set x_{test} and u_{test} are divided into test and validation set with a ration of 80/20. To sepererate train loaders are used. In each validation loop the average L^2 error for the batch gets calculated, if the error plus a tolerance is bigger the the previous on, the counter is set plus one.

Numerical Results

The training terminated at epoch 701 due to early stopping, with a total runtime of 295 seconds. The final relative L^2 error is 1.29×10^{-2}, indicating high accuracy. As shown in the absolute error plot, the predicted displacement closely matches the reference solution across the length with an absolute error less then 0.0085.



(a) True solution (b) Predicted solution (c) Pointwise absolute error

Figure 1.1: Experiment results of Task 1: (a); (b); (c).

As observed in the absolute error plot, the model loses the most accuracy primarily around the regions with non-smooth transitions. In contrast, the top region and the continuous side areas are well generalized by the model.

1.2 Task 2

1.2.1 Problem Description

The second task considers the same 1D linear elastostatic problem as in Task 1. However, instead of computing the displacement field from a known material distribution, the focus now is to recover the spatially varying Young's modulus $E(x)$ from noisy observations of the displacement field $u(x)$.
More precisely, the displacement field u_{obs} is given at randomly located sensors x_{obs}, contaminated with approximately 5% noise. Additionally, the Young's modulus is known at the boundary, i.e., $E(0) = E(1) = 1$. The inverse problem consists of reconstructing the function $E(x)$ such that the resulting displacement field satisfies the underlying elliptic PDE and aligns with the noisy observations.
This is a typical inverse problem with limited and noisy data, requiring robust and physics-informed learning approaches to recover a material parameters.
The dataset `LinearElasticity1d.h5` provides the reference solutions for $u(x)$ and $E(x)$, with $N_{obs} = 20$ noisy measurements of $u(x)$. The quality of the reconstruction is evaluated via the L^2 relative error between predicted and true displacement fields.

1.2.2 Methodology

The method selection

To solve this inverse problem, a Physics-Informed Neural Network (PINN) is used. Unlike forward problems where methods like DeepRitz are effective, inverse problems require the simultaneous recovery of unknown parameters from sparse and noisy data. PINNs are particularly well suited for this setting, as they embed the known governing PDE as a soft constraint within the loss function. [2]

In this task, both the displacement field $u(x)$ and the unknown Young's modulus $E(x)$ are represented by neural networks. The model is trained by minimizing a composite loss that includes the mismatch between the observed noisy displacements $u_{\text{obs}}(x_{\text{obs}})$, the predicted values, the residual of the PDE, and the satisfaction of boundary conditions on $E(x)$ and $u(x)$. This formulation enables the network to learn a physically consistent material distribution from limited data, making PINNs a good choice for this type of inverse problem.

The loss function

The model consists of two neural networks: one to approximate the displacement field $u(x)$ and one to approximate the Young's modulus $E(x)$. Both networks are trained collectively by minimizing a combined total loss function. This total loss includes the data mismatch on observed displacements and the residual of the governing PDE . The PINN framework allows to integrate the physical knowledge directly into the training process by minimizing the combined loss term that enforces data consistency and PDE satisfaction, with respect to the boundary conditions.

The total loss $\mathcal{L}_{\text{total}}$ is defined as:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{data}}\mathcal{L}_{\text{data}} + \lambda_{\text{PDE}}\mathcal{L}_{\text{PDE}} \tag{1.7}$$

where $\mathcal{L}_{\text{PDE}}$ measures the residual of the PDE $\frac{d}{dx}\left(E(x)\frac{du}{dx}\right) = f(x)$ at the location points and is defined as:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{d}{dx}\left(E(x_i)\cdot\frac{du}{dx}(x_i)\right) + f(x_i)\right)^2 \tag{1.8}$$

and $\mathcal{L}_{\text{data}}$ is the MSE between the predicted displacement $u(x_{\text{obs}})$ and the observed noisy data $u_{\text{obs}}(x_{\text{obs}})$ and defined as:

$$\mathcal{L}_{\text{Data}} = \frac{1}{N}\sum_{i=1}^{N}(u(x_i) - u_{\text{obs}}(x_i))^2 \tag{1.9}$$

The boundary conditions are implemented by using tow mollifier functions. The mollifier function to set the boundary for conditions for $u(0) = u(1) = 0$ is defined as: The mollifier function can be defined as:

$$u(x) = u * sin(\pi * x) \tag{1.10}$$

The mollifier function to set the boundary conditions for E(x) with $E(0) = E(1) = 0$ is defined as:

$$e(x) = 1 + e * sin(\pi * x) \tag{1.11}$$

A total of 10,000 collocation points are used for the length of the rod x_{in} to enforce the physics-informed constraints during training. The weighting hyperparameter for the loss terms are set as follows: the physics-based term is scaled with $\lambda_{\text{PDE}} = 2$, while the data-based loss is weighted with $\lambda_{\text{data}} = 1000$. These values provided the best results during training, ensuring that the boundary conditions are well enforced while keeping high accuracy in the predicted solution.

The network structure

Two separate neural networks are used in this task: one to approximate the displacement field $u(x)$ and another to reconstruct the Young's modulus $E(x)$. The network for $u(x)$ is comparatively smaller, comprising 11,161 trainable parameters, whereas the network for $E(x)$, which needs to capture more detail, is larger with 30,601 parameters. Both networks share a similar architecture, consisting of four fully connected layers.

Table 1.3: Architecture of Neural Network Model for $u(x)$

Layer	Size / Neurons	Activation Function	Notes
Input Layer	1	–	1 Input
Hidden Layer 1	60	$-sin(\pi tanh(x))$	Fully connected
Hidden Layer 2	60	$-sin(\pi tanh(x))$	Fully connected
Hidden Layer 3	60	$-sin(\pi tanh(x))$	Fully connected
Hidden Layer 4	60	$-sin(\pi tanh(x))$	Fully connected
Output Layer	1	$-sin(\pi tanh(x))$	1 Ouput

Table 1.4: Architecture of Neural Network Model for $E(x)$

Layer	Size / Neurons	Activation Function	Notes
Input Layer	1	–	1 Input
Hidden Layer 1	100	$-sin(\pi tanh(x))$	Fully connected
Hidden Layer 2	100	$-sin(\pi tanh(x))$	Fully connected
Hidden Layer 3	100	$-sin(\pi tanh(x))$	Fully connected
Hidden Layer 4	100	$-sin(\pi tanh(x))$	Fully connected
Output Layer	1	$-sin(\pi tanh(x))$	1 Ouput

The code link

https://github.com/jonaskobel/Projects—Deep-Learning-for-Partial-Differential-Equations-in-Engineering-Physics

1.2.3 Numerical Experiments

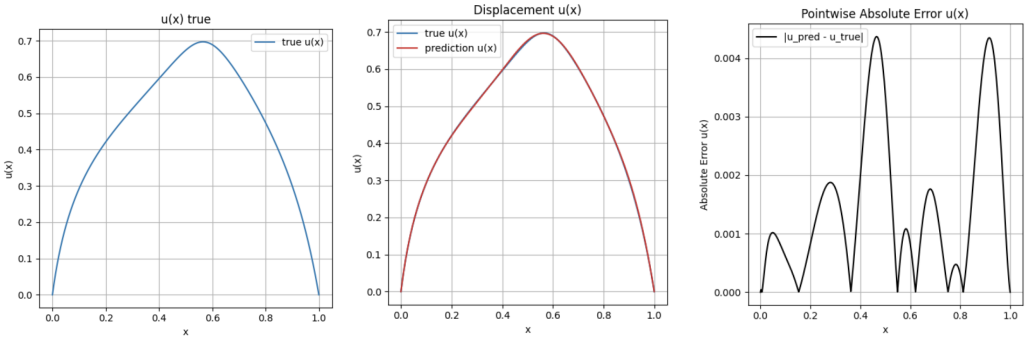
Experiment Setups

Numerical Results

The model was trained for 2000 epochs with a total runtime of 395.8 seconds. The final relative L^2 error is 3.48×10^{-2} for the prediction of Young modulus $E(x)$ and 3.78×10^{-3} for the prediction of the displacement $u(x)$, indicating high accuracy for both models. As shown in the absolute error plot, the predicted displacement matches closely the reference solution over the most part across the length with an absolute error less then 0.0085.

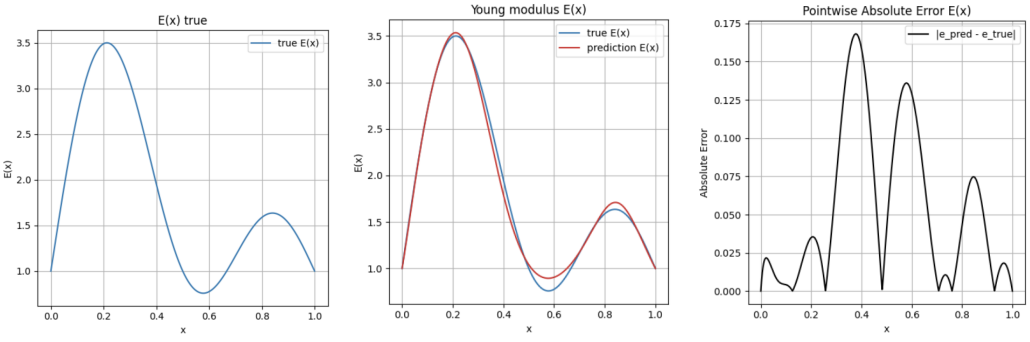
Table 1.5: Numerical Setup for Training the DNN in PyTorch

Parameter	**Value**
Optimizer	AdamW
Learning Rate (initial)	1×10^{-3}
Batch Size x_{in}	1000
Batch Size x_{obs}, u_{obs}	250
weight decay	1×10^{-4}
Number of Epochs	2000
Learning Rate Scheduler	StepLR
Decay Rate (α/N epochs)	0.5/250
Hardware	T4 GPU (Google Colab)
Precision	Float32



(a) True solution (b) Predicted solution (c) Pointwise absolute error

Figure 1.2: Experiment results of Task 2 for the Displacement u(x): (a); (b); (c).



(a) True solution (b) Predicted solution (c) Pointwise absolute error

Figure 1.3: Experiment results of Task 2 for the Young Modulus E(x): (a); (b); (c).

Chapter 2

Problem B

The second problem addresses the prediction of the effective thermal conductivity in a two-dimensional, two-phase composite material. This is a classical problem in materials science with practical relevance, where the goal is to estimate the bulk thermal behavior of a heterogeneous medium composed of two materials with differing conductivities. For this problem, a dataset of collected thermal conductivities $k(x)$ for two-phase regions is provided, along with a computed temperature field $T(x, y)$ solved by a classical FEM method. The aim of this task is to implement a model to make fast predictions of the temperature field $T(x, y)$ for the thermal conductivity $k(x, y)$ of two phases.

2.1 Problem Description

The physical behavior is governed by the steady-state heat equation, defined over a square domain $\Omega = [0, 1]^2$, and describes the temperature field $T(x, y) \in \mathbb{R}$ in fro the spatial distribution of thermal conductivity $k(x, y)$. The governing partial differential equation is given by:

$$-\nabla \cdot (k(x, y)\nabla T) = 0, \quad (x, y) \in \Omega \tag{2.1}$$

subject to mixed boundary conditions that fix the temperature at the left and right boundaries and enforce Neumann (no-flux) conditions at the top and bottom:

$$T(0, y) = 0, \quad T(1, y) = 1, \quad \frac{\partial T}{\partial n}(x, 0) = \frac{\partial T}{\partial n}(x, 1) = 0 \tag{2.2}$$

The material structure is modeled by a piecewise-constant conductivity field:

$$k(x, y) = \begin{cases} k_1 = 2.0, & (x, y) \in \Omega_1 \\ k_2 = 10.0, & (x, y) \in \Omega_2 \end{cases} \tag{2.3}$$

where Ω_1 and Ω_2 define the occupied region of the first phase and the second phase.
This setup introduces spatial heterogeneity that significantly impacts the heat flow through the material. The challenge lies in predicting the resulting temperature field $T(x, y)$ for two different phase regions of the material structure $k(x, y)$, sampled from a predefined distribution. Traditional numerical methods such as FEM offer high accuracy but are computationally expensive when applied to a large ensemble of material samples.
In this task, one method of physics-informed deep learning is explored to accurately and fast approximate the solution $T(x, y)$ from the input thermal conductivity map $k(x, y)$. The aim is to design and train a neural network model that accurately approximates the temperature

field for the given conductivity map and to evaluate its performance in terms of prediction accuracy, aswell as computational efficiency. The dataset `EffectiveConductivity.h5` provides training and testing samples, including the conductivity map and reference solutions computed with FEM. The performance of the model will be measured by computing the L^2 relative error over the test set for each epoch, as well as the absolute error compared to the test set for the final solution.

2.2 Methodology

The method selection

For this task, the Fourier Neural Operator (FNO) is chosen among the neural operator methods because it can efficiently learn mappings between function spaces of infinite dimension. FNO is especially good when both input and output are spatial fields, like the conductivity map $k(x, y)$ in and the temperature field $T(x, y)$ in this task. Unlike DeepONet, which represents functions by sampling them at scattered sensor points and needs positional encoding for input and output, FNO works directly on the whole input field. It uses spectral convolution layers that effectively capture global spatial patterns. This makes FNO a perfect fit for structured grid data, such as images, where it leverages the fast Fourier transform (FFT) to model long-range dependencies with less computational effort. Moreover, FNO has shown strong results and good scalability in related PDE surrogate modeling problems like Darcy flow or turbulence. Since the goal here is to predict the full temperature distribution across heterogeneous materials, FNO provides a good and efficient approach without requiring explicit pointwise coordinates or complex input encoding.[6]

The loss function

The loss function for Task B is based on data-driven supervision using paired input-output samples ($k(x)$, $T(x)$). The goal is to train a neural network model $T_\theta(k(x))$ that predicts the temperature distribution $T(x)$ for the given the thermal conductivity field $k(x)$.
The loss is defined as the average L^2-norm between the predicted and true temperature fields:

$$L_{\text{data}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \left\| T_\theta(k^{(i)}(x)) - T^{(i)}(x) \right\|_2 \tag{2.4}$$

where $T^{(i)}(x)$ is the ground-truth temperature corresponding to the i-th sample.
For evaluation, the relative L^2-error is computed as:

$$\text{Relative Error} = \sqrt{\frac{\| T_\theta(x) - T_{\text{true}}(x)\|_2^2}{\| T_{\text{true}}(x)\|_2^2}} \tag{2.5}$$

The network structure

The main component of the model is the *SpectralConv2d* layer, which performs convolution in the Fourier domain. The input is first transformed using the 2D real-valued FFT:

$$\hat{x} = \mathcal{F}[x]$$

Then, a complex-valued multiplication with learned weights W is applied to selected low-frequency modes:

$$\hat{y}_{\text{low}} = \hat{x}_{\text{low}} \cdot W_1, \quad \hat{y}_{\text{high}} = \hat{x}_{\text{high}} \cdot W_2$$

The modified spectrum is then transformed back using the inverse FFT:

$$y = \mathcal{F}^{-1}[\hat{y}]$$

Overall Architecture

The FNO2d model consists of:

- **Input layer:** A fully connected linear projection

$$\text{Linear}(d_{\text{in}} \to 40)$$

- **3 hidden layers:** Each consists of:
  - A *SpectralConv2d* layer with 8 retained Fourier modes in each spatial direction.
  - A pointwise 1×1 convolution (`Conv1d`) for residual connection.
  - ReLU activation after summing spectral and pointwise branches.
- **Output layers:**

$$\text{Linear}(40 \to 128) \to \text{ReLU} \to \text{Linear}(128 \to d_{\text{out}})$$

The model maps a 2D spatial input tensor of shape (N, H, W, d_{in}) to an output tensor of shape $(N, H, W, d_{\text{out}})$, where $H \times W$ is the spatial grid.

Fourier Modes and Parameters

The model uses:

- `modes1` $= 11$: first 11 Fourier modes in the vertical direction.
- `modes2` $= 1$: first 11 Fourier modes in the horizontal direction.
- Hidden dimensions: $[256, 256, 256]$

The total number of trainable parameters is:

$$47{,}810{,}561$$

The code link

https://github.com/jonaskobel/Projects—Deep-Learning-for-Partial-Differential-Equations-in-Engineering-Physics

2.3 Numerical Results

Experiment Setups

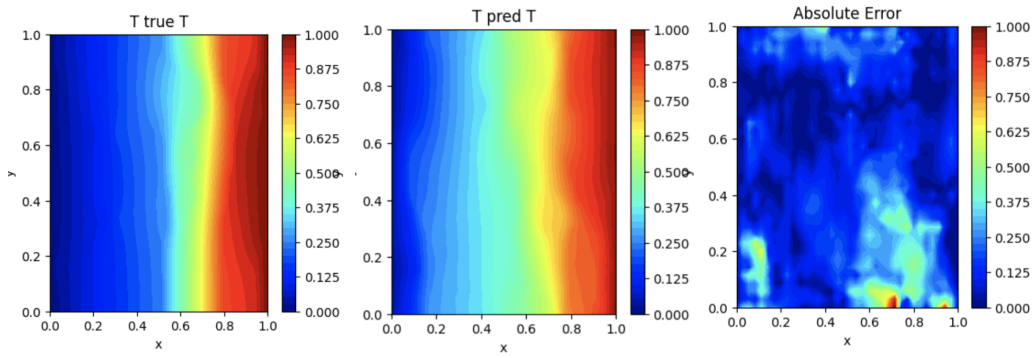
For the validation loop and the setup of the early stopping criteria, the training dataset $(kx_{\text{train}}, T_{\text{train}})$ is split into two parts: 800 samples are used for training, and the remaining 200 samples are used for validation, denoted as $(kx_{\text{val}}, T_{\text{val}})$, this combination worked best in the conducted tests.

Table 2.1: Numerical Setup for Training the DNN in PyTorch

Parameter	**Value**
Optimizer	AdamW
Learning Rate (initial)	1×10^{-3}
Batch Size (training)	50
Batch Size (validation)	50
weight decay	1×10^{-3}
Number of Epochs	2000
Learning Rate Scheduler	StepLR
Decay Rate (α/N epochs)	0.5/250
Hardware	2 x T4 GPU (Kaggle)
Precision	Float32
Early Stopping	Yes (patience = 30)

Numerical Results

The model was trained for 2000 epochs, with a total runtime of 1359 seconds. Early stopping halted the training at epoch 369, and the model from epoch 338 was loaded as the best-performing version. The final relative L^2 error is 0.0083.



(a) True solution (b) Predicted solution (c) Pointwise absolute error

Figure 2.1: Experiment results of Task 2 for the Young Modulus E(x): (a); (b); (c).

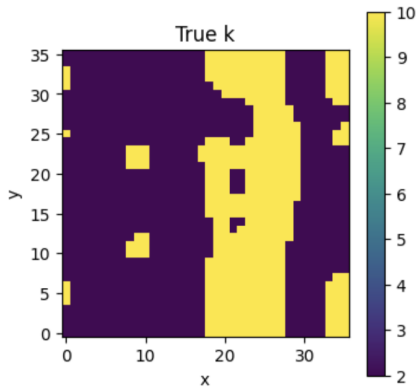


Figure 2.2: Two phase thermal conductivity map k(x,y)

Chapter 3

Problem C

This task focuses on modeling traffic flow using the Burgers' equation, a fundamental nonlinear partial differential equation (PDE) that incorporates both convective and diffusive effects. In the context of traffic dynamics, the equation captures the phenomena such as the deceleration of vehicles in response to traffic jams (convection) and the tendency of traffic to spread out due to individual driver behavior (diffusion). This task consider a time-dependent setting in which the evolution of the velocity field over time is governed by the initial traffic condition and a fixed diffusion coefficient.

3.1 Problem Description

The governing PDE for this problem is given by:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in (-1, 1),\ t \in (0, 1] \tag{3.1}$$

where $u(x, t)$ denotes the velocity of cars at position x and time t, and $\nu = 0.1$ is the diffusion coefficient reflecting driver caution. The system is restricted to homogeneous Dirichlet boundary conditions:

$$u(-1, t) = u(1, t) = 0, \quad t \in (0, 1] \tag{3.2}$$

and an initial condition $u(x, 0) = a(x)$, where $a(x)$ is sampled from a distribution $\mathcal{A}$.
Given a dataset of initial velocity profiles and corresponding time-evolved velocity fields computed by a classical finite difference method (FDM), the goal is to develop a deep learning model that can predict fast the entire space-time velocity field $u(x, t)$ based just on a new initial condition $a(x)$.
Unlike traditional FDM solvers, this approach explores a data-driven learning approach to approximate the solution operator of the Burgers' equation. The model must learn the complex nonlinear mapping from an initial condition to the full spatial and temporal solution, enabling a fast and accurate inference. The performance of the model will be measured by computing the relative L^2 error per epoch using test date along with the absolute L^2 error for the final solution . The results will be visualized for better interpretation.

3.2 Methodology

The method selection

The Fourier Neural Operator (FNO) is selected for this task due to its ability to efficiently learn mappings between function spaces, especially for problems governed by PDEs with structured

input-output domains. In contrast to DeepONet, which is designed to approximate general nonlinear operators by decoupling input functions and evaluation points by using a branch and trunk networks, FNO use the regular grid structure of the spatiotemporal data and use the fast Fourier transforms (FFTs) to perform global convolutions in a spectral a space. This makes it especially good for learning evolution dynamics on uniform grids, as present in the Burgers' equation dataset. 2025
Since both the input $a(x)$ and the output $u(x,t)$ are discretized on fixed grids $a(x)$ on a spatial mesh and $u(x,t)$ on a space-time mesh, FNO can treat the initial condition as a channel of the input tensor and directly model the evolution across time and space as a sequence-to-sequence mapping. Its architecture captures long range spatial dependencies and temporal correlations through spectral convolution layers, enabling efficient learning of nonlinear dynamics with less layers and parameters than other models.[6] [1]
Moreover, FNO shows a strong empirical performance and scalability when it is used on structured PDE problems with low or moderate input complexity and rich spatial-temporal interactions. In this context, it offers a more direct modeling approach than DeepONet, which would require sampling across all spacetime points and separate evaluations at each, likely increasing the computational load during inference. The ability of FNO's to output the entire velocity field in one forward pass makes it a practical and good choice for real-time predictions in traffic flow modeling.

The loss function

Similar to Task B, Task C employs a data-driven loss term that measures the discrepancy between the model-predicted velocity field $\hat{u}(x,t)$ and the ground truth velocity field $u(x,t)$. Here, the model takes the initial velocity profile $a(x) = u(x,0)$ as input.
The loss function is defined as the mean ℓ^2-norm of the difference between predicted and true fields over all training samples:

$$\mathcal{L}_{\text{data}} = \frac{1}{N}\sum_{i=1}^{N}\left\|u^{(i)} - \hat{u}^{(i)}\right\|_2,$$

where $u^{(i)}$ denotes the true velocity field and $\hat{u}^{(i)}$ the model prediction for the i-th sample. The prediction $\hat{u}$ is obtained by passing the input $a(x)$ through the trained model $\hat{u} = \texttt{u\_model}(a)$. The loss is then averaged over the batch.

The network structure

For Problem C, we employ the same *FNO2d* architecture described in the previous section, but adapt it to a different input–output setting. Here, the model learns the mapping from the initial velocity field $a(x) = u(x,0)$ to the full spatiotemporal velocity field $u(x,t)$ governed by the 1D Burgers' equation.
The model configuration uses the same Fourier resolution with $\texttt{modes1} = \texttt{modes2} = 8$, and a hidden size configuration of $[45, 45, 45]$. The input features have dimensionality $d_{\text{in}} = \texttt{ax\_train.shape[-1]}$, which encodes initial or boundary information, while the output has $d_{\text{out}} = \texttt{u\_train.shape[-1]}$, representing the time evolution of the velocity field.

The model contains a total of:

$$790{,}007 \text{ trainable parameters}$$

and is trained to approximate the nonlinear solution operator of the Burgers' equation from initial conditions.

The code link

https://github.com/jonaskobel/Projects—Deep-Learning-for-Partial-Differential-Equations-in-Engineering-Physics.

3.3 Numerical Results

Experiment Setups

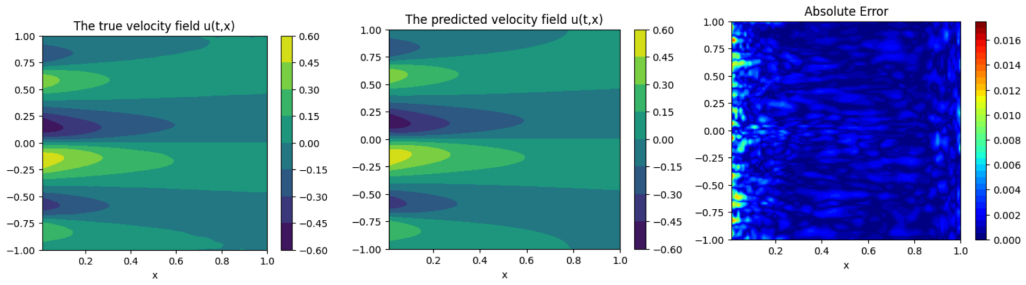
For the validation loop and the setup of the early stopping criteria, the training dataset $(ux_{\text{train}}, T_{\text{train}})$ is split into two parts: 800 samples are used for training, and the remaining 200 samples are used for validation, denoted as $(kx_{\text{val}}, T_{\text{val}})$, this combination worked best in the conducted tests.

Table 3.1: Numerical Setup for Training the DNN in PyTorch

Parameter	**Value**
Optimizer	Adam
Learning Rate (initial)	1×10^{-3}
Batch Size (training)	50
Batch Size (validation)	50
weight decay	1×10^{-3}
Number of Epochs	2000
Learning Rate Scheduler	StepLR
Decay Rate (α/N epochs)	0.5/250
Hardware	2 x T4 GPU (Kaggle)
Precision	Float32
Early Stopping	Yes (patience = 20)

Numerical Results

The model was trained for 2000 epochs, with a total runtime of 2115 seconds. Early stopping halted the training at epoch 671, and the model from epoch 338 was loaded as the best-performing version. The final relative L^2 error is 0.0094.
The prediction shows good results overall, with only minor discrepancies observed near the boundary at $x = 0$



(a) True solution (b) Predicted solution (c) Pointwise absolute error

Figure 3.1: Experiment results of Task 2: (a); (b); (c).

Chapter 4

Conclusions

In this project, different neural operator approaches were applied to solve PDE-governed problems, each with unique characteristics. For the linear elasticity inverse problem (Task B), DeepONet proved effective due to its flexibility in learning mappings from functions to scalar values or other functions, making it suitable for parameter identification tasks. For the traffic flow prediction task (Task C), the Fourier Neural Operator (FNO) was the better choice because of its ability to directly model mappings between discretized function spaces on regular grids. FNO's use of spectral convolutions allowed efficient learning of the nonlinear Burgers' dynamics, producing fast and accurate full-field predictions from initial conditions.

Throughout these tasks, it became clear that deep learning methods for PDEs offer significant advantages in terms of speed at inference time and the ability to learn complex solution operators from data. However, they also require careful tuning of architecture and hyperparameters and can be sensitive to data distribution and resolution. Unlike classical solvers, they do not guarantee convergence or stability and can sometimes fail to generalize to unseen inputs if training data is not sufficiently representative.

Possible future improvements include conducting a more extensive hyperparameter search using methods such as Grid Search or Randomized Search to further increase accuracy, but this would increase the computational costs enormously. Also Further work could also look into applying these models to higher-dimensional problems or real-world applications where classical solvers are computationally expensive.

References

[1] Z. L. M. L.-S. J. K. A. A. Kamyar Azizzadenesheli, Nikola Kovachki. Neural operators for accelerating scientific simulations and design. *publised as conference paper at ICLR*, 2020.

[2] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[3] B. Y. Weinan E. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *School of Mathematical Sciences, University of Science and Technology of China and Springer-Verlag GmbH Germany, part of Springer Nature 2018*, 2018.

[4] V. S. . D. Y. Zang. Lecture: The deep ritz methode. *Deep Learning-Based Numerical Algorithm for Solving Variational Problems*, 2025.

[5] V. S. . D. Y. Zang. Lecture: Fourier neoral operator for parametric partial differnatial equations. *Deep Learning-Based for partial differential equations in engineering physics*, 2025.

[6] K. A. B. L.-K. B. A. S. A. A. Zongyi Li, Nikola Kovachki. Fourier neoral operator for parametric partial differential equations. *publised as conference paper at ICLR*, 2021.