Improving Temperature Estimation Models using Machine Learning Techniques

Van Duy Dang

Basim Elessawi

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6229
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2024

Abstract

Temperature estimation models are crucial for various products manufactured by BorgWarner. These models often require manual calibration, where experts adjust parameters to ensure accuracy. However, this process can be slow and prone to errors. This thesis investigates how Machine Learning techniques can be used to improve accuracy and efficiency of temperature estimation models.

Both black-box and grey-box approaches are used to evaluate the effectiveness of machine learning-based calibration. The black-box model employs techniques such as Decision Trees, Random Forests, and Neural Networks to predict temperature directly from raw input data, bypassing traditional temperature estimation processes. The grey-box model, on the other hand, uses Deep Q-learning to adjust the calibration automatically.

Results show that the black box model achieves better performance compared to conventional temperature estimation methods. Meanwhile, the grey-box model not only significantly improves accuracy compared to the manual calibration method, but also reduces the need for manual calibration in temperature estimation models.

Acknowledgements

We would like to express our gratitude to our supervisors at BorgWarner, Meike Rönn and Arne Hörberg, as well as our manager, Henrik Nilsson, for their unwavering support throughout the entire duration of this thesis. Their expertise and insightful feedback have been instrumental in shaping the development of this work.

We would also like to thank Richard Pates from the Department of Automatic Control, Lund University for his supervision and valuable feedback. Moreover, we extend our gratitude to Professor Bo Bernhardsson for helping us get started on this thesis.

Furthermore, we are grateful to the managers at BorgWarner for giving us the opportunity to engage in this interesting project and for their continuous support and feedback.

Finally, we would like to extend our deepest appreciation to our family and friends for their unwavering love, understanding and encouragement throughout this project.

Abbreviations

DNN Deep Neural Network
DRL Deep Reinforcement Learning
DQN Deep Q-Network
ECU Electronic Control Unit
GRU Gated Recurrent Unit
KNN K-Nearest Neighbour
LSTM Long Short-Term Memory
MAE Mean Absolute Error
MLP Multi-Layer Perceptron
MSE Mean Squared Error
NN Neural Network
PCB Printed Circuit Board
RF Random Forests
RL Reinforcement Learning
RMSE Root Mean Squared Error
RNN Recurrent Neural Network
THFP Total High Frequency Power

Contents

1

Introduction

1.1 Background

BorgWarner Landskrona designs and manufactures driveline and propulsion systems for hybrid, electric, and combustion vehicles. They cooperate with the biggest vehicle suppliers in the automotive industry. BorgWarner is also one of the largest suppliers of four-wheel drive systems in the world.

Within BorgWarner's operations, a state-space temperature model is used in drive train products that use an electric motor to actuate a pump in order to control torque using hydraulic pressure. It plays a vital role in predicting temperatures at specific locations where sensor deployment may not be feasible. Figure 1.1 illustrates the areas of the Transfercase, one of those drive train products, where temperature estimation is crucial.

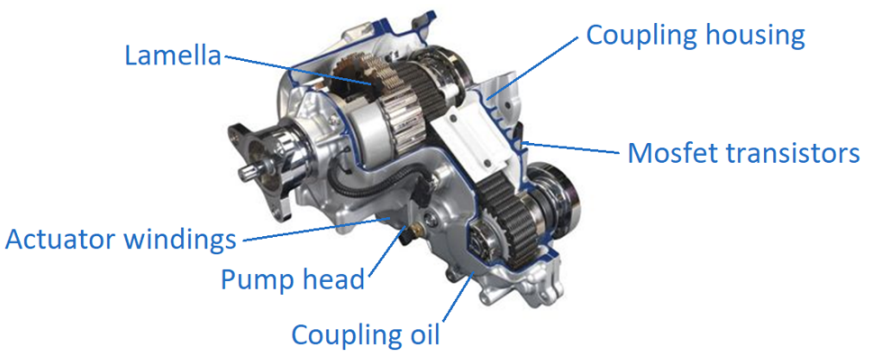

Figure 1.1 Temperature estimation spots of the Transfercase.

The estimated temperatures are utilized to accurately control the pump connected to the pump head. Additionally, they serve to identify the critical temperatures of the coupling, which helps prevent mechanical damage caused by overheating. The need for these temperature estimates arises from the high costs associated

with the implementation of temperature sensors. However, even with good sensors, it is challenging to measure clutch plate temperatures that can increase by 100 to 200 degrees Celsius in just a second. [Olsson, 2019].

This project aims to develop innovative grey-box and black-box models for the temperature estimation model, particularly in regions where direct measurement is not feasible due to cost or complexity. The current method relies on a state-space model with hand tuned parameters and inputs such as sensed temperature from the Printed Circuit Board (PCB), motor speed, pump current, among others. This master thesis provides a valuable opportunity to apply the knowledge we gained during our time at LTH to a real-world project.

1.2 Problem Formulation and Goals

The state-space model requires an experienced engineer to manually calibrate prior to its application in a project. To verify the model, predicted temperatures are compared with actual temperatures obtained from sensors installed in test vehicles. This iterative process continues until an optimal parameter set is found, which usually takes around 2-3 weeks. Consequently, there is a need for a solution that not only automates the calibration process but also enhances the model's accuracy, potentially even eliminating the need for calibration altogether.

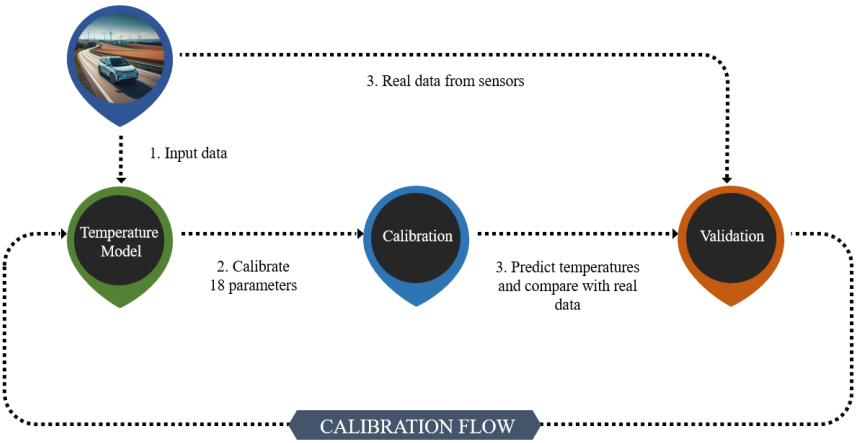

Figure 1.2 Manual calibration flow.

This thesis aims to address the following questions:

- Is it possible to utilize machine learning for parameter tuning and identification of an optimal parameter set?

- Can temperature predictions be made directly, bypassing the use of the state-space model?

The problem can be divided into two parts:

Part 1 - Grey-Box Model: The objective is to develop a common machine learning model capable of automatically tuning parameters and identifying the optimal parameter set across various projects.

Part 2 - Black-Box Model: The goal is to construct a machine learning model that can directly predict temperatures from the inputs in Section 3.2, thereby eliminating the need for Manual Calibration and the state-space model.

1.3 Limitations

The evaluation of the black-box model primarily focuses on comparing its predictions with those from a temperature model tuned by experts. While this benchmarking approach offers a useful point of reference, it limits the validation process to a single method. One of the main concerns with the black-box model is not its predictive accuracy—where it is expected to outperform traditional models—but rather its stability, variance, and susceptibility to noise. These factors can significantly impact the model's reliability in real-world conditions. Additionally, implementing the black-box model poses practical challenges, including the need for greater computational power and potential modifications to the Electronic Control Unit (ECU) to accommodate the model's preprocessing and filtering requirements.

The most effective way to thoroughly assess the performance and robustness of the black-box model is by integrating it into the ECU and conducting extensive real-world testing in an actual vehicle. However, incorporating such real-world testing procedures is beyond the scope of this thesis. Therefore, while the black-box model shows promise in controlled comparisons, its practical viability and long-term stability in a dynamic automotive environment remain areas for future investigation.

1.4 Related Work

Q-learning is an algorithm in the realm of reinforcement learning presented by Watkins, C.J.C.H. (1989). They used this algorithm to learn the action-value function $Q(s,a)$, which estimates the expected discounted reward for taking a particular action in a given state . The Q-learning algorithm has been widely adopted in the reinforcement learning community due to its simplicity, convergence properties [Watkins et al., 1992], and effectiveness in solving complex decision-making problems. Their research has laid the foundation for numerous applications, including the integration of deep neural networks with Q-learning, known as Deep Q-Networks (DQN).

Paszke and Towers (2017) created a tutorial on how to train a Deep Q-Learning agent using PyTorch. This tutorial inspired the development of Model 2 in this thesis. They used DQN along with Replay Memory to train the agent, ensuring that the pole attached to the cart remains upright.

In a paper written by Roderick et al. (2017), they presented critical aspects of implementing the DQN proposed by Mnih et al. (2015), which were essential for its overall performance but lacked detailed coverage in the original work. The paper helps researchers understand and create their own versions of the algorithm more easily. It also emphasized challenges in approximating a Q-function and provided a comparison between their implementation and the original one by Mnih et al. (2015).

Tian et al. (2020) presented a novel framework for the inference of model parameters based on Deep Reinforcement Learning (DRL).They reformulated the inference problem as a tracking problem with the objective of learning a policy that forces the response of the physics-based model to follow the observations. This work has laid a solid foundation in the field of model calibration using Reinforcement Learning. It provides valuable insights and methodologies that can be leveraged for calibrating a temperature model in this thesis.

Arendt et al. (2018) conducted a comparison among white-box, grey-box, and black-box models for predicting indoor temperature within a university building. They focused on two black-box models: a nonlinear autoregressive exogenous model (NARX) and a feed-forward neural network (NN). These models are commonly utilized in building energy-related predictions [Macas et al., 2016], as highlighted by Macas et al. in a study from 2016. They use Keras [Chollet et al., 2015] and Tensorflow [Abadi et al., 2015] libraries to implement the NN model. Overall, the results indicated that black-box models consistently outperformed grey and white-box models in the majority of validation periods, with only one exception.

In [Afram and Janabi-Sharifi., 2015], the authors compared different black-box and grey-box models for residential heating, ventilation and air conditioning (HVAC) system modeling. They used artificial neural networks (ANN), transfer functions (TF), autoregressive exogenous models (ARX), state-space models (SS), and several grey-box models in the project. Their validation with real measured data showed that ANN performed better than all the other models, while the grey-box models were the least accurate.

A paper written by Naing and Htike. (2015) explained how they used a random forest model for monthly temperature forecasting. Their results indicate that the random forests model could be a significant tool for temperature forecasting.

Zhang and Dong (2020) evaluated the use of a convolution recurrent neural network (CRNN) for predicting temperature. The results suggested that the CRNN was more effective than the other benchmark methods. Uluocak and Bilgili (2023) employed hybrid models that integrate Convolutional Neural Networks (CNN) with Long Short-Term Memory (LSTM) neural network and Gated Recurrent Unit (GRU) to perform one-day ahead air temperature (AT) predictions. The

results demonstrated that the proposed hybrid models outperformed all other models in one-day ahead AT predictions with high accuracy.

The paper by Nketiah et al. (2023) used Recurrent Neural Network (RNN) and LSTM to improve and highlight the importance of deep learning algorithms in temperature forecasting. Fente and Singh (2018) also employed the LSTM technique to forecast future weather by training the neural network on various combinations of weather parameters. In an article by Kreuzer et al. (2020), the performance of deep learning models using convolutional LSTM was compared with that of Seasonal autoregressive integrated moving average (SARIMA). The results demonstrated that the convolutional LSTM model outperformed the SARIMA model.

1.5 Structure

The next chapter delves into the concepts of temperature modeling, decision trees, random forests, neural networks, and reinforcement learning, providing the theoretical framework for the thesis. Chapter 3, Methodology, describes how the grey-box and black-box models were developed, trained, and validated. This includes detailed explanations of data collection, model design, training procedures, and validation methods. Chapter 4 presents the results of the models and provides a comparative analysis of the performance of the grey-box models, black-box models, and Manual Calibration. Chapter 5 discusses the findings in depth, providing conclusions and insights based on the results. This chapter also outlines potential future work, suggesting areas for further research and improvement.

2

Theory

2.1 Decision Trees and Random Forests

Decision Trees (DTs) are a type of supervised learning algorithm that is used for both classification and regression tasks. They work by repeatedly splitting the data into smaller and smaller subsets based on specific criteria. Each node in the decision tree represents a feature in the dataset, and the splits are based on a simple decision rule derived from this feature. This process results in a tree-like model of decisions and their possible consequences.

2.1.1 Decision Tree Learning

Decision tree learning involves constructing the tree by repeatedly selecting features that return the highest information gain or the lowest gini impurity. The choice of feature and the threshold for splitting is based on how well the feature separates the classes. The commonly used algorithms include the ID3, C4.5, and CART.

The purity of a node is measured using either of the following criteria:

- Gini Impurity: Used in the CART algorithm, it measures the disorder of a set. A Gini Impurity of 0 indicates that the node is pure, containing elements from only one class.

- Information Gain: Typically used in the ID3 algorithm, it is based on the concept of entropy from information theory. It measures the reduction in entropy or surprise by splitting a dataset according to a given value of a random variable.

A decision tree can be trained by recursively splitting the data according to these measures until a stopping criterion is met, which could be a maximum depth of the tree, a minimum number of samples per leaf, or a minimal gain in impurity.

$$IG(D_p, a) = I(D_p) - \frac{N_{left}}{N} I(D_{left}) - \frac{N_{right}}{N} I(D_{right}) \tag{2.1}$$

where $IG(D_p, a)$ is the information gain by using feature a to split dataset D_p into D_{left} and D_{right}, N_{left} and N_{right} are the number of points in D_{left} and D_{right} respectively.

2.1.2 Random Forests

Random Forests (RF) [Breiman, 2001] are an ensemble learning technique that builds upon the decision tree algorithm. It involves constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Random forests correct for decision trees' habit of overfitting to their training set by adding randomness in two ways:

- Each tree in a random forest is built from a set of data samples drawn with replacement (i.e., a bootstrap sample) from the training set.
- In the traditional decision tree algorithm, the best split is chosen from all features at a node. In contrast, in random forests, each node is split using the best among a subset of predictors randomly chosen at that node.

This randomness helps to make the model more robust to noise and less likely to overfit. The algorithm for training a random forest classifier involves the following steps:

Algorithm 1 Random Forest Algorithm.

1: For $i = 1$ to $n\_estimators$:
2: Create a bootstrap sample of the data set.
3: Build a decision tree by repeating the following for each terminal node until it reaches a minimal threshold size:
4: For each node, randomly select d features without replacement.
5: Choose the best split based on these d features.
6: Split the node into daughter nodes.
7: Use the forest of trees to predict the output for new data points by aggregation:
8: For classification, use majority voting from the output of individual trees.
9: For regression, calculate the average of the outputs from the individual trees.

Random Forests are widely used due to their simplicity, scalability, and good performance across a wide range of data types and tasks. They are less sensitive to outliers and can handle large datasets with higher dimensionality.

2.2 Neural Networks

Neural Networks (NNs) are a foundational component of machine learning that simulate the way human brains operate, allowing computers to recognize patterns and solve common problems in the fields of AI, machine learning, and deep learning. At the core, a neural network consists of layers of interconnected nodes or neurons, which are units of computation. Each connection represents a weight, and during the learning process, these weights are adjusted to predict the correct output.

2.2.1 Feedforward Neural Networks

Feedforward Neural Networks are the simplest type of artificial neural network. In this architecture, the information moves in only one direction—forward—from the input nodes, through the hidden nodes (if any), and to the output nodes. There are no cycles or loops in the network. The process of adjusting the weights and biases is known as training the neural network, and a commonly used method for this is backpropagation combined with stochastic gradient descent.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized kind of neural network for processing data with a known grid-like topology. While they are commonly associated with image data, where they operate in two dimensions as shown in Figure 2.1 [Lecun et al., 1998], CNNs can also be adapted for one-dimensional (1D) data, making them highly effective for sequential data such as time series or audio signals. An example of this is shown in Figure 2.2 [Kiranyaz et al., 2021].

1D CNNs employ the same fundamental operation as their 2D counterparts: convolution, which is a specialized linear operation. These networks are known for their ability to detect patterns and features in sequential data, making them particularly useful for tasks such as signal processing and time series analysis.

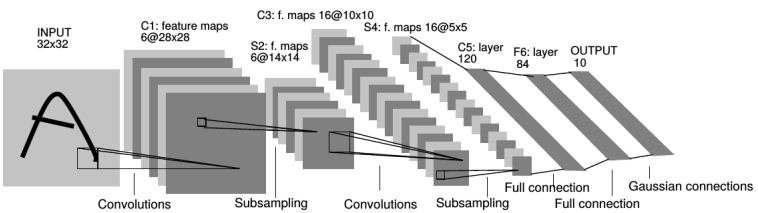

Figure 2.1 Traditional 2D Convolutional Neural Network architecture.

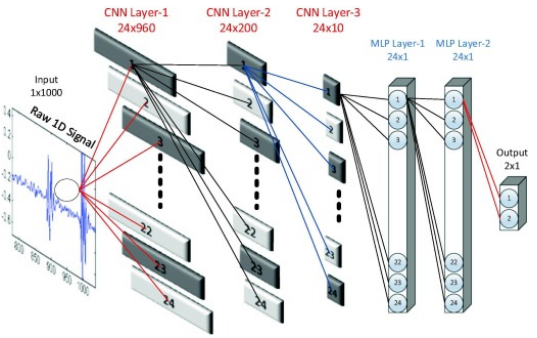

Figure 2.2 1D Convolutional Neural Network architecture.

1D CNNs use convolution in place of general matrix multiplication in at least one of their layers. They typically consist of three types of layers: convolutional layers, pooling layers, and fully connected layers. The convolutional layers apply a convolution operation to the input, passing the result to the next layer. This operation helps the network to concentrate on high-importance features while reducing dimensionality. The formula for a convolution operation in 1D involves a filter or kernel that passes over the sequential data, creating a feature map. After the convolutional and pooling layers have extracted and reduced the features, the output is often passed to a Multilayer Perceptron (MLP) for final classification or regression tasks. An MLP is a type of artificial neural network consisting of multiple layers of neurons, typically including one or more hidden layers, which enables the network to learn complex mappings from inputs to outputs.

$$C_{i,k} = \sum_{m} F_{m,k} \cdot I_{i+m} \tag{2.2}$$

where $F_{m,k}$ represents the filter matrix, I is the input sequence, and $C_{i,k}$ is the output feature map at position i for feature k.

1D CNNs are particularly useful in scenarios where the data is naturally sequential, such as audio signals, time-series data, or any other form of 1D data. They effectively capture local patterns along the sequence, making them powerful tools for tasks like anomaly detection, forecasting, and speech recognition.

2.2.3 Deep Learning

Deep Learning involves neural networks with a large number of layers. These networks can learn very complex patterns and features from the data, making them extremely powerful for many tasks including speech recognition, natural language processing, and image recognition. The depth of these networks is what allows them to learn such rich representations of the data, enabling them to handle very complex

tasks. The training of deep neural networks involves considerations such as avoiding overfitting by using techniques such as dropout or batch normalization.

In practice, training deep neural networks is performed using high-level libraries like TensorFlow or PyTorch, which provide tools that automatically calculate gradients by backpropagation and update weights by gradient descent.

2.2.4 Neurons and Activation Functions

A neuron in an artificial neural network is a computational unit that takes inputs, multiplies them by some weights, and then passes them through an activation function to produce an output. The purpose of the activation function is to introduce non-linearities into the output of a neuron. This is crucial because it helps the network learn complex patterns in the data.

- **Structure of a Neuron**: Each neuron receives input from some other neurons or from an external source and computes an output. Each input has an associated weight (a scalar), and there is an additional bias term. The neuron's output, O_i, is defined as:

$$O_i = f\left(\sum_j w_{ij} x_j + b_i\right) \tag{2.3}$$

  where f is the activation function, w_{ij} are the input weights, x_j are the input signals, and b_i is the bias.

- **Common Activation Functions**:

  - *Sigmoid*: $\sigma(x) = \frac{1}{1+e^{-x}}$, traditionally used because it squashes the output between 0 and 1, making it useful for binary classification.

  - *ReLU* (Rectified Linear Unit): $\mathrm{ReLU}(x) = \max(0,x)$, popular in most recent neural networks due to its computational simplicity and ability to reduce the vanishing gradient problem.

  - *Tanh* (Hyperbolic Tangent): $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, outputs values between -1 and 1, making it zero-centered and thus helping in the convergence during training, it can be considered a simple rescaling of the sigmoid function and can be used for other purposes.

2.2.5 Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory networks [Hochreiter and Schmidhuber, 1997] are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem, remembering information for long periods as a default behavior.

- **LSTM Architecture**: Each LSTM cell has three gates: the input gate, the output gate, and the forget gate. These gates determine whether to let new input in (input gate), delete the information because it is not necessary anymore (forget gate), or let it impact the output at the current timestep (output gate).
- **Mathematical Model**: The operations within an LSTM cell can be summarized by the following equations:

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

  where σ is the sigmoid function, W and b are the weights and biases specific to each gate, and $*$ denotes element-wise multiplication.

2.2.6 Gated Recurrent Unit (GRU) Networks

Gated Recurrent Unit networks [Chung et al., 2014] are a type of RNN that aim to solve the vanishing gradient problem and enhance the ability to capture long-term dependencies, similar to LSTMs but with a simpler architecture. GRUs achieve this by using gating mechanisms to control the flow of information.

GRU Architecture: Each GRU cell has two main gates: the update gate and the reset gate. These gates manage the flow of information by determining how much of the past information should be passed to the future (update gate) and how much of the past information should be forgotten (reset gate).

- **Mathematical Model:** The operations within a GRU cell can be summarized by the following equations:

$$
\begin{aligned}
z_t &= \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \\
r_t &= \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \\
\tilde{h}_t &= \tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$

  where σ is the sigmoid function, W and b are the weights and biases specific to each gate, and $*$ denotes element-wise multiplication.

In these equations:

- z_t is the update gate vector,
- r_t is the reset gate vector,
- $\tilde{h}_t$ is the candidate activation vector,
- h_t is the new hidden state vector.

The update gate z_t decides how much of the past information needs to be passed along to the future. The reset gate r_t determines how much of the past information to forget. The candidate activation $\tilde{h}_t$ is then calculated using the reset gate's output. Finally, the new hidden state h_t is a linear interpolation between the previous hidden state h_{t-1} and the candidate activation $\tilde{h}_t$, controlled by the update gate z_t.

GRUs offer a simpler and potentially more efficient alternative to LSTMs while achieving similar performance in handling long-term dependencies.

2.2.7 Dropout and Dynamic Regularization

In traditional artificial neural networks (ANNs), dropout is a widely used technique to prevent overfitting and improve generalization by randomly omitting a subset of neurons during training. However, due to the inherent memory properties and sequential nature of Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), applying dropout directly to these architectures can disrupt the temporal dependencies they are designed to capture. To address this challenge, we introduce a method called "dynamic regularization." This approach involves applying varying regularization strengths to different layers of the LSTM and GRU networks, rather than using a uniform regularization parameter across the entire network. By tailoring the regularization for each layer, dynamic regularization helps to maintain the integrity of memory retention and sequence learning while mitigating overfitting. This method ensures that the regularization is optimally balanced, allowing the model to generalize better without compromising its ability to learn long-term dependencies.

2.2.8 Loss Functions and Optimization

In neural network training, a loss function or cost function is used to measure how well the model predicts the target data. Optimizing this function involves adjusting the weights of the network to minimize the loss.

- **Common Loss Functions**:

  - *Huber Loss*: The Huber loss [Huber, 1964] acts like MSE when the error is small, but it acts like Mean Absolute Error (MAE) when the error is large.

The Huber loss, $L_\delta(y,\hat{y})$, is defined as follows:

$$L_\delta(y,\hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & \text{if } |y-\hat{y}| \leq \delta \\ \delta \cdot |y-\hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{2.4}$$

where y is the true value, $\hat{y}$ is the predicted value, and δ is the threshold parameter.

- *Cross-Entropy Loss*: Preferred for classification tasks, measures the performance of a classification model whose output is a probability value between 0 and 1.

$$L = -\sum y \log(\hat{y}) \tag{2.5}$$

where y is 1 for the true class and 0 otherwise, and $\hat{y}$ is the predicted probability of the label.

- *Mean Squared Error* (MSE) and *Root Mean Squared Error* (RMSE): Both are primarily used for regression problems. MSE quantifies the average of the squared differences between the predicted values and the actual values, while RMSE provides a measure of how well a model predicts the target variable by calculating the square root of the average squared differences between the predicted values and the actual values.

The MSE is defined as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.6}$$

The RMSE is defined as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.7}$$

where n is the number of data points, y_i represents the true value for data point i, and $\hat{y}_i$ represents the predicted value for data point i.

Both MSE and RMSE are sensitive to large errors, which can be beneficial when large errors are particularly undesirable. However, this sensitivity can also be a disadvantage if the dataset contains outliers, as they can disproportionately influence these metrics. Despite this, MSE and RMSE are often favored in practice due to their straightforward interpretation and their ability to reflect the overall fit of a model more effectively than other metrics.

- **Optimization Techniques**:

  - *Gradient Descent*: The most basic form of optimization algorithm used in neural networks. It updates the weights incrementally after each epoch according to the gradient of the loss function.

  - *Stochastic Gradient Descent* (SGD): A variant of gradient descent where the update to the weights is performed using a subset of the data rather than the full dataset. This is much faster and can also lead to better generalization.

  - *Adam*: Adaptive moment estimation (Adam) is a popular optimization algorithm used in neural networks. It provides an optimization that can handle sparse gradients on noisy problems. [Kingma and Ba, 2017]

- **Backpropagation**: This is the process used to compute the gradient of the loss function in a neural network. It involves performing a forward pass to calculate the output and error, and then a backward pass to calculate the gradient of the loss with respect to each weight.

$$\Delta w = -\eta \nabla L \tag{2.8}$$

  where Δw is the change to the weights, η is the learning rate, and ∇L is the gradient of the loss function with respect to the weights.

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning that focuses on decision making and the optimization of sequential actions to achieve a goal. In RL, an agent interacts with an environment over discrete time steps. At each time step, the agent selects an action from a set of possible actions. The environment, in response, transitions to a new state and returns a reward to the agent. The goal of the agent is to learn a policy, which is a mapping from states to actions, that maximizes the sum of rewards over time.

2.3.1 Q-Learning

Q-Learning is a model-free reinforcement learning algorithm, introduced by Watkins et al, that seeks to find the best action to take based on the current state. It focuses on learning the Q-value, which represents the expected future reward for taking a specific action in a particular state [Watkins, C.J.C.H., 1989].

The Q-value for a current state s and action a is denoted by $Q(s,a)$ and is updated using the Bellman equation:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2.9}$$

where α is the learning rate, γ is the discount factor. R(s, a) is the immediate reward received after taking action a in state s. $\max_{a'} Q(s', a')$ represents the maximum Q-value over all possible actions in the next state s'.

The goal of Q-Learning is to find the optimal policy by learning the optimal Q-values for each state-action pair. The optimal policy is the one that has the highest expected future reward.

2.3.2 Deep Q-Learning

While Q-Learning can be very effective, it has a major limitation in that it can only handle environments with small state and action spaces, as it requires more memory and time to create and store the Q-table. DQN [Mnih et al., 2015] addresses this limitation by using deep neural networks (DNN) to approximate the Q-value function. Figure 2.3 illustrates the integration of a DNN in the Deep Q-learning model, allowing it to handle environments with larger and more complex state and action spaces effectively.

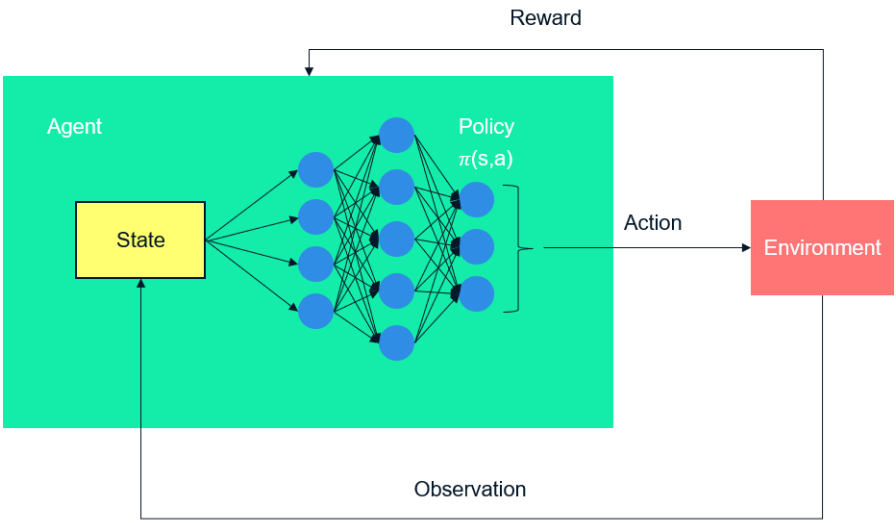

Figure 2.3 Deep Q-Learning model.

The DQN takes in the state as input and outputs the Q-value for each action. The weights of the network are then updated to minimize the difference between the predicted Q-values and the target Q-values,

$$L_i(\theta_i) = (R(s,a) + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i))^2 \tag{2.10}$$

where θ_i is the parameters of the neural network at each step i. When optimizing the loss function, the parameters from the previous iteration θ_{i-1} are fixed.

Deep Q-learning uses the experience replay technique where past transitions $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored in a replay memory. During training, mini-batches of transitions are randomly sampled from this replay memory instead of using just the latest transition. This approach ensures a diverse and uncorrelated set of experiences for learning, thereby improving the stability and efficiency of the learning process. In addition, an ε-greedy policy, as explained in steps 8 and 9 in Algorithm 2, is used to select and execute an action to ensure good coverage of the state and action space. Finally, through backpropagation, the weights of the main DNN are updated to minimize the loss from (2.10), thus improving the accuracy of Q-value estimation. The deep Q-learning algorithm is derived from [Roderick et al., 2017] as follows:

Algorithm 2 Deep Q-learning with Experience Replay.

1: Initialize replay memory D to capacity N.
2: Initialize action-value function Q with random weights θ.
3: Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$.
4: **for** episode $= 1, M$ **do**
5: Initialize sequence $s_1 = \{x_1\}$ where x_1 represents the initial observation of the environment when a new episode starts.
6: Initialize state $\phi_1 = \phi(s_1)$ where ϕ is the function to handle s_1 and convert it to ϕ_1.
7: **for** t=1,T **do**
8: With probability ε select a random action a_t
9: otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$.
10: Execute action a_t in emulator and observe reward r_t and the next
11: observation of the environment x_{t+1}.
12: Set next sequence $s_{t+1} = s_t, a_t, x_{t+1}$ and next state $\phi_{t+1} = \phi(s_{t+1})$.
13: Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D.
14: If D has reached its capacity N, replace the oldest transition with the new one.
15: Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D with a batch size of K.
16: Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{for non-terminal } \phi_{j+1} \end{cases}$
17: Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ.
18: Every C steps reset $\hat{Q} = Q$, i.e set $\theta^- = \theta$.
19: **end for**
20: **end for**

3

Methodology

At a high level, the inputs for both models are the data described in Section 3.2, and the outputs are the predicted temperatures. The grey-box model uses an existing state-space model to predict these temperatures. In contrast, the black-box model predicts the temperatures directly from the inputs. This chapter explains how data was collected and processed. It also describes the process of building the models and how these were evaluated.

To answer the first question in the problem formulation and goals "Is it possible to utilize machine learning for parameter tuning and identification of an optimal parameter set?", Deep Q-Learning models are created to calibrate the temperature model. Python libraries such as PyTorch, Numpy, Scipy, Matlab engine are used to handle data and create the DRL model.

To answer the second question, which is if it is possible to bypass the requirement of a state-space model, we propose using black-box model methods. Our main focus is on customizable neural network models created with the Deep Learning Toolbox in MATLAB. Additionally, we use models such as Random Forests, SVM, and KNN as benchmark comparisons. These additional models were developed using the scikit-learn toolbox in Python.

3.1 Temperature Model

This section briefly introduces the state-space model. The primary objective of the temperature estimation module is to estimate the oil temperature accurately based on the measured temperatures, internal coupling signals, and existing knowledge. This is crucial for effectively controlling the pump and identifying critical temperatures of the coupling to prevent mechanical damage from overheating. As the sole method for measuring temperatures relies on a temperature transducer on the ECU PCB and the external temperature of the vehicle, it is essential to precisely estimate the temperatures in the oil pan and the lamella. [Olsson, 2023]

The temperature estimation relies on the state-space model (3.1) and 18 parameters that need to be tuned. [Svendenius, 2020]

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{3.1}$$

where the state vector x comprises the temperatures to be estimated and the control vector u consists of powers. These powers are calculated from the data described in Section 3.2. The matrices A and B are constructed using 18 tunable parameters. The matrix C is defined as the identity matrix, and the matrix D is set to zero. Further details regarding these matrices are omitted from this report due to confidentiality reasons.

3.2 Data

A test car used for driving tests is equipped with sensors at different places, such as Pump sump, Final gear box, Lamella sump, Coupling, Outside air temp sensor, etc. DIAdem/CANalyzer is used to collect data depending on project needs. Below are some input signals required for the Matlab model:

- Pump voltage
- Pump current
- Coupling torque
- Absolute rotational speed of coupling
- Differential rotational speed of coupling
- Vehicle velocity
- Pump motor speed
- Battery voltage when the ignition is on

The test logs will contain approximately 20-45 minutes of data, and each test will be started from different temperatures. The test car will be driven in various driving scenarios, such as aggressive driving, highway driving, country road driving, or city driving.

3.2.1 Dataset Description for Black Box Models

For the purpose of training black box models aimed at temperature estimation, we utilized a comprehensive collection of 18 datasets. These datasets were systematically divided into two segments: 12 datasets allocated for training purposes and 6 datasets reserved for testing. Each dataset encompasses a range of features and goals essential for model development and evaluation.

Each dataset comprises 8 features and 4 goals, crucial for accurate temperature estimation. The features included are:

- **Pump Current** - The current measured in the pump.
- **Pump Voltage** - The voltage supplied to the pump.
- **Motor Speed** - The rotational speed of the motor.
- **Car Velocity** - The speed of the car.
- **Outside Temperature** - The ambient temperature outside.
- **Case Temperature** - The temperature of the case enclosing the system.
- **Flow Rate** - The rate of fluid flow through the system.
- **PCB Temperature** - The temperature of the printed circuit board.

The goals, which are the target variables for our models, include:

- **Lamella Temperature** - The temperature of the lamella.
- **Pumphead Temperature** - The temperature at the pump head.
- **Coupling Oil Temperature** - The temperature of the coupling oil.
- **Final Gear Temperature** - The temperature of the final gear.

To ensure a more robust and comprehensive model training and evaluation, we employed two distinct testing regimens:

1. **Compiled Dataset Testing**: In this approach, all 12 training datasets are used collectively to train the models, and all 6 testing datasets are used to evaluate the model performance. This comprehensive approach allows for a generalized and comprehensive evaluation of the model, and how we can expect it to behave in a variety of situations.
2. **Warm Driving Situation Testing**: This focused testing method involves selecting 4 specific training datasets and 1 test dataset, all related to warm driving situations. This subset of datasets helps in understanding the model performance in specific operational conditions, providing insights into its reliability and accuracy. This subset of data also allows for more detailed micro-investigations into the behaviour of various models on a controlled dataset due to it's smaller size which facilitates more numerous tests in the same time frame.

3.3 Loss Functions and Evaluation methods

RMSE is the loss function used to evaluate the Black box model, however, due to the uncertain nature of black box models (that arise because of the lack of explainability) additional methods of evaluation are necessary. To achieve this we also extracted the variance of results, as well as the total high frequency power.

MSE and Huber loss are used to train DRL models. Although MSE is a popular loss function, Huber loss can be especially useful in this context because the model's loss often starts out high in the initial stages.

3.4 Grey-Box Models

Two models were created to compare their performance. The first model relies solely on a single DQN, while the second model employs a policy network and a target network. The input to these models is a parameter set comprising 18 parameters, as described in Section 3.1, with all values initialized to zero. Each parameter is bounded within the range of 0 to 100. This boundary is defined by the architect. These parameters are updated every epoch and are reset to zero when any parameter reaches its defined limit. When this happens, the current episode is marked as done (termination). New Q values are updated to include the rewards for the current step as described in step 16 in Algorithm 2.

Each model will be trained for about two days to five days on a laptop and the performance of these models will be evaluated using data from three different driving tests. After each epoch, the average MSE loss between the predicted data and the actual data will be calculated. At the end of the training, the best average MSE loss from the grey-box models will be compared to the average MSE loss from Manual Calibration to evaluate its performance.

3.4.1 Model 1

Model 1 uses a single DQN, serving as both the policy network and the target network as shown in Figure 3.1. This model implements a training approach that uses both short-term and long-term memory to update the DQN. The short-term memory allows the model to learn from individual experiences as they occur.

On the other hand, the long-term memory involves accumulating a collection of experiences in a replay memory. The network is then trained in batches by sampling from this replay memory at the end of each episode. By combining short-term and long-term memory, this approach leverages the benefits of both immediate learning and more generalized learning from diverse experiences, contributing to a more stable and effective training process.

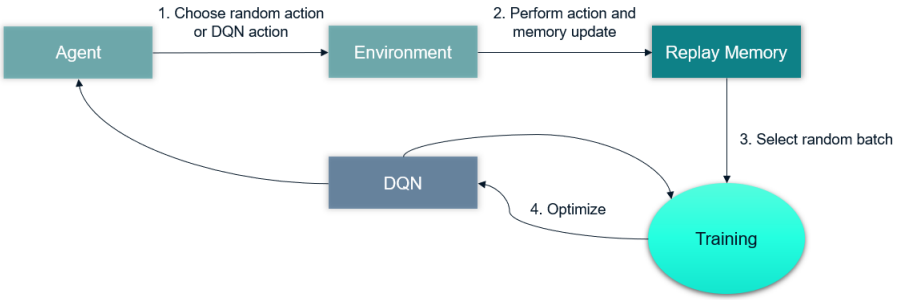

Figure 3.1 DQN network.

3.4.2 Model 2

This second model, as shown in Figure 3.2, involves two separate networks: a policy network for selecting actions and a target network for providing stable reference values during training [Lillicrap et al., 2019]. The policy network and the target network have similar structures, but they operate independently. The policy network chooses actions based on the current state, while the target network offers a stable estimation of future rewards, which helps in updating the policy network.

When a new episode starts, a random sample of experiences from the replay memory is used to update the policy network, which helps to improve training stability and model robustness by reducing the likelihood of overfitting to specific sequences of events. However, this model does not use short-term and long-term memory. The policy network is optimized every epoch with the batch size from Table 3.1. Moreover, the target network is updated less frequently, which contributes to greater stability during training. The target network is synchronized with the policy network by a soft update [Lillicrap et al., 2019].

$$\theta' \longleftarrow \tau\theta + (1-\tau)\theta' \tag{3.2}$$

where θ and θ' are the weights of the policy network and the target network, respectively, and τ is a hyperparameter between 0 and 1.

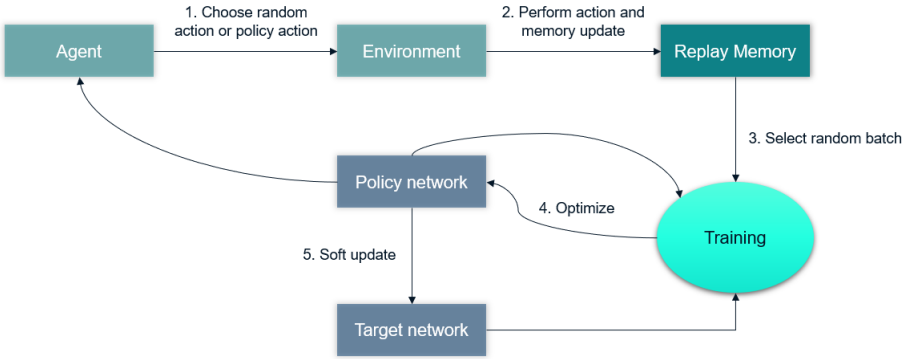

Figure 3.2 Policy and Target network.

This approach may slow down learning because the target values change slowly, but it enhances stability.

3.4.3 Hyperparameter tuning

Various configurations were tested through a series of experiments to optimize the hyperparameters. Table 3.1 presents the optimal set of hyperparameters determined through these experiments.

Table 3.1 Model Parameters.

Parameter	Model 1	Model 2	Description
Batch Size	1000	250	Number of samples in each training batch
Randomness (Epsilon)	80	80	Initial randomness for exploration
Discount Rate (Gamma)	0.9	0.9	Rate at which future rewards are discounted
Learning Rate	0.001	0.001	Rate at which the optimizer updates the parameters
Soft Update Coefficient (TAU)	None	0.005	Coefficient for updating the target network
Input Layer Neurons	25	25	Number of neurons in the first linear layer
First Hidden Layer Neurons	128	128	Neurons in the first hidden layer
Second Hidden Layer Neurons	128	128	Neurons in the second hidden layer
Output Layer Neurons	17	17	Neurons in the output layer

3.5 Black-Box Models

The black box models are divided in four main categories, each category contains 3 models. It should be noted that the black box models are expected to perform significantly better than the state-space models, thus the comparisons made will rarely be purely on the error scale, and comparisons are only used to contrast the ability of the black box methods. The categories are as follows:

1. **Classical models:** Random forest, SVM, and KNN.

   These models are used as a benchmark standard for more robust neural network models. The classical models have an upper limit in terms of learning capabilities. Grid searches were used to optimize the hyperparameters, and after some feature engineering, what is believed to be the theoretically best results achievable within the limits of the model have been acquired.

2. **Perceptron Neural Networks:** Small, Medium, and Large.

   The sizes reflect various architectures of a perceptron based Deep Neural network model (The specific architectures will be mentioned in the appendix). The varying sizes are used to achieve a balance between generalizing data, power consumption, and capturing temporal properties. All of which vary across datasets and model sizes.

3. **LSTM Networks:** Small, Large, and Dynamic Regularization.

   As mentioned, LSTM or Long Short-Term Memory is a special type of neural network unit, this unit is especially helpful in keeping track of temporal dependencies. Similar to the Perceptron networks above, the sizes reflect different architectures of the LSTM networks, for consistency the sizes 'small' and 'large' have been made to match that of the other networks, this was done by choosing an architecture that has the most similar number of parameters as that of the other networks, with the same number of hidden layers. The reason why matching the number of parameters was preferred over matching the architecture is due to the increasing complexity of the model being based on the number of parameters instead of the architecture. Additionally, the number of hidden layers was kept consistent to avoid any qualitative differences between results, and to most closely match the behavior of the other models (for easier contrast). Dynamic regularization is an additional structure used on Recurrent neural networks such as LSTM and GRU, the purpose is to use regularization at different layers as opposed to dropout which is used in the perceptron networks.

4. **GRU Networks:** Small, Large, and Dynamic Regularization.

   The GRU model is known for its efficiency in processing sequences similarly to LSTMs but with a simplified structure. The small and large configurations are designed with a similar number of parameters to their perceptron and LSTM equivalents, maintaining uniformity in complexity and capability across the models. The use of dynamic regularization in GRU networks is aimed at enhancing model adaptability and improving performance on varied sequences by selectively applying regularization techniques at different network layers. This approach helps in mitigating overfitting while preserving the network's ability to capture essential temporal features in the data, thus ensuring robustness and generalization across different datasets and scenarios.

3.5.1 Data processing and Procedure

Data pre-processing is a critical step in the methodology of training machine learning models, as it directly influences the model's performance and effectiveness. The data pre-processing performed consisted of several key steps: data cleaning, normalization, and resampling. Data cleaning was used to rectify inconsistencies such as missing values, outliers, and incorrect data entries, which could otherwise skew the results and lead to unreliable model predictions. Normalization scales the data attributes to a standard range of 0 to 1 using a min-max normalization formula, this is crucial for preventing certain features from dominating due to their scale, thus ensuring all attributes contribute equally to the learning process. Additionally,

transformation techniques such as encoding categorical variables and decomposing timestamps were used to enhance the model's ability to understand and extract patterns from the data.

3.5.1.1 Resampling.

Resampling was a critical process in the data pre-processing step, especially due to dealing with time-series data from multiple experiments. The goal is to standardize the number of data points in each dataset, ensuring consistency across different experimental conditions and facilitating robust comparative analyses. The resampling performed in our paper was done using linear interpolation and is described as:

$$x(t') = x(t_1) + \frac{(t' - t_1)}{(t_2 - t_1)} (x(t_2) - x(t_1))$$

Where, t represents the original time points, t' represents the new resampled time points, and t_1 and t_2 are the time points in the original data that bracket t'. This ensures that the temporal relationships in the original data are preserved while reducing (or increasing) the number of points to the desired sample size.

For our thesis, a resampling factor was used to verify the effect of resampling across datasets, this is because the effect of resampling can manifest in two ways; it can involve either under-sampling or oversampling, both of which have their individual benefits and drawbacks.

A smaller resampling factor would result in under-sampling more data sets, which could lead to a loss of detail and potential under-representation of the data's variability. On the other hand, a larger resampling factor would result in more frequent oversampling, potentially introducing artificial data points that might not represent real changes in the underlying process.

However, by standardizing the number of data points across all experiments, resampling ensures that each dataset is comparable. This uniformity is crucial for the black box models, as to prevent the model from developing unnatural biases due to the data size and sampling time variations, thus facilitating more robust statistical analyses. Overall, the goal of resampling is to enhance the reliability and validity of the subsequent analyses and modeling efforts.

3.5.1.2 Filtering and Smoothing (with considerations).

Noise reduction techniques were employed in the pre-processing of feature data for neural networks to enhance the quality of the input data and improve the model's performance. This was crucial for mitigating the influence of noise and variability in sensor measurements. While these techniques successfully preserved the accuracy of the model's predictions, ensuring robust performance with real-world data required us to forego some noise reduction methods that could not be implemented

in real-time. Instead, we suggest using alternative methods such as Kalman filtering and low-pass filters.

For the purposes of this thesis following methods for noise reduction were used:

1. *Moving Average Filter:* a moving average filter was applied to sensor data to reduce noise and smooth the signal. The filter was configured with a window size of 5, meaning each output value was the average of the current and the four preceding sensor readings. This approach effectively smoothed out rapid fluctuations in the sensor output, enhancing the signal-to-noise ratio. The implementation can be given as:

$$y_t = \frac{1}{N} \sum_{i=t-N+1}^{t} x_i \tag{3.3}$$

   where:

   - y_t is the output of the moving average filter at time t.
   - x_i is the sensor data at time i.
   - N is the window size of the moving average filter.

2. *Exponential Smoothing:* Exponential smoothing was also utilized, prioritizing recent data points by assigning exponentially decreasing weights to older observations. A smoothing factor of α was chosen to balance between responsiveness and smoothness of the output signal. This method provided a continuously updated average that responded sensibly to changes in trend while damping out noise. Similarly we can realize this smoothing with the following formula:

$$S_t = \alpha x_t + (1-\alpha) S_{t-1} \tag{3.4}$$

   where:

   - S_t is the smoothed value at time t.
   - x_t is the raw sensor data at time t.
   - α is the smoothing constant, $0 < \alpha \leq 1$.
   - S_{t-1} is the previous smoothed value.

3. The *Kalman Filter* was designed to estimate and correct sensor readings in real time. The filter started with an initial estimation of state and error covariance matrices. The process involved a prediction step, where the next state was estimated based on a physical model, followed by an update step, where this prediction was corrected using new sensor data. It is instrumental to understand that the Kalman filter was implemented on pre-recorded data. Due

to the intricacies of which, or lack thereof, we are not able to accurately assert the effectiveness of this method on real time processes (this is not to say that the Kalman filter may produce contradictory results, but the degree to which it may improve the data may not be the same).

4. *Low-Pass Filter* AA Butterworth low-pass filter was chosen for its maximally flat frequency response at $\omega = 0$, ensuring minimal signal distortion at low frequencies. The MATLAB function designfilt was used to create the digital low-pass filter, and filtfilt was employed to apply the filter to the dataset. The filtfilt function was chosen for its zero-phase filtering capability, which ensures that the filtered signal has no phase distortion. The filtering process was applied to each column (feature) of the dataset independently. The mathematical details and pseudocode of which are given by:

   The transfer function of a digital Butterworth low-pass filter of order N is given by:

$$H(z) = \frac{B(z)}{A(z)}$$

   where $B(z)$ and $A(z)$ are polynomials in z.

   The frequency response of the Butterworth filter is:

$$|H(e^{j\omega})|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}}$$

   The difference equation for the filter in the time domain is:

$$y[n] = \sum_{i=0}^{N} b_i x[n-i] - \sum_{j=1}^{N} a_j y[n-j]$$

   where:

   - $y[n]$ is the output signal.
   - $x[n]$ is the input signal.
   - b_i are the feedforward coefficients.
   - a_j are the feedback coefficients.

3.5.2 Goals and Evaluation Criteria

While RMSE (Root Mean Square Error) is typically the main evaluation criterion for neural network predictions in regression tasks, the delicacy of real-time car temperature estimation requires additional evaluation factors. Simply achieving a low RMSE may not be sufficient to confirm the model's validity due to external factors

Algorithm 3 Apply Low-Pass Filter

```
function APPLYLOWPASSFILTER(data, cutoffFrequency)
    Design the low-pass filter using Butterworth design
    d ← designfilt(lowpass, cutoffFrequency, butterworth)
    Initialize the filtered data matrix
    filteredData ← zeros(size(data))
    Apply zero-phase filtering to each column of the dataset
    for i = 1 to size(data, 2) do
        filteredData(:, i) ← filtfilt(d, data(:, i))
    end for
    return filteredData
end function
```

such as result uncertainty. This uncertainty can necessitate a higher safety factor, potentially leading to overall worse performance compared to a more stable state-space model. Thus, evaluating the model's effectiveness involves a comprehensive approach considering multiple criteria.

1. **Root Mean Square Error (RMSE)**:

   RMSE measures the average magnitude of the errors between predicted and actual values. The metric is used to directly assess the model's prediction accuracy, with Low RMSE meaning better accuracy.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_{\text{pred/est},i} - y_{\text{test},i})^2}$$

   where $y_{pred/est,i}$ is the value for the i-th prediction/estimation, $y_{test,i}$ is the true value, and N is the total number of predictions.

2. **Variance**:

   Variance measures the dispersion of prediction errors. It indicates how much the prediction vary from the mean. With low variance indicating consistent performance with small deviations in prediction errors, leading to reliable predictions. Conversely, high variance could imply that the model is overly sensitive to minor fluctuations in the input data, which is usually undesirable in an automotive system.

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(y_{\text{pred/est},i} - \bar{y})^2$$

where $y_{\text{pred/est},i}$ is the value for the i-th prediction, $\bar{y}$ is the mean value, and N is the total number of predictions.

3. **Total High Frequency Power**:

Total high frequency power quantifies the amount of high-frequency components in the prediction error signal. It helps to assess the presence of high-frequency noise or rapid fluctuations in the predictions. Low High Frequency Power indicates that the predictions are smooth and free from high-frequency noise, contributing to stable performance. Inversely, high values of total High Frequency Power suggests that the predictions contain significant high-frequency noise, which can lead to erratic performance and reduced reliability.

To compute the total high-frequency power, the following steps are performed:

a) **Detrending the Signal**: The prediction signals are detrended to remove any linear trend that could obscure the high-frequency components.

b) **Computing the Power Spectral Density (PSD)**: The PSD of the detrended signals is calculated using the Welch method. The PSD represents the distribution of power across different frequency components in the signal.

c) **Summing the High-Frequency Components**: A threshold frequency is defined to separate high-frequency components from low-frequency ones. The total high-frequency power is then obtained by summing the PSD values that correspond to frequencies above this threshold.

Mathematically, if $\hat{y}_{\text{pred}}(t)$ is the detrended prediction signal, the power spectral density $\text{PSD}_{\text{pred}}(f)$ is computed. The total high-frequency power P_{HF} is given by:

$$P_{\text{HF}} = \sum_{f > f_{\text{th}}} \text{PSD}_{\text{pred}}(f)$$

where f_{th} is the threshold frequency separating high-frequency components.

By analyzing the total high-frequency power, we can evaluate the smoothness and reliability of the predictions. Low high-frequency power indicates smoother predictions with fewer rapid fluctuations, while high high-frequency power suggests the presence of noise and potential instability in the predictions.

4

Results

This chapter will comprehensively present the performances of all models. A concise discussion will accompany the results to enhance understanding. Additionally, visual comparisons of results will be provided to identify the most effective model. Further elaboration will be discussed in Chapter 5.

4.1 Grey-Box Model

4.1.1 Model comparision

The best average loss values of all the models, calculated using MSE, are shown in Table 4.1.

Table 4.1 Best average MSE loss metrics.

	Model 1	Model 2	Manual Calibration
Dataset 1	16.3	19.5	18.9
Dataset 2	58.9	72.9	71.6
Dataset 3	138.5	177	191.4

Out of the numerous log files in each dataset, a log file from Dataset 3 will be used as the basis for visualizing the results across all models. Figures 4.1 and 4.2 present the predictions from all the models evaluated with the MSE and Huber loss functions, respectively. The blue lines represent the actual data from the sensors. The green lines show predictions using the parameter set from Model 1, while the black lines illustrate predictions using the parameter set from Model 2. The dashed lines show predictions from Manual Calibration. Model 1 generally fits the real data better in most cases. More details can be found in sections 4.1.2 and 4.1.3.

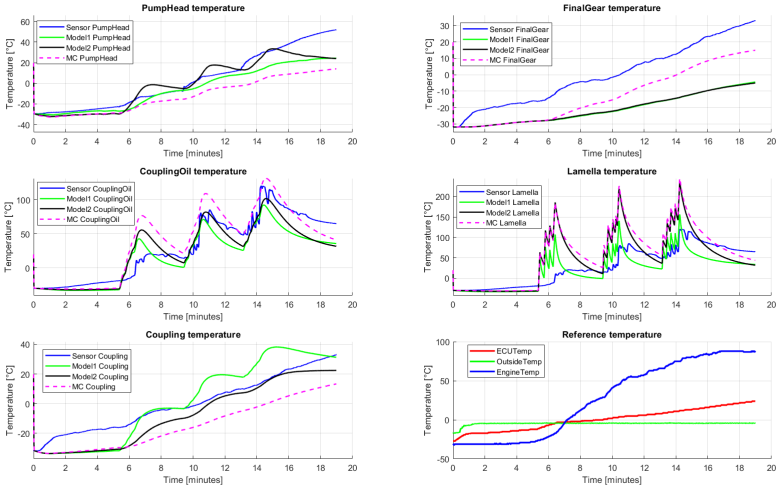

Figure 4.1 Temperature predictions of all the models trained using the MSE loss function.

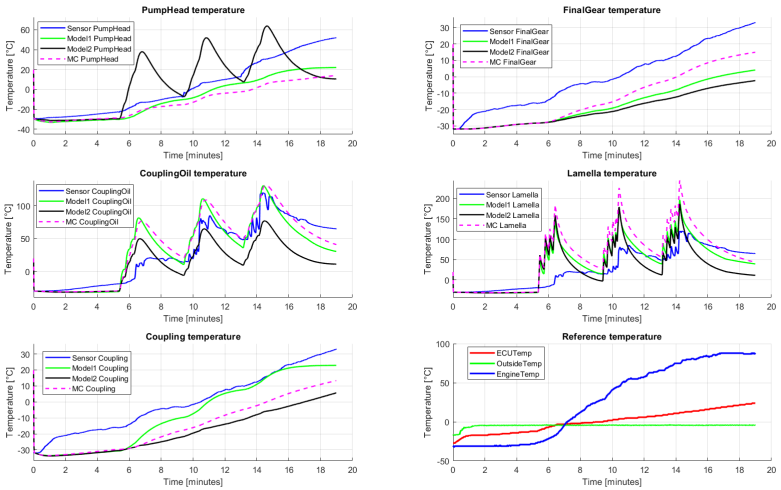

Figure 4.2 Temperature predictions of all the models trained using the Huber loss function.

4.1.2 Model 1

Table 4.2 shows the average loss metrics of Model 1, calculated using MSE, with three different measurements. The first column displays the average MSE loss of Model 1, trained using the MSE function on its DQN, across three datasets. In the second column, Model 1 utilizes the Huber loss function to train the DQN, while the third column shows the average loss for Manual Calibration using MSE. These metrics provide a baseline for comparing Model 1's results..

Table 4.2 Average MSE Loss of Model 1 and Manual Calibration.

	Model 1 (MSE)	Model 1 (Huber Loss)	Manual Calibration
Dataset 1	16.8	16.3	18.9
Dataset 2	69.7	58.9	71.6
Dataset 3	143.4	138.5	191.4

Figures 4.3 and 4.4 visualize the results obtained from Dataset 3. The blue lines represent the actual data from the sensors. The green lines show predictions using the parameter set from Model 1. The dashed lines show predictions from Manual Calibration.

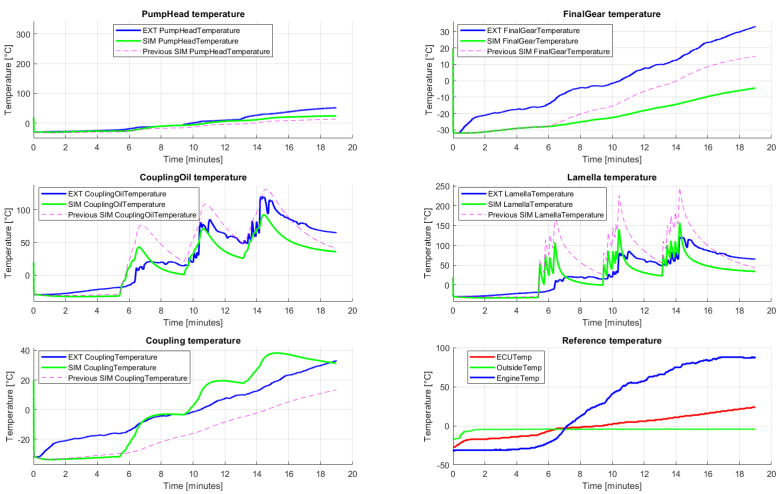

Figure 4.3 Predictions from Model 1 trained using Dataset 3 and the MSE loss function.

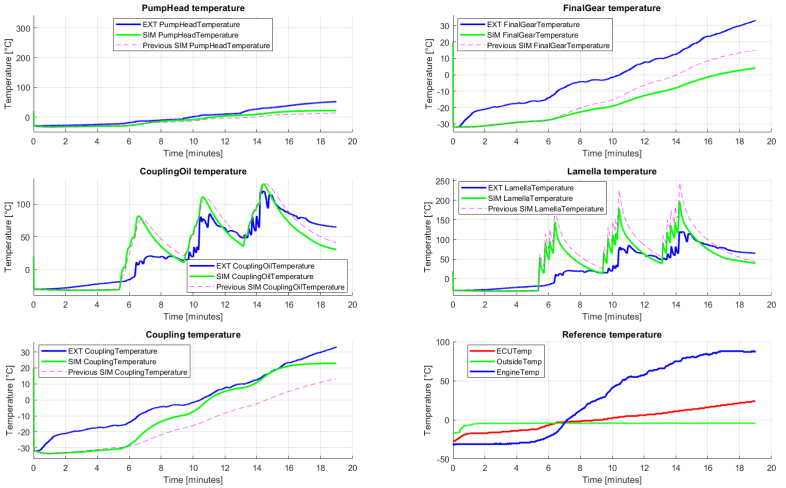

Figure 4.4 Predictions from Model 1 trained using Dataset 3 and the Huber loss function.

Two separate replicas of Model 1 were trained, each using a different loss function: one with Huber loss and the other with MSE. Figures 4.3 and 4.4 show that Huber loss leads to more accurate predictions for Coupling, CouplingOil, PumpHead and Lamella temperatures. However, the predictions for FinalGear temperature tend to undershoot, which might be due to limitations within the state-space model. This poses a trade-off, implying that it might not be feasible to find one parameter set that provides good predictions for all components.

At the start of each episode, all parameters were reset to zero. Subsequently, the average MSE loss was recalculated after the agent performed a new action. Consequently, this reset led to a spike in the average MSE loss. The average MSE loss and Huber loss are plotted in figures 4.5 and 4.6. These figures illustrate that while both models were able to converge before resetting for a new episode, the model trained with the MSE loss function typically demonstrated a longer and more stable calibration period. Notably, the model utilizing the Huber loss function achieved the lowest loss value.

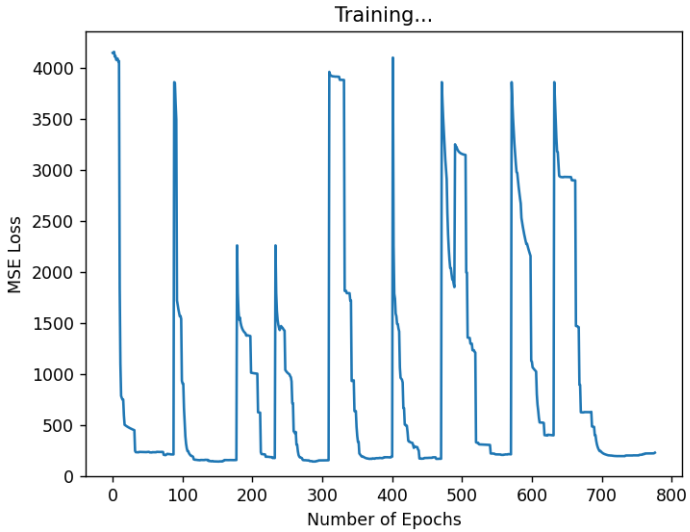

Figure 4.5 Average MSE loss of Model 1 trained using Dataset 3 and the MSE loss function.

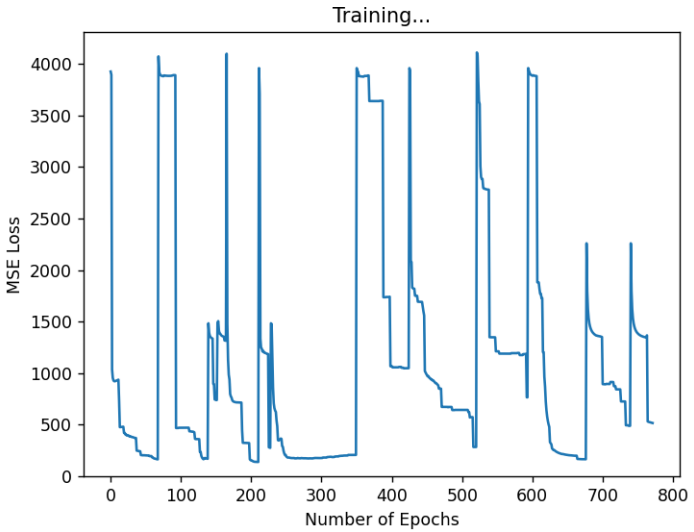

Figure 4.6 Average MSE loss of Model 1 trained using Dataset 3 and the Huber loss function.

Moreover, Dataset 2 was utilized for a five-day training period on a laptop to evaluate Model 1's performance under conditions where it could explore and learn more extensively. Figure 4.7 shows the average MSE loss of Model 1 throughout this training process.

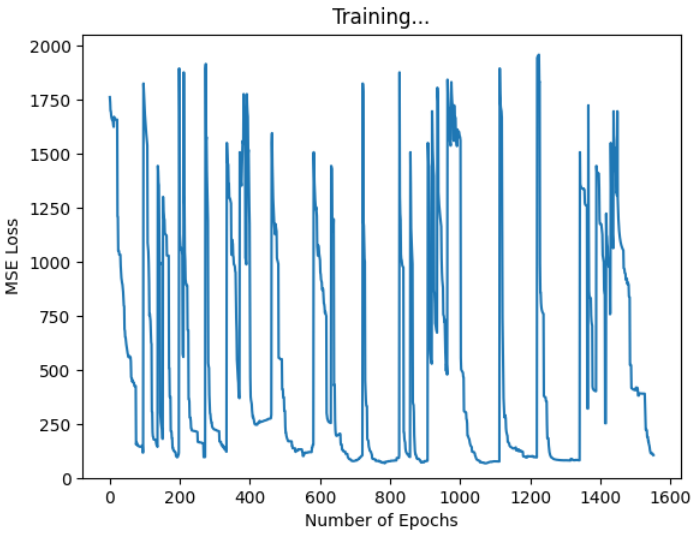

Figure 4.7 Average MSE loss of Model 1 trained using Dataset 2 and the MSE loss function after five days.

Figure 4.7 demonstrates that given sufficient time for exploration and learning, the grey-box models are capable of identifying even more optimal parameter sets. It was able to find the optimal set around epoch 1100.

4.1.3 Model 2

Similar to 4.1.2, the average MSE loss metrics of Model 2 are shown in Table 4.3.

Table 4.3 Average MSE loss of Model 2 and Manual Calibration.

	Model 2 (MSE)	Model 2 (Huber Loss)	Manual Calibration
Dataset 1	30	19.5	18.9
Dataset 2	73	72.9	71.6
Dataset 3	177	219.5	191.4

Figures 4.8 and 4.9 visualize the results obtained from Dataset 3. The blue lines represent the actual data from the sensors. The green lines show predictions using

the parameter set from Model 2. The dashed lines show predictions from Manual Calibration.

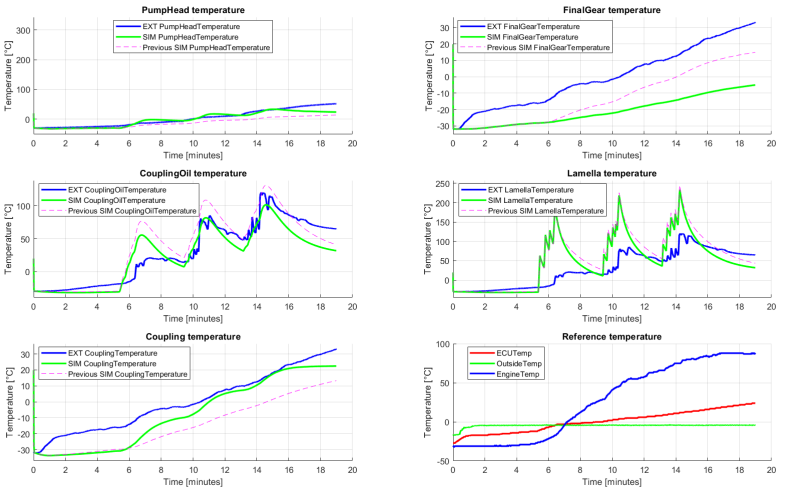

Figure 4.8 Predictions from Model 2 trained using Dataset 3 and the MSE loss function.

Model 2 demonstrated poor fitting to the real data, performing worse than Manual Calibration in most cases.

The average MSE loss and Huber loss are seen in figures 4.10 and 4.11. These figures show that the model using the MSE loss function generally experienced a longer and more stable calibration period before resetting. In contrast, the model employing the Huber loss function achieved the lowest loss value.

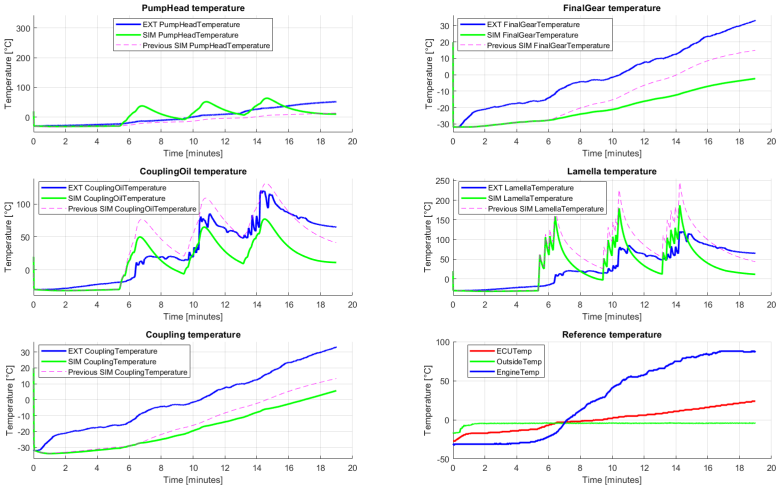

Figure 4.9 Predictions from Model 2 trained using Dataset 3 and the Huber loss function.

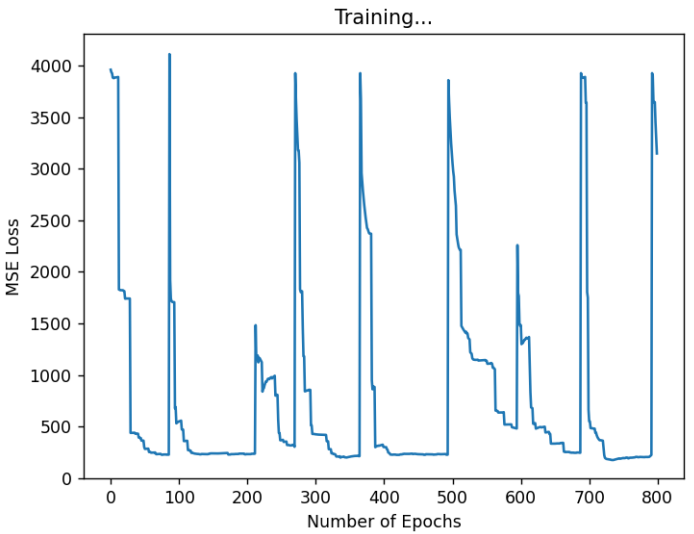

Figure 4.10 Average MSE loss of Model 2 trained using Dataset 3 and the MSE loss function.

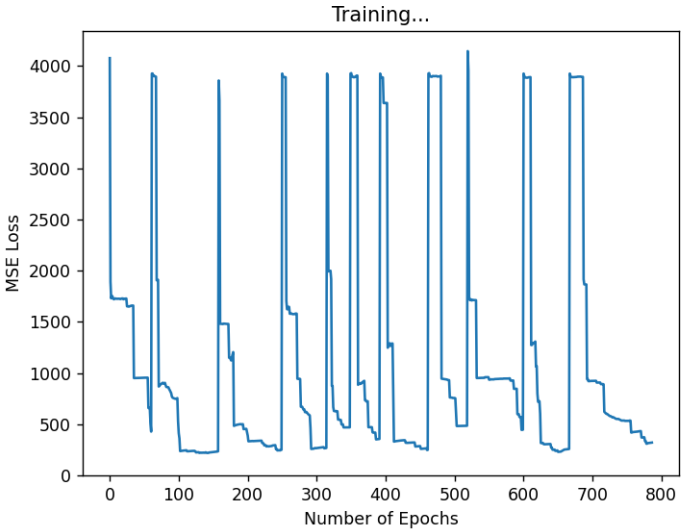

Figure 4.11 Average MSE loss of Model 2 trained using Dataset 3 and the Huber loss function.

4.2 Black-Box Models

4.2.1 Benchmark model results

Model	RMSE	Variance	High Frequency Power
Random Forest	8.60	56.39	7.64
KNN	8.82	121.94	32.31
SVM	6.90	50.09	10.24

Table 4.4 Benchmark results

From the results in Table 4.4 and Figure 4.12 we can compare the performance of three benchmark models (Random Forest, KNN, and SVM) on the warm drive dataset. The SVM model achieves the lowest RMSE (6.897613) and variance (50.088986), indicating better accuracy and consistency compared to the Random Forest and KNN models. However, the SVM model has a higher High Frequency Power (10.24166) than the Random Forest model (7.63996), indicating more high-frequency noise.

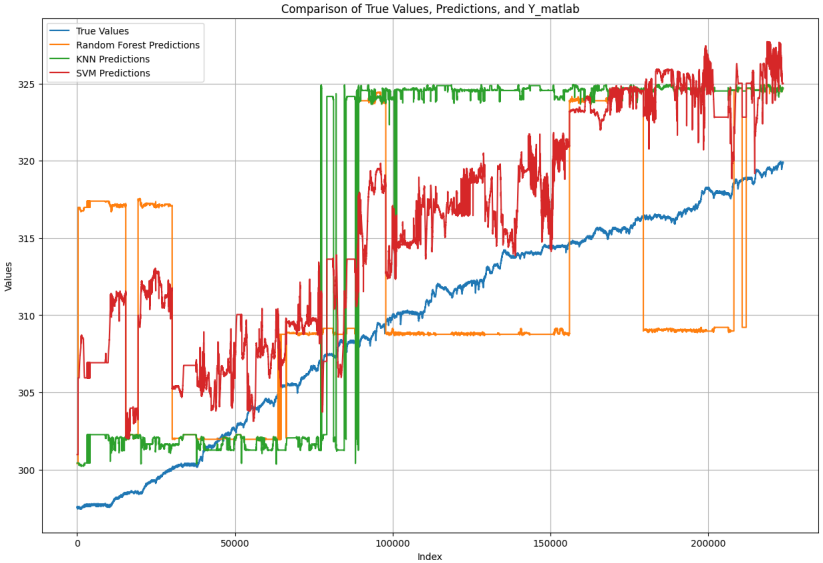

Figure 4.12 Performance of the benchmark model on the warm drive data set.

4.2.2 Effect of Resampling on RMSE

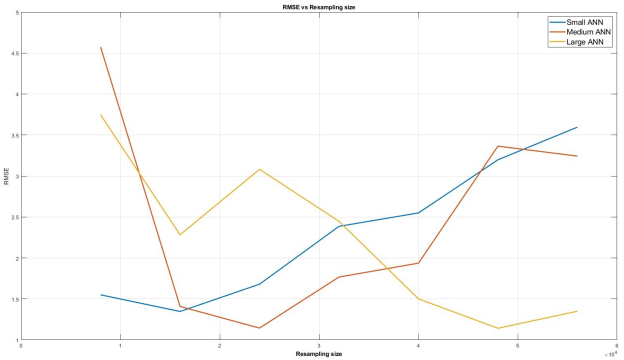

Figure 4.13 Trend of RMSE as resample size varies and across model sizes.

Figure 4.13 and Table 4.5 show the trend of RMSE as the resample size varies across small, medium, and large ANN models. The data indicates that as resampling size increases, the RMSE initially decreases for all models but then deteriorates for

Model	Resample size	RMSE
Small	16000	1.35
Medium	24000	1.14
Large	48000	1.14

Table 4.5 Best results on RMSE-Sample size test.

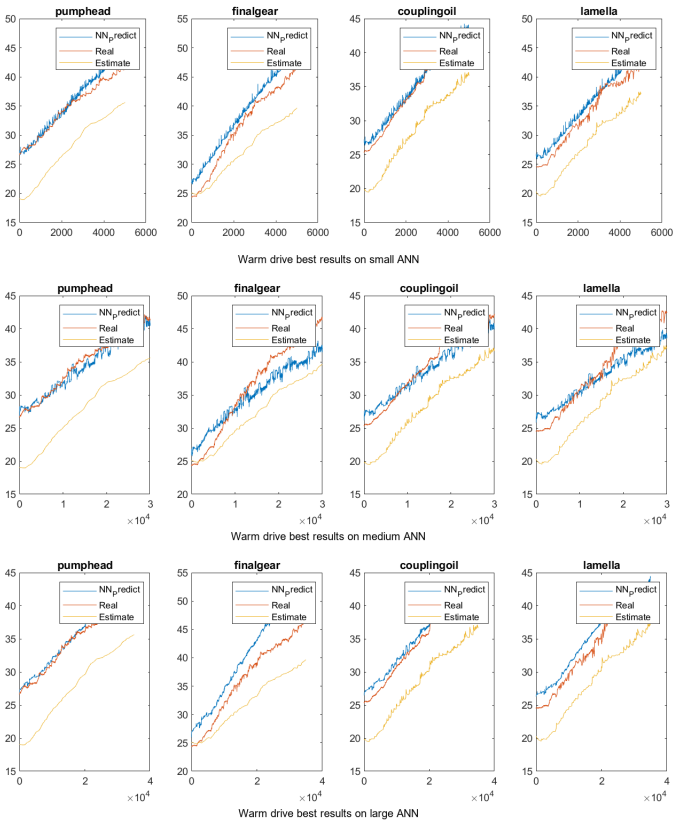

Figure 4.14 Visual results of the best Neural network models plotted with the real and estimated results. [Sample index (x-axis) Temperature in $°C$ (y-axis)]

the small and medium models at higher resampling sizes. The large ANN model consistently shows the lowest RMSE at higher resampling sizes. Figure 4.14 shows the final test results using the best resampling size (in terms of RMSE) for each ANN model.

The results highlight that as the resampling size increases, the complexity of the model required to maintain a low RMSE also increases, this is discussed further in section 5.2.1. Initially, all models benefit from increased resampling, suggesting improved pattern recognition due to more comprehensive data. However, the small ANN model struggles with higher resampling sizes, likely due to its limited capacity to handle the increased temporal complexity. Conversely, the medium and large ANN models show better adaptability, with the large ANN consistently performing the best at higher resampling sizes. This indicates that larger models are more suited for managing large datasets with complex temporal dynamics.

4.2.3 Effect of Resampling on THFP

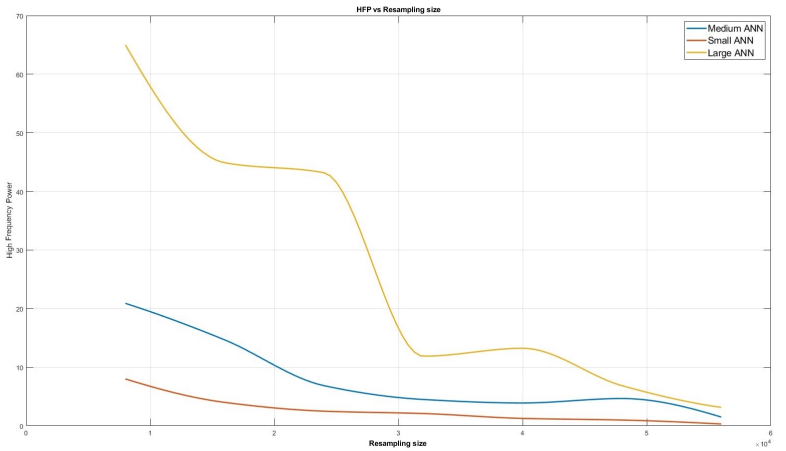

Figure 4.15 Trend of total high frequency power as resample size varies and across model sizes.

Model	Sampling size	Total High Frequency Power
Small	56000	0.32
Medium	56000	1.50
Large	56000	3.14

Table 4.6 Best results on Total high Frequency Power-Sample size test.

The results from Figure 4.15 and Table 4.6 concern the effect of resampling size on THFP across different ANN models. As resampling size increases, THFP de-

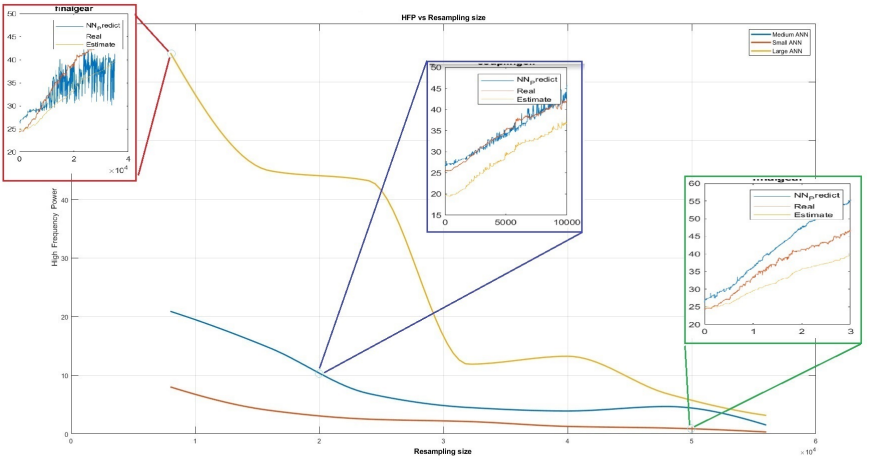

Figure 4.16 Trend of total high frequency power with examples.

creases, suggesting more stable models with less high-frequency noise. Figure 4.16 illustrates these results in small snapshot views across points on the curve shown in Figure 4.15. Finally Figure 4.17 shows the final test results using the best resampling size (in terms of THFP) for each ANN model.

The data demonstrates that larger models require higher resampling rates to maintain stability and avoid high-frequency noise. While large models capture more intricate patterns, they risk instability without detailed data. In contrast, whilst still showing improvement, smaller models are still stable at lower resampling sizes but may lack the capacity to capture complex dynamics as shown from the RMSE results.

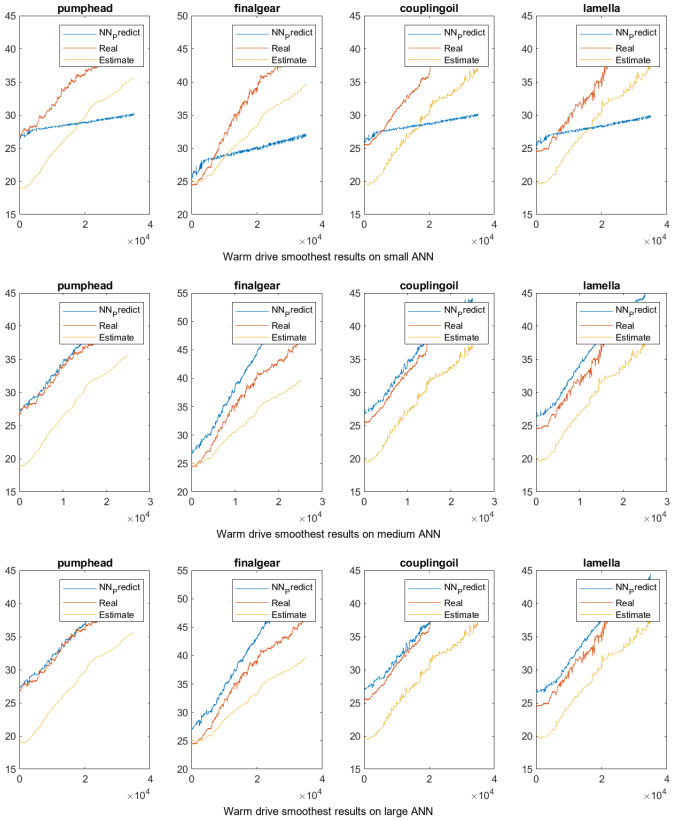

Figure 4.17 Visual results of the smoothest Neural network models plotted with the real and estimated results. [Sample index (x-axis) Temperature in $°C$ (y-axis)]

4.2.4 Effect of smoothing on results

Figure 4.18 reveals the impact of the moving average filter and exponential smoothing (we will collectively refer to this as "smoothing" going forward) on model predictions. Smoothing generally aligns predictions with actual data, reducing variance and high-frequency noise. This results in a much less volatile prediction pattern.

However, inappropriate smoothing can amplify variability, especially in complex data scenarios such as the one shown in Figure 4.19; displaying a paradoxical behavior in the smoothing results.

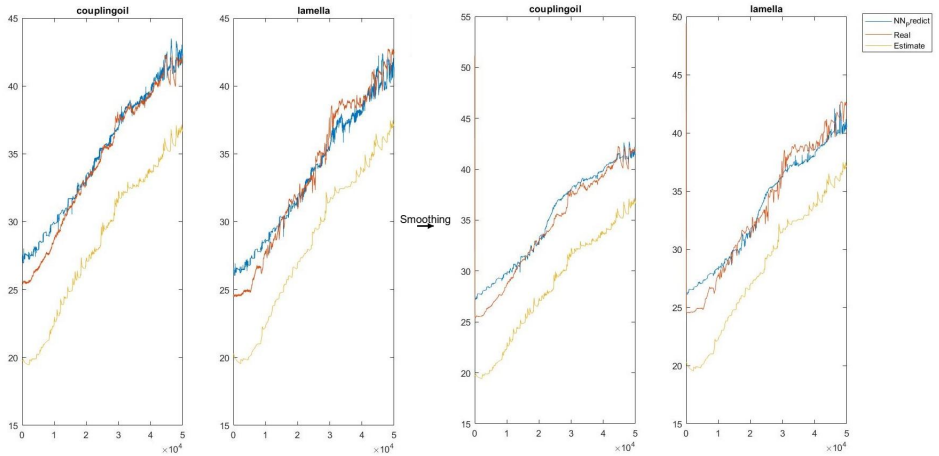

Figure 4.18 Effect of smoothing on data. [Sample index (x-axis) Temperature in $°C$ (y-axis)]

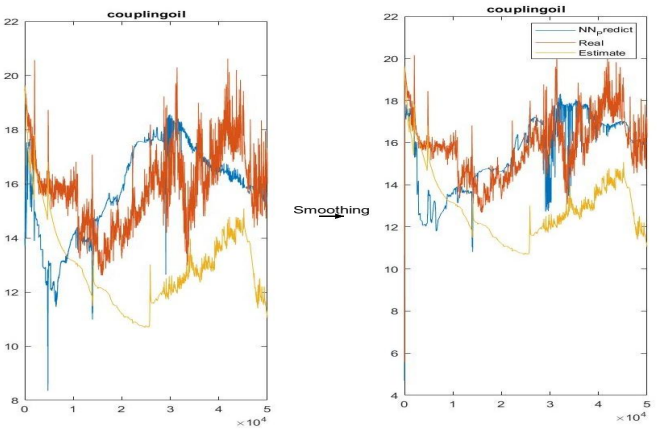

Figure 4.19 Case where smoothing leads to more variance. [Sample index (x-axis) Temperature in $°C$ (y-axis)]

4.2.5 Best model results on combined data

Model	$RMSE_{NN}$	$RMSE_{est}$	σ^2_{NN}	σ^2_{est}	$THFP_{NN}$	$THFP_{est}$
Small, 5000	3.50	9.12	2.63	59.12	317.27	507.42
Medium, 24000	3.40	9.12	6.74	59.12	418.40	509.11
Large, 50000	2.50	9.12	12.42	59.11	381.31	508.89

Table 4.7 Table of RMSE, Variances, and HF Powers for combined best results.

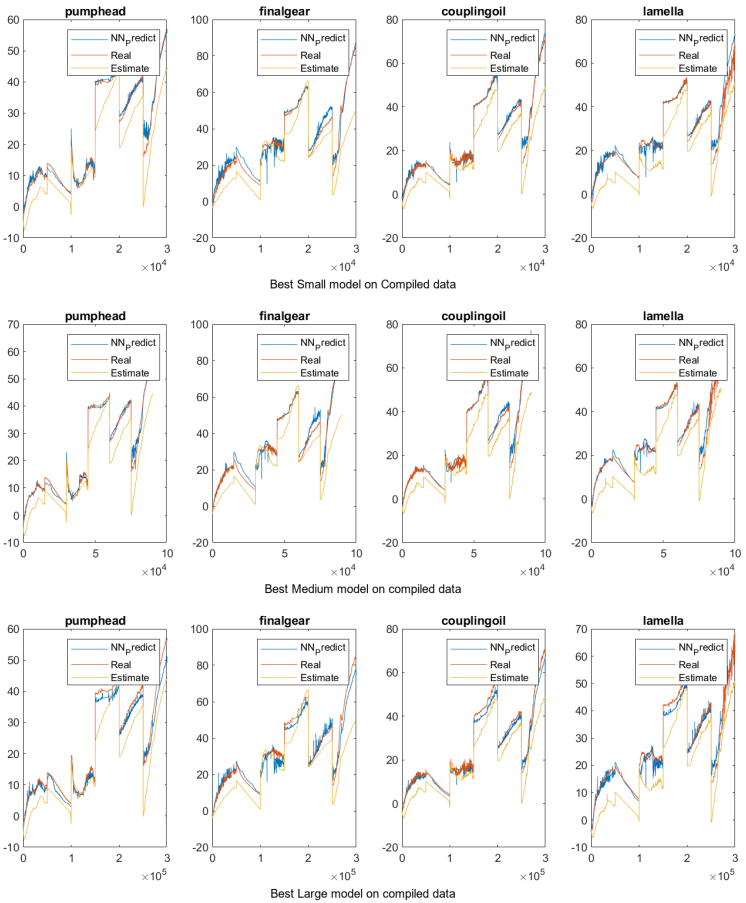

Figure 4.20 Visual results of the best Neural network models tested on compiled data. [Sample index (x-axis) Temperature in $°C$ (y-axis)]

Table 4.7 and Figure 4.20 summarize the performance of ANN models on compiled datasets. The large ANN model achieves the lowest RMSE, but with higher variance and THFP, indicating greater fluctuations in predictions. Smaller models, while less accurate, provide more consistent and stable predictions. For the purpose of this paper however, we will proceed with the assumption that the large model performs the best.

4.2.6 LSTM and GRU

Model	$RMSE_{NN}$	$RMSE_{est}$	σ^2_{NN}	σ^2_{est}	$THFP_{NN}$	$THFP_{est}$
GRU small	12.87	5.75	6.28	27.27	9.97	0.021
GRU Large	4.65	5.75	23.96	27.27	2.13	0.021
LSTM small	8.84	5.75	1.79	27.27	11.11	0.021
LSTM Large	1.46	5.75	26.28	27.27	2.23	0.021

Table 4.8 Warm drive results for LSTM and GRU models.

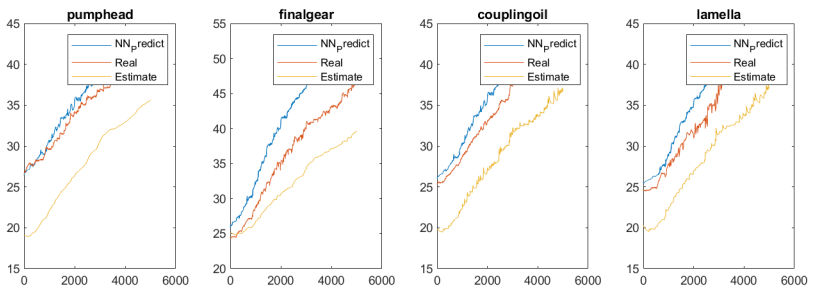

Figure 4.21 Best GRU result.[Sample index (x-axis) Temperature in $°C$ (y-axis)]

The results in this subsection present the performance of LSTM and GRU models. Dynamic regularization significantly improves their performance, reducing RMSE and THFP. from Table 4.8 we can observe that LSTMs generally perform better on the smaller dataset in terms of accuracy (RMSE). Conversely, the GRU models perform better in terms of THFP. Figures 4.21 and 4.22 display the results acquired of the "GRU Large" and "LSTM Large" models.

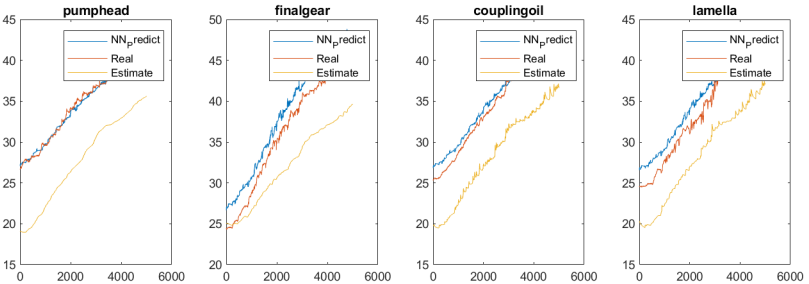

Figure 4.22 Best LSTM result.[Sample index (x-axis) Temperature in $°C$ (y-axis)]

4.2.7 Effect of Dynamic regularization

	Dynamic Regularization			
	With		Without	
Model	LSTM	GRU	LSTM	GRU
$RMSE_{NN}$	0.96	2.34	1.46	2.65
σ^2_{NN}	0.05	16.31	26.28	31.31
$THFP_{NN}$	0.21	2.23	11.11	2.13

Table 4.9 Table with Dynamic Regularization.

Subsection 4.2.7 shows the improvement that Dynamic regularization provides to the LSTM and GRU models. Table 4.9 displays improvements in both LSTM and GRU models after dynamic regularization, with the improvements in the LSTM model especially noteworthy, achieving the best results across all model types on all criteria for the small data set. Figure 4.23 shows an interesting phenomena that occurs in the LSTM model under higher Dynamic regularization, where a discretization effect takes place on the prediction, with the appearance of step functions that at first seem similar to the decision boundaries visible in the benchmark results.

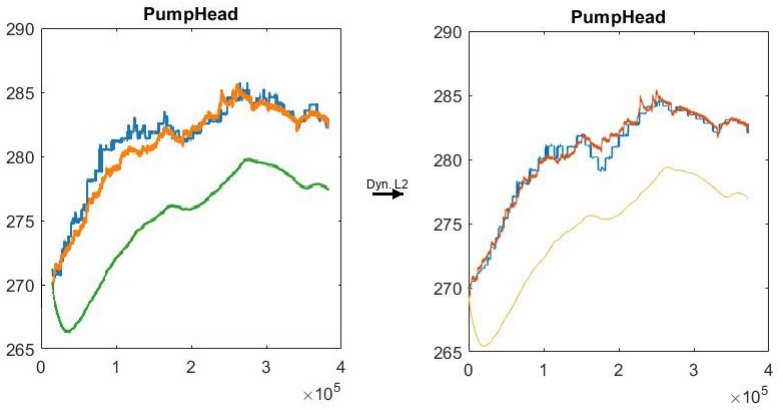

Figure 4.23 LSTM Discretization effect after Dynamic L2 regularization. [Sample index (x-axis) Temperature in Kelvin (y-axis)]

4.2.8 Comparison against best ANN

Metric	Model		
	ANN	LSTM	GRU
RMSE	2.50	2.43	2.75
Var	12.42	20.43	11.84
THFP	381.31	469.90	464.21

Table 4.10 Comparison of Models with Highlighted Minimum Values.

The compiled results compare the best performing LSTM and GRU models against the best ANN model. Figures 4.24, 4.25, and 4.26 visually demonstrate these findings. The ANN model's predictions align closely with the real data while maintaining smooth transitions, reflecting its lower THFP and balanced performance. The LSTM model, although accurate, shows more fluctuations, corresponding with its higher variance and THFP. The GRU model, while consistent, shows some degree of instability in the form of high-frequency noise.

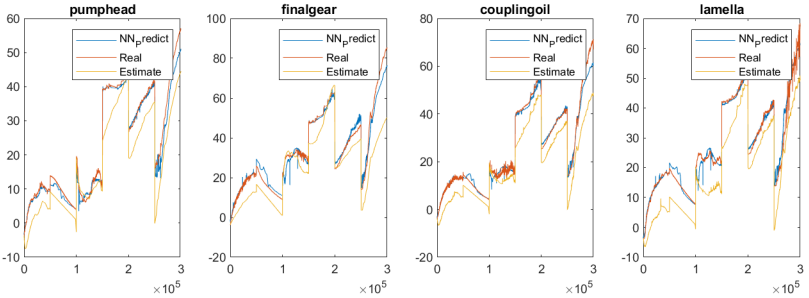

Figure 4.24 Best ANN results on full data.[Sample index (x-axis) Temperature in $°C$ (y-axis)]

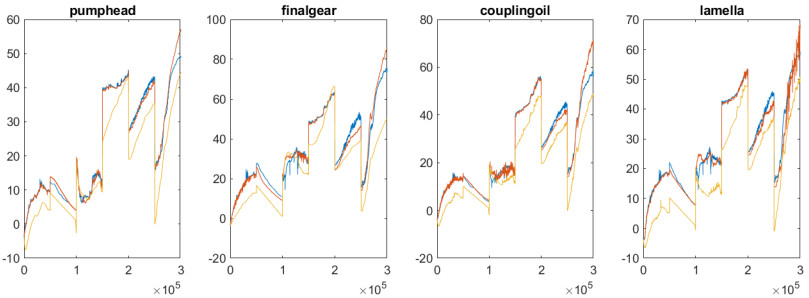

Figure 4.25 Best LSTM results on full data.[Sample index (x-axis) Temperature in $°C$ (y-axis)]

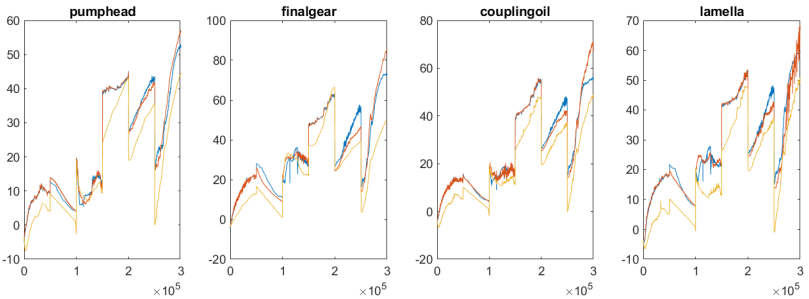

Figure 4.26 Best GRU results on full data.[Sample index (x-axis) Temperature in $°C$ (y-axis)]

5

Discussion and Conclusion

5.1 Grey-box models

It is a fact that experts invest 2-3 weeks in Manual Calibration, yielding exceptionally accurate results that are challenging to surpass. Figures 4.1 and 4.2 demonstrate that Model 1 achieves a superior fit to the real data compared to both Model 2 and Manual Calibration. Despite this, none of the models are able to accurately fit the actual data due to inherent limitations within the state-space model. However, as the average MSE loss values of Model 1 in Table 4.2 are lower than the average MSE loss values of Manual Calibration across all tests, the results show that RL can help reduce the effort required to calibrate the temperature model while improving the accuracy of predictions. This is achieved by identifying parameter sets with lower average MSE loss values. The grey-box models can find these parameter sets in just 1-2 days of training on a laptop, reducing the Manual Calibration effort by 90%.

5.1.1 Model 1

The results in Table 4.2 indicate that employing the Huber loss function during training leads to a lower average MSE loss for Model 1 compared to using the MSE loss function. This suggests that the Huber loss function may be more effective for this model, potentially due to its robustness against outliers. The lower average MSE loss with Huber loss may be explained by its design, which combines the advantages of MSE and MAE. It behaves like MSE for small errors, providing sensitivity to minor changes, but acts more like MAE for larger errors, reducing the impact of outliers. This characteristic can lead to more stable and accurate model performance. More details can be seen in Figure 4.6 where the average MSE loss is high when the model is reset and low when the agent modifies the parameter values.

Within the scope of the thesis, only a select number of projects were tested. Despite the fact that the outcomes surpassed those of Manual Calibration, it would be beneficial to evaluate additional projects to confirm the performance. An alternative strategy could involve experimenting with different hyperparameters, states, or reward values to achieve a better average MSE loss value.

5.1.2 Model 2

Table 4.3 shows that the average MSE loss values of Model 2 are higher than those of Model 1 and Manual Calibration. This indicates that the simpler architecture of a single DQN may offer advantages in terms of stability and convergence speed. The lower average MSE loss for Model 1 might be due to fewer layers and reduced complexity, which can lead to faster convergence. In contrast, Model 2's increased complexity from the addition of a target network and a policy network could lead to slower learning and higher average MSE loss. This low performance might be resolved through further tuning of hyperparameters, but the current results indicate that a simpler model structure may offer better performance in this context. However, it would be intriguing to explore whether updating the state, reward, or adjusting hyperparameters could yield better results. Perhaps, it might be worthwhile to explore the use of short-term memory from Model 1 as an alternative approach.

5.2 Black-box models

5.2.1 Effect of Resampling on RMSE

As depicted by the graph presented in Figure 4.13, there is a clear trend indicating that as the resampling size increases, the complexity of the model required to maintain a low RMSE also increases.

Initially, at lower resampling sizes, all three models (small, medium, and large) show a decreasing trend in RMSE, suggesting that resampling improves the model's ability to capture relevant patterns in the data, this makes sense as larger resampling size means less data is under-sampled and more information is available to the model. However, as the resampling size continues to increase, the performance of the small ANN begins to deteriorate. This can be attributed to the small model's limited capacity to handle the increased volume of data and the associated temporal complexity.

In contrast, the medium and large ANN models demonstrate better adaptability to higher resampling sizes. The medium ANN maintains a relatively stable RMSE up to a certain point before also experiencing a performance decline, albeit less severe than the small model, this again can be attributed to the fact that at some point some datasets start to get over-sampled, the medium model cannot handle the presence of extra information and loses the ability to grasp the temporal behavior of the data. The large ANN, on the other hand, consistently shows the lowest RMSE at higher resampling sizes, indicating its superior ability to manage large datasets with complex temporal dynamics.

This trend underscores the importance of selecting an appropriately sized model based on the sampling rate of the data. For sensors that have higher sampling rates to adequately capture the temporal nuances, larger models with greater complexity are necessary to achieve optimal performance, as the temporal aspect will be more prevalent in higher sample sizes (i.e. higher sampling rates). Smaller models, while computationally less intensive, may struggle to maintain accuracy as data complexity increases.

The results emphasize the presence of a trade-off between sampling time and model complexity. The resampling size directly correlates with the temporal complexity of the data, which is determined by the sampling time. A shorter sampling time captures more detailed temporal information, necessitating the use of more complex models to accurately process and predict the data without overfitting or increased prediction variance. Smaller models, while offering better computational efficiency, may struggle to handle complex temporal properties, making them less suitable as the data's temporal complexity increases. Therefore, when high-resolution temporal data is required, larger models, despite their higher computational demands, are essential to ensure precise temperature estimation and model stability.

5.2.2 Effect of Resampling on Total High Frequency Power

The observed trend in the results indicates that as the resampling size increases, the THFP consistently decreases. This finding suggests that higher resampling rates contribute to more stable models with reduced high-frequency noise, a crucial aspect for real-time sensor data applications in automotive environments.

Interestingly, the data demonstrates that large models exhibit a significant reduction in THFP only at extremely high resampling sizes. This implies that without sufficient resampling, large models may be prone to instability due to their complexity and the increased difficulty in managing temporal variations in the data. This relationship highlights a key trade-off: while larger models have the potential to capture more intricate patterns, they require more detailed data (achieved through higher resampling rates) to maintain stability and avoid introducing high-frequency noise.

Conversely, less complex models (such as the small and medium ANNs) inherently exhibit lower THFP, indicating that they are more stable at lower resampling sizes. This stability, however, comes at the cost of potentially reduced capability to capture complex temporal dynamics present in the data, as discussed in the section 5.2.1.

The results illustrate the importance of stability over mere accuracy in model selection. While larger models may offer greater predictive accuracy, their instability at lower resampling rates makes them less suitable for practical applications without high-resolution data. This instability, characterized by higher THFP, can lead to unreliable performance in real-time systems. Therefore, it is crucial to balance model complexity with the resampling capabilities of the data collection system to ensure both stability and efficiency. This approach ensures that temperature estimation models are not only accurate but also robust and reliable, effectively avoiding the pitfalls of high-frequency noise and making efficient use of computational resources.

5.2.3 Effect of Smoothing on Results

The application of smoothing techniques was generally found to improve the alignment of the predicted values with the real data, reducing the variance and high-frequency noise.

However, Figure 4.19 highlights a case where smoothing introduces more variance rather than reducing it. This indicates that while smoothing can enhance the model's performance by reducing noise, it can sometimes amplify the variability in the predictions, especially in complex scenarios where the underlying data has significant fluctuations. This paradoxical effect suggests that smoothing needs to be carefully calibrated to balance noise reduction with the risk of introducing additional variance.

5.2.4 Analysis on full ANN results

The next set of experiments involved testing the compiled datasets on the ANN models, comparing their performance against the state-space model.

The RMSE values show that the large model achieves the lowest error (2.50), followed by the medium model (3.40), and the small model (3.50), these results can be found in Table 4.7. This suggests that the larger model, given the correct sampling rate, is more capable of capturing the underlying patterns in the data, resulting in more accurate temperature predictions.

However, when considering the variance of residuals (σ^2), the small model exhibits the least variance in its predictions (2.63), indicating more consistent performance. In contrast, the large model, despite its lower RMSE, shows a higher variance (12.42), reflecting greater fluctuations in its predictions. This trade-off between accuracy and consistency must be carefully managed in practical applications, especially in real-time systems where stability is crucial.

The Total High Frequency Power (THFP) values further highlight the differences in model behavior. The small model has the lowest THFP (317.26), indicating the least amount of high-frequency noise and suggesting it is more stable in its predictions. The medium and large models have higher THFP values (418.40 and 381.31, respectively), implying they introduce more high-frequency components into their predictions, which could lead to instability in real-time applications.

The visual results in Figure 4.20 corroborate these findings. For each component (pumphead, finalgear, couplingoil, and lamella), the predictions of the large model closely follow the real data, but with more pronounced fluctuations compared to the smaller models. The small model provides smoother predictions with fewer high-frequency variations, however as we can see from the results, it is unable to keep up with the trends in the data due to the higher resampling size.

5.2.5 LSTM and GRU Models

The introduction of LSTM and GRU models brings another dimension to our analysis of temperature prediction models. The results show that large LSTM models perform comparably well, achieving a comparable RMSE (1.46) to all the ANN models tested on the initial dataset. The remaining large and small models however, do not perform as well.

Interestingly, despite the high RMSE, the THFP values for both LSTM and GRU models are lower than those for the large ANN model discussed earlier. This indicates that while their predictions may not be as accurate, they are more stable and less noisy, which is critical for real-time applications.

The introduction of dynamic regularization dramatically improves the performance of both LSTM and GRU models. The RMSE for the LSTM model with dynamic regularization drops to 0.960, and for the GRU model, it decreases to 2.345. This improvement is accompanied by a significant reduction in THFP, suggesting that dynamic regularization effectively stabilizes the predictions and reduces high-frequency noise.

A notable observation from the dynamic regularization is its discretizing effect on the LSTM model's predictions. This effect could be due to the regularization method enforcing a form of periodicity or structure in the predictions, which warrants further investigation but is beyond the current scope.

From these results we can realize the potential of LSTM and GRU models, particularly when enhanced with dynamic regularization, for accurate and stable temperature estimation. However, their performance on compiled data needs to be com-

pared with ANN models to fully understand their efficacy in a more comprehensive dataset. This analysis will be discussed in the next section.

5.2.6 Compiled results and comparisons

In this final section, we compare the best performing LSTM and GRU models against the best ANN model using the compiled dataset. Table 4.10 summarizes the key metrics: RMSE, variance (Var), and Total High Frequency Power (THFP).

The large LSTM model achieves the lowest RMSE (2.43), outperforming both the ANN (2.50) and GRU (2.75) models. This indicates that the LSTM model is slightly more accurate in predicting temperature across the compiled dataset.

When it comes to variance, the GRU model excels with the lowest value (11.84), indicating the most consistent predictions. The ANN model follows with a variance of 12.42, while the LSTM model, despite its lower RMSE, shows higher variance (20.43). This suggests that while the LSTM model is accurate, its predictions are more variable, potentially due to its sensitivity to temporal dynamics.

The THFP metric reveals another critical aspect of model performance. The ANN model exhibits the lowest THFP (381.31), indicating the least amount of high-frequency noise and the most stable predictions. Both the LSTM (469.90) and GRU (464.21) models have higher THFP values, suggesting that their predictions are noisier and potentially less stable for real-time applications.

Interestingly, while the LSTM model performed exceptionally well on smaller datasets with dynamic regularization, it shows a marked increase in THFP on the compiled dataset. This increase is even more pronounced in the estimated state-space model. This discrepancy is attributed to the nature of the compiled data, where the time resets multiple times between datasets. Such resets introduce discontinuities that the ANN model appears more robust against compared to the LSTM and GRU models. The ANN model's ability to handle these resets better contributes to its superior performance in maintaining stability and low high-frequency noise in the compiled dataset. How this translates in real time deployment is yet to be seen.

5.3 Future Work

Although this project provided insights into the performance of the grey-box and black-box models, further investigation is needed to deepen the understanding of the optimal design and application of these approaches.

One promising option is hyperparameter optimization. To refine the performance of both models, a more comprehensive exploration of hyperparameters could

be conducted. This could involve experimenting with different values and combinations of hyperparameters, and observing the impact on model performance.

The current state-space model implemented in Matlab has been identified as a bottleneck due to its slow execution time. One potential solution is to reimplement the model in Python, which could lead to significant reductions in training time.

Exploring alternative architectures is another area worth pursuing. Double DQN or Prioritized Experience Replay could offer insights into achieving greater stability and enhanced performance of the grey-box models.

Additionally, the grey-box models depends heavily on the state-space model for temperature prediction. The accuracy and reliability of these predictions are closely tied to the quality of the state-space model. Therefore, making improvements to the state-space model has the potential to significantly enhance the performance of the grey-box models.

5.4 Conclusion

This thesis successfully addressed the questions from Section 1.2 by applying the grey-box and black-box models in a real-world project to improve calibration and prediction accuracy in the temperature models.

The grey-box models demonstrated a significant advantage over Manual Calibration by achieving better results with substantially less effort. It outperformed Manual Calibration, reducing the effort to calibrate by 90%, and reducing the need for an expert's involvement in the calibration process. This approach provides a more accessible and efficient method for parameter tuning, enabling broader application in various projects.

The black-box model also showed promising results, offering predictions with higher accuracy compared to Manual Calibration. By using a Neural network architecture to predict temperatures by directly mapping them from the input data, the black-box model reduced the complexity associated with traditional state-space models.

However, both models require further exploration of hyperparameter tuning and alternative architectures to optimize its performance. In addition, while both models demonstrated their potential, it is essential to validate the final results with a broader range of test data to ensure their reliability and robustness.

Bibliography

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng (2015). "Tensorflow: large-scale machine learning on heterogeneous distributed systems".

Afram, A. and F. Janabi-Sharifi. (2015). "Black-box modeling of residential hvac system and comparison of gray-box and black-box modeling methods."

Arendt, K., M. Jradi, H. R. Shaker, and C. Veje. (2018). "Comparative analysis of white-, gray-and black-box models for thermal simulation of indoor environment."

Breiman, L. (2001). "Random forests". *Machine learning* **45**:1, pp. 5–32.

Chollet, F. et al. (2015). *Keras. https://github.com/keras-team/keras.*

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling".

Fente, D. N. and D. K. Singh (2018). "Weather forecasting using artificial neural network". In: *2018 second international conference on inventive communication and computational technologies (ICICCT)*. IEEE, pp. 1757–1761.

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory". *Neural computation* **9**:8, pp. 1735–1780.

Huber, P. J. (1964). "Robust estimation of a location parameter. ann. math. statist. 35 (1) 73 - 101, march, 1964. https://doi.org/10.1214/aoms/1177703732".

Kingma, D. P. and J. Ba (2017). *Adam: a method for stochastic optimization*. arXiv: `1412.6980 [cs.LG]`.

Kiranyaz, S., O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman (2021). "1d convolutional neural networks and applications: a survey". *Mechanical Systems and Signal Processing* **151**, p. 107398. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2020.107398. URL: https://www.sciencedirect.com/science/article/pii/S0888327020307846.

Kreuzer, D., M. Munz, and S. Schlüter (2020). "Short-term temperature forecasts using a convolutional neural network—an application to different weather stations in germany". *Machine Learning with Applications* **2**, p. 100007.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". *Proceedings of the IEEE* **86**, pp. 2278–2324. DOI: 10.1109/5.726791.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2019). *Continuous control with deep reinforcement learning*. arXiv: 1509.02971 [cs.LG].

Macas, M., F. Moretti, A. Fonti, A. Giantomassi, G. Comodi, M. Annunziato, S. Pizzuti, and A. Capra. (2016). "The role of data sample size and dimensionality in neural network based forecasting of building heating related variables."

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning. nature, 518(7540):529–533, 2015".

Naing, W. Y. N. and Z. Z. Htike. (2015). "Forecasting of monthly temperature variations using random forests."

Nketiah, E. A., L. Chenlong, J. Yingchuan, and S. A. Aram. (2023). "Recurrent neural network modeling of multivariate time series and its application in temperature forecasting."

Olsson, F. (2019). "Thermal model and tuning presentation, BorgWarner internal document".

Olsson, F. (2023). "Genvi temperature estimation tuning guide, BorgWarner internal document".

Paszke, A. and M. Towers (2017). *Reinforcement learning (dqn) tutorial. https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.*

Roderick, M., J. MacGlashan, and S. Tellex (2017). "Implementing the deep q-network". *CoRR* **abs/1711.07478**. arXiv: 1711.07478. URL: http://arxiv.org/abs/1711.07478.

Svendenius, J. (2020). "Genvi sdd temperature estimation, BorgWarner internal document".

Tian, Y., M. A. Chao, C. Kulkarni, K. Goebel, and O. Fink (2020). *Real-time model calibration with deep reinforcement learning*. arXiv: 2006.04001 [eess.SP].

Uluocak, I. and M. Bilgili (2023). "Daily air temperature forecasting using lstm-cnn and gru-cnn models".

Watkins, C.J.C.H., and P. Dayan (1992). "Q-learning. mach learn 8, 279–292 (1992)."

Watkins, C.J.C.H. (1989). "Learning from delayed rewards. PhD thesis, University of Cambridge."

Zhang, Z. and Y. Dong (2020). "Temperature forecasting via convolutional recurrent neural networks based on time-series data". *Complexity* **2020**, pp. 1–8.

Appendices

A.1 Additional Figures

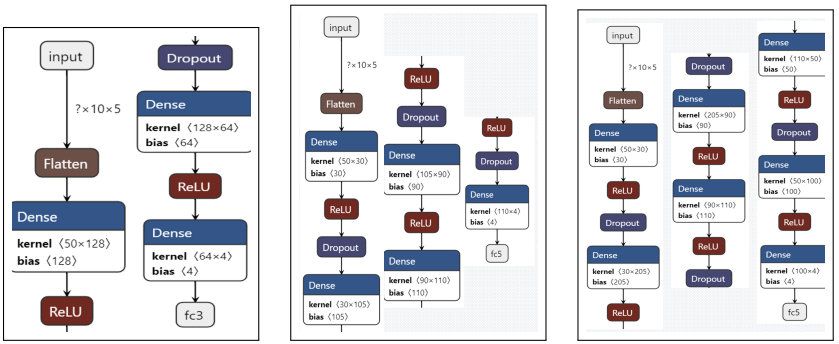

Figure .1 Architectures of ANN model. (from left to right: small, medium, large)

Model	Layer	Regularization Value
LSTM	LSTM Layer 1	1×10^{-4}
	LSTM Layer 2	2×10^{-3}
GRU	GRU Layer 1	2×10^{-4}
	GRU Layer 2	1×10^{-3}

Table .1 Regularization values for each layer in the LSTM and GRU models.

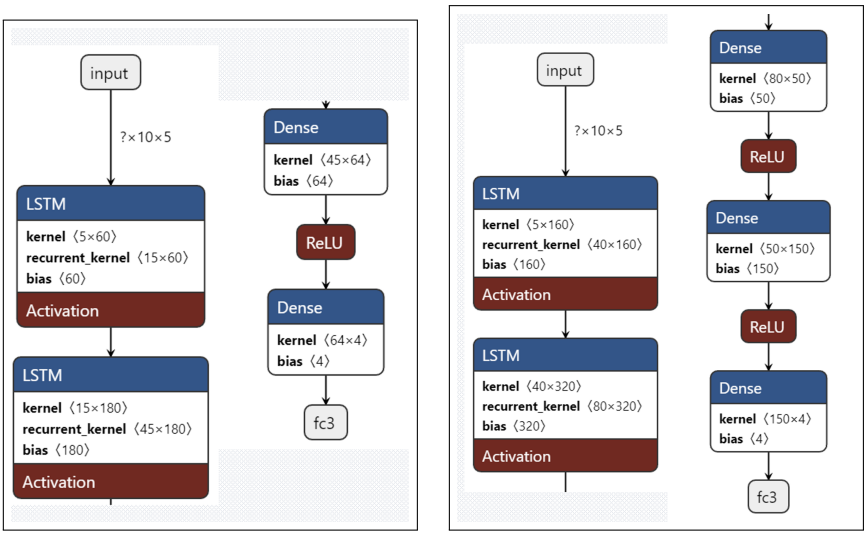

Figure .2 Architectures of LSTM model. (left: small, right: large)

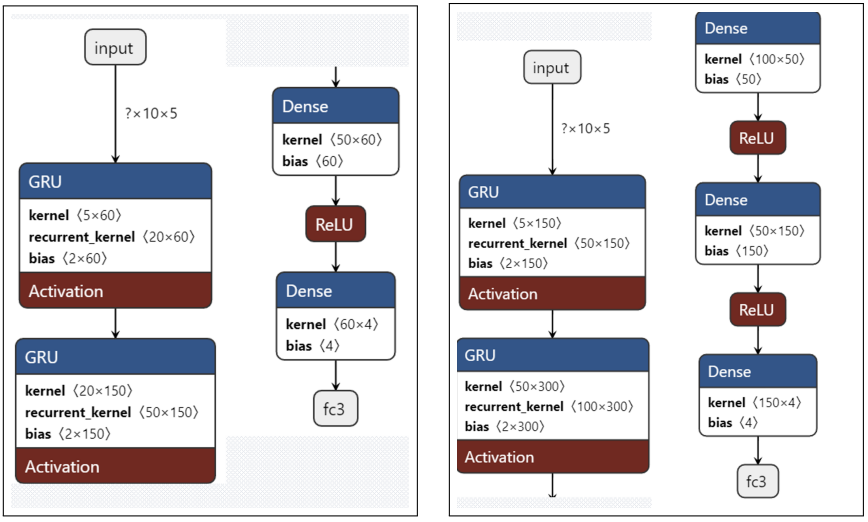

Figure .3 Architectures of GRU model. (left: small, right: large)

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
June 2024

Document Number
TFRT-6231

Author(s)
Van Duy Dang
Basim Elessawi

Supervisor
Meike Rönn. BorgWarner Sweden AB
Arne Hörberg, BorgWarner Sweden AB
Richard Pates, Dept. of Automatic Control, Lund University, Sweden
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Improving Temperature Estimation Models using Machine Learning Techniques

Abstract

Temperature estimation models are crucial for various products manufactured by BorgWarner. These models often require manual calibration, where experts adjust parameters to ensure accuracy. However, this process can be slow and prone to errors. This thesis investigates how Machine Learning techniques can be used to improve accuracy and efficiency of temperature estimation models.

Both black-box and grey-box approaches are used to evaluate the effectiveness of machine learning-based calibration. The black-box model employs techniques such as Decision Trees, Random Forests, and Neural Networks to predict temperature directly from raw input data, bypassing traditional temperature estimation processes. The grey-box model, on the other hand, uses Deep Q-learning to adjust the calibration automatically.

Results show that the black box model achieves better performance compared to conventional temperature estimation methods. Meanwhile, the grey-box model not only significantly improves accuracy compared to the manual calibration method, but also reduces the need for manual calibration in temperature estimation models.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-71

Recipient's notes

Security classification