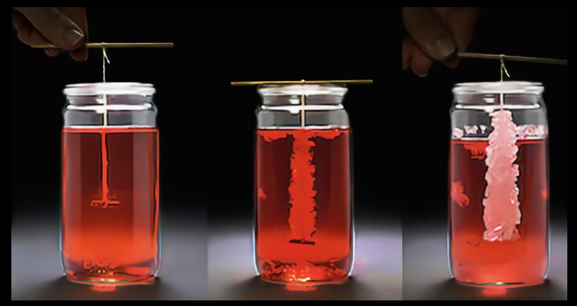
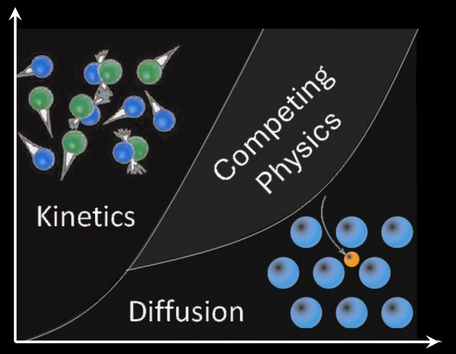
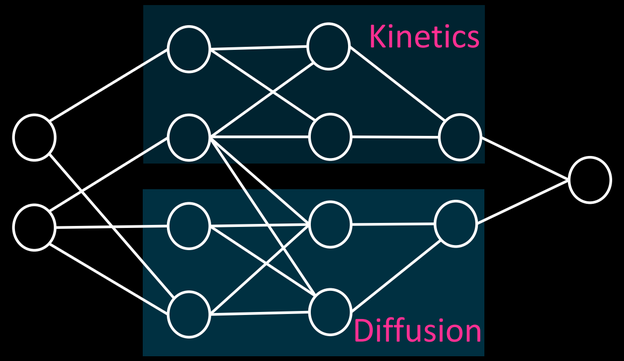
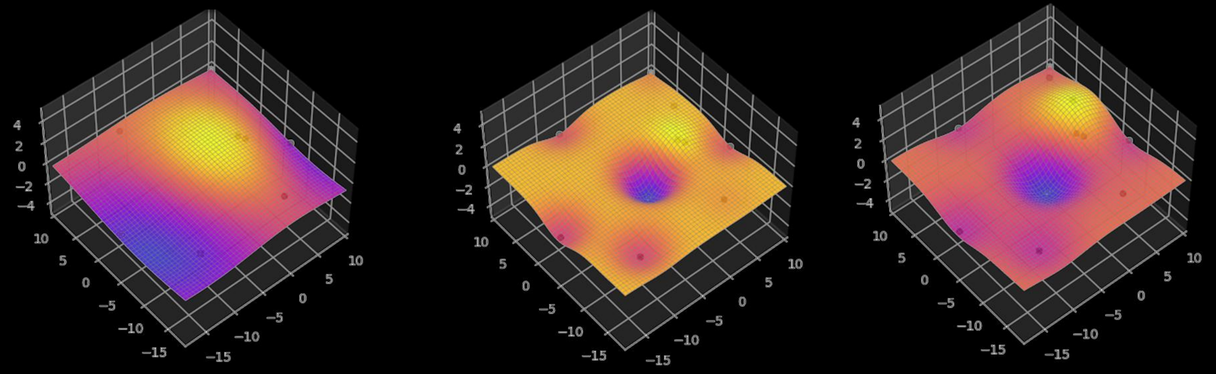
Physics Informed Machine Learning Heat Transfer







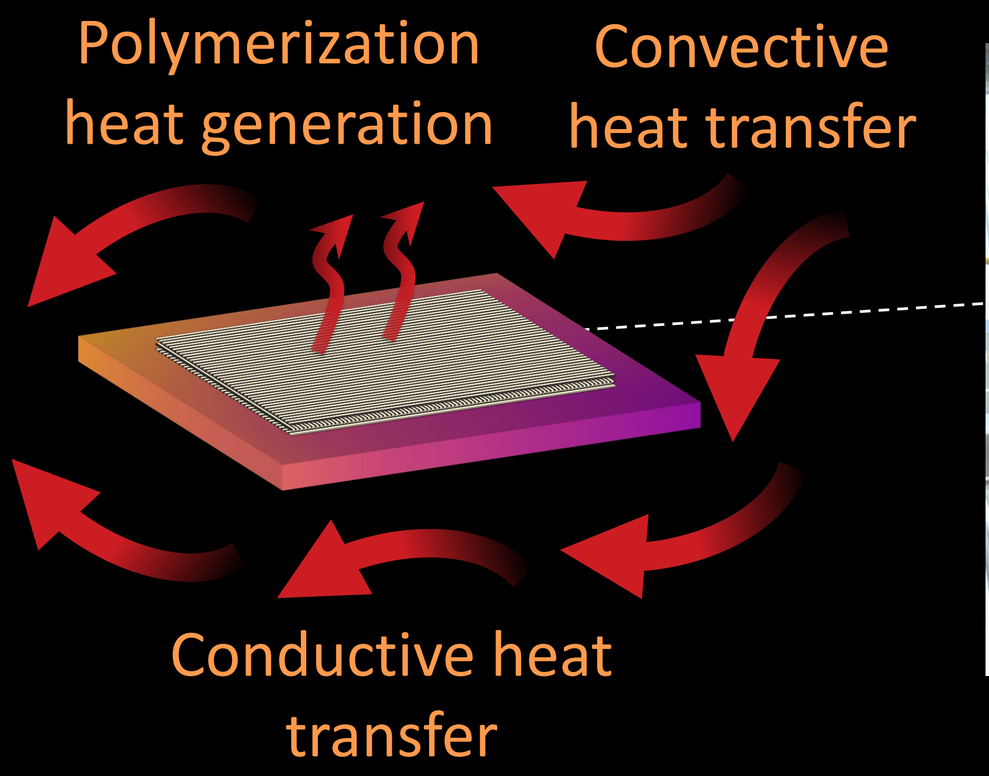


Navid Zobeiry, Associate Professor
Email: navidz@uw.edu
Materials Science & Engineering
https://composites.uw.edu/AI/

UNIVERSITY of WASHINGTON

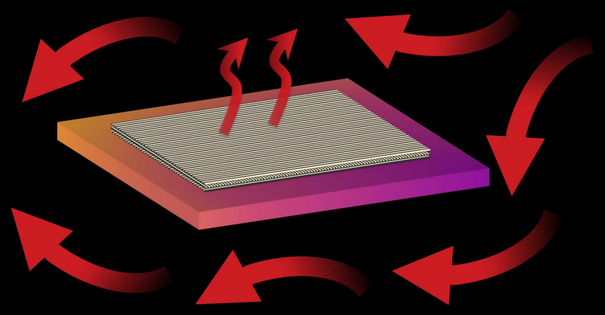
Background

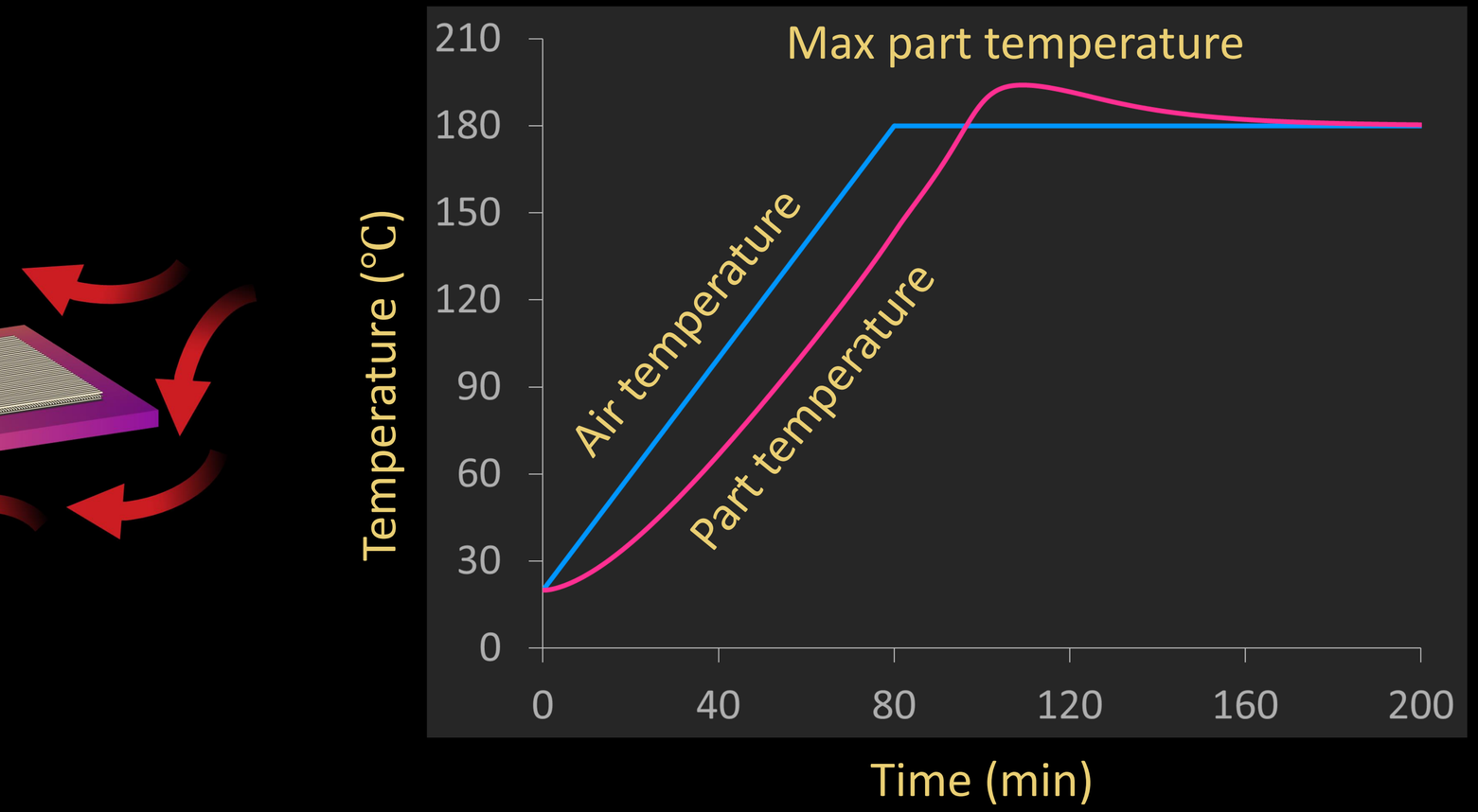
> Consider an epoxy-based part curing on a steel tool, subjected to a temperature cycle in a convection oven. Three competing physical phenomena occur:



Background

- Initially, the part lags behind the air temperature. As polymerization peaks, the part becomes hotter than the air.





Dataset

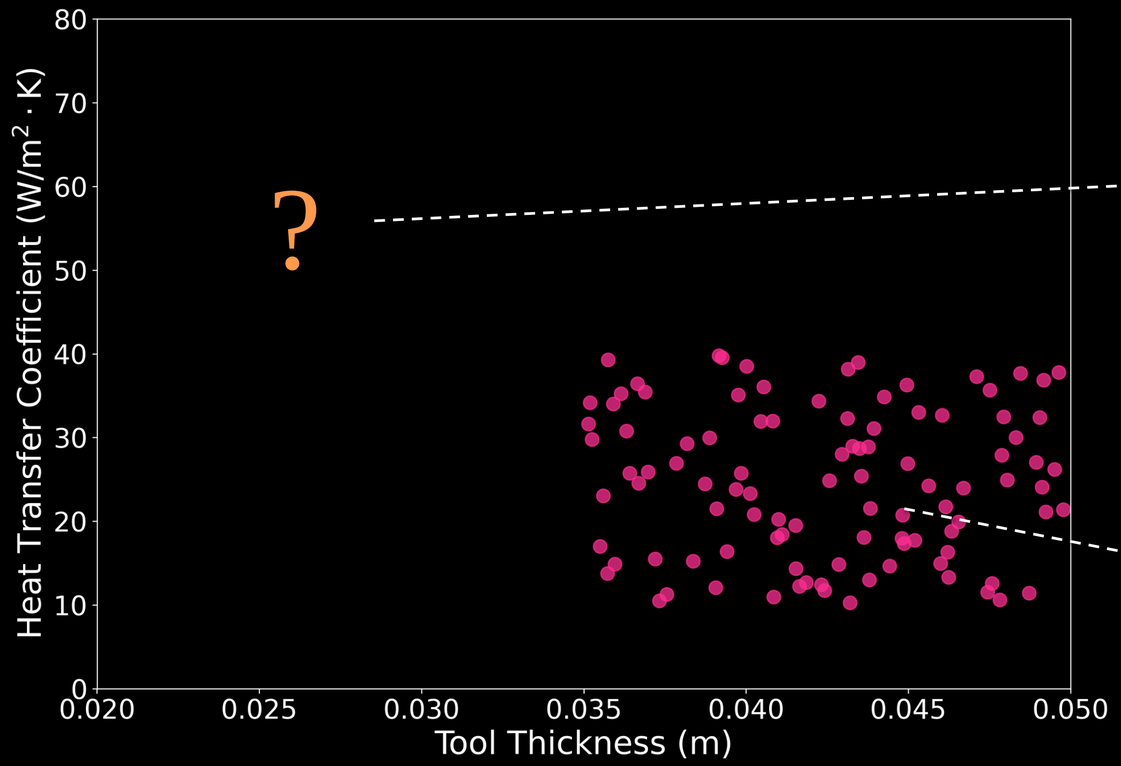
- 100 tests are conducted with two variables:
  - Tool thickness (L)
  - Airflow velocity (correlated with convective heat transfer coefficient, h)
- Test Output:
  - Maximum part temperature (T_{max})

	A	B	C
1	heat_transfer_coefficient	tool_thickness	max_temperature
2	W/m^2.K	m	°C
3	13.33	0.046	161.3
4	32.28	0.043	178.9
5	23.81	0.040	177.5
6	16.31	0.046	167.7
7	19.94	0.047	172.2
8	26.21	0.050	175.5
9	36.45	0.037	179.9
10	20.25	0.041	175.1
11	12.57	0.048	157.7
12	21.13	0.049	172.0
13	11.27	0.038	163.8
14	24.08	0.049	174.4

heat/data.csv

Problem Statement

- **100 data points are provided for specific ranges. Our goal is to train a ML model and generalize it to areas beyond the provided training data.**



beyond training zone

heat/data_all.csv

100 training data

heat/data.csv

Machine Learning Approaches

> **We will start with a neural network and progressively introduce physics-informed methods:**

1.	**Traditional ML with a Neural Network**	heat/heat_1.py
2.	**Physics-Informed Features**	heat/heat_2.py
3.	**Physics-Informed Loss**	heat/heat_3.py
4.	**Physics-Informed Domain Transformation**	heat/heat_4.py
5.	**Physics-Informed Neural Network**	Not provided

Python Implementation

> We use two standard Python libraries to train the neural network: TensorFlow and scikit-learn (sklearn).

```
8   # Import Libraries
9   import warnings
10  warnings.filterwarnings("ignore")
11  import os
12  os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
13  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
14  import numpy as np
15  import pandas as pd
16  import tensorflow as tf
17  from tensorflow.keras.models import Sequential
18  from tensorflow.keras.layers import Dense, Input
19  from tensorflow.keras.optimizers import Adamax
20  from sklearn.model_selection import train_test_split
21  from sklearn.metrics import r2_score
22  from sklearn.preprocessing import StandardScaler
23  import matplotlib.pyplot as plt
24  from mpl_toolkits.mplot3d import Axes3D
25
```

Python Implementation

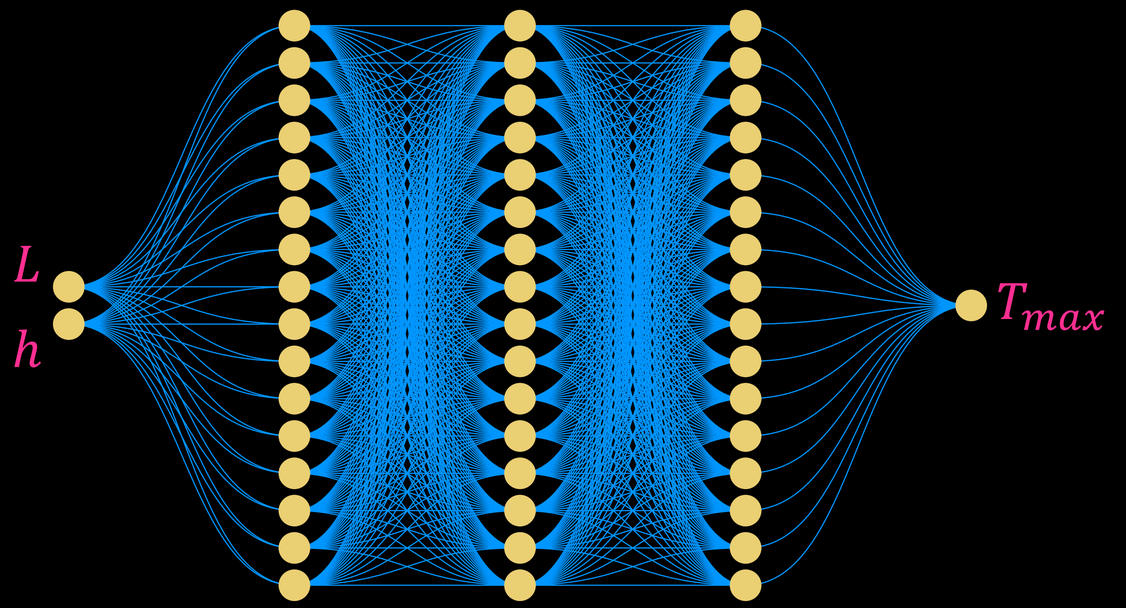
- Load the data from the CSV file into a pandas DataFrame. Scale the data, then split it into 75% for training and 25% for testing.

```
30  # Load the data
31  data_file = 'data.csv'
32  df = pd.read_csv(data_file)
33
34  X_columns = ['tool_thickness', 'heat_transfer_coefficient']
35  Y_column = 'max_temperature'
36
```

```
61  # Extract features (X) and target variable (Y)
62  X = df[X_columns]
63  Y = df[Y_column].values
64
65  # Standardize the data
66  scaler_X = StandardScaler()
67  X_scaled = scaler_X.fit_transform(X)
68
69  scaler_Y = StandardScaler()
70  Y_scaled = scaler_Y.fit_transform(Y.reshape(-1, 1)).flatten()
71
72  # Split the main data into training and testing sets
73  X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y_scaled, test_size=0.25, random_state=42)
```

Approach 1: traditional ML

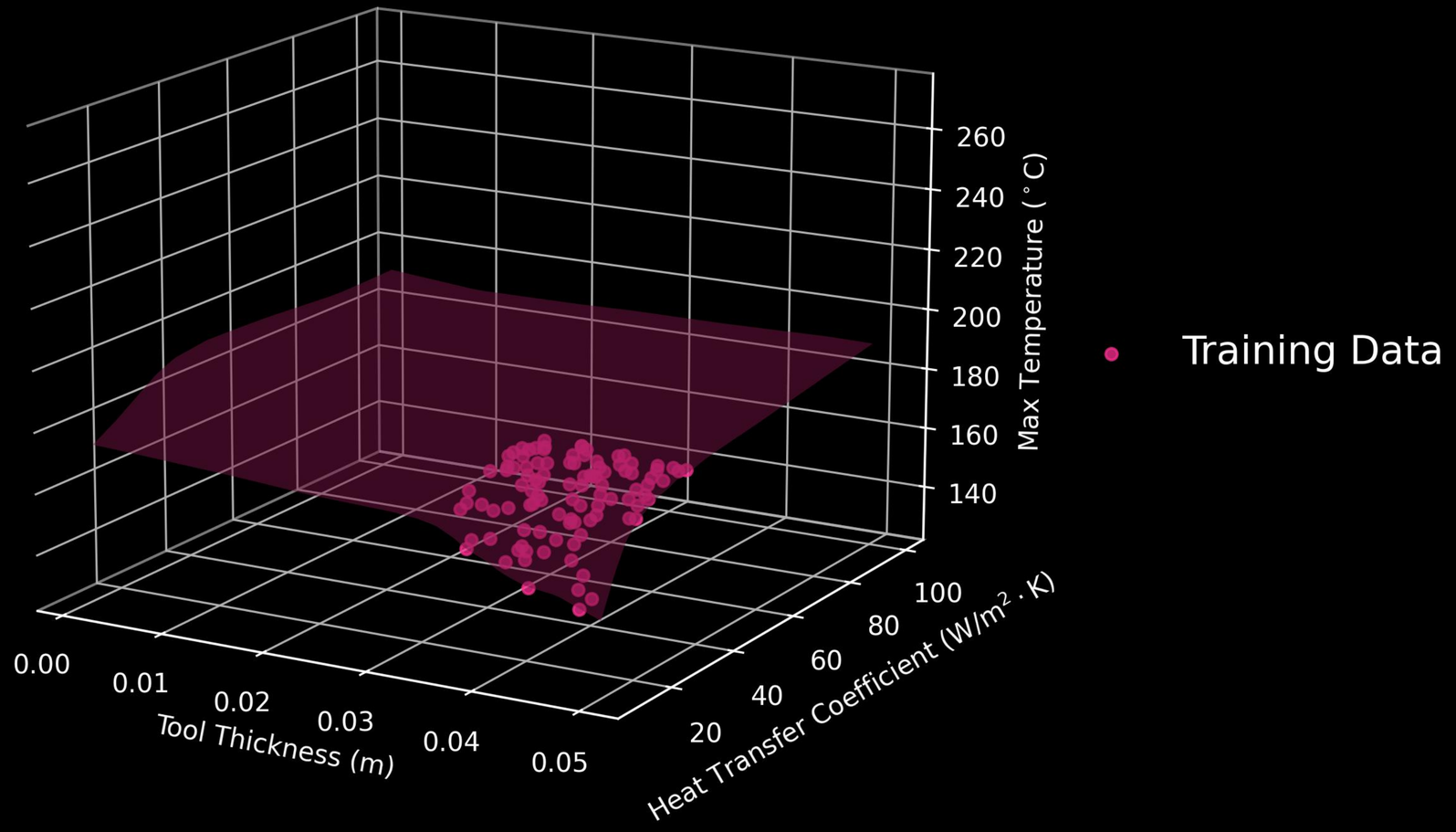
- A neural network is built and trained.
  - 3 layers × 16 nodes
  - ReLU activation
  - Adamax optimizer
  - MSE loss




```python
def build_and_train_nn(X_train, Y_train, X_test, Y_test):
    # Build a neural network model
    model = Sequential([
        Input(shape=(X_train.shape[1],)),
        *[Dense(16, activation='relu') for _ in range(3)],  # Three hidden layers
        Dense(1)
    ])

    model.compile(optimizer=Adamax(learning_rate=0.01), loss='mse')
    model.fit(X_train, Y_train, epochs=100, batch_size=16, verbose=0, validation_data=(X_test, Y_test))

    # Evaluate the model
    Y_pred_test = model.predict(X_test)
    r2_test = round(r2_score(Y_test, Y_pred_test) * 100, 1)
    return model, r2_test
```

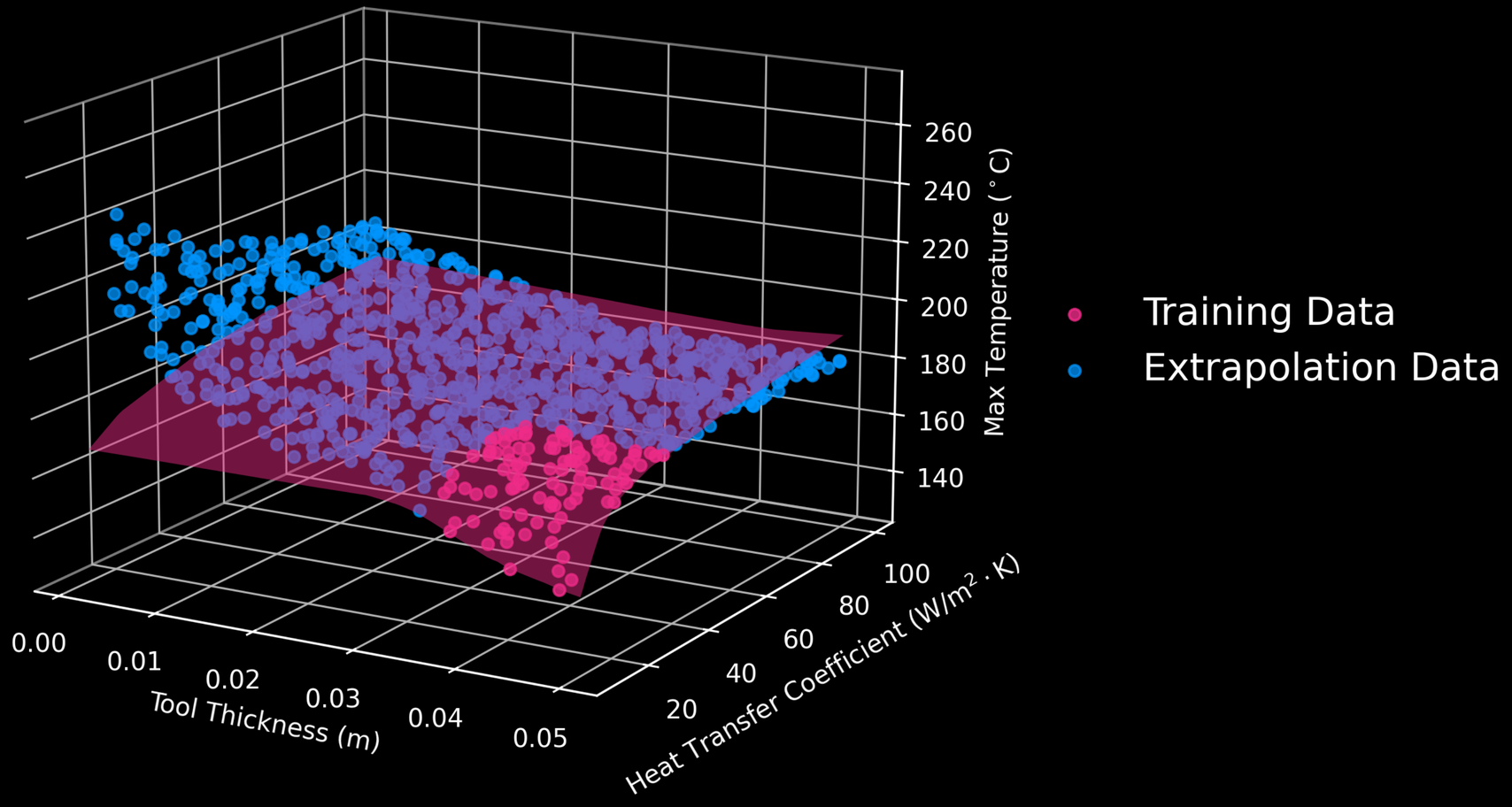
Approach 1: Training

> **98.6% accuracy on test data. We predict linear extrapolation beyond the training boundary.**



Approach 1: Extrapolation

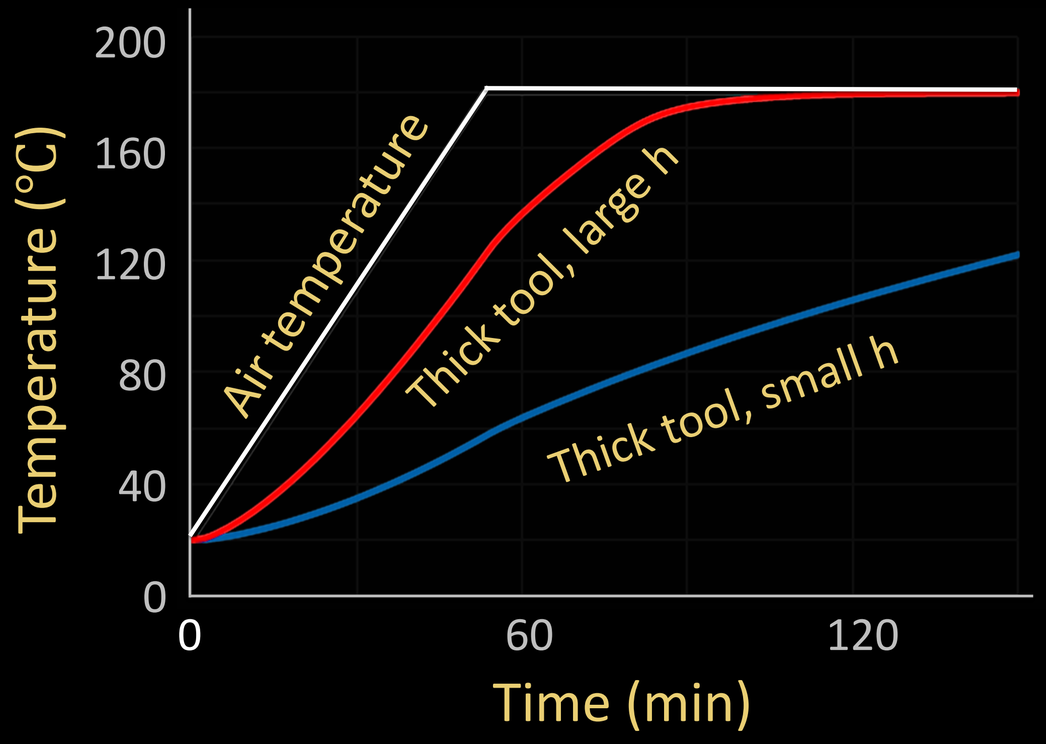
> Model predictions are incorrect, especially at high heat transfer and low tool thickness.



Approach 2: Physics-informed Feature

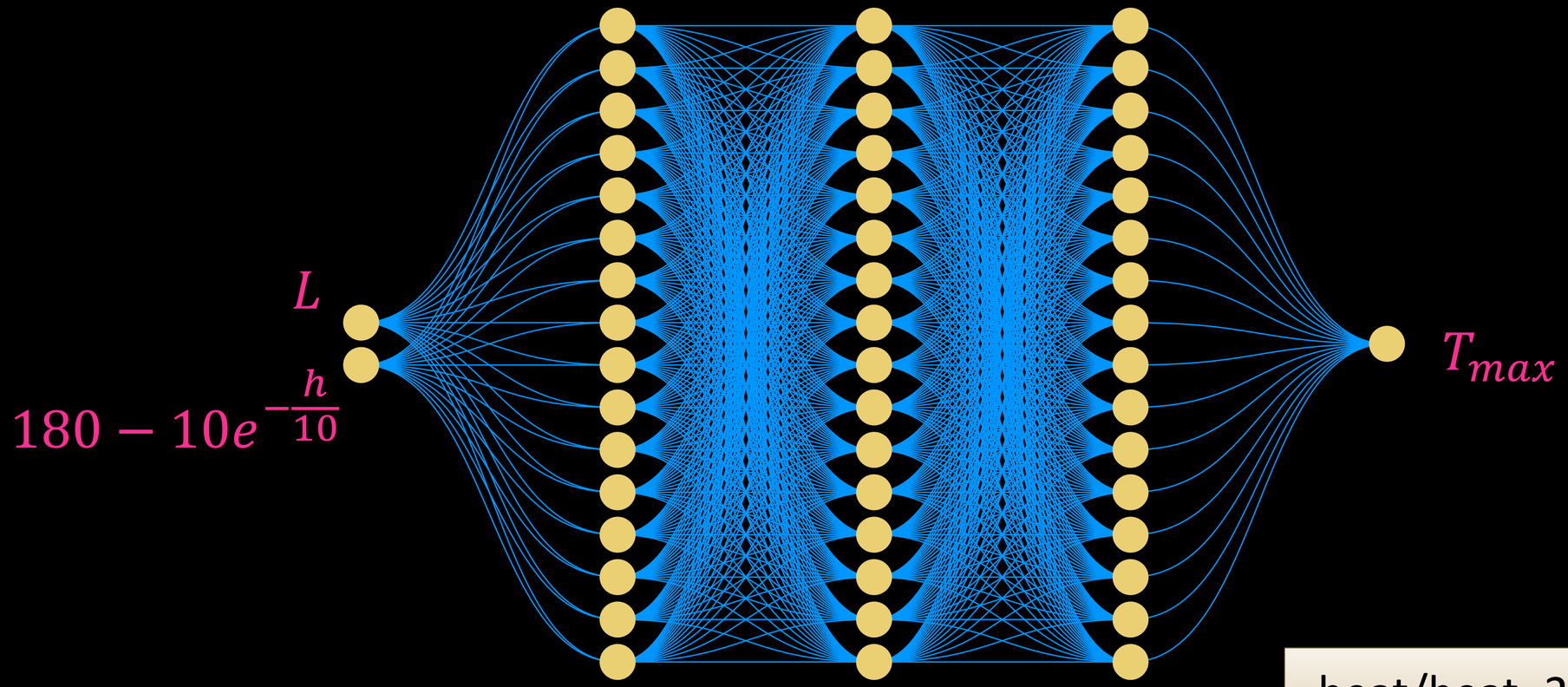
- At large thicknesses, the tool's thermal mass caps the temperature at 180°C. Low heat transfer coefficients further reduces the max temperature.
- This suggests an exponential relationship, estimated as:

$$T_{\max} \propto 180 - 10e^{-\frac{h}{10}}$$



Approach 2: Physics-informed Feature

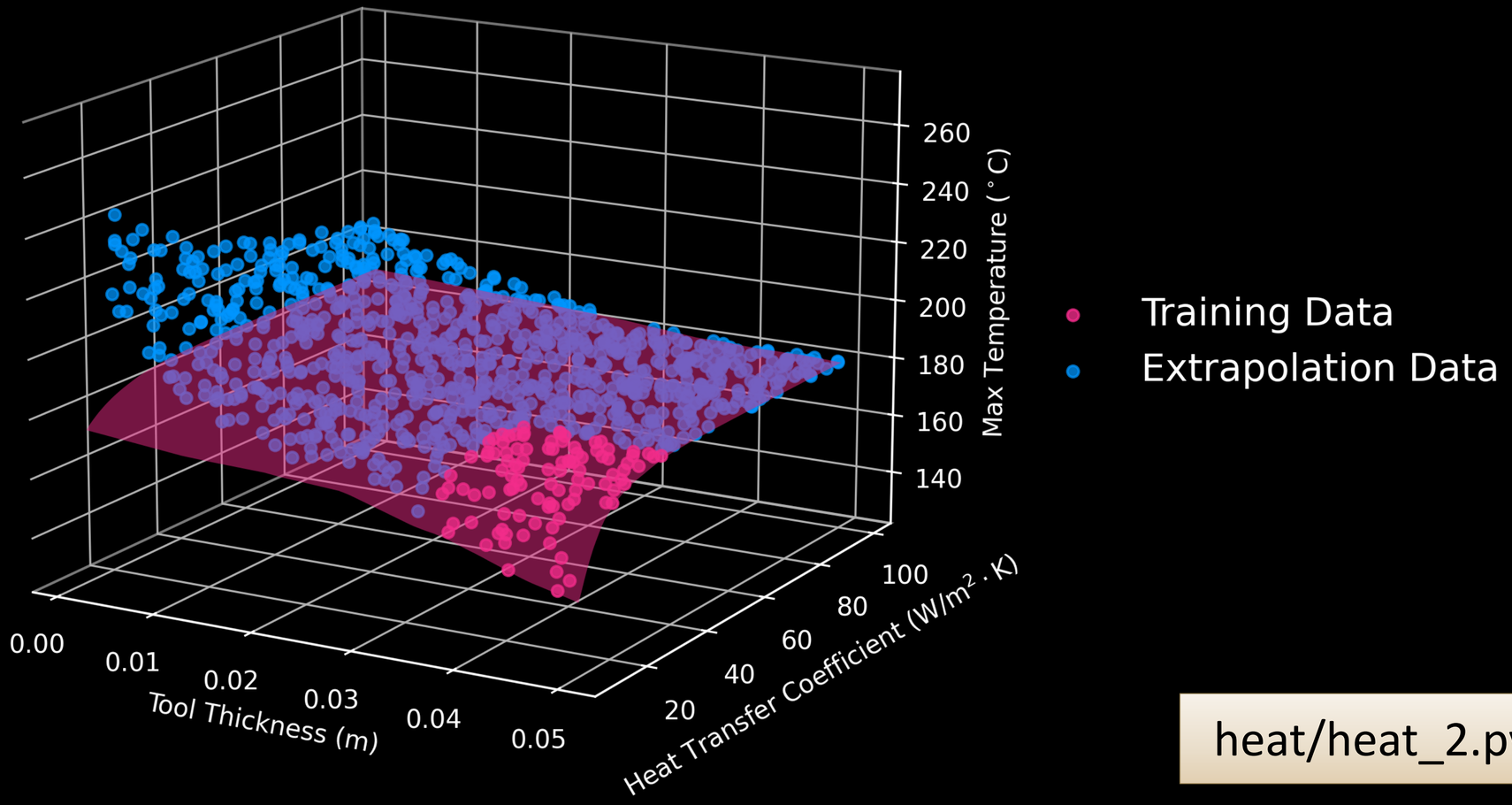
- We repeat our neural network training, but with this new transformed feature.



heat/heat_2.py

Approach 2: Physics-informed Feature

- **This shows correct extrapolation for large tool thicknesses but incorrectly predicts thin ones.**

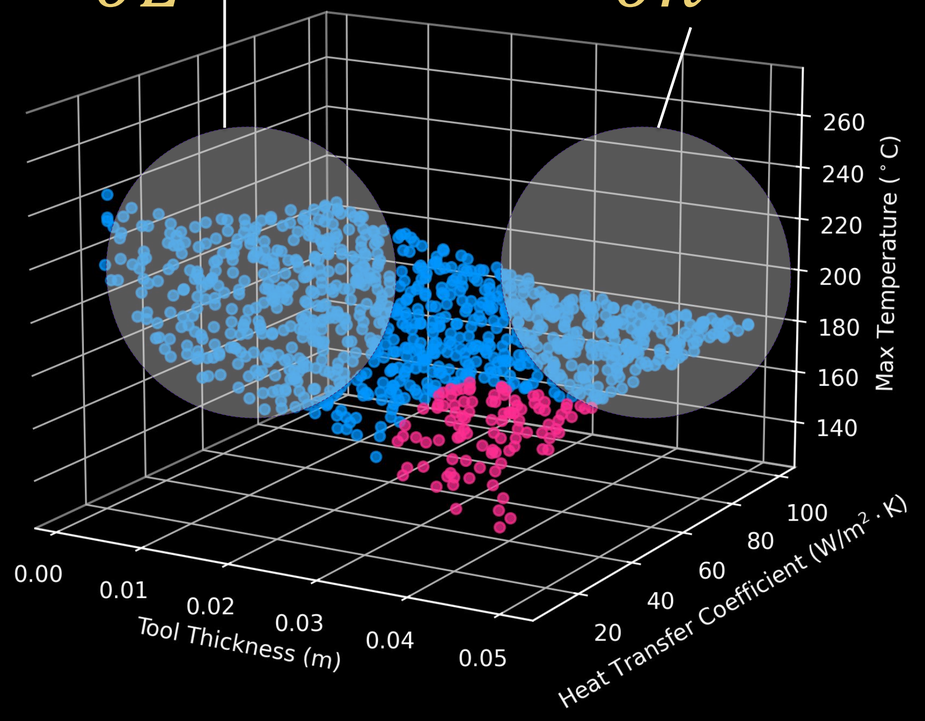


heat/heat_2.py

Approach 3: Physics-informed Loss

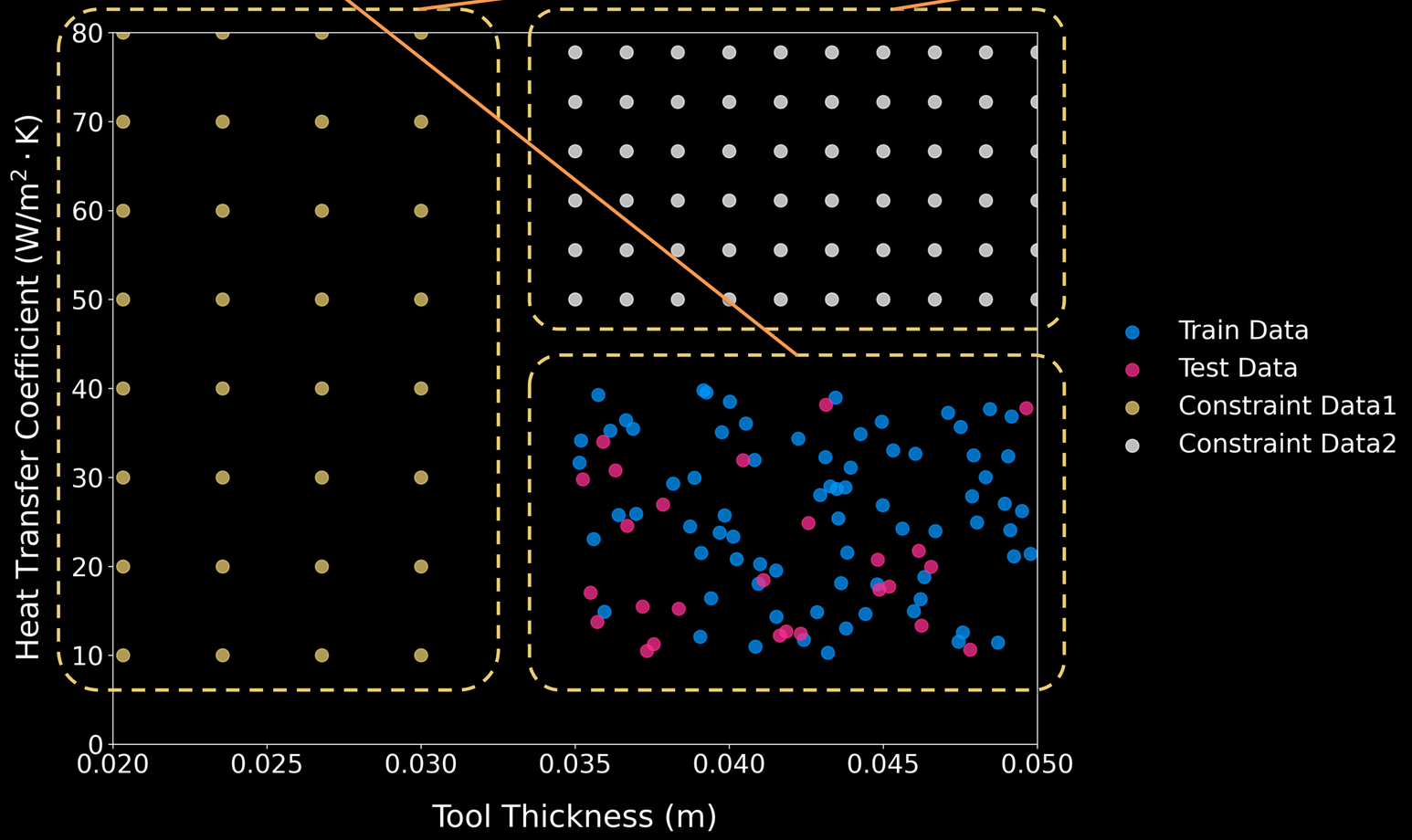
- In this approach, we modify the loss function for training the NN model.
- As tool thickness decreases, it loses its heat sink capability, causing the part's temperature to rise.
- Similarly, for high heat transfer coefficients, the derivative vanishes.

$$\frac{\partial T_{max}}{\partial L} < 0 \qquad \frac{\partial T_{max}}{\partial h} \approx 0$$



Approach 3: Physics-informed Loss

$$Loss = \underbrace{\frac{1}{\mathrm{n}}\sum^{n}\left(T_i - \widehat{T}_i\right)^2}_{} + \lambda_1 \underbrace{\sum^{m} \max(0, \frac{\partial \widehat{T}_i}{\partial L_i})}_{} + \lambda_2 \underbrace{\sum^{p} \left|\frac{\partial \widehat{T}_i}{\partial h_i}\right|}_{}$$



Approach 3: Python Implementation

> We implement the custom loss function using TensorFlow's GradientTape to compute the derivative of the NN model:

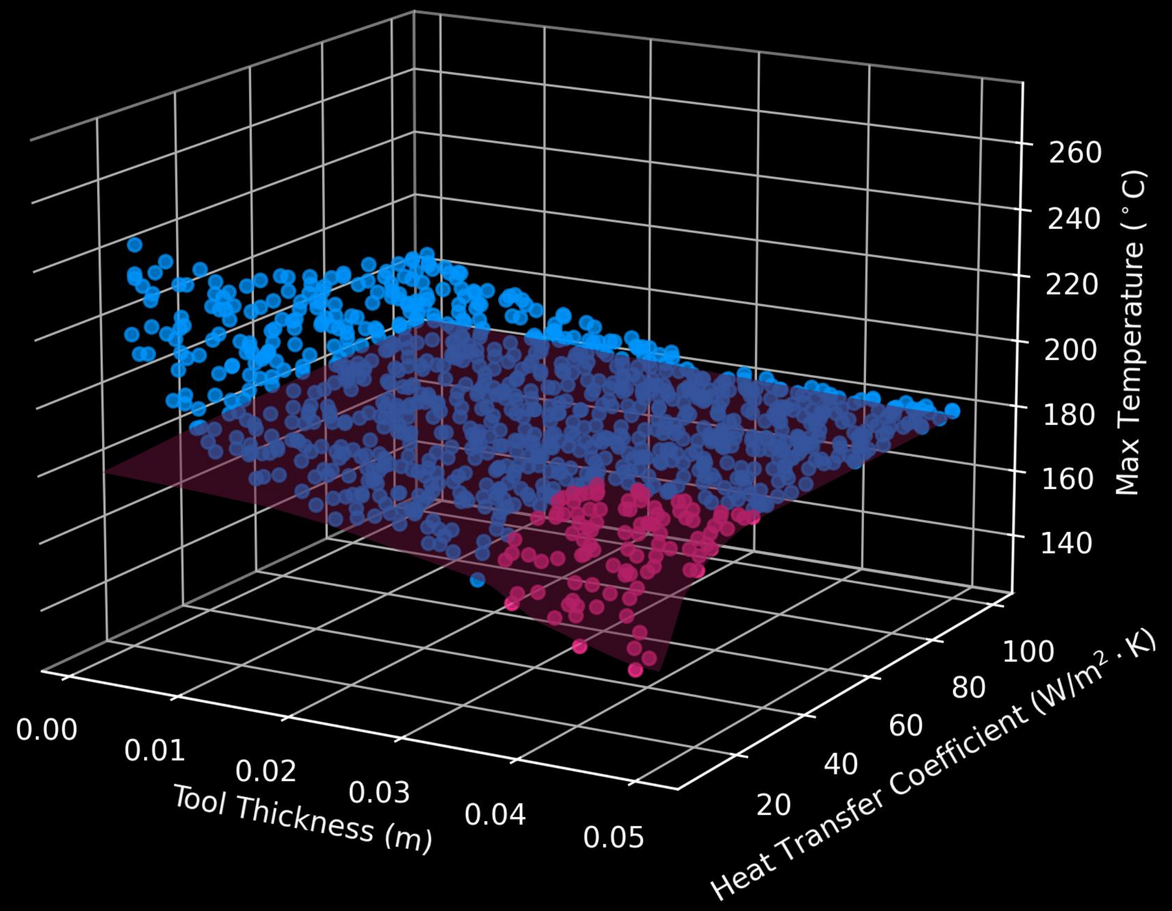
$\frac{\partial \hat{T}_i}{\partial L_i}$

$\frac{\partial \hat{T}_i}{\partial h_i}$

```python
def custom_loss_with_constraint(y_true, y_pred, X_constraint1, X_constraint2, model):
    # Calculate the mean squared error loss
    mse_loss = tf.reduce_mean(tf.keras.losses.mean_squared_error(y_true, y_pred))

    # Compute first derivative for the first column of X_constraint1
    with tf.GradientTape() as tape1:
        tape1.watch(X_constraint1)
        y_pred_constraint1 = model(X_constraint1, training=True)
    first_derivative_x1 = tape1.gradient(y_pred_constraint1, X_constraint1)[:, 0]
    constraint_loss1 = tf.reduce_sum(tf.maximum(0.0, first_derivative_x1))

    # Compute first derivative for the second column of X_constraint2
    with tf.GradientTape() as tape2:
        tape2.watch(X_constraint2)
        y_pred_constraint2 = model(X_constraint2, training=True)
    first_derivative_x2 = tape2.gradient(y_pred_constraint2, X_constraint2)[:, 1]
    constraint_loss2 = tf.reduce_sum(tf.abs(first_derivative_x2))

    # Combine the mean squared error and physics-based loss
    lambda1 = 1
    lambda2 = 1
    total_loss = mse_loss + lambda1 * constraint_loss1 + lambda2 * constraint_loss2
```

Approach 3: Physics-informed Loss - Results

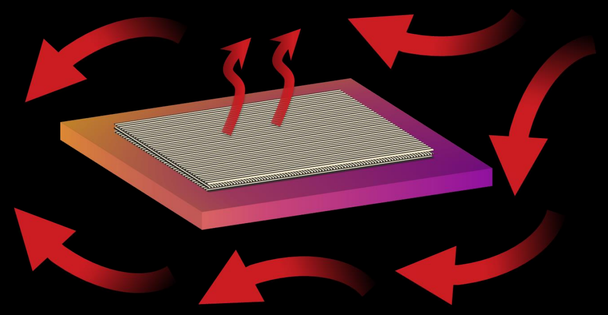
> **Extrapolation works nearby but falters with very small tool thickness due to physics changes.**

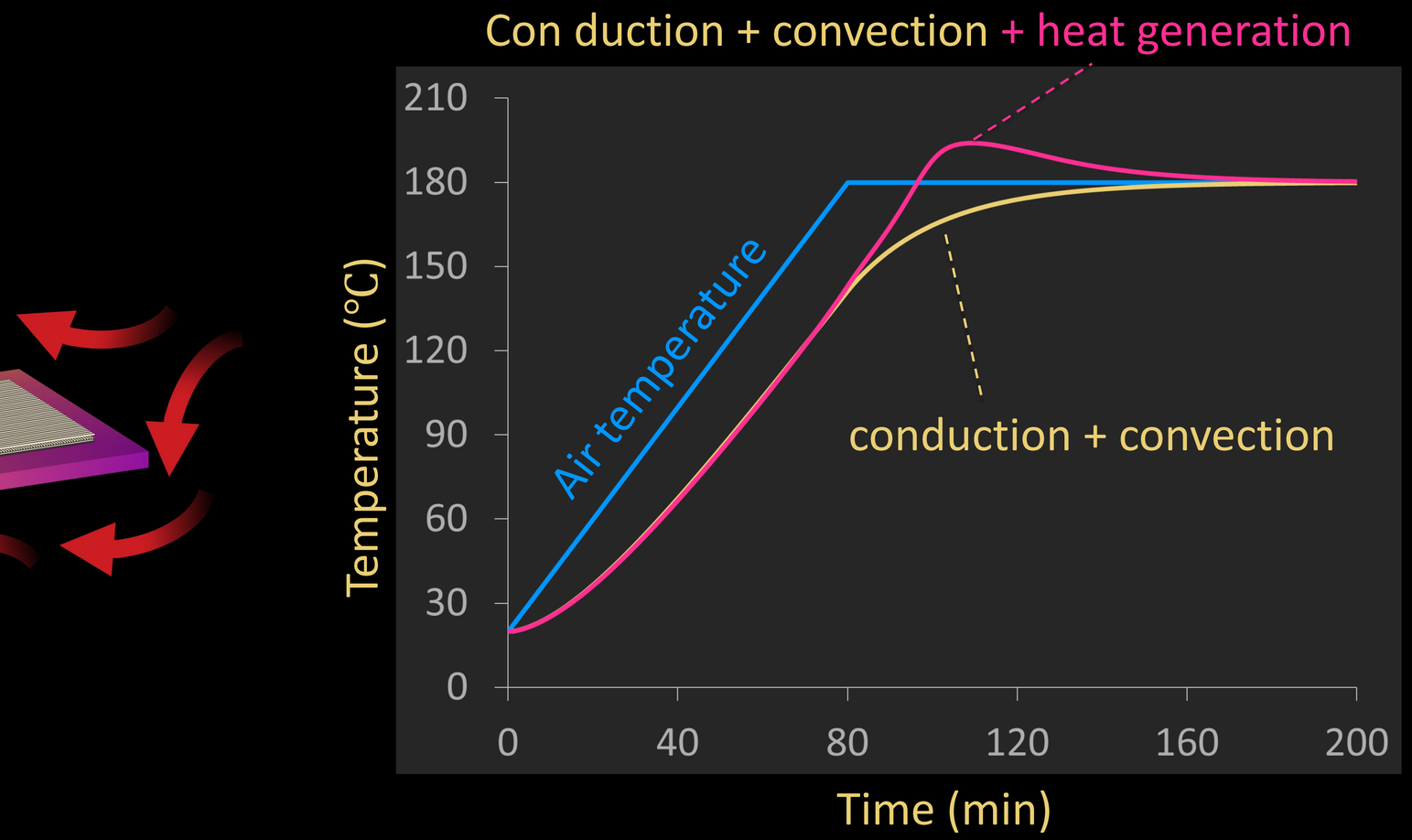


heat/heat_3.py

Approach 4: Physics-informed Domain Transformation

- **We can decouple the problem into much simpler conduction + convection, and heat generation physics:**



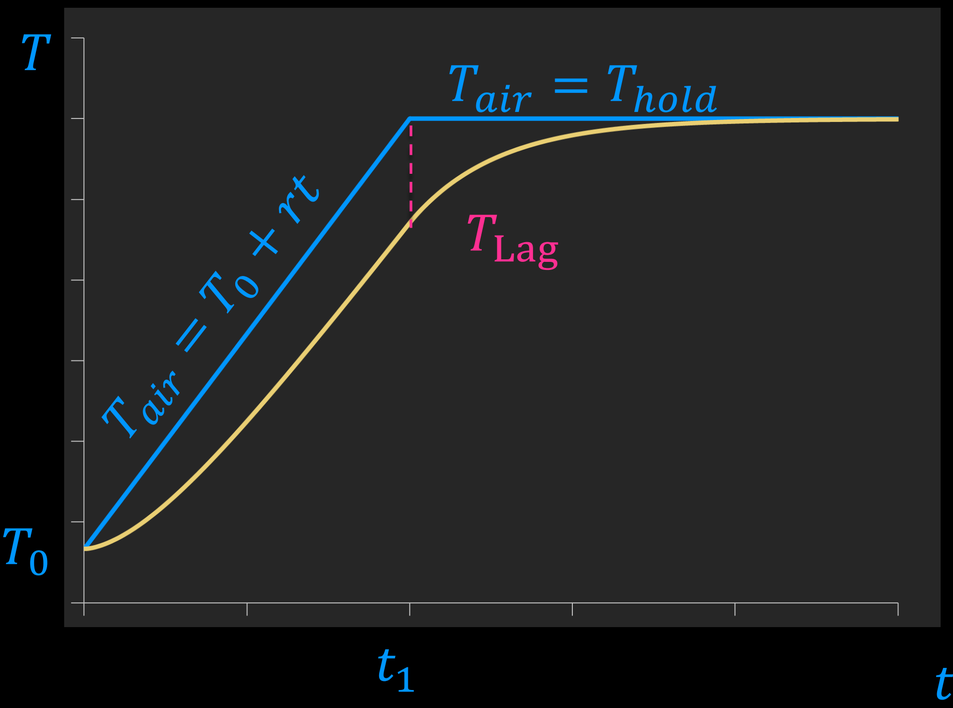


Physics 1: thermal lag (convection + conduction)

> **Approximate closed-form solutions are available to estimate thermal lag for the case without heat generation:**

$$T_{\mathrm{Lag}} = T_{\mathrm{hold}} - \Delta\left(e^{\frac{-r}{\Delta}(t-t_1)} - e^{\frac{-r}{\Delta}t}\right)$$

$$\Delta \propto \frac{M}{h} = \frac{thermal\ mass}{Heat\ transfer\ coefficient}$$



Zobeiry, N., Park, J., & Poursartip, A. (2019). An infrared thermography-based method for the evaluation of the thermal response of tooling for composites manufacturing. Journal of composite materials, 53(10), 1277-1290.

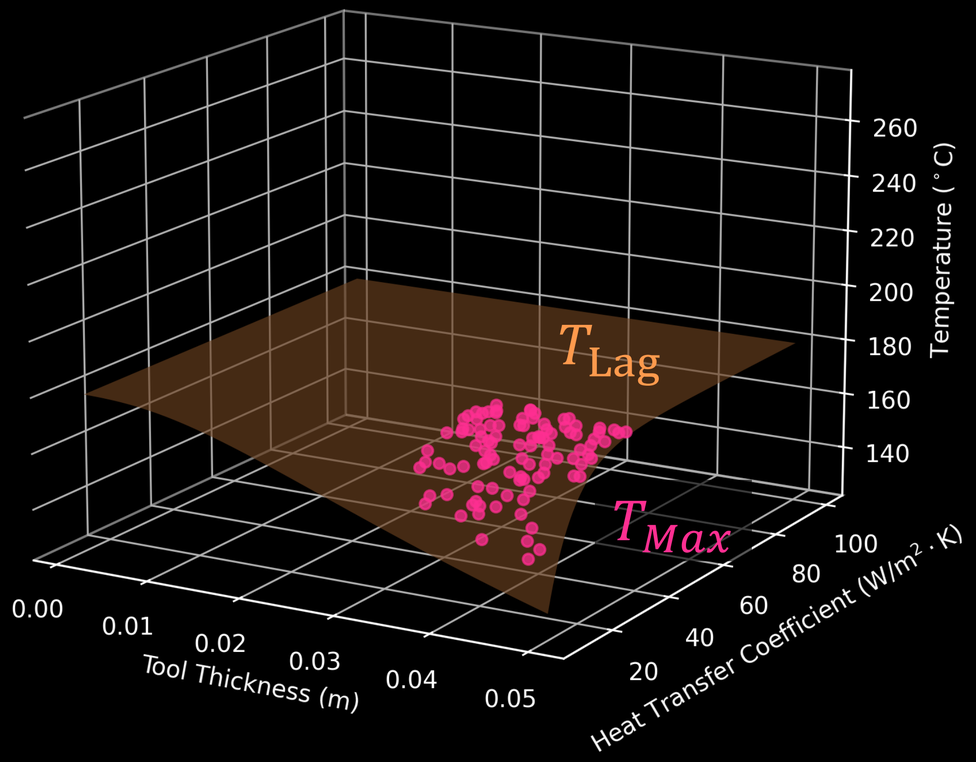
Physics 2: heat generation

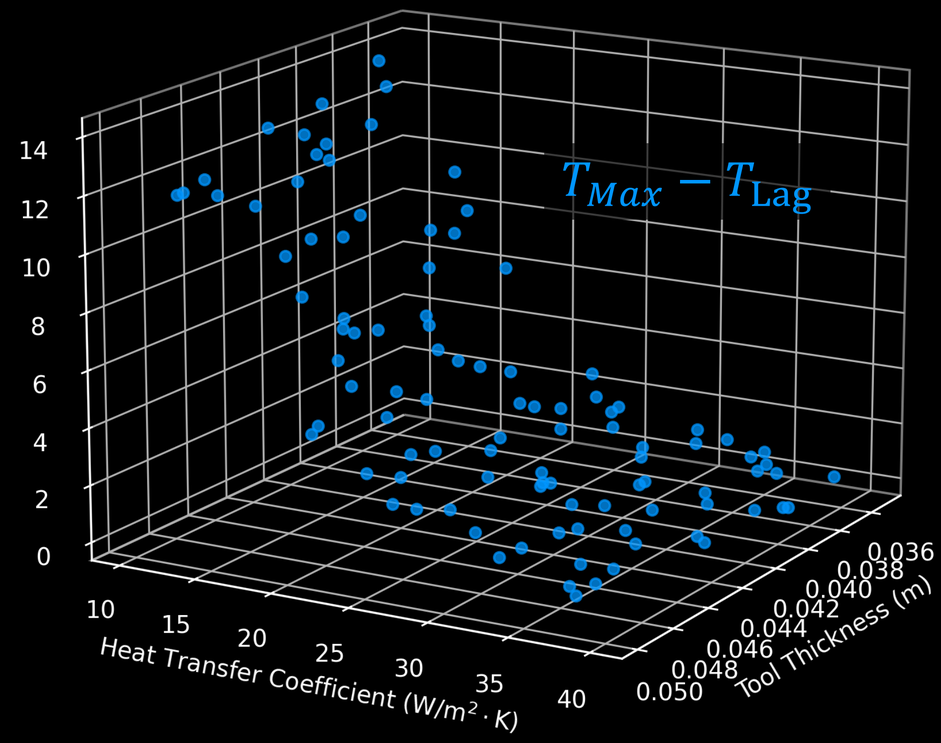
- Without thermal lag, any heat generated must either transfer to the air or be absorbed by the part, raising its temperature.
- Larger thermal mass and smaller thermal resistance both lead to a lower maximum temperature:

$$T_{\max} - T_{Lag} \propto \frac{R}{M} = \frac{thermal\ resistance}{thermal\ mass}$$

Physics-informed Domain Transformation

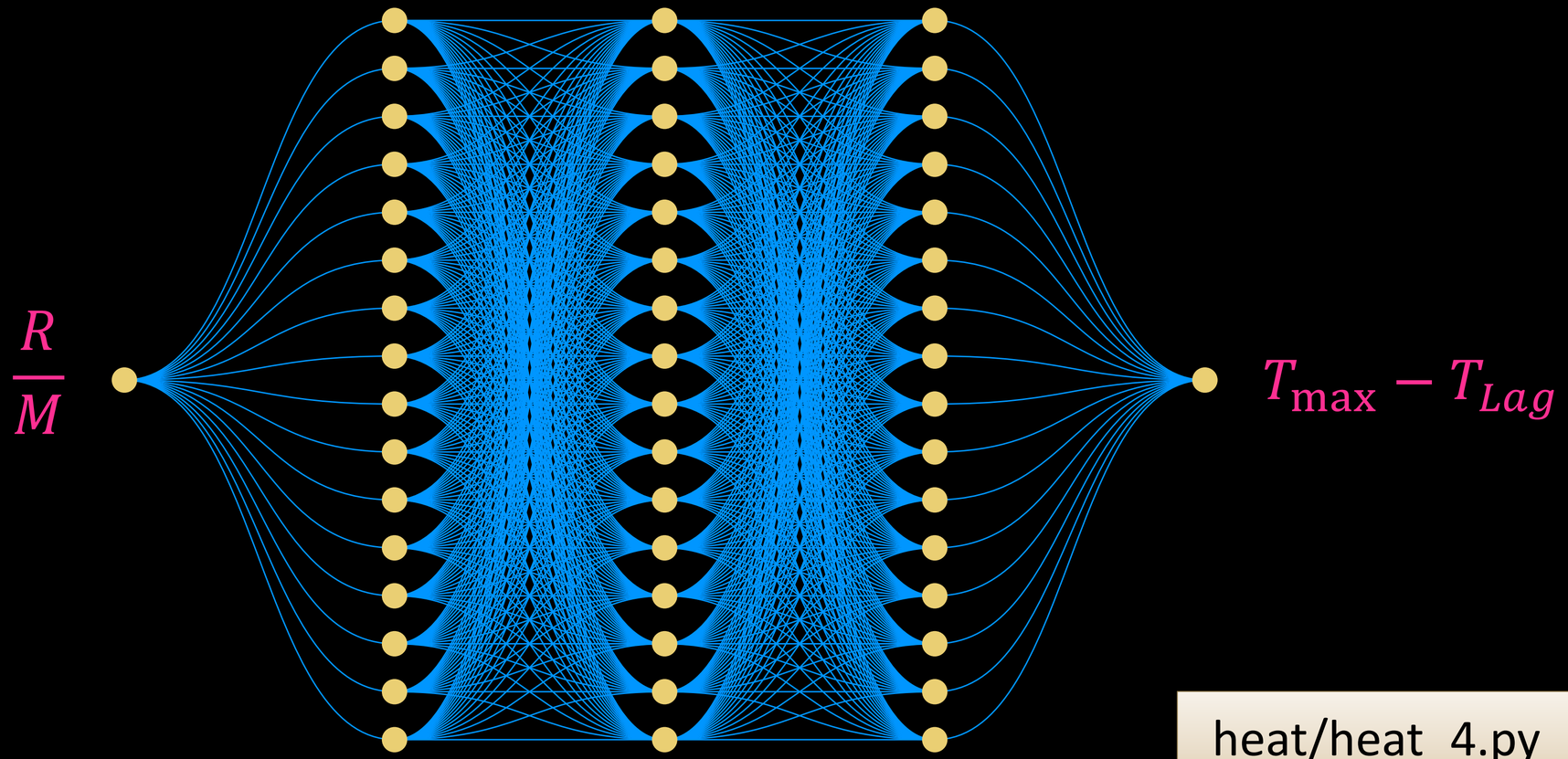
> First, we calculate thermal lag and subtract it from the max temperature data, effectively removing one physical phenomenon.





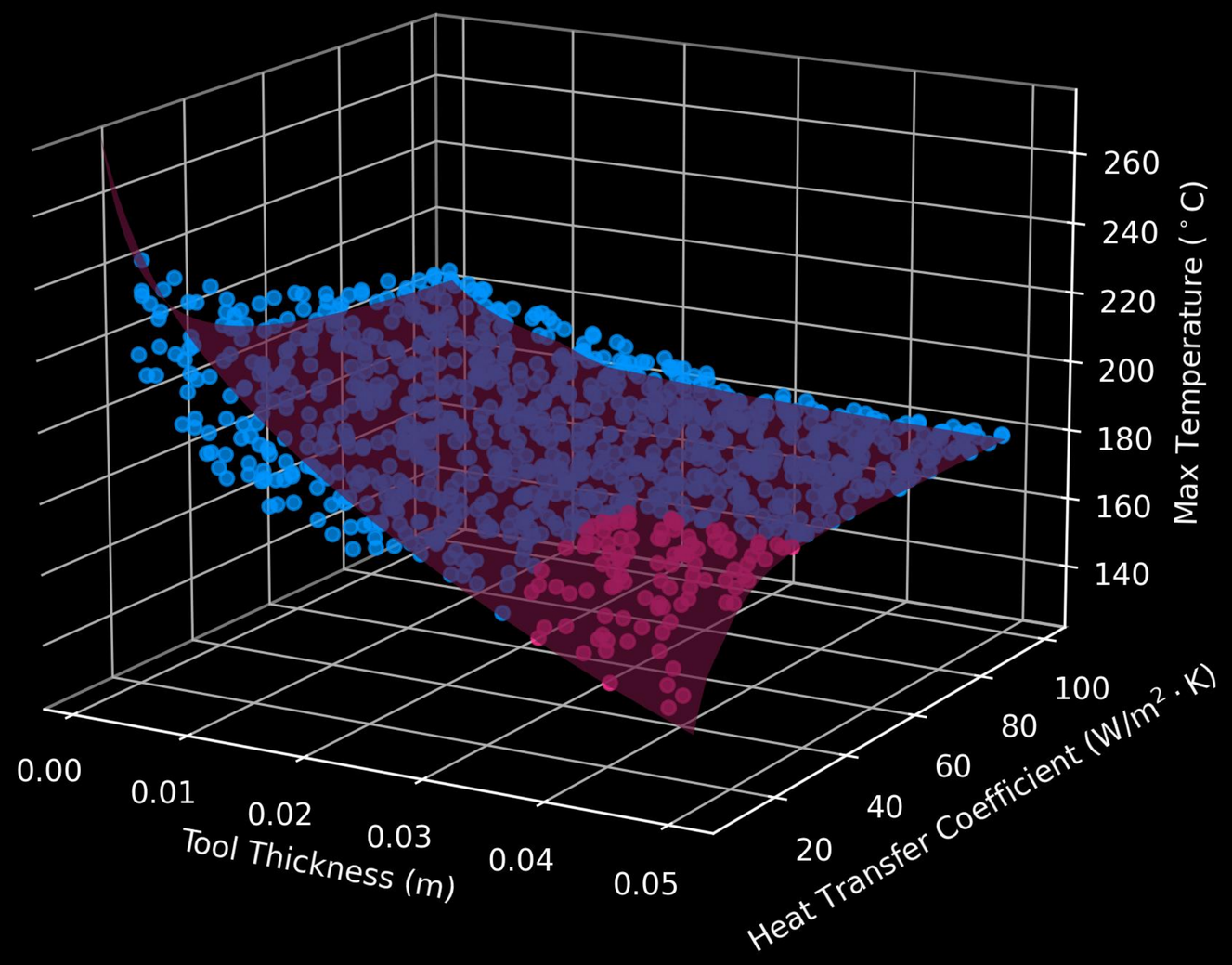
Physics-informed Domain Transformation

> **Next, we repeat our neural network training, but by transforming both inputs and outputs:**



heat/heat_4.py

Approach 4: Physics-informed Domain Transformation - Results



heat/heat_4.py

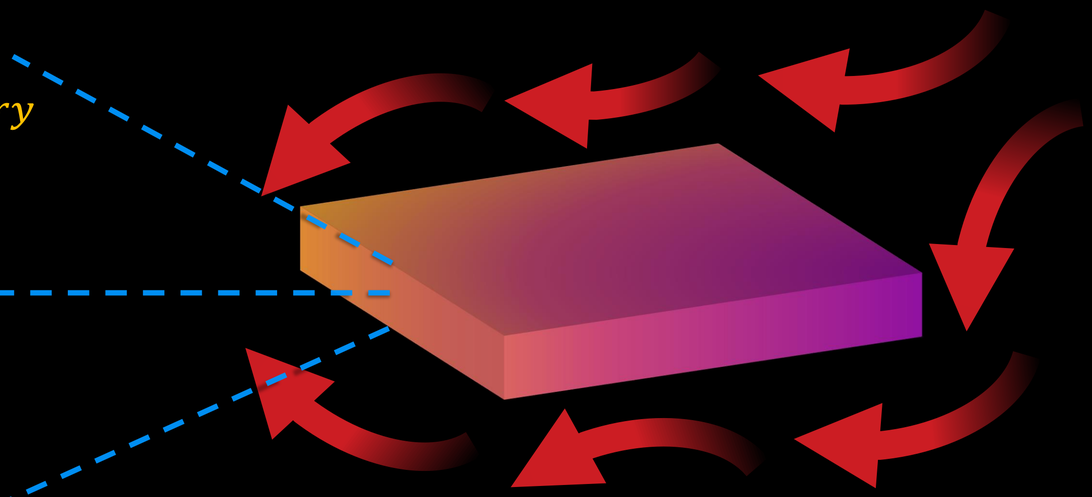
Approach 5: Physics-informed Neural Networks (PINN)

> **Instead of relying on data, we solve the governing differential equations (PDEs) directly. Here's a demonstration for convection and conduction:**

Convection: $h(T_\infty - T) = k \left.\frac{\partial T}{\partial x}\right|_{boundary}$

Conduction: $\frac{\partial T}{\partial t} - \alpha \frac{\partial^2 T}{\partial x^2} = 0$

Convection: $h(T_\infty - T) = k \left.\frac{\partial T}{\partial x}\right|_{boundary}$



Solving PDEs using PINN

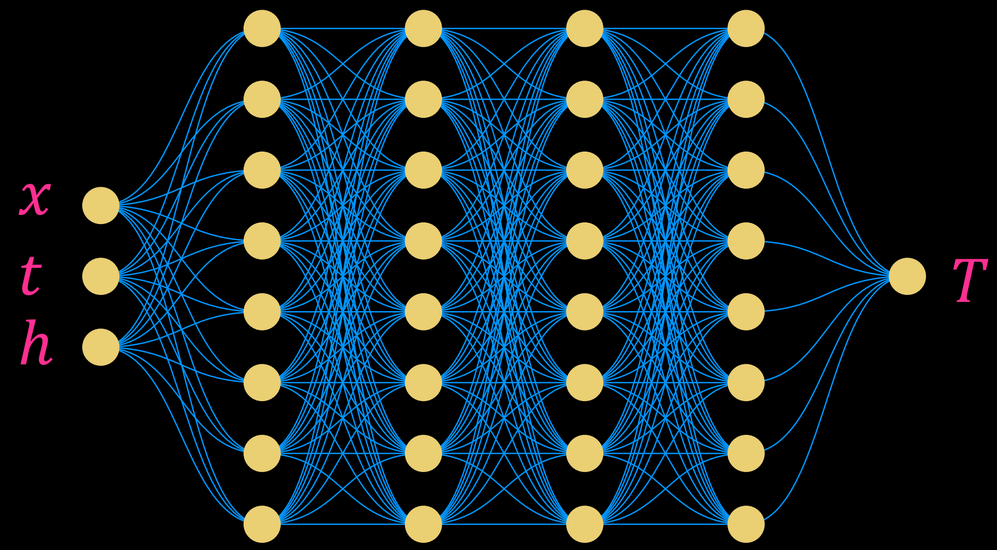
> At each step, we randomly select inputs, calculate the loss function based on the PDEs, and minimize it to train the neural network to predict temperature.

$$Loss = \sum Error_{PDE}^2 + \lambda_1 \sum Error_{BC1}^2 + \lambda_2 \sum Error_{BC2}^2$$

$$Error_{PDE} = \alpha \frac{\partial^2 f}{\partial x^2} - \frac{\partial f}{\partial t}$$

$$Error_{BC1} = (T_\infty - f) - \frac{k}{h} \frac{\partial f}{\partial x}\bigg|_{x=0}$$

$$Error_{BC1} = (T_\infty - f) - \frac{k}{h} \frac{\partial f}{\partial x}\bigg|_{x=L}$$



Zobeiry N., Humfeld K.D. (2021). A Physics-Informed Machine Learning Approach for Solving Heat Transfer Equation in Advanced Manufacturing and Engineering Applications, Engineering Applications of Artificial Intelligence, 101:104232