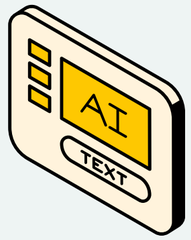
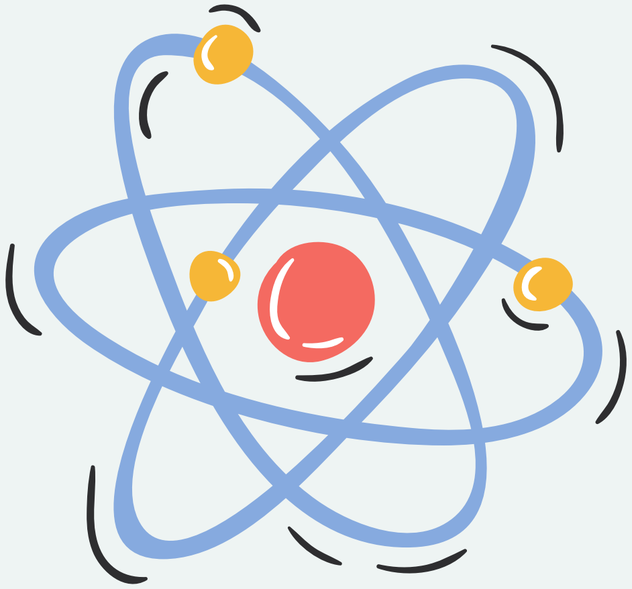




HEAT EQUATION

WITH PINNS & INVERSE METHOD

PRESENTED BY:

SURAJ CHOTHE (B22CI041)
AADITYA KAMBLE (B22MT024)

PRESENTATION OUTLINE

- Introduction
- Heat Equation
- Inverse Method
- Neural Networks
- Physics Informed Neural Network
- Exploratrary Data Analysis
- Setup
- Model & Training
- Results

INVERSE METHOD

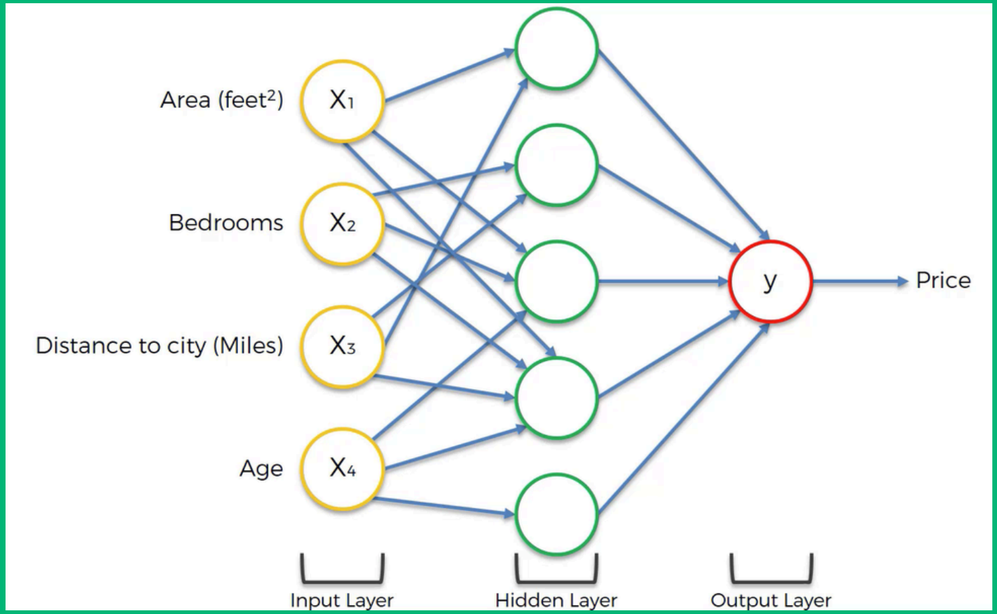
The Inverse Method refers to solving inverse problems, starting from observable outcomes (like temperature over time) and working backwards to find the causes or system parameters (like thermal conductivity k and specific heat capacity Cp).

$$\rho C_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right)$$

For the above equation,

In forward problems, the material properties like thermal conductivity (k), specific heat (Cp), and density (ρ\rho) are known, and we solve for the temperature distribution T(x,t). In inverse problems, we already have temperature data, and we aim to estimate unknown properties such as kk and Cp. This is treated as an optimization problem, where we adjust kk and Cp so that the predicted temperature matches the measured data while satisfying the heat equation.

NEURAL NETWORKS



Neural networks are a type of machine learning algorithm inspired by the structure and function of the human brain

They use interconnected nodes, or neurons, to process data and learn patterns, allowing them to perform tasks like image recognition, natural language processing, and prediction.

Each neuron performs a weighted sum of its inputs, passes it through an activation function, and sends it forward.

NEURAL NETWORKS

STRUCTURE

Hidden Layers

- One or more layers between input and output.
- Each neuron performs a weighted sum of inputs and applies an activation function (like ReLU, sigmoid, tanh).
- These layers extract patterns and features from the input data.
- The more layers (deep networks), the more complex relationships it can learn

Input Layer

- Takes in raw input features (e.g., temperature, time).
- One neuron per input variable.

Output Layer

- Produces the final prediction (e.g., temperature, material property).
- The number of neurons depends on the output format (single value, vector, classification).

PHYSICS INFORMED NEURAL NETWORKS

- Physics-informed neural networks (PINNs) are a type of neural network that incorporate physical laws, often expressed as differential equations, into their learning process.
- This approach helps guide the network towards solutions consistent with the underlying physics. PINNs solve problems involving partial differential equations by training a neural network to minimize a loss function that includes terms reflecting the initial and boundary conditions and the PDE residual.
- PINNs can be trained with minimal or no labelled data, making them suitable for problems where labelled data is scarce.
- PINNs can solve PDEs without the need for traditional mesh-based discretization techniques, making them computationally efficient.

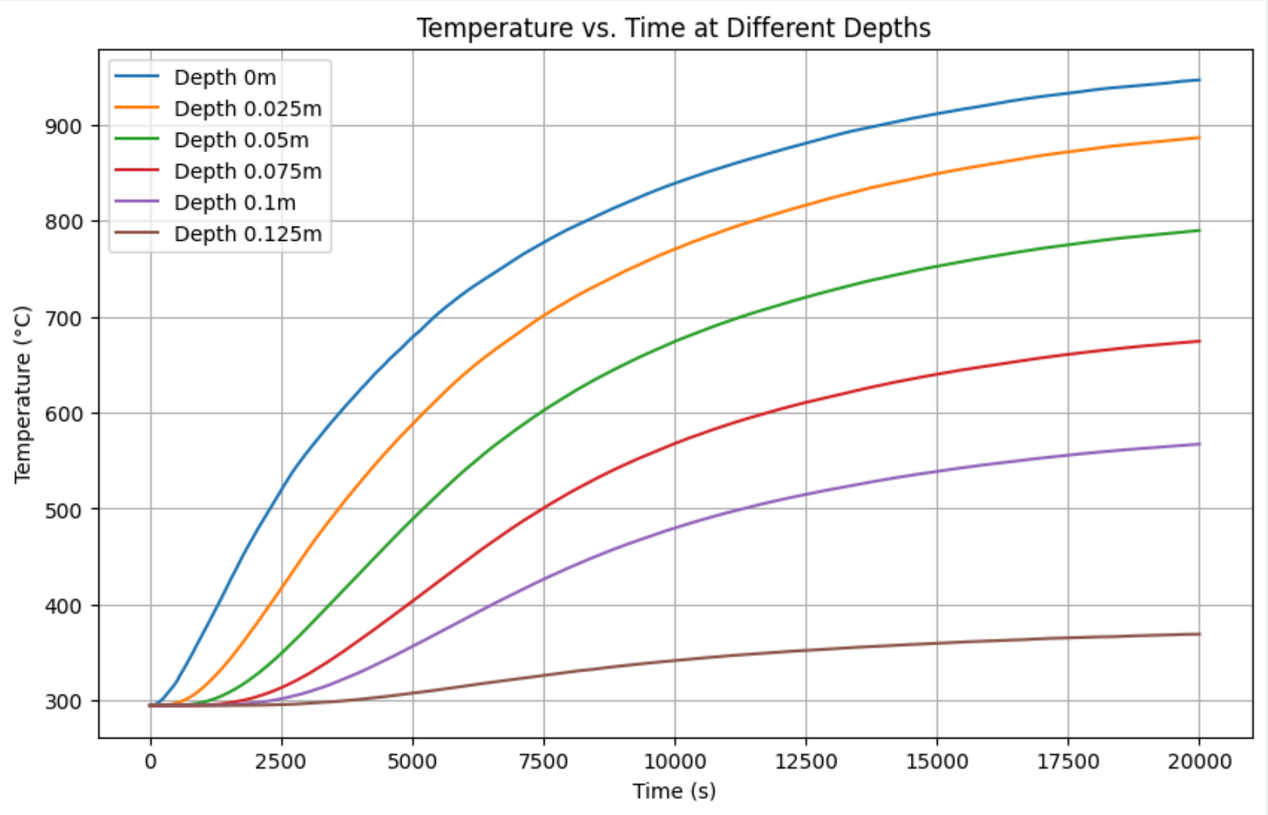
PHYSICS INFORMED NEURAL NETWORKS

WORKING

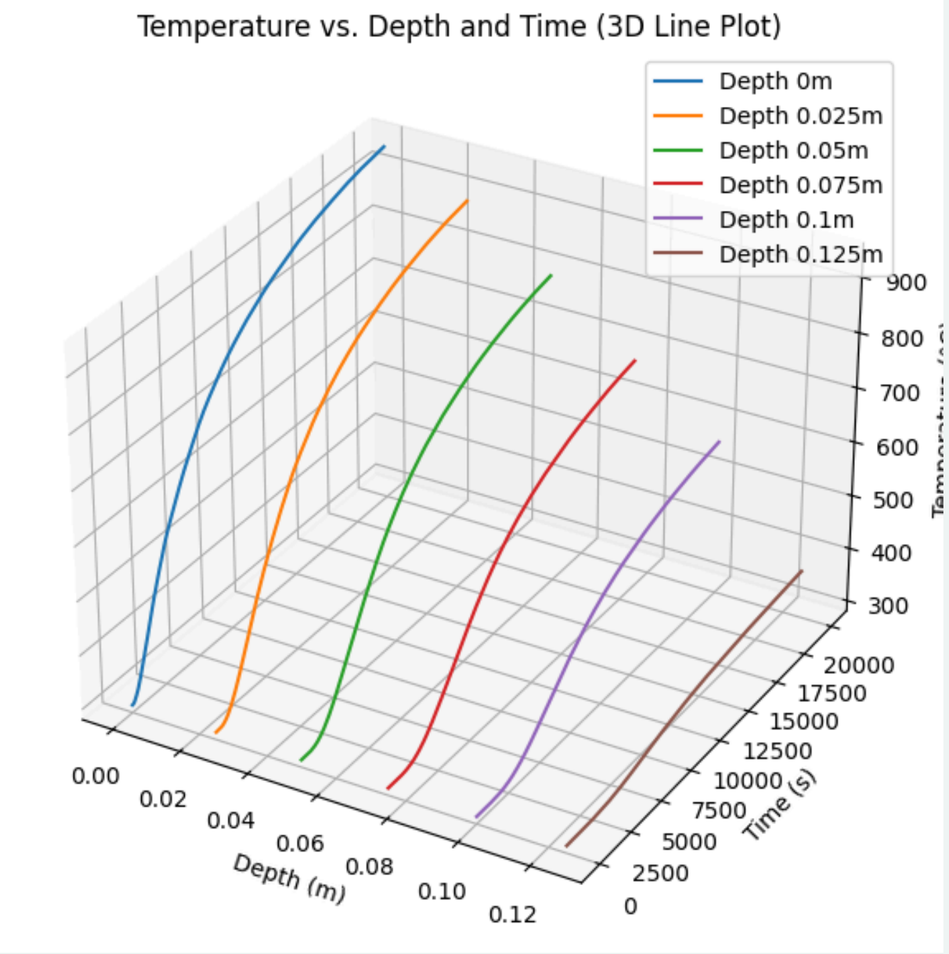
1. Neural Network Architecture:
   - A standard neural network is chosen, potentially with specialized layers for encoding physical information.
2. Loss Function Design:
   - A loss function is defined that includes terms representing:
     - The network's predictions at specific points (collocation points).
     - The network's predictions on the boundaries of the domain (boundary conditions).
     - The residual of the PDE at collocation points (representing how well the network satisfies the differential equation).
3. Training:
   - The neural network is trained by minimizing the loss function using optimization algorithms (e.g., gradient descent).
4. Solution Approximation:
   - Once trained, the network can approximate the solution of the PDE at any point within the domain.

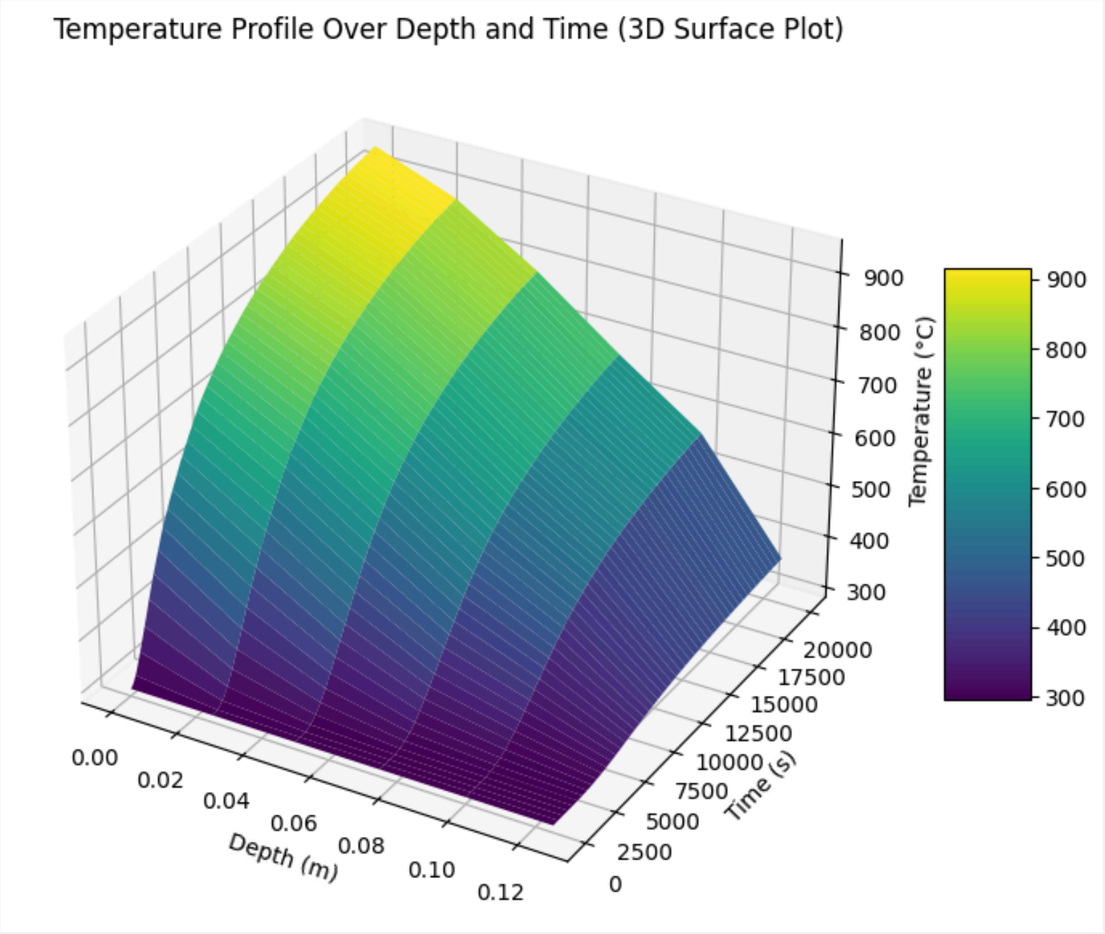
EXPLORATORY DATA ANALYSIS

- Checked basic statistics (mean, median, min, max, std dev).
- Checked for missing (null) or infinite values (found none).
- Visualized data distributions (histograms).
- Plotted Temperature vs. Time for each depth (shows heating/cooling trends).
- Created 3D plots (Temperature vs. Depth vs. Time) to visualize the overall thermal profile.
- removed the outliers and plotted Tvs t



EXPLORATORY DATA ANALYSIS





SETUP

- Libraries used:
  - NumPy (numerics),
  - Pandas (data handling),
  - DeepXDE (PINN framework),
  - TensorFlow (backend),
  - Matplotlib (plotting)
- Equation parameters were set as variables.

```
# Scaling constants based on domain size
L = x_max - x_min  # Spatial range (0.125 m)
T_time = t_max - t_min # Temporal range (~20000 s)
Cp_scaled_init = 50000*((L**2)/T_time) # An initial guess/scaling


# Define k and Cp as *trainable* variables
k = dde.Variable(0.8) # Initial guess for k
Cp = dde.Variable(Cp_scaled_init) # Initial guess for Cp
```

SETUP

EQUATION

```
# The Heat Equation: Cp * ∂T/∂t - k * ∂²T/∂x² = 0
def pde(x, y):
    T_val = y[:, 0:1] # Predicted Temperature T(x,t)
    dT_t = dde.grad.jacobian(y, x, i=0, j=1) # Computes ∂T/∂t
    dT_xx = dde.grad.hessian(y, x, component=0, i=0, j=0) # Computes ∂²T/∂x²
    return Cp * dT_t - k * dT_xx # The residual - should be close to 0
```

Initial Condition (IC):

-
```
ic1 = dde.icbc.IC(geomtime, fun_init, lambda _, on_initial: on_initial)
```

- Specifies the temperature at time t=0.
- T0 = 294.6390 (Kelvin).
- Temperature is also normalized (T0_scaled) to be consistent with the NN output range (typically [0,1] or [−1,1]).

SETUP

Experimental Data Integration:

```
observe_u = dde.icbc.PointSetBC(X_normalized, observe_T_normalized)
```

- Combines all measured (depth, time) pairs into an array X.
- Normalizes X to X_normalized (coordinates in the [0, 1] domain).
- Flattens all corresponding temperature measurements into observe_T.
- Normalizes observe_T to observe_T_normalized.
- Key Step: Uses PointSetBC to enforce the data constraint:

MODEL & TRAINING

Neural Network Architecture

- Uses a Feedforward Neural Network (FNN or CustomFNN with Dropout).
- layer_size = [2, 8, 16, 16, 8, 4, 1]: Input (x, t) -> Hidden Layers -> Output (T).
- activation = "tanh": Activation function for hidden layers.
- initializer = "Glorot normal": Method to set initial network weights.

```
data = dde.data.TimePDE(
    geomtime, pde, [ic1, observe_u], # Geometry, PDE, IC, Data BC
    num_domain=1000, num_initial=100, # Sampling points for PDE/IC loss
    anchors=X_normalized, # Ensure training includes experimental points
    num_test=40000 # Points for evaluating test loss
)
```

Model Compilation:

```
model = dde.Model(data, net)
model.compile("adam", lr=0.01,
              external_trainable_variables=[Cp, k]) # CRUCIAL!
```

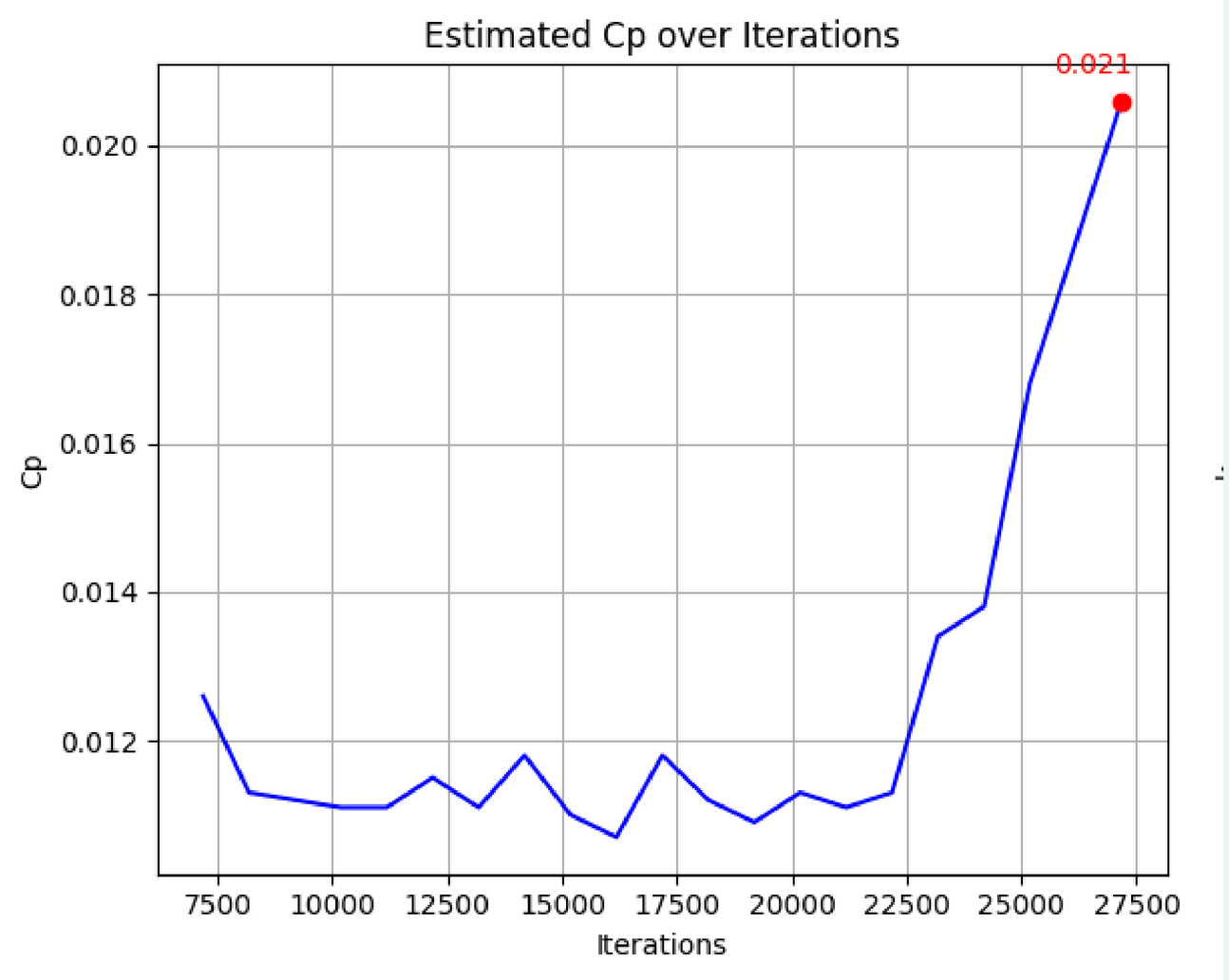
- Optimizer (adam), Learning Rate (lr).
- external_trainable_variables: Tells the optimizer to adjust Cp and k alongside the network weights to minimize the total loss.

Traning:

```
variable = dde.callbacks.VariableValue([Cp,k], period=1000, filename="variables.dat")
losshistory, train_state = model.train(iterations=25000, callbacks=[variable])
```

- Runs the optimization process for 25,000 iterations.
- VariableValue callback saves the estimated values of Cp and k every 1000 iterations to variables.dat.

RESULTS

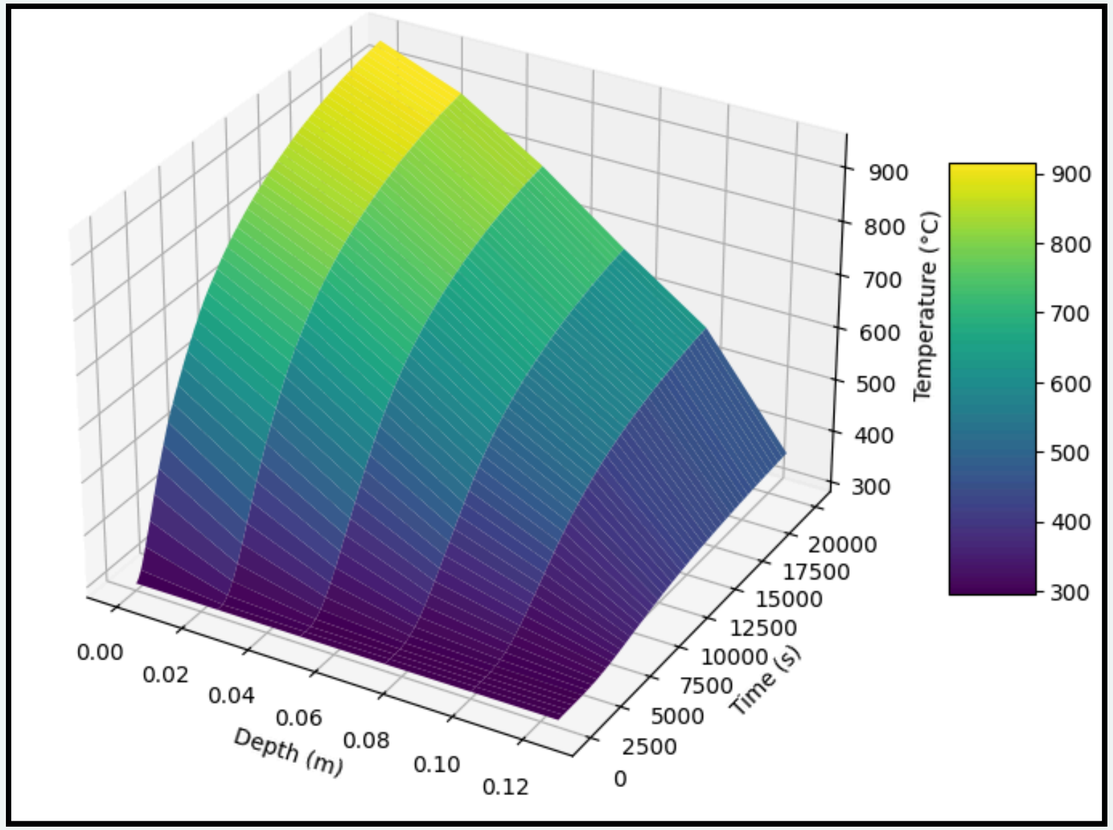



```
Cp at iteration 27183: [2.06e-02,
k at iteration 27183: 8.00e-01]
```

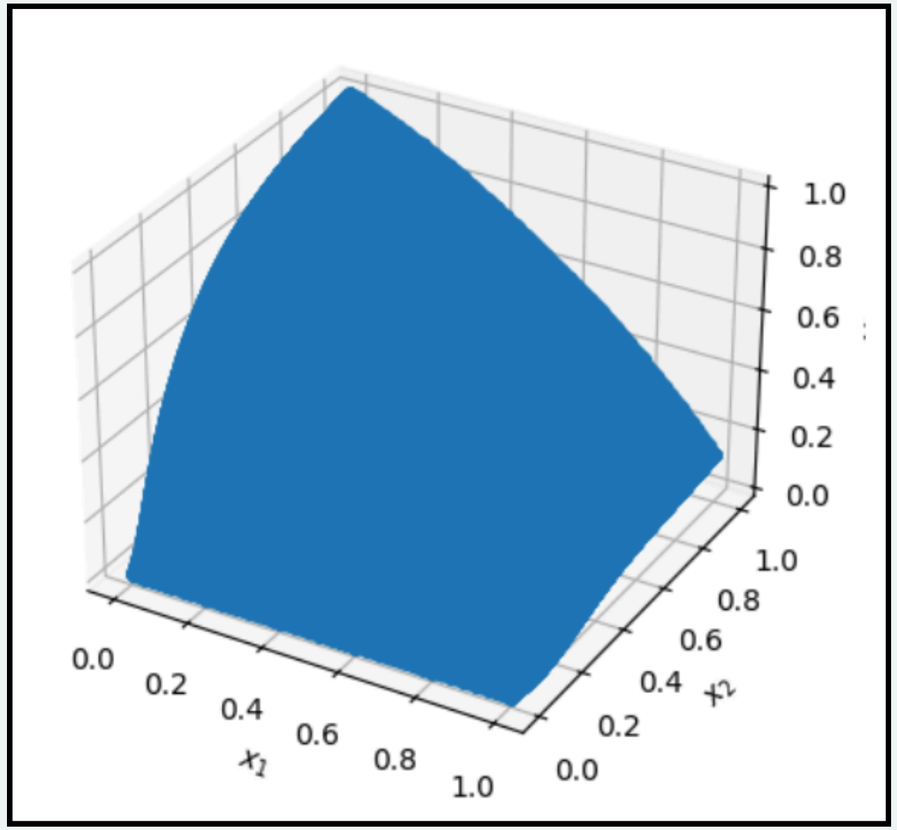
```
Cp_estimated = 0.0206*(T_time/(L**2))
print(Cp_estimated)

26368.006763392
```

Note: k is Not Scaled so No need to unscale the result

RESULTS

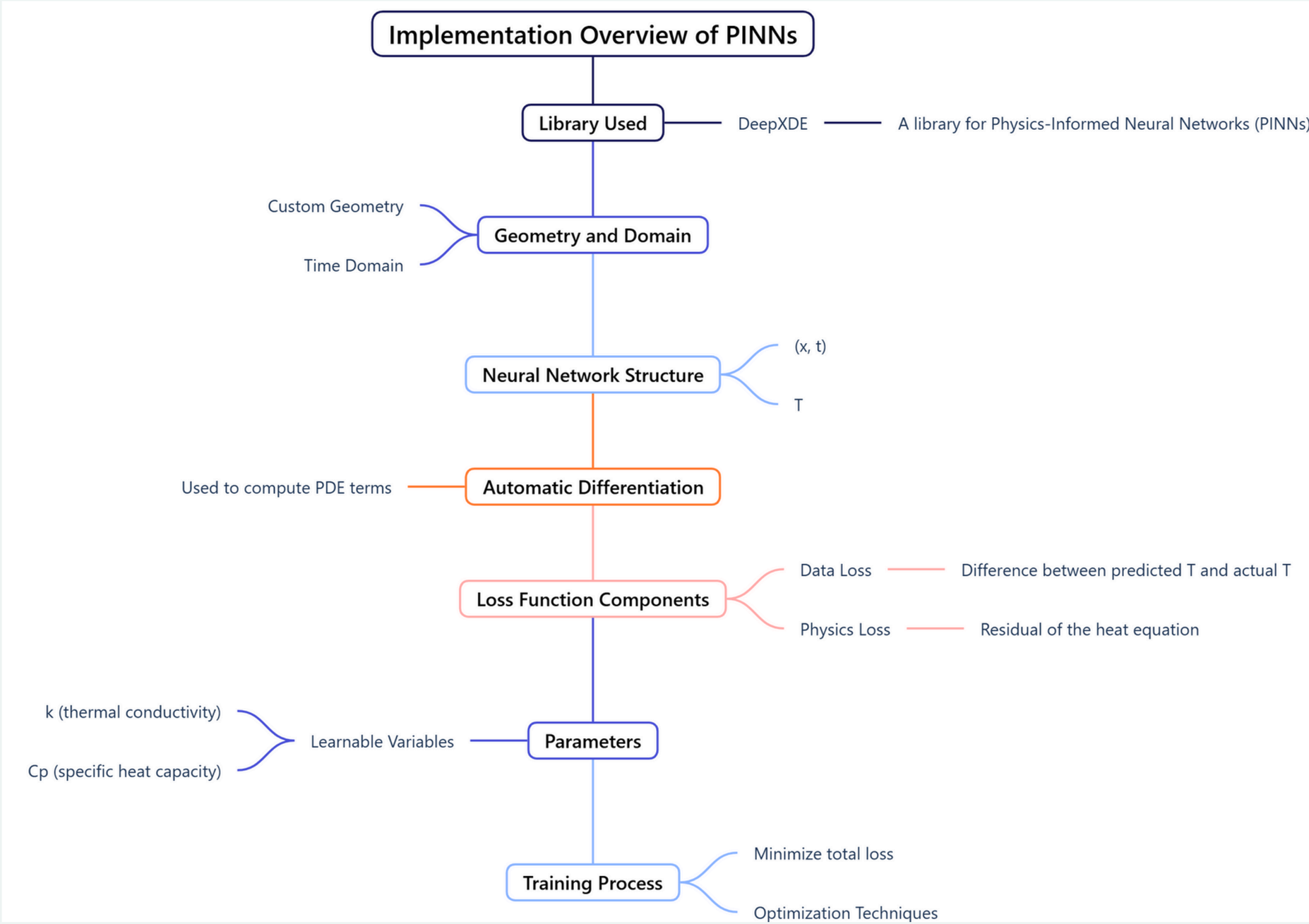


Actual



Predicted

L2 Relative Error for for this model is; 0.0250, which is 2.5%


Implementation Overview of PINNs
Library Used
DeepXDE
A library for Physics-Informed Neural Networks (PINNs)
Custom Geometry
Time Domain
Geometry and Domain
Neural Network Structure
(x, t)
T
Used to compute PDE terms
Automatic Differentiation
Loss Function Components
Data Loss
Difference between predicted T and actual T
Physics Loss
Residual of the heat equation
k (thermal conductivity)
Cp (specific heat capacity)
Learnable Variables
Parameters
Training Process
Minimize total loss
Optimization Techniques

THANK YOU

Google Colab
Github