MACHINE LEARNING MODELS FOR PREDICTIVE ANALYSIS OF PRESSURE DROP AND TEMPERATURE IN POLYMER ELECTROLYTE MEMBRANE FUEL CELL STACKS TO FIND OPTIMAL FABRICATION PARAMETERS

HANHEE LEE

ESROP GLOBAL

NATIONAL UNIVERSITY OF SINGAPORE

2024

Declaration

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in this thesis.

This thesis has also not been submitted for any degree in any university previously.

Hanhee Lee

August, 2024

Acknowledgements

I would like to express my sincere gratitude to Professor Erik Birgersson. Professor Birgersson has provided me with profound insights and knowledge on various topics, including polymer electrolyte membrane fuel cell stacks, machine learning, deep learning, and the use of COMSOL. He has also taught me how to effectively present to a scientific audience and the importance of learning independently. His enthusiasm and encouragement have made this journey a truly pleasant and enriching experience.

Abstract

Contents

List of Figures

List of Tables

0.1 List of Symbols

Name	**Description**	**Unit**
p	Pressure	Pa
$\mathbf{u}, \mathbf{U}$	Velocity	m/s
ρ	Density	kg/m^3
$\mathbf{I}$	Identity tensor	-
$\mathbf{K}$	Viscous stress tensor	-
$\mathbf{F}$	Volume force vector	-
μ	Dynamic viscosity	Pa·s
T	Temperature	K
$\mathbf{n}$	Unit vector normal to the given surface	-
$\mathbf{G}$	Reciprocal wall distance	-
C_p	Specific heat capacity	J/(kg·K)
$\mathbf{q}$	Heat flux vector	-
Q	Heat source	W/m^3
k	Thermal conductivity	W/(m·K)
R	Thermal resistance	(m^2·K)/W
d	Thin layer thickness	m
Δ	Delta	-
ℓ	Length/Distance	m
u^+	Tangential velocity in viscous units	-
σ_w	Smoothing parameter	-
Re	Reynolds number	-
q	Quadratic loss coefficient	-
ϵ_p	Porosity	-
κ	Permeability	m^2
β_F	Forchheimer coefficient	kg/m^4
Q_m	Mass source	kg/(m^3·s)

Table 1: Variable Descriptions and Units

Name	**Description**
0	Standard Conditions
n	Unit vector normal to the given surface
p	Point
ted	Thermoelastic damping
b	Boundary
d	Down side
s	Solid
u	Up side
ref	Reference
w	Wall
T	Turbulent
$exit$	Exit
pc	Pressure curve

Table 2: Subscript Descriptions

Name	**Description**
T	Transpose

Table 3: Superscript Descriptions

Chapter 1

Introduction

1.1 Fuel Cells

Chapter 2

Polymer Electrolyte Membrane Fuel Cells

Chapter 3

Neural Networks

3.1 Introduction to Machine Learning

Machine learning is a subset of artificial intelligence that focuses on developing algorithms and statistical models that enable computers to learn from and make predictions based on data. It is used in a wide range of applications, including image and speech recognition, medical diagnosis, and financial forecasting. Machine learning can be broadly categorized into three types: supervised learning, unsupervised learning, and reinforcement learning.

Machine learning is useful for tasks where it is difficult to explicitly program the rules or patterns, such as recognizing handwritten digits or detecting spam emails.

By training a model on a dataset, the model can learn the underlying patterns and relationships in the data and make predictions on new, unseen data.

3.2 Structure of Neural Networks

Neural networks are a class of machine learning models inspired by the structure and function of the human brain. They consist of interconnected nodes, or neurons, organized in layers. Each neuron receives input, processes it, and produces an output that is passed to the next layer. The connections between neurons are represented by weights, which are learned during the training process.

The basic building block of a neural network is the perceptron, which takes a set of inputs, applies weights to them, and passes the weighted sum through an activation function to produce an output. Multiple perceptrons are connected in layers to form a neural network. The most common type of neural network is the feedforward neural network, where information flows in one direction from the input layer to the output layer.

Neural networks can have multiple layers, with each layer performing a different transformation on the input data. The input layer receives the input data, the hidden layers process the data, and the output layer produces the final output. The number of layers and the number of neurons in each layer are hyperparameters that can be tuned to optimize the performance of the network.

3.2.1 Types of Layers

Neural networks have multiple types of layers, including:

- Input Layer: The first layer of the network that receives the input data.
- Hidden Layers: Intermediate layers that process the input data and extract features.
- Output Layer: The final layer that produces the output of the network.

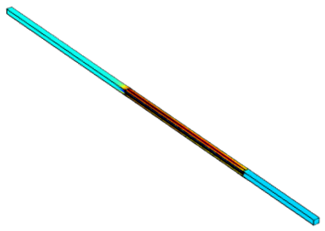

Figure 3.1: Structure of a feedforward neural network.

3.2.2 Shallow and Deep Neural Networks

Neural networks can be classified as shallow or deep based on the number of hidden layers they contain. Shallow networks have a small number of hidden layers, while deep networks have many hidden layers. Deep neural networks are capable of learning complex patterns and representations in the data but are more computationally intensive to train

3.3 Process of Supervised Learning

Supervised learning is a type of machine learning where the model is trained on labeled data. The objective is to learn a function that maps input data to the correct output based on the provided labels. The general process involves the following steps:

1. **Data Collection**: Gather a dataset consisting of input-output pairs. Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be the set of input vectors and $\mathbf{Y} = \{y_1, y_2, \ldots, y_n\}$ be the corresponding set of output values.
2. **Data Preprocessing**: Clean and preprocess the data to remove noise, handle missing values, and normalize the features. This step ensures that the data is in a suitable format for training the model.
3. **Model Selection**: Choose a neural network architecture suitable for the problem. This includes deciding on the number of layers, the number of neurons per layer, and the type of activation functions.
4. **Initialization**: Initialize the weights $\mathbf{W}$ and biases $\mathbf{b}$ of the network. This is typically done using small random values.
5. **Forward Propagation**: Compute the predicted output $\hat{y}$ by passing the input $\mathbf{x}$ through the network.
6. **Loss Computation**: Calculate the loss $\mathcal{L}(\hat{y}, y)$ which measures the difference between the predicted output $\hat{y}$ and the actual output y.
7. **Backward Propagation**: Compute the gradients of the loss with respect to the weights and biases.
8. **Weight Update**: Update the weights and biases using an optimization algorithm such as Stochastic Gradient Descent (SGD).
9. **Model Evaluation**: Evaluate the performance of the model on a validation set to tune hyperparameters and avoid overfitting.

Mathematically, the process can be summarized as follows:

Given an input vector $\mathbf{x}$, the network's output $\hat{y}$ is computed as:

$$\hat{y} = f(\mathbf{x}; \mathbf{W}, \mathbf{b}) \tag{3.1}$$

where f is the function represented by the neural network, parameterized by weights $\mathbf{W}$ and biases $\mathbf{b}$.

The loss function $\mathcal{L}$ is defined to measure the discrepancy between $\hat{y}$ and the true output y:

$$\mathcal{L}(\hat{y}, y) \tag{3.2}$$

The gradients of the loss with respect to the parameters are computed during backpropagation:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}} \tag{3.3}$$

Finally, the parameters are updated using an optimization algorithm:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}}, \quad \mathbf{b} \leftarrow \mathbf{b} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}} \tag{3.4}$$

where η is the learning rate.

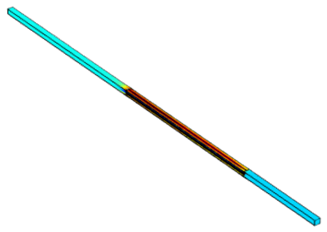

Figure 3.2: Overview of the supervised learning process in neural networks.

3.4 Forward Propogation

Forward propagation is the process by which the input is passed through the neural network to obtain the output. It involves the computation of outputs at each layer of the network until the final output is achieved. The process can be described as follows:

Given an input vector $\mathbf{x}$, the output of the first layer is computed as:

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \tag{3.5}$$

$$\mathbf{a}^{(1)} = \sigma(\mathbf{z}^{(1)}) \tag{3.6}$$

where $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ are the weights and biases of the first layer, $\mathbf{z}^{(1)}$ is the linear combination of inputs and weights, and σ is the activation function.

This process is repeated for each subsequent layer. For the l-th layer, the computations are:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \tag{3.7}$$

$$\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)}) \tag{3.8}$$

Finally, the output of the network is obtained:

$$\hat{y} = \mathbf{a}^{(L)} \tag{3.9}$$

where L is the number of layers in the network.

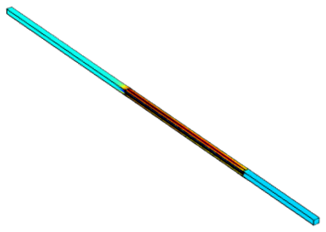

Figure 3.3: Forward propagation through a neural network.

3.4.1 Activation Functions

Activation functions introduce non-linearity into the neural network, allowing it to learn complex patterns. Common activation functions include:

Sigmoid

The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.10}$$

It maps any real-valued number into the range (0, 1).

Hyperbolic Tangent (Tanh)

The tanh function is defined as:

$$\sigma(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.11}$$

It maps any real-valued number into the range (-1, 1).

Rectified Linear Unit (ReLU)

The ReLU function is defined as:

$$\sigma(z) = \max(0, z) \tag{3.12}$$

It outputs the input directly if it is positive; otherwise, it outputs zero.

Leaky ReLU

The Leaky ReLU function is defined as:

$$\sigma(z) = \begin{cases} z & \text{if } z \geq 0 \\ \alpha z & \text{if } z < 0 \end{cases} \tag{3.13}$$

where α is a small constant.

3.5 Backward Propagation

Backward propagation, or backpropagation, is the process by which neural networks update their weights and biases to minimize the loss function. It involves calculating the gradient of the loss function with respect to each weight by the chain rule, iterating backward from the output layer to the input layer. The main steps are as follows:

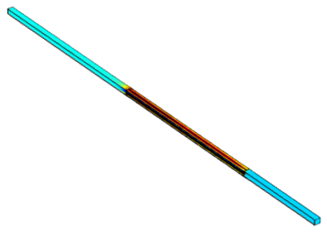

Figure 3.4: Common activation functions used in neural networks.

1. **Compute the loss**: Calculate the loss $\mathcal{L}$ between the predicted output $\hat{y}$ and the actual output y.
2. **Calculate the gradient of the loss with respect to the output layer**: For the output layer, compute the gradient of the loss with respect to the activations.
3. **Propagate the gradient backward through the network**: Use the chain rule to compute the gradient of the loss with respect to the weights and biases of each layer.
4. **Update the weights and biases**: Use the gradients to update the weights and biases in a direction that reduces the loss.

Mathematically, the gradient of the loss $\mathcal{L}$ with respect to the weights $\mathbf{W}^{(l)}$ and biases $\mathbf{b}^{(l)}$ in layer l is computed as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \delta^{(l)} \mathbf{a}^{(l-1)} \tag{3.14}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \delta^{(l)} \tag{3.15}$$

where $\delta^{(l)}$ is the error term for layer l and $\mathbf{a}^{(l-1)}$ is the activation of the previous layer.
The error term $\delta^{(l)}$ is computed as:

$$\delta^{(l)} = \begin{cases} (\mathbf{a}^{(L)} - y) \odot \sigma'(\mathbf{z}^{(L)}) & \text{for the output layer} \\ (\mathbf{W}^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(\mathbf{z}^{(l)}) & \text{for hidden layers} \end{cases} \tag{3.16}$$

where $\odot$ denotes the element-wise multiplication and σ' is the derivative of the activation function.

3.5.1 Loss Functions

Loss functions, also known as cost functions, measure how well the neural network's predictions match the actual target values. Common loss functions include:

Mean Squared Error (MSE)

The Mean Squared Error is used for regression tasks and is defined as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{3.17}$$

3.5.2 Methods to Update Weights

Updating the weights and biases of a neural network is a crucial part of the training process. Various methods can be employed to perform these updates:

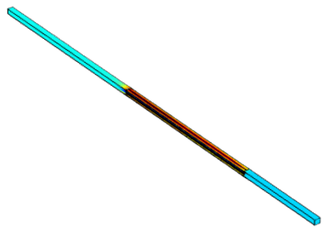

Figure 3.5: Illustration of common loss functions.

Stochastic Gradient Descent (SGD)

SGD updates the weights using a single training example at a time:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \tag{3.18}$$

where η is the learning rate.

Batch Gradient Descent

Batch Gradient Descent computes the gradient using the entire training dataset:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{n} \sum_{i=1}^{n} \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}} \tag{3.19}$$

Mini-Batch Gradient Descent

Mini-Batch Gradient Descent is a compromise between SGD and Batch Gradient Descent. It uses a small random subset (mini-batch) of the training data to compute the gradient:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}} \tag{3.20}$$

where m is the mini-batch size.

Adaptive Methods

Adaptive methods like AdaGrad, RMSProp, and Adam adjust the learning rate based on the history of the gradients. For example, the Adam optimization algorithm updates the weights as follows:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \tag{3.21}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}} \right)^2 \tag{3.22}$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t} \tag{3.23}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t} \tag{3.24}$$

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \tag{3.25}$$

where β_1 and β_2 are hyperparameters, $\mathbf{m}_t$ and $\mathbf{v}_t$ are the first and second moment estimates, and ϵ is a small constant.

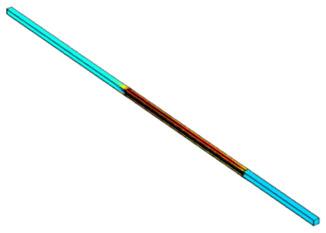

Figure 3.6: Comparison of different methods to update weights.

3.6 Hyperparameters

Hyperparameters are parameters whose values are set before the learning process begins. Unlike model parameters, which are learned during training, hyperparameters control the training process and influence the performance of the neural network. Common hyperparameters include learning rate, batch size, number of epochs, network architecture, activation functions, weight initialization, dropout rate, and regularization parameters.

3.6.1 Common Hyperparameters

Learning Rate

The learning rate η controls the size of the steps taken during gradient descent to update the weights. A smaller learning rate can lead to more precise convergence but slower training, while a larger learning rate can speed up training but might overshoot the optimal solution.

Batch Size

Batch size determines the number of training examples used to calculate the gradient in one forward/backward pass. It influences the stability and speed of the training process. Common choices are small (stochastic gradient descent), large (batch gradient descent), or in between (mini-batch gradient descent).

Number of Epochs

The number of epochs defines how many times the entire training dataset passes through the network. More epochs typically improve learning, but excessive epochs can lead to overfitting.

Network Architecture

Network architecture includes the number of layers, the number of neurons per layer, and the type of layers used (e.g., dense, convolutional, recurrent). These choices significantly impact the capacity and capability of the network.

Activation Functions

Different activation functions (e.g., Sigmoid, Tanh, ReLU) can be used in different layers of the network. The choice of activation function affects the network's ability to capture non-linear patterns.

Weight Initialization

Weight initialization affects the starting point of the training process. Common initialization methods include random initialization, Xavier initialization, and He initialization, each suitable for different types of activation functions.

Dropout Rate

Dropout is a regularization technique where a fraction of neurons is randomly set to zero during training. The dropout rate controls this fraction and helps prevent overfitting by promoting the independence of neurons.

Regularization Parameters

Regularization parameters, such as L1 and L2 regularization, add a penalty to the loss function to constrain the model complexity. They help in preventing overfitting by discouraging overly complex models.

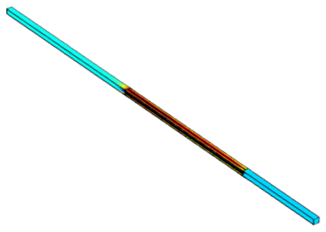

Figure 3.7: Overview of common hyperparameters in neural networks.

3.6.2 Hyperparameter Optimization

Hyperparameter optimization involves finding the optimal set of hyperparameters that result in the best performance of the neural network. Common methods for hyperparameter optimization include:

Grid Search

Grid search involves specifying a set of values for each hyperparameter and training the model on all possible combinations of these values. It is computationally expensive but exhaustive.

$$\text{Optimal Hyperparameters} = \arg \min_{\eta, \text{batch size}, \ldots} \text{Validation Loss} \tag{3.26}$$

Random Search

Random search samples random combinations of hyperparameters from a predefined distribution. It is often more efficient than grid search and can find good hyperparameters with fewer iterations.

$$\text{Optimal Hyperparameters} = \arg \min_{\eta, \text{batch size}, \ldots} \text{Validation Loss} \tag{3.27}$$

Bayesian Optimization

Bayesian optimization builds a probabilistic model of the objective function and uses it to select the most promising hyperparameters to evaluate next. It is more sophisticated and can find optimal hyperparameters more efficiently.

$$\text{Optimal Hyperparameters} = \arg \max_{\eta, \text{batch size}, \ldots} P(\text{Low Validation Loss} \mid \eta, \text{batch size}, \ldots) \tag{3.28}$$

Hyperband

Hyperband is a method that combines random search with early stopping. It evaluates many configurations with a small number of iterations and progressively increases the budget for the most promising configurations.

$$\text{Optimal Hyperparameters} = \arg \min_{\eta, \text{batch size}, \ldots} \text{Validation Loss} \tag{3.29}$$

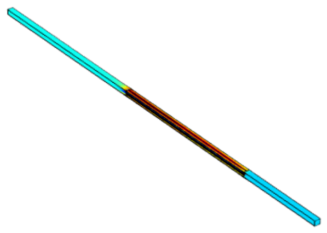

Figure 3.8: Comparison of different hyperparameter optimization methods.

3.7 Mathematical Formulation of Neural Networks

3.7.1 Introduction

This paper explores simple regression models to demonstrate the underlying mechanics by implementing these models both through MATLAB and using a paper-and-pen approach for a shallow and deep neural network.

3.7.2 Notation

The following table summarizes the notation used in the neural network models:

Table 3.1: Detailed Explanation of Variables

Header	**Dimension**	**Explanation**
		Superscripts
$[l]$	1	Current layer
(i)	1	ith training example
		Subscripts
j	1	jth node of the current layer
k	1	jth node of the previous layer
		Sizes
m	1	Number of training examples in the dataset
n_x	1	Number of nodes in the input layer
n_y	1	Number of nodes in the output layer
$n^{[l]}$	1	Number of nodes in the current layer
$n^{[l-1]}$	1	Number of nodes in the previous layer
L	1	Number of layers in the network
		Objects
$\mathbf{X}$	$n_x \times m$	Input matrix
$\mathbf{x}^{(i)}$	$n_x \times 1$	ith example represented as a column vector
$\mathbf{W}^{[l]}$	$n^{[l]} \times n^{[l-1]}$	Weight matrix of the current layer
$z_j^{[l](i)}$	1	A weighted sum of the activations of the previous layer, shifted by a bias
$w_{j,k}^{[l]}$	1	A weight that scales the kth activation of the previous layer
$b^{[l]}$	$n^{[l]} \times 1$	Bias vector in the current layer
$b_j^{[l]}$	$n^{[l]} \times 1$	Bias in the current layer
$a_j^{[l](i)}$	1	An activation in the current layer
$a_k^{[l-1](i)}$	1	An activation in the previous layer
$g_j^{[l]}$	1	An activation function used in the current layer

3.7.3 Neural Network Formulas

This section describes the foundational equations used in neural networks, detailing the computation involved in forward propagation.

3.7.4 Layer-Wise Computation

$$z_j^{[l](i)} = \sum_k w_{j,k}^{[l]} a_k^{[l-1](i)} + b_j^{[l]} \tag{3.30}$$

- This equation calculates the weighted sum of the activations from the previous layer, combined with a bias term, to produce the pre-activation value for neuron j in layer l for a given input i.

$$a_k^{[l-1](i)} = g_j^{[l]} \left(z_1^{[l](i)}, \ldots, z_j^{[l](i)}, \ldots, z_{n^{[l]}}^{[l](i)} \right) \tag{3.31}$$

- Here, the activation of the kth neuron in layer $l-1$ for input i is calculated using the activation function $g_j^{[l]}$, which is applied to the vector of all pre-activation values from layer l.

$$L = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2} \tag{3.32}$$

- This equation represents the root mean squared error, a common cost function used to measure the difference between the predicted outputs ($\hat{y}_i$) and the actual targets (y_i) over all m training examples.

3.7.5 Simple Regression Models

To demonstrate the neural network's learning mechanism, we will use simple regression models. These models illustrate how a neural network can be trained to fit data according to specific mathematical relationships.

Model Descriptions

The first model we will consider is described by the following linear relationship:

$$y = 4x_1 + x_2^2 + x_3 \tag{3.33}$$

- This model expresses y as a linear combination of x_1 and x_3, with x_2 contributing quadratically. This demonstrates how different types of input features can be integrated into the prediction.

Additional models can be added here following the same format, explaining the mathematical relation and its potential learning implications for a neural network.

3.7.6 Neural Network Diagram

This section details the architecture of a simple neural network, which is diagrammed below:

- **Input Layer:** Consists of 3 nodes, representing the input features (x_1, x_2, x_3).
- **Hidden Layer:** Comprises 4 nodes, which facilitate the learning of non-linear relationships.
- **Output Layer:** Contains a single node, which outputs the prediction of the network.

Please refer to the accompanying diagram for a visual representation of the network's structure:

3.7.7 Hyperparameters

Hyperparameters are configurations external to the model that influence how the network is structured and trained. Hyperparameters play a crucial role in determining the model's performance by affecting how quickly and effectively it learns from the training data.

Table 3.2: Detailed Explanation of Variables

Hyperparameter	Description
Number of hidden layers	1
Optimizer	Stochastic gradient descent
Number of nodes in the hidden layer	4
Activation function of the hidden layer	ReLU function
Activation function of the output layer	Sigmoid function
Loss function	Root mean squared error
Learning rate	1
Number of epochs	1

3.8 Example: Mapping Function in Neural Network

This example illustrates how a specific input maps through a neural network to produce an output. The neural network is structured with three input nodes and one output node. The inputs for this example are chosen with feature dimensions of 2, 1, and 3, respectively.

Input Layer: The network receives a single training point with features:

- $x_1 = 2$
- $x_2 = 1$
- $x_3 = 3$

These inputs are fed into the network to process through the neural architecture.

3.8.1 Forward Propagation

Forward Propagation involves processing input data through the network from the input to the output layer using current weights and biases, generating predictions that are used to calculate the error against actual targets.

1. **Input to Hidden Layer:**

   Let $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, where each x_i is an input feature.

   - Here, $\mathbf{x}$ represents the input vector to the network, consisting of features x_1, x_2, and x_3. These features are the data points that you want the network to learn from.

   Let $\mathbf{W}^{[1]} = \begin{bmatrix} w_{1,1}^{[1]} & w_{1,2}^{[1]} & w_{1,3}^{[1]} \\ w_{2,1}^{[1]} & w_{2,2}^{[1]} & w_{2,3}^{[1]} \\ w_{3,1}^{[1]} & w_{3,2}^{[1]} & w_{3,3}^{[1]} \\ w_{4,1}^{[1]} & w_{4,2}^{[1]} & w_{4,3}^{[1]} \end{bmatrix}$, be the weight matrix connecting the input to the hidden layer.

   - $\mathbf{W}^{[1]}$ is the weight matrix associated with the first layer. Each element, such as $w_{1,1}^{[1]}$, represents the weight connecting the input node x_1 to the first neuron in the hidden layer.

   Let $\mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$, for the hidden layer.

   - $\mathbf{b}^{[1]}$ represents the bias vector for the hidden layer, with each entry like $b_1^{[1]}$ adding a bias term to the corresponding neuron's output. This helps to adjust the threshold at which the neuron activates.

   $\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$

   - This equation computes the linear combination of inputs and weights, adjusted by the bias. The result, $\mathbf{z}^{[1]}$, is the pre-activation output of the hidden layer.

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{[1]} & w_{1,2}^{[1]} & w_{1,3}^{[1]} \\ w_{2,1}^{[1]} & w_{2,2}^{[1]} & w_{2,3}^{[1]} \\ w_{3,1}^{[1]} & w_{3,2}^{[1]} & w_{3,3}^{[1]} \\ w_{4,1}^{[1]} & w_{4,2}^{[1]} & w_{4,3}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- This step shows the explicit matrix multiplication and addition for the given example. It is a practical computation where each neuron's input is the sum of products of each input feature and the corresponding weight plus a bias term.

$$\mathbf{z}^{[1]} = \begin{bmatrix} 6 \\ 6 \\ 6 \\ 6 \end{bmatrix}$$

- The result is a vector of pre-activation values for each neuron in the hidden layer.

2. **Activation in Hidden Layer:**

$\mathbf{a}^{[1]} = \text{ReLU}(\mathbf{z}^{[1]})$

- The ReLU function is applied to each pre-activation value. This non-linear function outputs the input directly if it is positive; otherwise, it outputs zero.

$$\begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \text{ReLU}\left(\begin{bmatrix} 6 \\ 6 \\ 6 \\ 6 \end{bmatrix} \right)$$

- Since all inputs are positive (6), the ReLU output matches the input. This vector represents the activated output of the hidden layer.

$$\mathbf{a}^{[1]} = \begin{bmatrix} 6 \\ 6 \\ 6 \\ 6 \end{bmatrix}$$

- These are the activation values from the hidden layer that will be fed to the next layer.

3. **Hidden Layer to Output Layer:**

Let $\mathbf{W}^{[2]} = \begin{bmatrix} w_{1,1}^{[2]} & w_{1,2}^{[2]} & w_{1,3}^{[2]} & w_{1,4}^{[2]} \end{bmatrix}$ and $\mathbf{b}^{[2]} = \begin{bmatrix} b_1^{[2]} \end{bmatrix}$

- $\mathbf{W}^{[2]}$ and $\mathbf{b}^{[2]}$ are the weight vector and bias for the output layer. Here, we're transitioning from a hidden layer with multiple neurons to an output layer with potentially one neuron.

$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$

- This equation calculates the linear combination of the activated outputs from the hidden layer, weighed by $\mathbf{W}^{[2]}$, and adjusted by the bias $\mathbf{b}^{[2]}$. The result is the input to the output layer's activation function.

$\mathbf{z}^{[2]} = \begin{bmatrix} 24 \end{bmatrix}$

- The computation is simplified to show the input to the output layer's activation function as 24.

4. **Activation in Output Layer:**

$a^{[2]} = \hat{y} = \sigma(\mathbf{z}^{[2]})$

- The sigmoid function σ is used at the output layer to map the input value into a (0,1) range, which is typical for binary classification tasks or probability estimation.

$\hat{y} = \sigma(\begin{bmatrix} 24 \end{bmatrix}) = 0.99$

- Given the high input value of 24, the sigmoid function outputs a value close to 1, indicating a high confidence level in the positive class, assuming a binary classification context.

3.8.2 Backward Propogation

Backward Propagation adjusts the network's parameters by calculating the loss function's gradient and updating weights and biases to minimize prediction errors, optimizing network performance over training iterations.

1. **Calculate Loss:**

$$L = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2}$$
$$L = \sqrt{(0.99 - 12)^2} = \sqrt{121.2201}$$
$$L = 11.01$$

- This formula calculates the Root Mean Squared Error (RMSE) between the predicted values ($\hat{y}_i$) and the actual values (y_i). Here, $\hat{y}_i = 0.99$ and $y_i = 12$.
- The RMSE provides a measure of how well the model is predicting the output, quantifying the difference in terms of the model's accuracy. In this case, an RMSE of 11.01 indicates a significant error, showing that the model's prediction is far from the actual value.

2. **Output to Hidden Layer:**

Using the chain rule, we first calculate the gradient of the loss function with respect to the output predictions:

$$\frac{\partial L}{\partial \hat{y}} = \frac{2}{m}(\hat{y} - y)\frac{1}{2\sqrt{\text{mean squared error}}}$$

For the next layer's pre-activation output, we derive the gradient with respect to the sigmoid function:

$$\frac{\partial L}{\partial z^{[2]}} = \frac{\partial L}{\partial \hat{y}} \cdot \sigma'(z^{[2]})$$

- $\sigma'(z^{[2]})$ is the derivative of the sigmoid activation function, applied to the pre-activation outputs at the output layer.

3. **Hidden Layer to Input Layer:**

   The gradients of the weights and biases are calculated as follows:

   $$\frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial z^{[2]}} \cdot a^{[1]}$$

   $$\frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial z^{[2]}}$$

   For the activations of the previous layer, we use the transpose of the weights:

   $$\frac{\partial L}{\partial a^{[1]}} = W^{[2]T} \cdot \frac{\partial L}{\partial z^{[2]}}$$

   The derivative through the ReLU function is computed next:

   $$\frac{\partial L}{\partial z^{[1]}} = \frac{\partial L}{\partial a^{[1]}} \cdot \text{ReLU}'(z^{[1]})$$

   - $\text{ReLU}'(z^{[1]})$ is the derivative of the ReLU function, which is 1 for positive inputs and 0 otherwise.

4. **Update Parameters:**

   Finally, we update the weights and biases using the calculated gradients and a learning rate α:

   $$W^{[l]} = W^{[l]} - \alpha \cdot \frac{\partial L}{\partial W^{[l]}}$$

   $$b^{[l]} = b^{[l]} - \alpha \cdot \frac{\partial L}{\partial b^{[l]}}$$

   - These updates adjust the weights and biases to minimize the loss, thereby improving the model with each iteration.

Chapter 4

Literature Review

The database review serves as an overview of the review conducted before my project, which is organized into 6 columns: title, author(s), year, keywords, main findings, and the relevance to the project. This has 4 distinct types of data that will be color-coded: literature (red), code (blue), data (orange), miscellaneous (green), in which literature will be the only type of data to summarize the main findings using the CRAAP test. The data will be hyperlinked (if possible) or have the corresponding file name & folder, which can be referenced using the readme file. Any of the notes that I create may not be put into the database.

4.1 Research trends

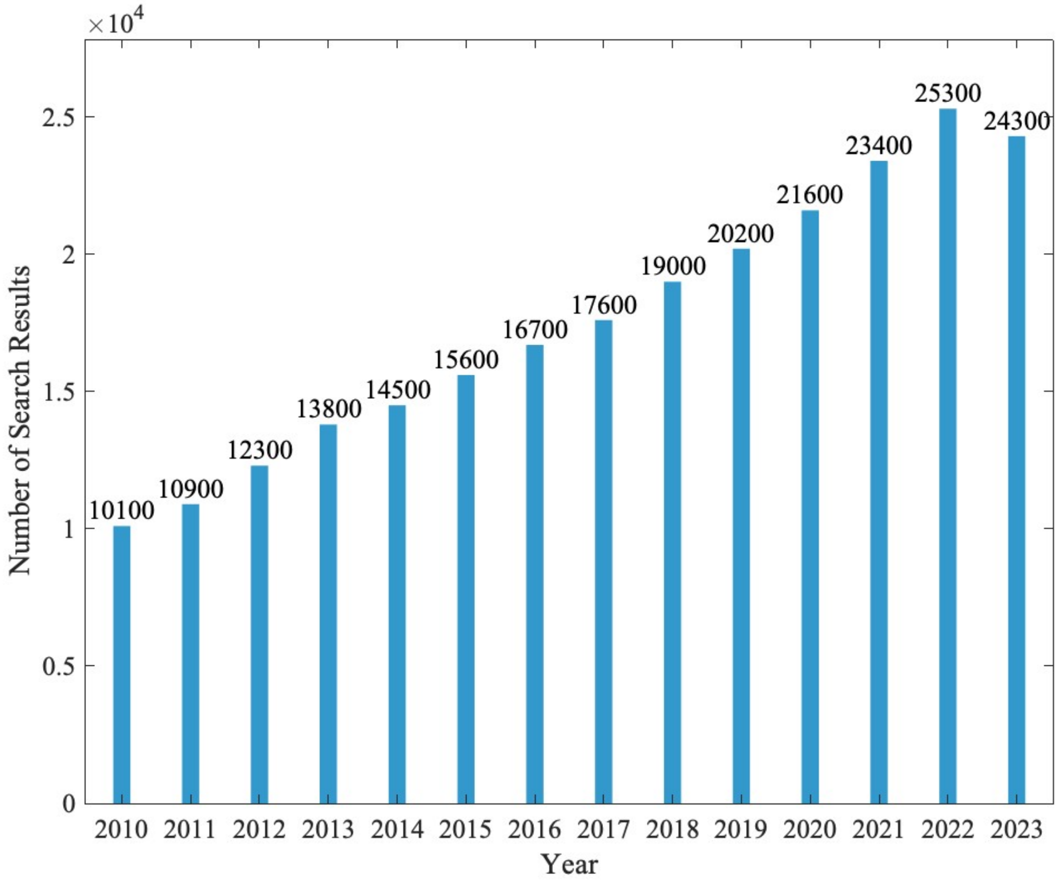

Figure 4.1: This is an example image.

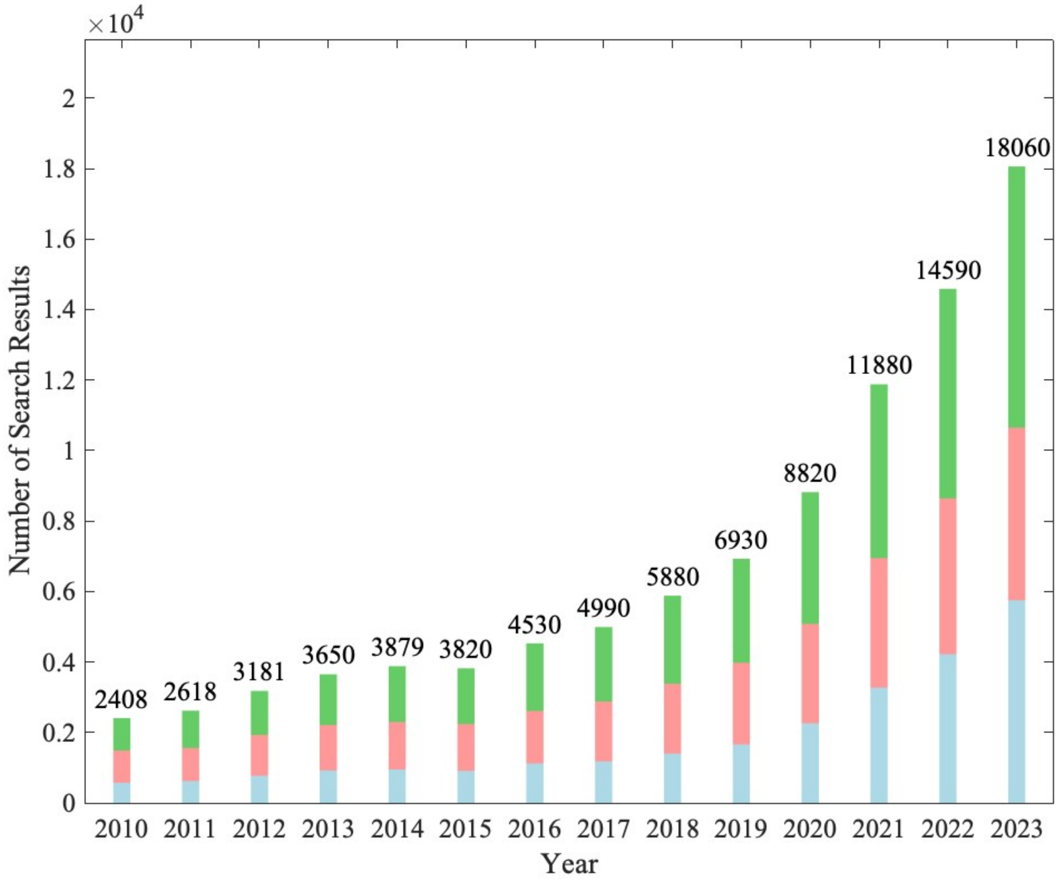

Figure 4.2: This is an example image.

4.2 Relevant literature

Since the previous work of Yong Rui Than provides a comprehensive overview of the literature on PEMFC involving machine learning. My review focused on the literature that was not covered in the previous work (i.e. 2021 - 2024). This will ensure that the literature review is up-to-date and relevant to the current state of the field.

Title	**Author**	**Summary**
Deep learning from three-dimensional multiphysics simulation in operational optimization and control of polymer electrolyte membrane fuel cell for maximum power	Tian et al.	This study integrates an artificial neural network (ANN) with a genetic algorithm (GA) to optimize the operational control of polymer electrolyte membrane fuel cells (PEMFCs) for maximum power output. Utilizing a 3D multiphysics model, 1500 data points were generated to train the ANN, which identified a peak power density of 0.78 W/cm^2 at 368.8 K. The ANN-GA method effectively predicted performance and optimized conditions across temperatures from 323 K to 373 K, proving crucial for practical system design and rapid control.
Predicting optimal membrane hydration and ohmic losses in operating fuel cells with machine learning	Paciocco et al.	This study pioneers the use of machine learning to predict optimal membrane hydration in polymer electrolyte membrane (PEM) fuel cells, focusing on high frequency resistance (HFR) and optimal hydration current density (OHCD). The long short-term memory recurrent neural network model achieved a mean absolute percentage error of 3.11% and an R^2 of over 0.95 in predicting HFR, while deep learning and k-nearest neighbors models predicted OHCD with over 98% precision and recall. These models provide robust tools for real-time parameter estimation and control, enhancing the performance and reliability of PEM fuel cells and other electrochemical devices.
Pre-diagnosis of flooding and drying in proton exchange membrane fuel cells by bagging ensemble deep learning models using long short-term memory and convolutional neural networks	Kim et al.	This study develops a pre-diagnosis system for detecting flooding and drying in polymer electrolyte membrane fuel cells (PEMFCs) using deep learning models. The system employs long short-term memory (LSTM) and convolutional neural networks (CNN) reinforced by a bagging ensemble method, achieving detection rates of 98.52% for flooding and 95.36% for drying within a 30-second prediction window. Experimental data from full-scale single-cell tests, including output voltage, relative humidity, and cell temperature, were used to train the model, highlighting its potential for enhancing PEMFC stability through early fault detection.
Machine learning modeling for proton exchange membrane fuel cell performance	Legala et al.	This study utilizes various machine learning techniques, including Artificial Neural Networks (ANN) and Support Vector Machine Regressor (SVR), to model proton exchange membrane fuel cell (PEMFC) performance and internal states. The ANN model, incorporating the dropout technique, achieved an $R^2 \geq 0.99$ for predicting cell voltage, membrane resistance, and hydration level across different operating conditions, outperforming SVR in multivariable regression tasks. The models were developed using data from a physics-based semi-empirical model and a 1-D reduced-dimension Computational Fluid Dynamics model, demonstrating that advanced machine learning can create accurate predictive models without extensive physical experimentation.

The table continues on the next page...

Title	**Author**	**Summary**
Analysis and modeling of high-performance polymer electrolyte membrane electrolyzers by machine learning	Gunay et al.	This study employs box and whisker plots, principal component analysis (PCA), and classification and regression tree modeling to analyze and model high-performance polymer electrolyte membrane (PEM) electrolyzers using a database of 789 data points from 30 recent publications. The PCA identified performance risks associated with specific materials, such as cathode surfaces with high Ni content and anode surfaces with cobalt-iron alloys or RuO2. Classification trees highlighted current density, potential, and mole fractions of Ni and Co as key performance variables, while the regression tree technique accurately modeled polarization behavior with an RMSE of 0.18.
Machine learning optimization of operating parameters to achieve high power density and efficiency of polymer electrolyte membrane fuel cell	Kaiser et al.	This study employs five machine learning regression models—Artificial Neural Network (ANN), Decision Tree (DT), Random Forest (RF), Gradient Boosting (GB), and Extreme Gradient Boosting (xGB)—to optimize the operating parameters of polymer electrolyte membrane fuel cells (PEMFCs) for maximum power density and efficiency. Among these models, the GB regressor demonstrated the highest accuracy with an $R^2 = 0.973$ and an $MSE = 0.0088$ for testing data, excelling in replicating experimental polarization curves. The optimal operating conditions identified were anode and cathode pressures at 2 bar and cathode stoichiometry between 2.50–2.75, significantly enhancing PEMFC performance. The study highlights the GB model's superior capability in handling extensive datasets and optimizing complex nonlinear operational conditions, thus offering a novel approach for PEMFC performance optimization.
A Review of physics-based and data-driven models for real-time control of polymer electrolyte membrane fuel cells	Zhao et al.	This review critically examines recent advancements in physics-based and data-driven models for the real-time control of polymer electrolyte membrane (PEM) fuel cells. Emphasizing the need for models that balance accuracy and computational efficiency, the study highlights trends such as coupling single cell models with balance-of-plant systems, incorporating aging effects for long-term predictions, and leveraging artificial intelligence algorithms for enhanced computational speed. Among the models reviewed, the study notes the superior accuracy and speed of data-driven models, particularly when trained on extensive datasets, while also addressing challenges in their applicability across diverse operational conditions.
Application of Machine Learning in Optimizing Proton Exchange Membrane Fuel Cells: A Review	Ding et al.	This review explores the application of machine learning (ML) in optimizing proton exchange membrane fuel cells (PEMFCs) to address their cost and performance challenges for large-scale commercialization. The study highlights ML's capability to reduce experimental and computational costs by predicting outcomes based on datasets from experiments or theoretical simulations. Notable ML applications in this field include predicting active electrocatalysts, optimizing membrane electrode assemblies (MEA), designing efficient flow channels, and developing stack operation strategies.

The table continues on the next page...

Title	**Author**	**Summary**
Towards Reliable Prediction of Performance for Polymer Electrolyte Membrane Fuel Cells via Machine Learning-Integrated Hybrid Numerical Simulations	Kaiser et al.	This study explores hybrid numerical simulations integrating machine learning (ML) to enhance the reliability and accuracy of polymer electrolyte membrane fuel cell (PEMFC) performance predictions. By addressing the limitations of existing computational fluid dynamics (CFD) models, which suffer from simplifications and inaccurate parameter approximations, the study demonstrates how ML can provide more appropriate parameters, leading to improved electrochemistry, mass/species transfer, thermal management, and water transport modeling. The ML-assisted CFD models are shown to optimize component design and material properties, significantly enhancing PEMFC efficiency and providing a robust framework for future PEMFC development and its impact on the transportation sector.
An Optimized Data Analysis on a Real-Time Application of PEM Fuel Cell Design by Using Machine Learning Algorithms	Saco et al.	This study applies machine learning algorithms to optimize the design of polymer electrolyte membrane fuel cells (PEMFCs) by analyzing their performance under different humidity levels. Three machine learning models—Support Vector Machine Regressor (SVMR), Linear Regression (LR), and k-Nearest Neighbors (KNN)—were evaluated for their predictive accuracy using root mean square error (RMSE) as the metric. The LR model demonstrated superior performance with an RMSE of 0.0034, compared to 0.0046 for SVMR and 0.004 for KNN, effectively enhancing PEMFC efficiency through optimal humidification conditions.
A systematic review of machine learning methods applied to fuel cells in performance evaluation, durability prediction, and application monitoring	Ming et al.	This systematic review explores the application of machine learning (ML) methods, including traditional ML and deep learning (DL), to fuel cells for performance evaluation, durability prediction, and application monitoring. The review highlights ML's effectiveness in addressing complex phenomena such as mass/heat transfer and electrochemical reactions, crucial for improving energy efficiency and durability. Notably, ML models excel in material selection, chemical reaction modeling, polarization curves, state of health monitoring, fault diagnostics, and predicting remaining useful life. The study also compares ML and DL methods, and their integration with physics simulations, providing a comprehensive outlook on future research directions for ML applications in fuel cells.
Application of Machine Learning in Fuel Cell Research	Su et al.	This comprehensive review examines the application of machine learning (ML) algorithms in optimizing proton exchange membrane fuel cells (PEMFCs), focusing on performance prediction, service life estimation, and fault diagnosis. The study highlights the effectiveness of ML models, such as artificial neural networks (ANN), support vector machines (SVM), and random forests (RF), in solving nonlinear problems associated with PEMFCs. Notable findings include an RMSE of 0.0034 for linear regression in performance prediction, a 99% accuracy rate in power-current curve modeling using SVM, and high precision in fault diagnosis and service life prediction.

The table continues on the next page...

Title	**Author**	**Summary**
Optimization of Proton Exchange Membrane Electrolyzer Cell Design Using Machine Learning	Mohamed et al.	This study leverages machine learning (ML) models, specifically polynomial and logistic regression, to optimize the design of proton exchange membrane (PEM) electrolyzer cells. By predicting eleven parameters of cell components based on four input parameters (hydrogen production rate, cathode area, anode area, and cell design type), the models achieved an average accuracy of 83.6% and a mean absolute error (MAE) of 6.825. Validated on a test set, the ML models demonstrated excellent agreement with experimental results, showing a negligible MAE of 0.615 in hydrogen production rate predictions. The study successfully designed optimal PEM electrolyzer cells for commercial-scale hydrogen production rates (500 to 5000 mL/min), highlighting the potential of ML to significantly reduce the cost and time required for developing efficient water electrolyzers.

Table 4.1: Database review of literature on PEMFC and machine learning from 2021 to 2024

Chapter 5

Materials and Methods

Header	**Symbol**	**Explanation**	**Unit**
		Input Parameters	
Q	Q	**Heat generation** influences membrane hydration and operational efficiency; critical to prevent membrane dry-out and maintain ion conductivity.	Wm^{-2}
Tamb	T_{amb}	**Ambient temperature of the air** sets baseline thermal conditions; higher temperatures boost performance to a limit before causing potential overheating.	$^{\circ}C$
Uin	U_{in}	**Airflow velocity** determines oxygen supply rate, essential for maintaining optimal reaction rates and power output.	m/s
Wcc	W_{cc}	**Cathode channel width** determines oxygen flow and diffusion rates to the cathode, crucial for optimizing reaction efficiency.	mm
Hcc	H_{cc}	**Cathode channel height** affects gas flow resistance and water removal, essential for maintaining membrane hydration and preventing flooding.	mm
Lcc	L_{cc}	**Cathode channel length** influences the residence time of reactants and products along the channel, impacting overall fuel cell efficiency.	mm
Wr	W_r	**Rib width** supports mechanical stability and maximizes the active area available for reactions, balancing structural support with performance.	mm
Hr	H_r	**Rib height** controls the depth of flow channels, enhancing reactant distribution and efficient water management within the stack.	mm
		Output Performance Metrics	
Tsta	T_{stack}	**Stack temperature** is crucial for optimal reaction rates, membrane hydration, and to prolong the lifespan.	$^{\circ}C$
Delp	Δp	**Pressure drop** indicates the system's resistance to reactant flow; minimizing pressure drop is essential to enhance efficiency and ensure uniform distribution.	Pa

Table 5.1: Detailed Explanation of Variables

Chapter 6

COMSOL Model

6.1 Model Assumptions

1. The airflow entering the channel is turbulent. The Reynolds number exceeds 2300 at an inlet velocity (U_{in}) of 0.3 m/s.
2. Portions of the fuel cell not part of the cathode flow field are impermeable to air. While carbon paper is porous, the in-plane pressure drop is much greater than that of the cathode flow field.
3. The cathode flow field channels are modeled with a uniform height. The average offset (0.025 mm) is negligible compared to the channel height (1 mm).
4. Cathode flow fields are modeled with straight edges and right-angle folds. The fold radius (0.05 mm) is small compared to the channel width (1 mm).
5. Conservation of mass and momentum of airflow is assumed to be valid when transitioning between open spaces and cathode channels.
6. A representative unit cell approximates the entire stack, excluding edge effects near the terminals. Airflow entering the unit cell is uniform and symmetrical.
7. Channels are identical throughout the stack, resulting in no pressure drop differences between channels.
8. The channels are fully dry without condensation. The oxidant stoichiometry exceeds 100, ensuring water vapor produced is absorbed by the airflow, negating the need for accounting water saturation.
9. The losses from the fuel cell operation are converted to heat.

6.2 Workflow - Pressure Drop

The following steps outline the workflow process for the unit cell simulations conducted in COMSOL:

1. **Selection of Laminar Flow Model:**
   - This model was chosen because the airflow within the channels is laminar and is the largest contributor to the pressure drop across the stack.
   - The laminar flow model allows for faster convergence and has been shown to be accurate when compared with experimental results.
2. **Construction of Unit Cell:**
   - The unit cell was constructed using blocks with dimensions defined by equations to accommodate dimensional changes in the channels.

3. **Meshing Strategy:**
    - A custom mesh was employed, focusing on the transition areas between inflow and outflow within the channels with a finer mesh.
    - A coarser mesh was used within the channel and the air space between inflow and outflow.
    - A mesh independence study was performed during the mesh fine-tuning. The results are shown in Figure ?.
    - The difference between the blue dash (custom meshing) and green dots (physics-based extra fine meshing) is less than 3%, thus the blue dash mesh was selected for its speed and accuracy.
4. **Parameter Sweep Function:**
    - The parameter sweep function was utilized to explore all possible combinations for each stack size.
5. **Exporting Simulation Data:**
    - The inputs and outputs of the simulations were exported from COMSOL for further analysis.

6.3 Workflow - Temperature Stack

The following steps outline the workflow process for the unit cell simulations conducted in COMSOL:

1. **Selection of Physics Models:**
    - The unit cell simulations were done using laminar flow physics as well as heat transfer in laminar flow.
    - The parameter sweep function was utilized to explore all possible combinations in the parameter space.
    - This model was selected because the flow through the channel is laminar and the simulation focuses on heat transfer within the channels.
2. **Construction of Unit Cell:**
    - The unit cell was constructed using blocks with dimensions defined by equations to accommodate dimensional changes in the channels.
3. **Heat Generation Simulation:**
    - Heat generation was simulated as a boundary heat source placed at the cathode side of the membrane.
4. **Heat Conduction Values:**
    - The heat conduction values for each layer in the fuel cell were obtained from various papers, as shown in Table 8-1.
5. **Meshing Strategy:**
    - A mesh independence study was conducted during the fine-tuning of the mesh. The results are shown in Figure ?.
    - The difference between the blue dash and green dots is less than 1.06%, hence the blue dash mesh was selected for its speed and accuracy.
6. **Exporting Simulation Data:**
    - The inputs and outputs of the simulations were exported from COMSOL for further analysis.

6.4 Parameters

Name	Value	Unit
Cell Area	0.005	m^2
Cell Voltage	0.6	V
Compression	0.1	-
Current	40	A
Hbp	5×10^{-5}	m
Hcc	1.25×10^{-3}	m
Hcp	3.15×10^{-4}	m
Hmem	1.5×10^{-5}	m
Kcp	1.5	W·m^{-1}K^{-1}
Kmem	0.1	W·m^{-1}K^{-1}
Lcc	0.03	m
pRef	1.0133×10^{5}	Pa
Q	5040	W·m^{-2}
Tamb	293.15	K
Test	25.2	W
Uin	7.75	m·s^{-1}
Wcc	0.001	m
Wr	5×10^{-5}	m

Table 6.1: Parameter Values

6.5 Materials

Name	Unit
Air	
Dynamic viscosity	$\mathrm{Pa \cdot s}$
Ratio of specific heats	-
Heat capacity at constant pressure	$\mathrm{J/(kg \cdot K)}$
Density	$\mathrm{kg/m^3}$
Thermal conductivity	$\mathrm{W/(m \cdot K)}$
Carbon Paper	
Density	$\mathrm{kg/m^3}$
Heat capacity at constant pressure	$\mathrm{J/(kg \cdot K)}$
Thermal conductivity	$\mathrm{W/(m \cdot K)}$
Membrane	
Thermal conductivity	$\mathrm{W/(m \cdot K)}$
Steel Grade 316L	
Density	$\mathrm{kg/m^3}$
Heat capacity at constant pressure	$\mathrm{J/(kg \cdot K)}$
Thermal conductivity	$\mathrm{W/(m \cdot K)}$

Table 6.2: Material Descriptions

6.6 Laminar Flow

6.6.1 Governing Equations

$$\rho(\mathbf{u} \cdot \nabla)\mathbf{u} = \nabla \cdot [-p\mathbf{I} + \mathbf{K}] + \mathbf{F} \tag{6.1}$$

where ρ is the density of air, **u** is the velocity vector, p is the pressure, **I** is the identity matrix, **K** is the viscous stress tensor, and **F** is the volume force vector.

$$\nabla \cdot (\rho\mathbf{u}) = 0 \tag{6.2}$$

$$\mathbf{K} = \mu(\nabla\mathbf{u} + (\nabla\mathbf{u})^T) - \frac{2}{3}\mu(\nabla \cdot \mathbf{u})\mathbf{I} \tag{6.3}$$

where μ is the dynamic viscosity of air.

6.6.2 Initial Values

1. Velocity field

$$\mathbf{u} = \mathbf{0} \tag{6.4}$$

2. Pressure

$$p = 0 \tag{6.5}$$

6.6.3 Boundary Conditions

1. Wall (No slip)

$$\mathbf{u} = \mathbf{0} \tag{6.6}$$

2. Inlet (Velocity)

$$\mathbf{u} = -U_0\mathbf{n} \tag{6.7}$$

where U_0 is the initial velocity (i.e. normal inflow velocity) and **n** is the unit normal vector.

3. Outlet (Pressure)

$$[-p\mathbf{I} + \mathbf{K}]\,\mathbf{n} = -\hat{p}_0\mathbf{n} \tag{6.8}$$

$$\hat{p_0} \leq p_0, \nabla\mathbf{G} \cdot \mathbf{n} = 0 \tag{6.9}$$

where $\hat{p}_0$ is the estimated standard condition pressure, $p_0 = 0$ is the standard condition pressure (i.e. suppress backflow), **G** is the reciprocal wall distance.

4. Symmetry

$$\mathbf{u} \cdot \mathbf{n} = 0 \tag{6.10}$$

$$\mathbf{K}_n - (\mathbf{K}_n \cdot \mathbf{n})\mathbf{n} = 0, \mathbf{K}_n = \mathbf{K}\mathbf{n} \tag{6.11}$$

6.7 Heat Transfer

6.7.1 Governing Equations - Solid

$$\rho C_p\mathbf{u} \cdot \nabla T + \nabla \cdot \mathbf{q} = Q + Q_{ted} \tag{6.12}$$

where C_p is the specific heat capacity of air, T is the temperature of air, **q** is the heat flux vector, Q is the heat source, and Q_{ted} is the thermoelastic damping heat source.

$$\mathbf{q} = -k\nabla T \tag{6.13}$$

where k is the thermal conductivity.

$$Q = (1.23 - Operating\ Voltage) \times \frac{Fuel\ Cell\ Stack\ Current}{Cell\ Area} \tag{6.14}$$

6.7.2 Governing Equations - Fluid

$$\rho C_p \mathbf{u} \cdot \nabla T + \nabla \cdot \mathbf{q} = Q + Q_p + Q_{vd} \tag{6.15}$$

where Q_p is the point heat source.

$$\mathbf{q} = -k\nabla T \tag{6.16}$$

6.7.3 Governing Equations - Thin Layer

$$-\mathbf{n}_d \cdot \mathbf{q}_u = \frac{(T_u - T_d)}{R_s} + \frac{1}{2} d_s Q_s \tag{6.17}$$

where R_s is the thermal resistance, and d_s is the thin layer thickness.

$$-\mathbf{n}_u \cdot \mathbf{q}_u = \frac{(T_d - T_u)}{R_s} + \frac{1}{2} d_s Q_s \tag{6.18}$$

$$R_s = \frac{d_s}{k_s} \tag{6.19}$$

6.7.4 Initial Values

1. Temperature

$$T = T_{amb} \tag{6.20}$$

   where T_{amb} is the initial ambient temperature.

6.7.5 Boundary Conditions

1. Insulation

$$-\mathbf{n} \cdot \mathbf{q} = 0 \tag{6.21}$$

2. Symmetry

$$-\mathbf{n} \cdot \mathbf{q} = 0 \tag{6.22}$$

3. Boundary Heat Source

$$-\mathbf{n} \cdot \mathbf{q} = Q_b \tag{6.23}$$

   where Q_b is the boundary heat source.

4. Outflow

$$-\mathbf{n} \cdot \mathbf{q} = 0 \tag{6.24}$$

5. Inlet

$$T = T_{amb} \tag{6.25}$$

6.8 Mesh

6.9 Study

6.9.1 Difference between External and Internal Sweep

An internal sweep involves varying parameters directly within the simulation model itself. This type of sweep is integrated into the simulation process, where the solver automatically adjusts the parameters as part of its internal algorithm. Internal sweeps are typically used to explore the parameter space of the model in a more automated and controlled manner, often leveraging the solver's built-in capabilities to iterate through different parameter values.

An external sweep, on the other hand, involves varying parameters outside of the simulation model. This is usually managed by an external script or control mechanism that systematically modifies the input parameters, runs the simulation for each set of parameters, and collects the results. External sweeps provide more flexibility and control over the parameter variations and can be useful for complex scenarios where the internal capabilities of the solver might be limited.

6.9.2 Parametric Sweep

A parametric sweep is a systematic method used in simulations and computational experiments to explore the effects of varying parameters within a defined range. By adjusting one or more input parameters incrementally, researchers can observe how changes influence the outcome, enabling a comprehensive analysis of the system's behavior. This technique is essential for optimizing designs, identifying critical factors, and understanding the sensitivity of the results to different variables. Ultimately, parametric sweeps provide valuable insights that drive informed decision-making and innovation.

Name	**Values**	**Unit**
Hcc	1 1.25 1.5 1.75 2	mm
Wcc	0.5 0.75 1 1.25 1.5	mm
Lcc	30 60 90	mm

Table 6.3: Parameter Sweep Values for All Combinations

6.9.3 Auxilary Sweep

An auxiliary sweep is a computational technique used to study the influence of secondary parameters or variables that are not directly part of the main parametric study. It involves varying these auxiliary parameters to investigate their impact on the primary simulation or experiment results. This approach helps in understanding the interactions and dependencies between primary and auxiliary parameters, providing a deeper insight into the system's overall behavior. Auxiliary sweeps are particularly useful in complex models where multiple factors may indirectly affect the outcomes, aiding in fine-tuning and enhancing the accuracy of simulations.

Name	**Values**	**Unit**
Uin	1 3.25 5.5 7.75 10	m/s
Q	1272 3132 5040	W/m^{-2}
Tamb	-20 0 20 40	°C

Table 6.4: Auxilary Sweep Values for All Combinations

6.10 Results

6.10.1 Accumulated Probe Table: Set 1

1. Hcc (1 - hc)
2. Wcc (2 - wc)
3. Lcc (3 - length)
4. Tamb (4 - Tamb)
5. Q (5 - Q)

6. Uin (6 - Uin)
7. Delp (15 - Pressure (Pa), Pressure Probe 3)

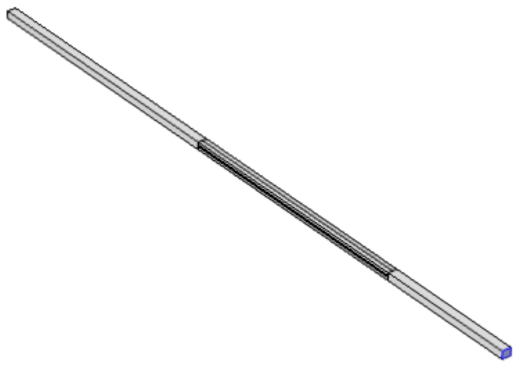

Figure 6.1: Pressure Probe 3 Location

8. Tsta (18 - Temperature (degC), Stack Temperature Probe 1)

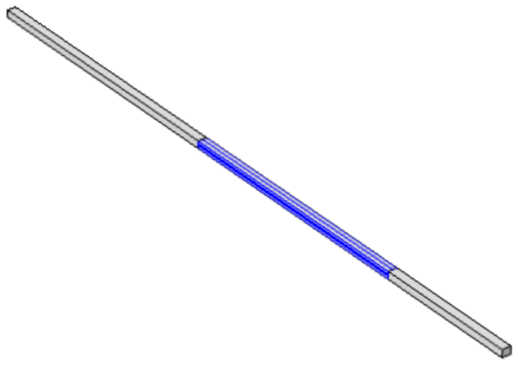

Figure 6.2: Stack Temperature Probe 1 Location

6.10.2 3D Plots

1. Pressure Drop

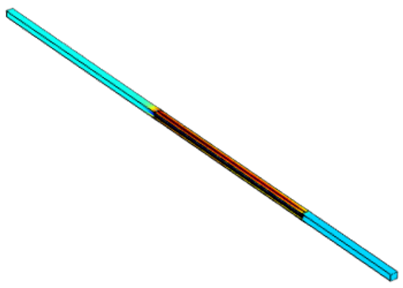

Figure 6.5: Velocity 3D Plot

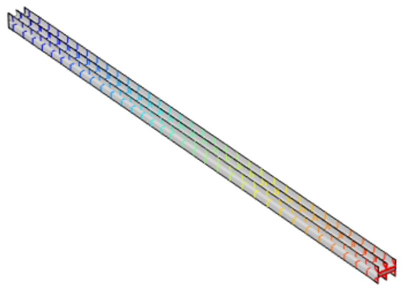

Figure 6.3: Pressure Drop 3D Plot

2. Temperature Stack

Figure 6.4: Temperature Stack 3D Plot

3. Velocity

Chapter 7

Dataset & Data Preprocessing

7.1 Data Preprocessing

7.1.1 Small Stack

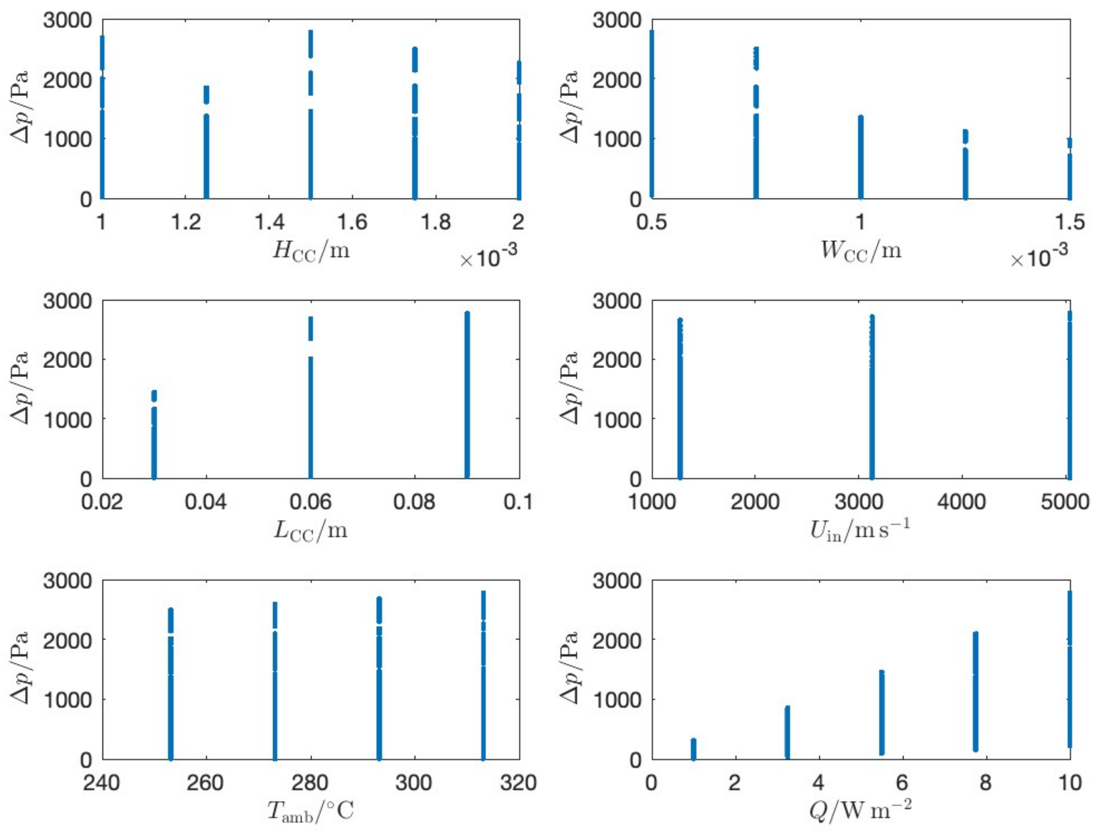

Figure 7.1: This is an example image.

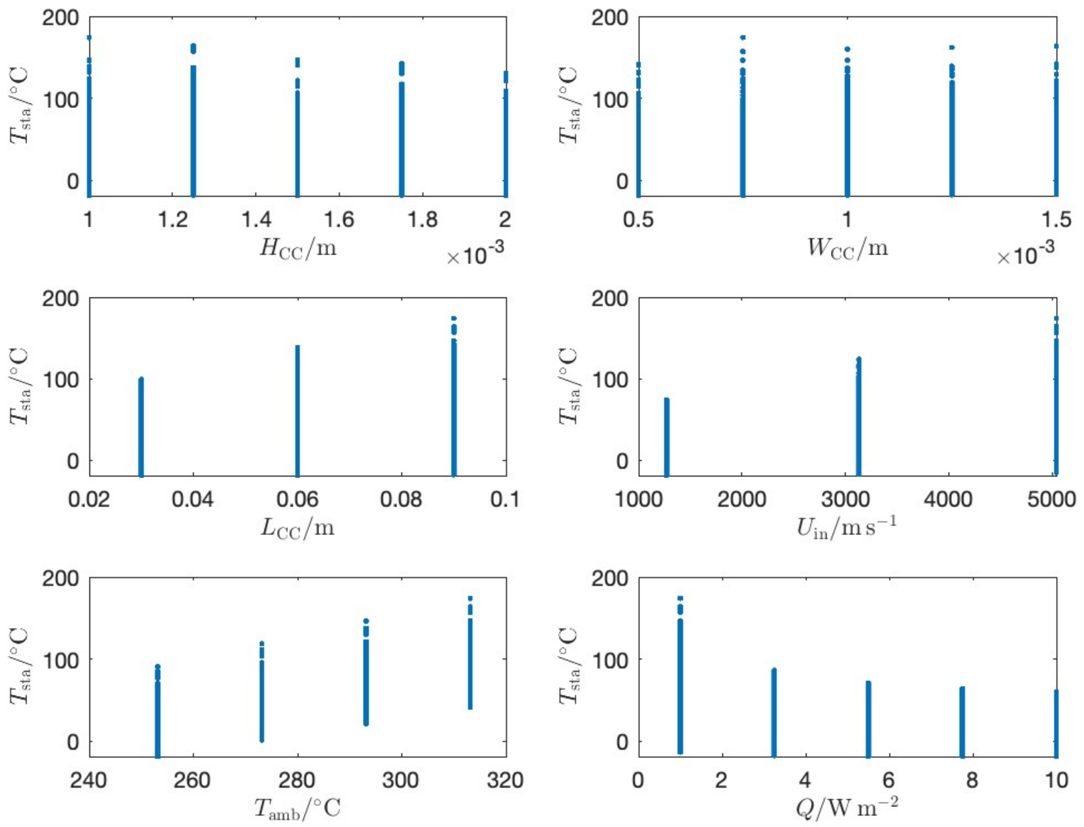

Figure 7.2: This is an example image.

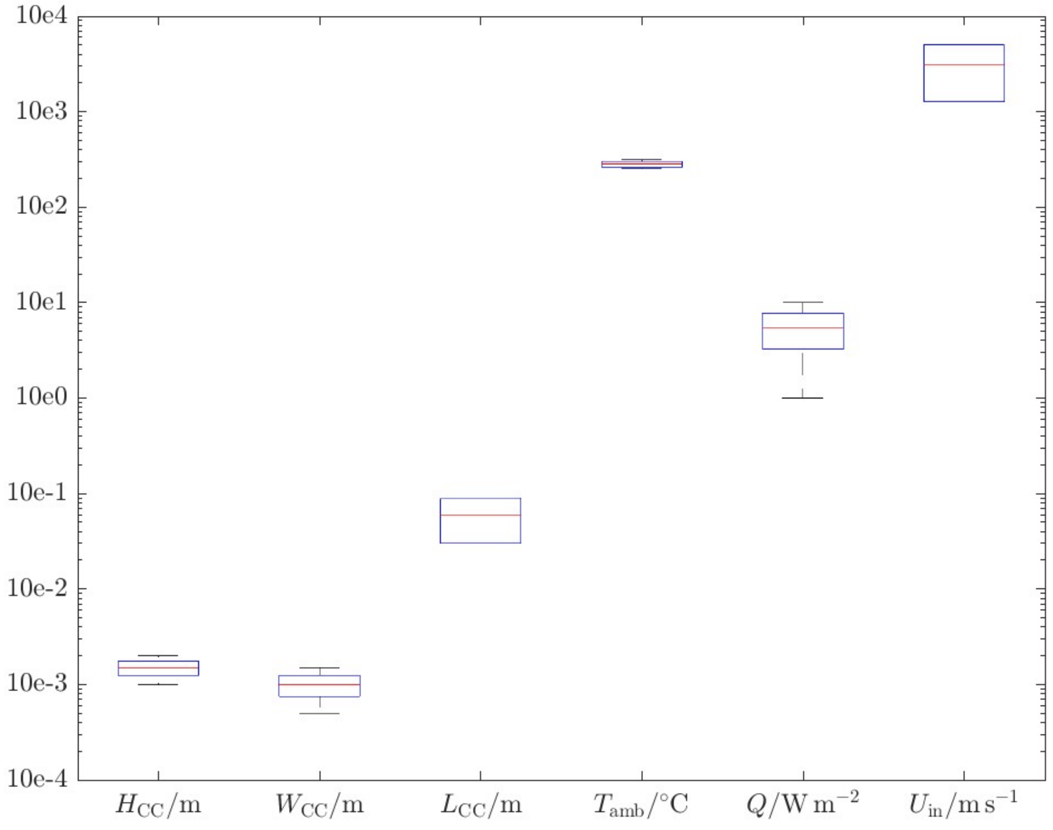

Figure 7.3: This is an example image.

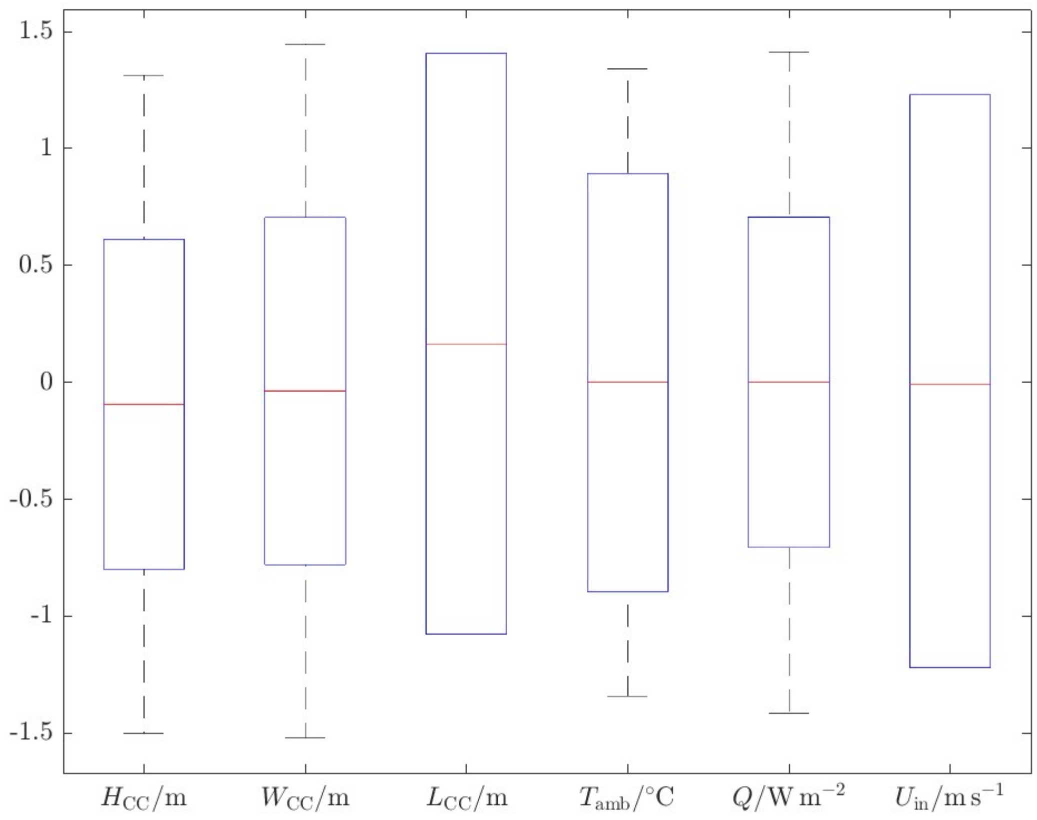

Figure 7.4: This is an example image.

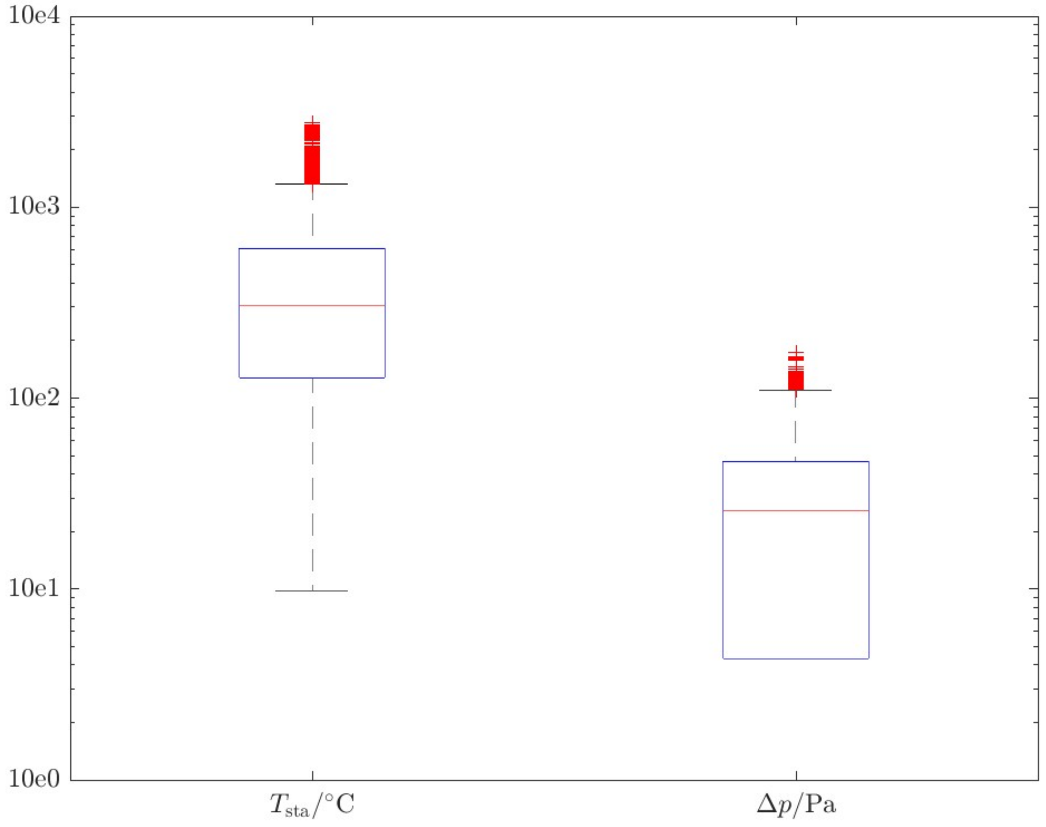

Figure 7.5: This is an example image.

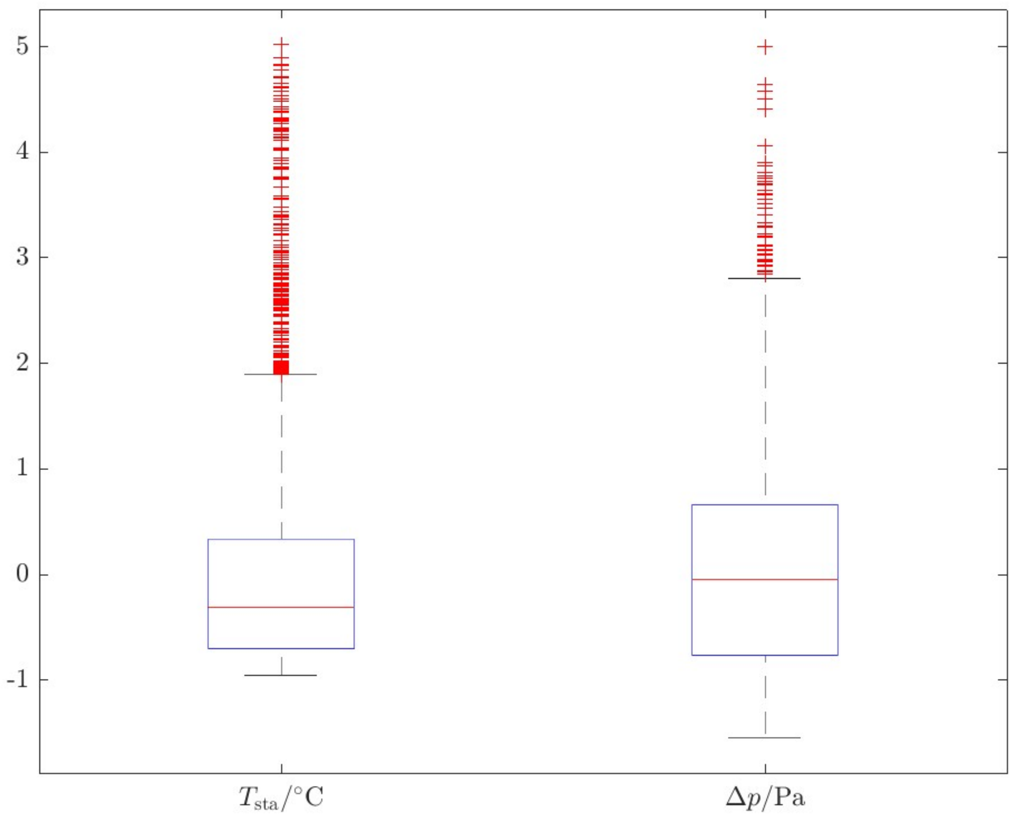

Figure 7.6: This is an example image.

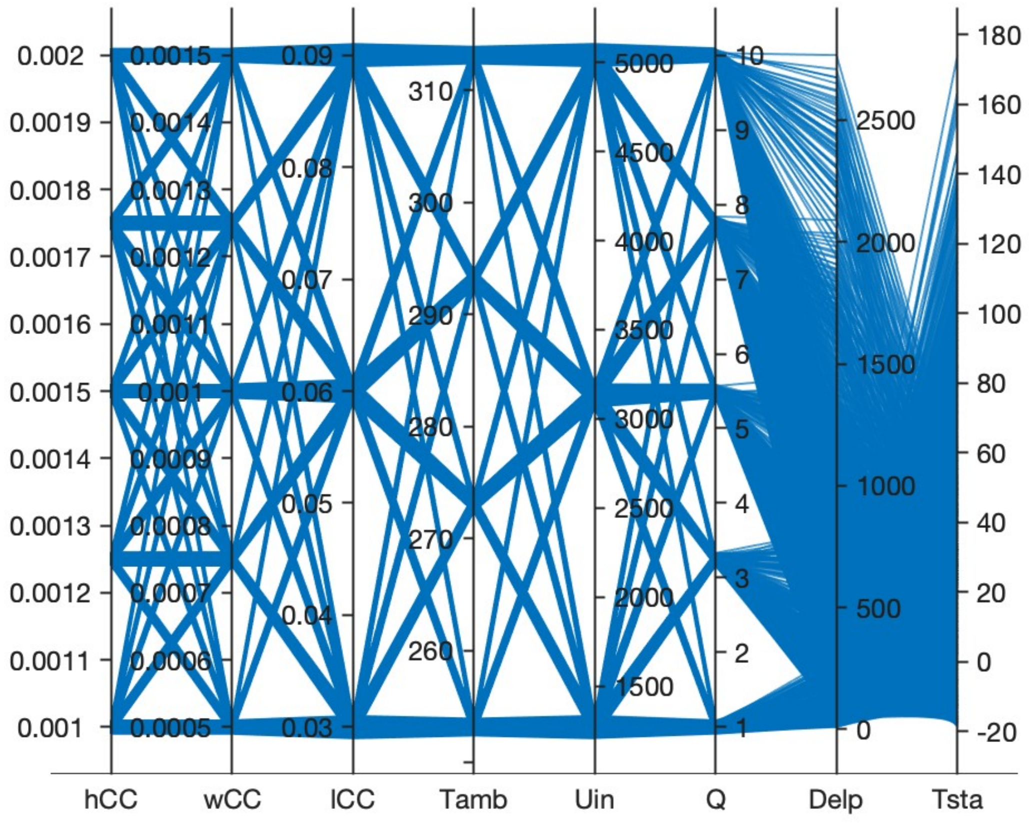

Figure 7.7: This is an example image.

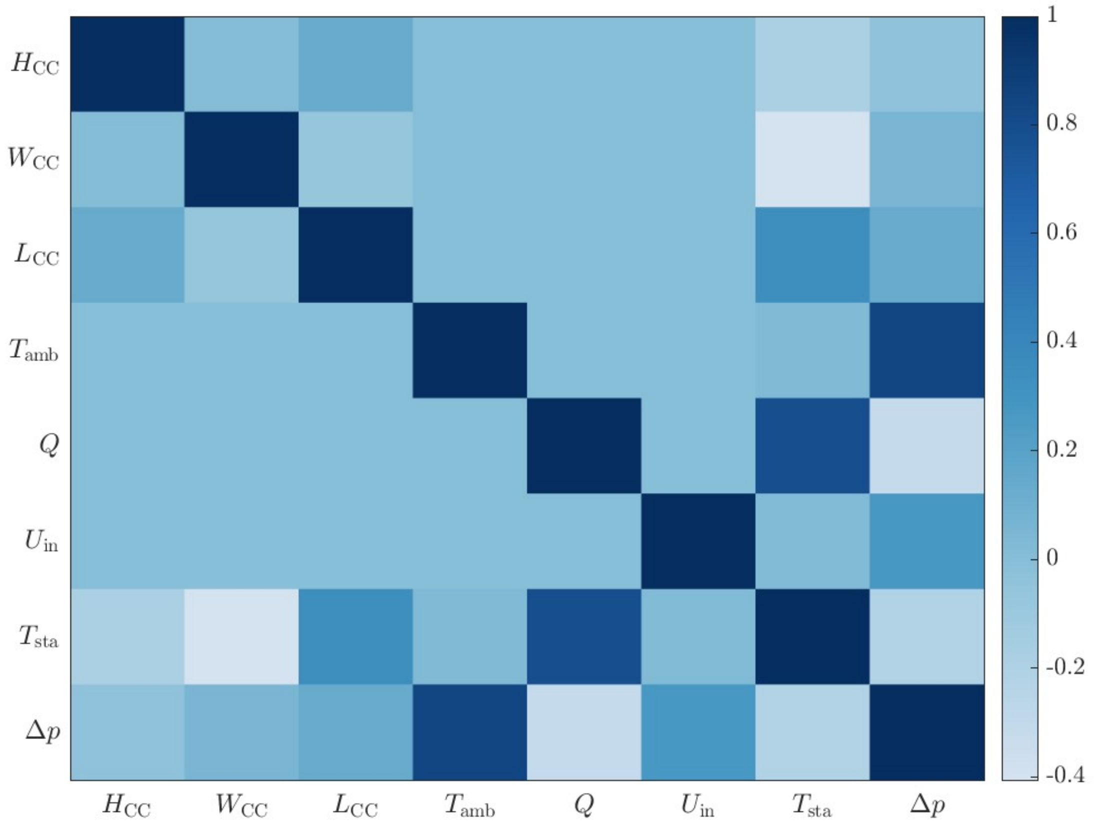

Figure 7.8: This is an example image.

7.1.2 Large Stack

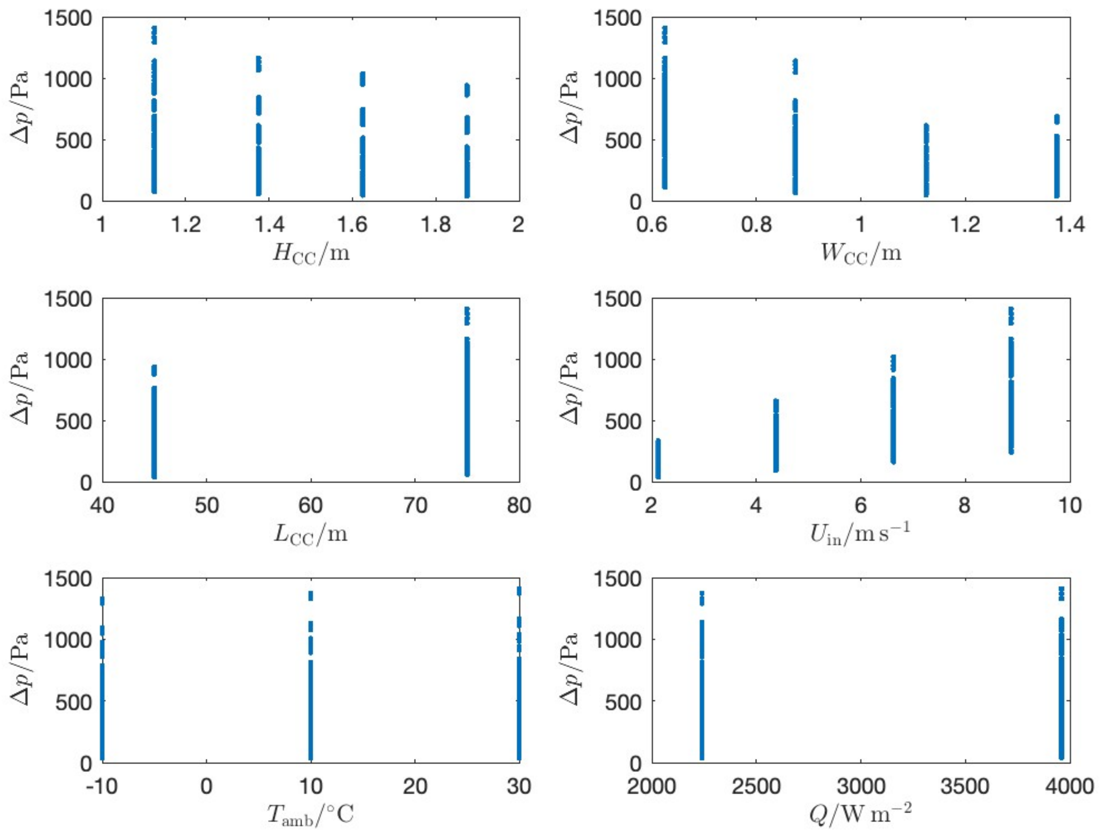

Figure 7.9: This is an example image.

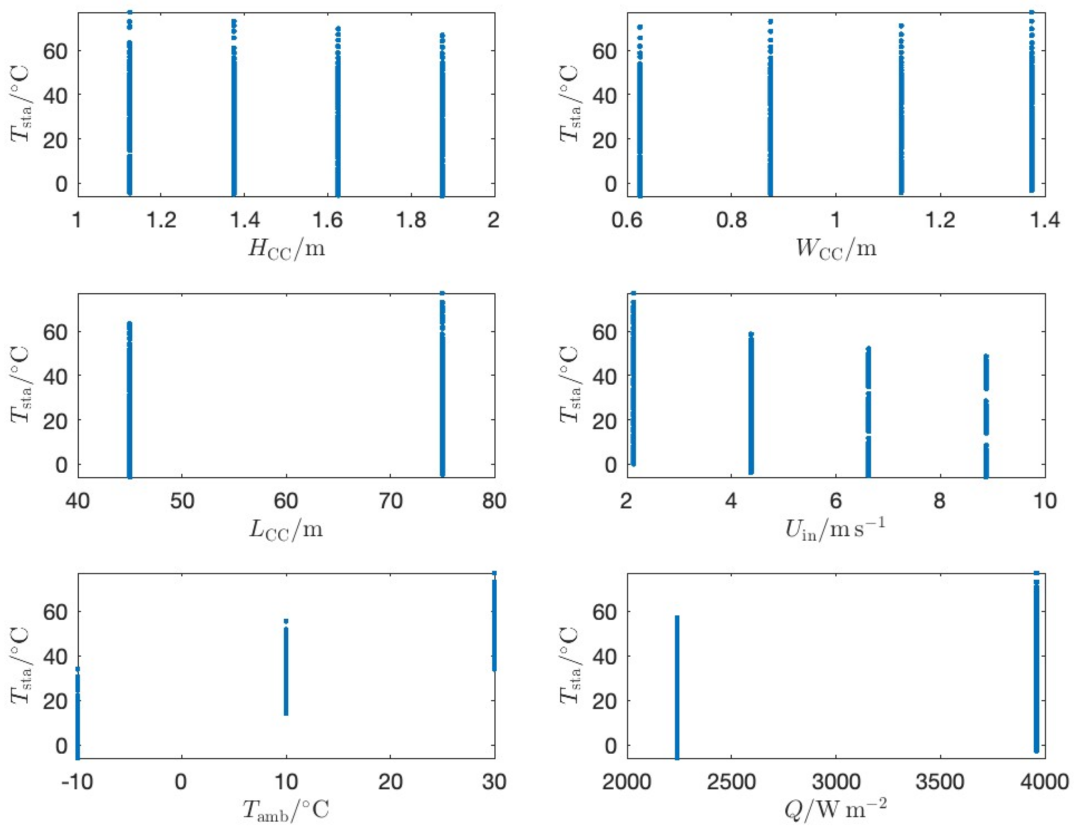

Figure 7.10: This is an example image.

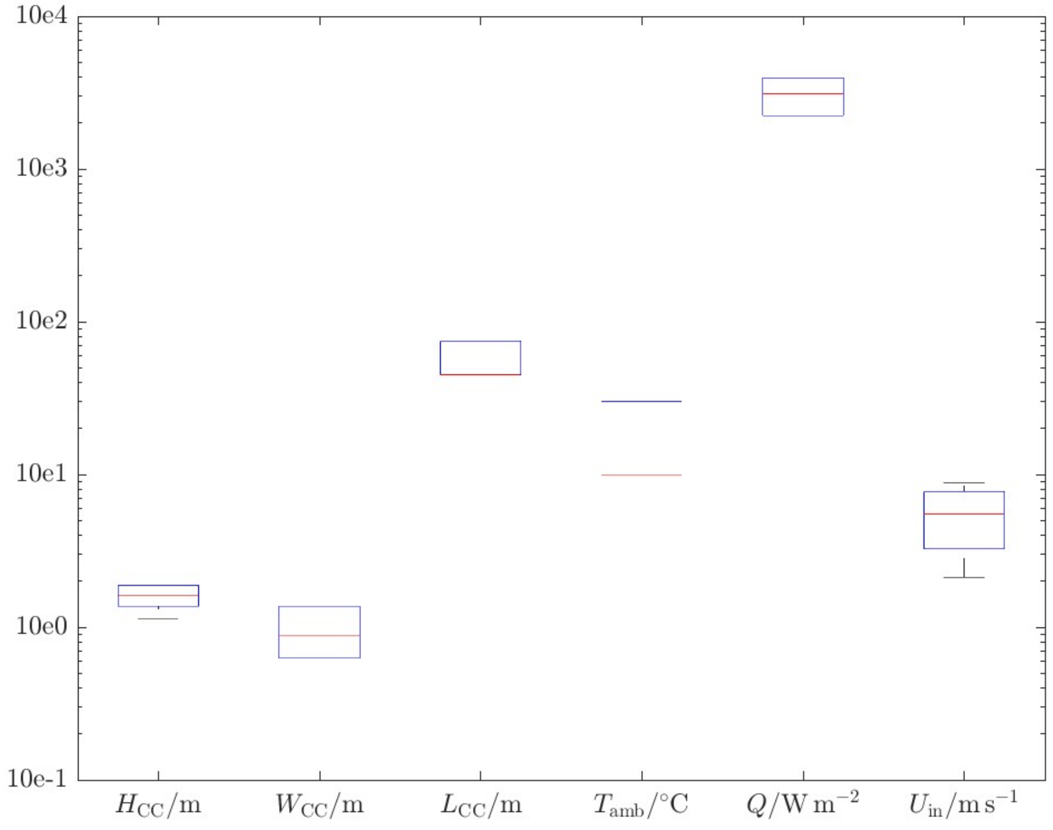

Figure 7.11: This is an example image.

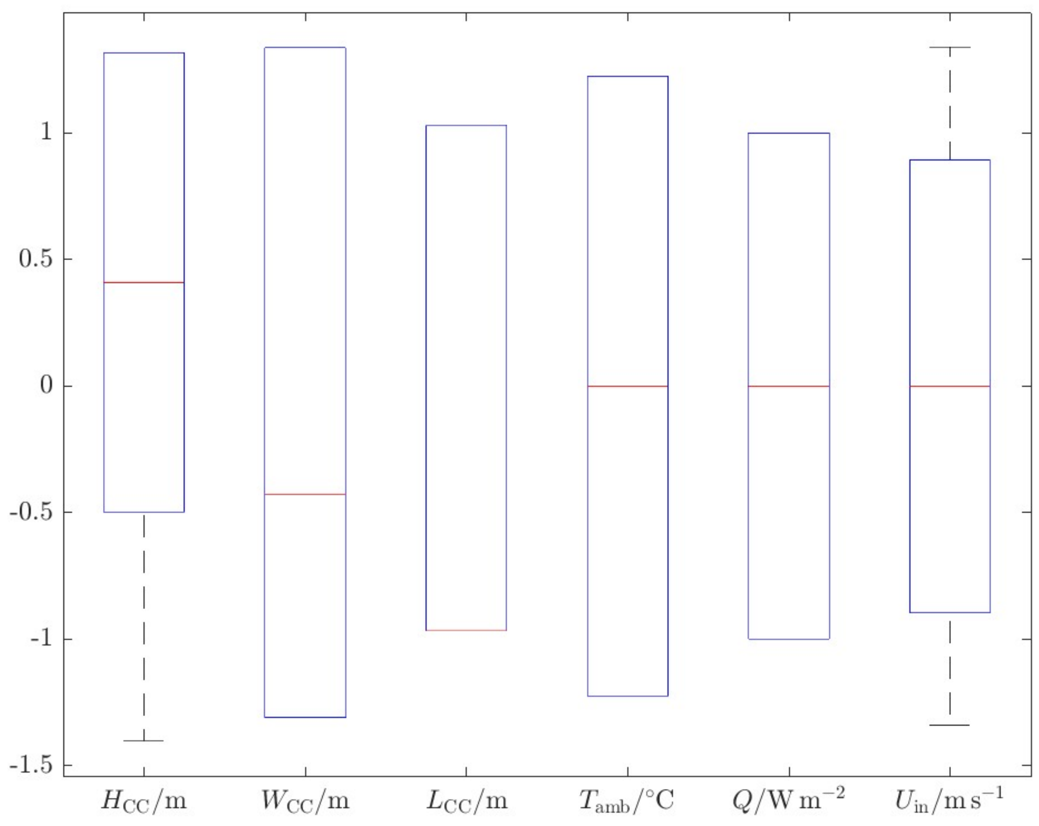

Figure 7.12: This is an example image.

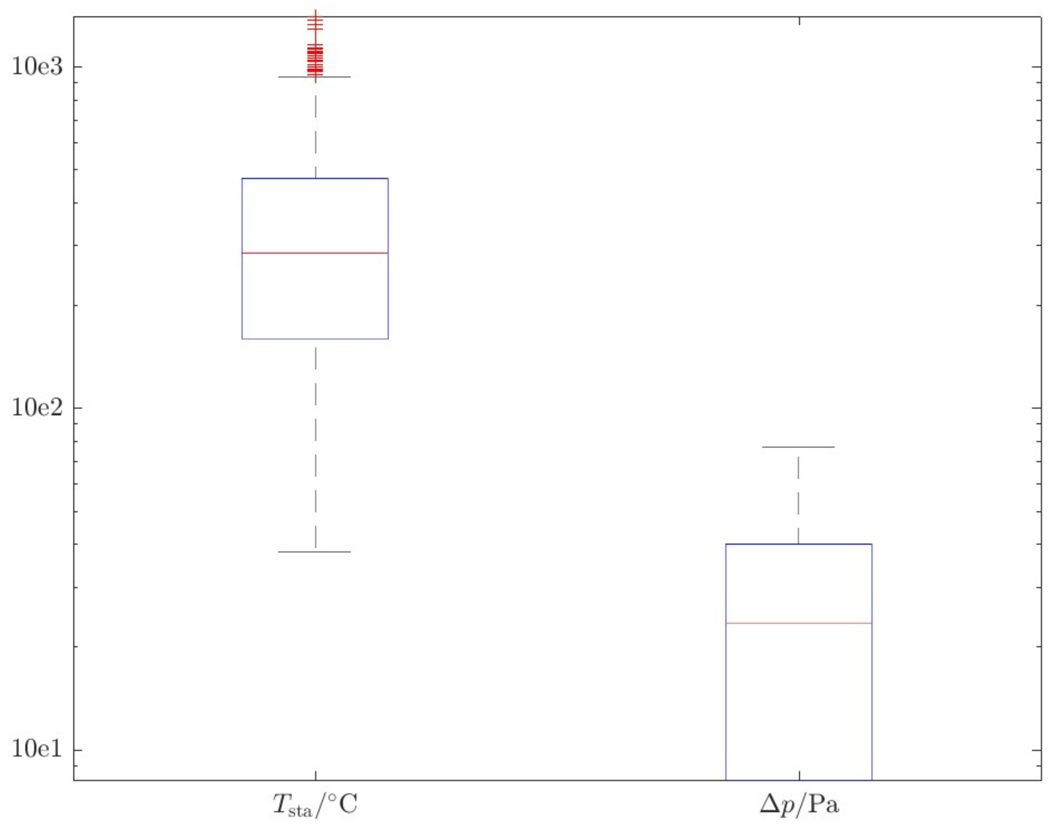

Figure 7.13: This is an example image.

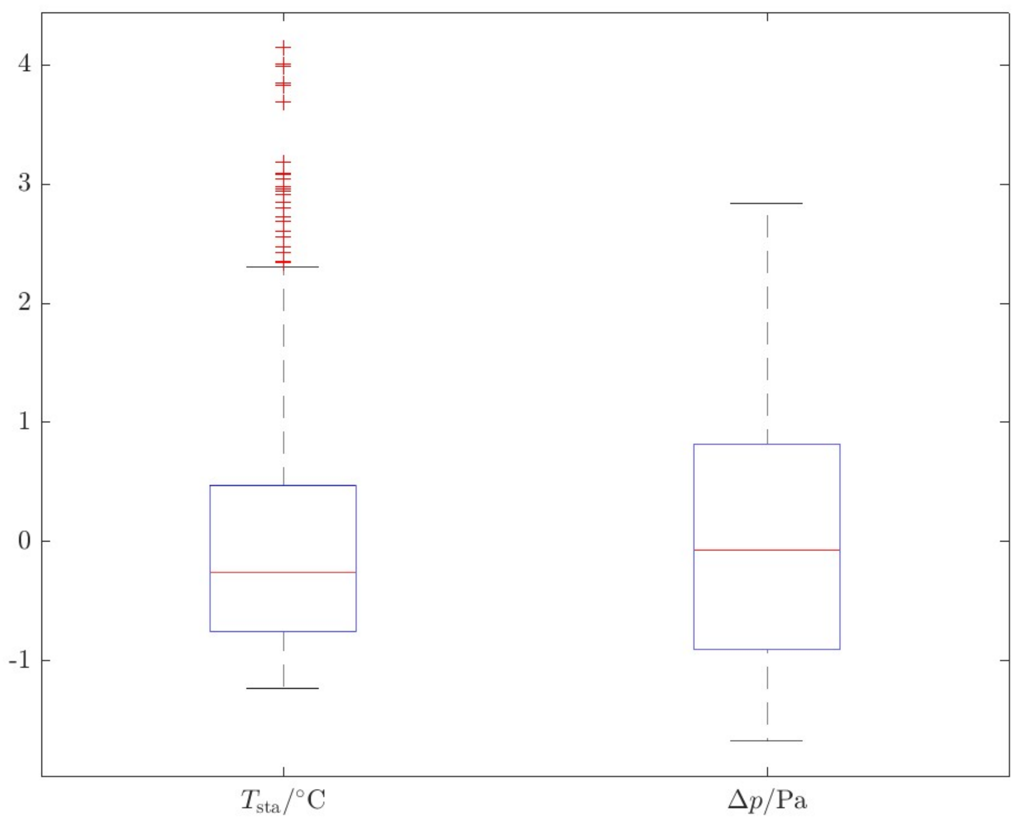

Figure 7.14: This is an example image.

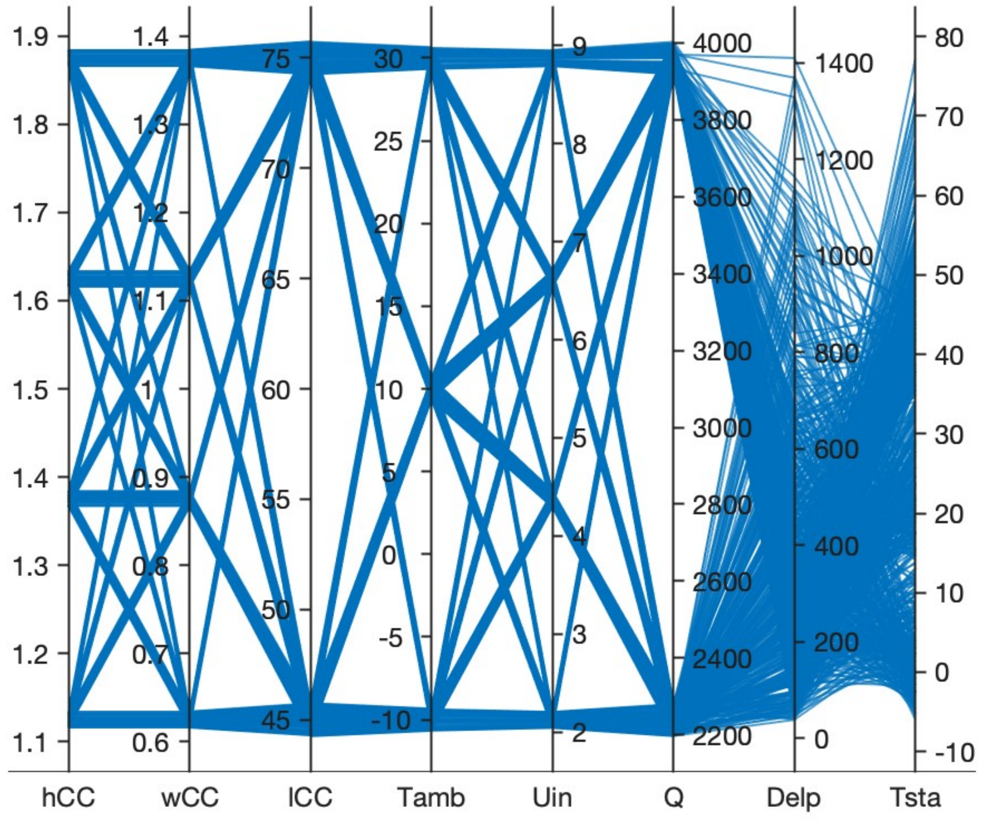

Figure 7.15: This is an example image.

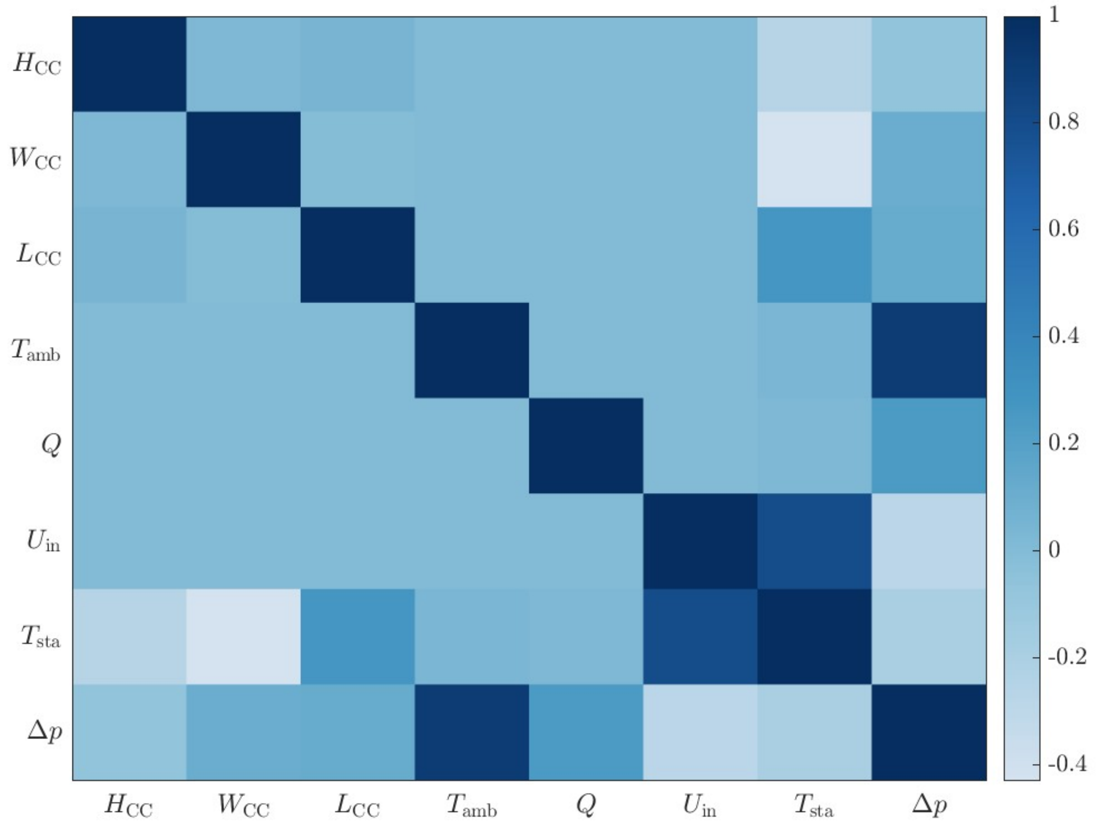

Figure 7.16: This is an example image.

Chapter 8

Training and investigation of different models

Chapter 9

Pareto Optimization

Chapter 10

Conclusion

Chapter 11

Outlook

11.1 References