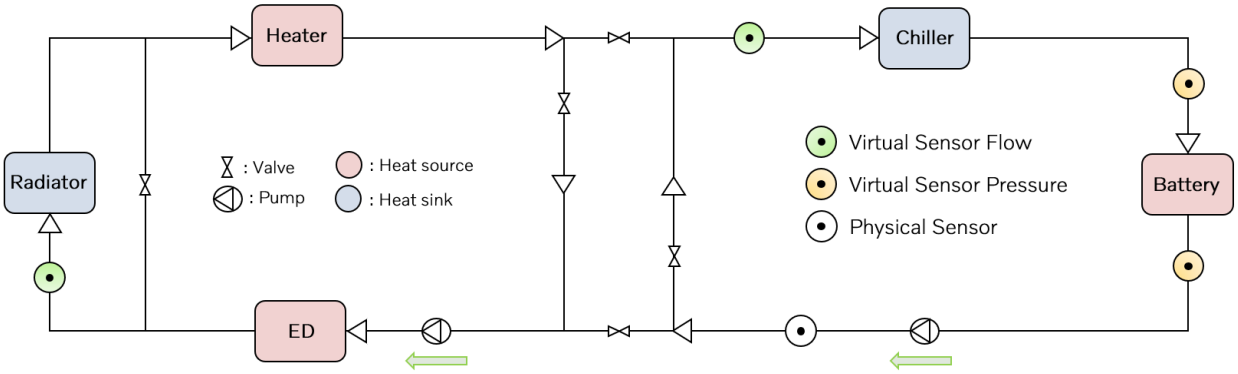
CHALMERS
UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Machine-Learning Based Virtual Sensor for Thermal Management

Master's thesis in Complex Adaptive Systems and Sustainable Energy Systems

Sky Sunsaksawat, Albert Ådén

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Machine-Learning Based Virtual Sensor for Thermal Management

Sky Sunsaksawat, Albert Ådén

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Machine-Learning Based Virtual Sensor for Thermal Management

Sky Sunsaksawat, Albert Ådén

Supervisor: Peter Damaschke, Department of Computer Science and Engineering
Advisor: Babak Heydarnezhad, CAE Analysis Engineer at Volvo Cars
Examiner: Ashkan Panahi, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Description of the picture on the cover page (if applicable)

Typeset in LaTeX
Gothenburg, Sweden 2025

Machine-Learning Based Virtual Sensor for Thermal Management

Sky Sunsaksawat, Albert Ådén
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Battery Electric Vehicles (BEVs) are increasingly prominent in the transition toward sustainable transportation. However, they face challenges compared to internal combustion engine vehicles, including range anxiety, the relatively shorter technical lifetime of their battery cells, and the risk of thermal runaway. A critical component in addressing these issues is the thermal management system (TMS), which relies on accurate real-time measurements or estimations of coolant pressure and flow. Due to cost and space constraints, the number of physical sensors in the TMS is limited, and alternative methods for providing these key performance indicators (KPIs) must be explored. This thesis investigates the development of a machine learning (ML)-based virtual sensor to predict the KPIs using data from steady-state simulations of a digital twin modeled in GT-SUITE.

Five traditional ML algorithms and three artificial neural network (ANN) architectures were developed and evaluated based on accuracy, storage size, and, ultimately, inference time, to meet the computational constraints of vehicle control units (VCUs). The dataset, derived from GT-SUITE simulations, was preprocessed and split into training, validation, and test sets. Hyperparameter tuning was conducted to optimize model performance, and multiple architectures were explored, including models specifically designed to handle pressure and flow targets separately. To facilitate deployment in embedded systems, the final selected model was converted into automotive hardware-optimized C code for integration and testing in a simulated software-in-the-loop environment.

Among the models evaluated, ANN-based approaches showed strong performance in predicting the KPIs of the TMS. The multilayer perceptron (MLP) model, in particular, offered the best trade-off between accuracy, simplicity, and integration feasibility, making it suitable for embedded implementation.

The results demonstrate that ML models can effectively function as virtual sensors in TMS, offering a viable alternative where physical sensors are not feasible. ML-based sensing shows strong potential for VCU implementation, with the selected MLP model exhibiting characteristics favourable for successful integration into a BEV. The thesis also highlights ethical considerations, emphasizing the importance of model transparency and accountability in safety-critical automotive systems.

Keywords: Battery electric vehicle, Thermal management system, Artificial intelligence, Machine learning, Virtual sensor, Coolant flow, Coolant pressure.

Acknowledgements

First and foremost, we would like to wholeheartedly thank our mentor, supervisor, and friend at Volvo Cars, Babak Heydarnezhad. Thank you for your warm welcome, kindness, and encouragement, which motivated us to take this thesis to new heights. Without your valuable insights, guidance, and vision, this work would not have been possible. We are deeply grateful for the opportunity you provided and for making this thesis a genuinely enjoyable and rewarding experience.

We would also like to extend our sincere thanks to the entire Thermal Management CAE team at Volvo Cars. Your feedback and comments significantly contributed to the improvement of our results and deepened our understanding of thermal management systems.

Special thanks to Anton Blaho Mildton, and Prajwal Prashant Shetye from the Thermal Management Software team for your assistance during the conversion of the model to automotive-optimized C code. Your support was instrumental in making that process smooth and manageable.

Furthermore, we would like to thank our academic supervisor at Chalmers, Professor Peter Damaschke, for always being available to answer our questions and for providing thoughtful and constructive feedback throughout the thesis.

Lastly, we would like to express our gratitude to our friends and families for their unwavering support, patience, and cheer throughout this journey. Your belief in us has been a constant source of motivation.

Sky Sunsaksawat, Albert Ådén, Gothenburg, 2025-06-05

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background

In recent years, driven by the global energy transition, Battery Electric Vehicles (BEVs) have experienced a steady rise in sales [1]. Despite this growth, BEVs still face significant challenges compared to traditional internal combustion engine (ICE) vehicles. Key issues include range anxiety, the relatively shorter technical lifespan of their batteries compared to the durability of ICE systems, and the critical need to prevent thermal runaway in the batteries [2]. A crucial component in addressing these challenges is the vehicle's thermal management system (TMS), which serves as a common denominator in mitigating these issues.

TMSs have become increasingly complex, incorporating circuits that can be connected or disconnected in various configurations depending on the operational mode, such as cooling, heating, or heat-pumping from different sources, as well as the operating temperatures [3]. To control such a system, accurate measurements or estimations of key performance indicators (KPIs) are crucial. The KPIs of a liquid-cooled TMS are values of pressure and flow rates of the coolant through the circuit. Physical sensors are used within a TMS to provide these, but due to space constraints and costs, their presence is limited. To address this, alternative methods are employed to provide these measurements in real-time.

Methods like 3D Computational Fluid Dynamics can calculate KPIs but are too computationally intensive to simulate an entire TMS across its many operating scenarios [4]. One approach the industry adopts is to reduce the dimensionality of simulations and develop digital twins.

Currently, Volvo Cars uses a 1D model developed in the commercial software GT-SUITE [5] to simulate the system's responses, serving as a digital twin for their TMS. However, since this model is built using commercial software, seamlessly integrating its outputs into the vehicle's onboard systems is challenging. To address this, one can map system configurations to the corresponding measured values from a digital twin, a physical test rig or real test data from a BEV. The BEV then utilizes this mapping in real time to assist controllers in taking action. However, the current mapping approach is complex and requires several components and steps. Therefore, Volvo Cars would benefit from an entirely data-driven model capable of generating this mapping in a single step. Leveraging machine learning (ML) offers an effective

solution for producing this mapping. The model should be developed using open-source software to enable seamless integration into the vehicle's systems and should be capable of being trained using data from a digital twin, a physical test rig, or real-world test data from a BEV.

1.2 Machine Learning for Thermal Management

ML has significantly evolved and grown in recent years [6], gaining widespread popularity for predicting and optimizing physical systems [7]. A major driver of this growth is the increasing adoption of artificial neural networks (ANN) [8]. ANN, a branch of ML inspired by the human brain, functions by mapping inputs to outputs. Through iterative adjustments, the network learns to perform specific tasks [9]. This method is known for its ability to capture complex and non-linear relationships.

Miari and Ali (2023) [10] reviewed ML algorithms for battery temperature prediction and thermal management. Their research suggests that no single machine-learning technique is universally optimal for predicting battery temperature and managing thermal conditions. Various factors, including dataset quality, feature dimensionality, training algorithms, and other parameters, significantly impact model performance. Furthermore, deep learning techniques outperform traditional ML algorithms, such as linear regression and support vector regression.

1.3 Aim and Purpose

This project aims to explore the use of ML as a method for modelling a BEV's TMS by developing a computationally efficient model capable of predicting system responses for Volvo Cars' TMS. This model will be designed to accurately predict the system's KPIs, including the pressure and flow of the coolant, under steady-state conditions. The purpose of the model is to serve as a virtual sensor in areas where physical sensors are absent within the BEV's TMS.

To ensure the project's objectives are successfully met, the model will be evaluated using several metrics, including accuracy, inference time, and storage size. The specific goals for each metric cannot be disclosed due to confidentiality. However, it can be stated that the accuracy goal is hybrid, depending on particular cases. There is a strict target for inference time, and storage space should be minimized while still meeting the other objectives.

1.4 Limitations

The model will be based solely on steady-state simulations of the TMS, where steady state refers to a condition where a process or variable remains constant over time [11]. Additionally, the model will only predict the pressure and flow of the coolant.

Since TMS simulations are computationally intensive, the available data is limited, and the project's timespan will constrain the generation of new simulation data.

However, the existing data has been deemed sufficient for the scope and objectives of the project.

Another limitation is the available space in the Vehicle Control Unit (VCU). The ultimate goal of the project is to implement the model within the vehicle, specifically in the VCU, where storage and computational resources are highly constrained.

1.5 Ethical aspects

An essential ethical consideration when utilizing ML is ensuring the transparency and explainability of a model's outputs. For this project, it is crucial to understand and explain why a model predicts a specific value for a KPI.

In the automotive industry, transparency in the TMS is crucial, as the TMS is closely tied to the vehicle's safety. Many flow and pressure requirements for components are established based on safety concerns. However, some of the ML models explored in this project are based on ANNs, which function as black boxes and are notoriously difficult to interpret. Despite their high accuracy, determining the explanation for a specific prediction can often be complex.

Another concern is accountability in systems utilizing ML. As ML-driven systems become more common, it becomes increasingly complex to determine responsibility in the event of system failures. For instance, if an ML-based thermal management model incorrectly predicts the battery's cooling requirements, increasing the risk of thermal runaway, it is unclear who should be held accountable.

While ML has the potential to offer advantages in TMS's software, it is crucial to be aware of the ethical concerns associated with it. Building trust in ML-driven decisions among engineers and end-users is vital for the safe and responsible deployment of these technologies.

2

Theory: Thermal Management

The performance, safety, and efficiency of BEVs depend on effective thermal management, as their more complex TMS require active control to maintain optimal operating temperatures. The chapter begins with an overview of GT-SUITE, the simulation tool used in this project to generate data. It continues with a description of key TMS components and their interactions. A simplified, generic TMS schematic is also presented to provide context for the system behavior that the ML model later aims to predict.

2.1 GT-SUITE

GT-SUITE is a multi-physics simulation software developed by GAMMA Technologies, enabling system-level simulations. It is widely used in the automotive industry for modelling and analyzing TMS. It allows the design and evaluation of complex TMSs by providing advanced tools for simulating coolant flow and pressure, heat transfer, and energy distribution throughout a vehicle. With its ability to model interactions between various thermal subsystems, GT-SUITE is particularly valuable for investigating how thermal performance in BEVs can be optimized [12].

In addition to simulation, GT-SUITE can be used to create a digital twin of a physical system. A digital twin is a virtual representation of a real-world system that mirrors its behaviour in real time or under simulated conditions [13]. This enables engineers to evaluate system performance, test design changes, and diagnose issues without requiring direct physical interaction.

2.2 Thermal Management System

Exploring the application of ML models for modelling a vehicle's TMS requires a comprehensive understanding of the dynamics of the TMS. This is essential for accurately interpreting model outputs and verifying their reliability. A TMS in a vehicle is responsible for regulating the temperature of critical components [14]. Several types of TMS exist, including liquid-cooled, air-cooled, and phase-change material-based systems. In the remainder of this report, only a liquid-cooled TMS will be considered. In its simplest form, it operates by circulating a fluid referred to as coolant through hoses and heat exchangers to cool and heat the system's components. In basic setups, a single pump may control the flow and pressure

of the coolant. TMS designs for BEVs are more advanced. They can regulate temperatures by adjusting configurations to achieve different operating modes, and they also include heating capabilities. The following section provides an overview of a generic liquid-cooled TMS and its most essential components. However, the specific TMS used in this project and by Volvo Cars cannot be disclosed due to confidentiality constraints.

2.2.1 Importance of Thermal Management in BEVs

There is a significant difference between TMSs in BEVs and ICE vehicles. Since BEV drivetrains are far more energy-efficient, they generate less waste heat, necessitating active heating to maintain optimal performance. While this might seem like a minor inconvenience, it has a significant impact on the range of the BEV. One of the reasons for this impact is the energy required to heat and cool the cabin compartment. In this context, the cabin compartment refers to the area that houses the driver and passengers of the vehicle. This underscores the need for an efficient TMS, which relies on principles such as recovering waste heat from sources within the system and utilizing heat pumping technologies between compartments in the vehicle [15]. Furthermore, the components of a BEV have relatively narrow and different optimal operating temperature ranges. As shown in Figure 2.1, these ranges are depicted for several key components of a BEV, and also compared to an ICE.

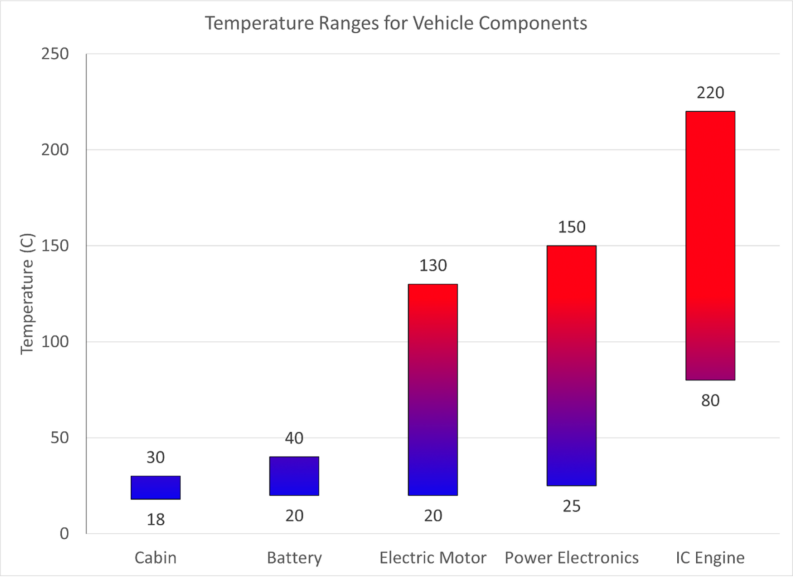


Figure 2.1: Optimal operating temperature ranges for critical components in a BEV compared to the engine of an ICE vehicle [16].

2.2.2 Thermal Loops in BEV Thermal Management

The different operating conditions explained in the section above further contribute to the intricacy of the TMS and introduce the concept of multiple thermal loops. The thermal requirements of the cabin compartment differ significantly from those of components such as the electric motor. For instance, if the cabin reached the same

temperatures as the electric motor during driving, it would be highly uncomfortable for the driver and passengers. These differing conditions result in specific cooling and heating requirements. One way to address this is by introducing additional thermal loops. A thermal loop, in this context, refers to a system of interconnected components that is regulated within a specified temperature range. Currently, several solutions exist, but it is common to divide the BEVs TMS into three distinct loops: one for the cabin compartment, one for the battery, and one for the electric drivetrain (ED) [17]. Another solution is to use just two loops, as Tesla did in their TMS for the Model Y [18]. By introducing more loops, the option to exchange heat between them becomes possible. A simple example is the heat exchange between the cabin compartment loop and the ED loop when the cabin requires heating.

2.2.3 Generic TMS of a BEV

Figure 2.2 shows a schematic representation of a simple, generic TMS for a BEV. This is not the actual system examined in this project, but rather a simplified version for showcasing purposes. Note that this example only addresses the thermal management of the battery and ED; thus, the cabin compartment is not included. Moreover, it assumes that the battery and the ED are managed within a single thermal loop, which is impractical in reality due to their differing heating and cooling requirements. This example includes components such as the battery, pump, valve, radiator, chiller, and heater, all of which are interconnected by hoses that circulate the coolant. A brief description of each component and its role in the TMS is provided below.

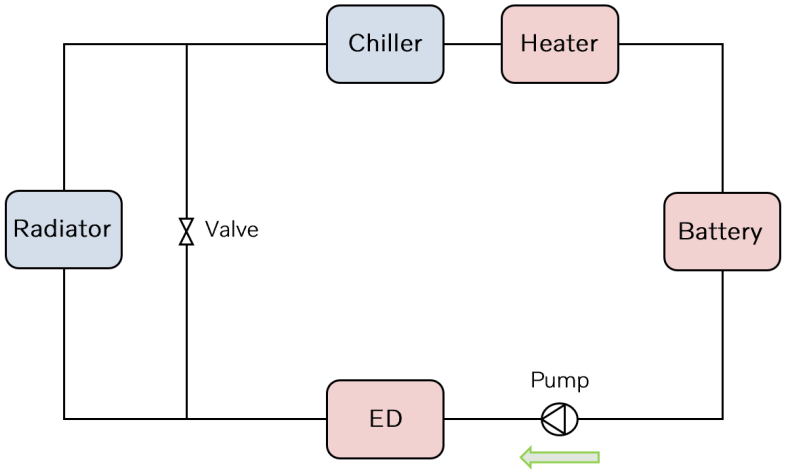


Figure 2.2: Thermal loop of a simple generic liquid TMS. Red indicates heat sources within the circuit, while blue represents heat sinks. The green arrow shows the coolant flow direction, while black lines represent the hoses connecting the various components.

Coolant
This is a specialized fluid designed for efficient heat transfer. It flows through hoses that interconnect all components in the TMS.

Pump
The pump circulates the coolant through the TMS, ensuring a continuous and controlled flow to optimize thermal performance. However, this comes at the cost of additional energy required to operate the pump. It operates at a certain pump speed ranging from 0 to 100%, which is controlled.

Valve
A valve directs coolant flow based on system demands, essentially controlling whether coolant is routed to specific components requiring temperature regulation. In the example shown, the valve can be used to bypass the radiator when necessary. Valves can be either adjustable, allowing for partial opening and closing, or fixed to operate in a fully open or fully closed state, which can be controlled.

Battery
The battery refers to the vehicle's battery pack, which consists of multiple battery modules, each containing individual battery cells. These cells convert chemical energy from active materials into electrical energy through redox reactions [19]. The battery pack also incorporates a heat exchange mechanism that facilitates thermal regulation between the battery cells and the coolant [14]. While various methods exist for achieving this heat exchange, the specific approach used is of less relevance to this project. The key takeaway from the battery block is that coolant flows through the battery pack, allowing for the temperature regulation of the battery cells. Since the battery generates heat during both charging and discharging, it functions as a heat source within the circuit.

Electric Drivetrain
Comprises the key components responsible for the vehicle's propulsion [20]. While the battery pack can be considered part of the ED, in this example, it is treated as a separate block. Similar to the battery block, the ED block generates heat due to losses within its respective components and is therefore treated as a heat source in the circuit.

Radiator
A radiator's primary function is to dissipate heat from the coolant into the surrounding air, acting as a passive heat sink.

Chiller
The Chiller, just like the radiator, dissipates heat from the system. However, unlike the radiator, the chiller is actively controlled, allowing it to be turned on and off at

various power levels. It functions as an active heat sink.

Heater
The heater works similarly to the chiller but instead provides heat to the loop, acting as an active heat source.

This example features just one loop, one pump, and one valve; however, a real-world TMS for a BEV typically incorporates multiple loops, pumps, and valves. To give the reader an idea of the number of configurations such a system can take, consider this simple example: assuming the pump speed can be adjusted in 25% increments and the valve is either fully open or fully closed (disregarding partially open states), the system would have eight possible operating configurations. To include even more configurations, the temperature of the coolant can also be varied, which is the case in reality.

3

Theory: Machine Learning

ML provides powerful tools for modelling complex, nonlinear systems where traditional methods may fall short. In the context of this project, ML is explored as a means to model the system responses of a TMS in a BEV, serving as a virtual sensor in the absence of physical sensors. This chapter introduces the theoretical background necessary to understand and apply ML in this context. It begins with core concepts such as the bias-variance tradeoff and proceeds to present both traditional ML algorithms and ANNs. Emphasis is placed on the aspects most relevant to the project, specifically those affecting the goals of accuracy, storage space, and inference time.

3.1 Bias-Variance Tradeoff

In ML, an important phenomenon to consider is the bias-variance tradeoff. When training an ML model, it learns from one dataset and should then generalize effectively to unseen data. Constructing a model that achieves this requires careful consideration of the bias-variance tradeoff. In this context, bias should not be confused with the bias term in a neuron of an ANN, which will be explained later. Here, bias refers to a model's inability to accurately capture the underlying relationship between its inputs and outputs. A model with high bias fails to capture this relationship effectively and is commonly referred to as an underfit model. An example of this would be applying a linear model to try to predict something nonlinear.

On the other hand, variance refers to a model's inability to generalize to different datasets. A model that performs well on the training data but significantly worse on unseen data has high variance; this is known as an overfit model. For example, this can occur when a complex, nonlinear model is applied to a problem that is inherently linear. In such cases, the model may attempt to learn patterns that it assumes represent the underlying relationship; however, these patterns can be noise in the data. A well-fitting model is one with low bias and low variance, but since these are often opposing factors, achieving the right balance is crucial [21].

3.2 Traditional Machine Learning

Traditional ML algorithms, such as linear regression, decision trees, and Support Vector Regression, provide powerful tools for capturing relationships while offering

interpretable insights. This section provides an overview of the traditional ML algorithms explored in this project.

3.2.1 Linear Regression

A linear regression model is the most common statistical concept for relating a set of two or more variables [22]. The objective of linear regression is to find a multiple linear model that involves a linear combination of the input variables, as shown in the equation 3.1.

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D \tag{3.1}$$

where $\mathbf{x} = (x_1, ..., x_D)^T$ is the vector of the input variables. $w_0, ..., w_D$ are parameters of the model, which determine the properties of the fitted line. The parameter w_0 accounts for any fixed offset in the data and is often referred to as the *bias* parameter.

3.2.2 Polynomial Regression

Polynomial regression (PR) is an extension of linear regression that models the relationship between independent variables and the target variable using an n-th degree polynomial [23]. While linear regression fits a straight line to the data, PR introduces higher-order terms to capture non-linear relationships. This can be seen in equation 3.2.

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_D x^D \tag{3.2}$$

3.2.3 k-Nearest Neighbour Regression

This algorithm stores all available data points and uses a similarity measure to predict values. It makes predictions by identifying the k nearest data points to a given input and averaging their target values. The prediction is based on feature similarity, which reflects how closely the test data resembles other training examples. There are two approaches for k-nearest neighbour (kNN) regression. The first is by computing the average of the target of the k-nearest neighbours. The second approach involves calculating an inverse distance-weighted average [24].

3.2.4 Support Vector Regression

Support vector regression (SVR) is an algorithm for regression analysis. Unlike other regression methods, it aims to find a continuous-valued function that models the relationship between the input variables and a continuous target variable while minimizing the prediction error, as long as the predictions fall within a certain margin of tolerance, called epsilon (ϵ). Imagine, the value of ε determines the width of the tube around the predicted line, SVR tries to find the narrowest tube while

keeping most of the data points inside it. The continuous-valued function can be written as in equation 3.3.

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} + b \tag{3.3}$$

To keep the model flat and straightforward, while handling cases where data points fall outside the epsilon margin, slack variables ξ and x_i^* are introduced to account for these latter cases. These variables enable the model to tolerate a few outliers that fall outside the margin. The optimization problem can be obtained in equation 3.4.

$$\min \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \tag{3.4}$$

subject to

$$\begin{aligned} y_i - \mathbf{w}^T\mathbf{x}_i - b &\leq \varepsilon + \xi_i^* & i &= 1, \ldots, N \\ \mathbf{w}^T\mathbf{x}_i + b - y_i &\leq \varepsilon + \xi_i & i &= 1, \ldots, N \\ \xi_i, \xi_i^* &\geq 0 & i &= 1, \ldots, N \end{aligned}$$

where C is a parameter that balances the trade-off between the model's flatness (minimizing $||\mathbf{w}||^2$) and the tolerance for the points outside the ϵ-radius tube. N denotes the number of data points [25]. This constrained quadratic optimization problem can be solved by using the Lagrangian method.

3.2.5 Decision Tree Regressor

A decision tree regressor is a non-parametric algorithm used for predicting continuous values. It features a hierarchical tree structure consisting of a root node, branches, internal nodes, and leaf nodes. The algorithm begins by initializing a root node that represents the entire dataset. The tree is then built by splitting the data into smaller groups, choosing the best split one step at a time using a greedy approach [26]. This splitting continues recursively until the stopping criteria, such as maximum depth or minimum samples per leaf, are met. Predictions are calculated by averaging the target values in the leaf nodes.

3.2.6 Random Forest Regressor

Random Forest (RF) is an algorithm utilizing an ensemble of decision trees to make predictions [27]. It works by creating a forest of decision trees from random subsets of the training data using the bootstrapping technique, also known as bagging, shown in Figure 3.1. Each tree makes its prediction, then the model aggregates these outputs to generate a final result. With this approach, Random forests are robust against overfitting and have improved prediction accuracy compared to individual decision trees.

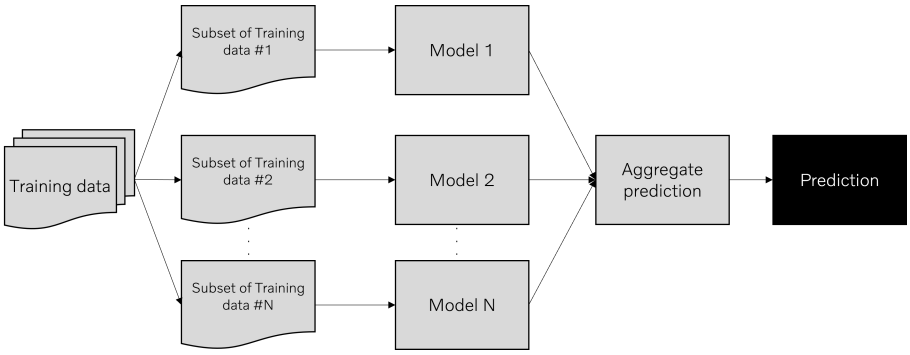


Figure 3.1: Illustration of Random Forest

3.2.7 Gradient Boosting Regressor

Gradient Boosting is an ensemble ML method that combines multiple decision trees. Unlike RFs, Gradient Boosting trains trees sequentially, with each new tree correcting the errors of the previous ones, known as the boosting technique, which is shown in Figure 3.2. The final prediction is the sum of outputs from all trees, scaled by the learning rate and the initial prediction [28].

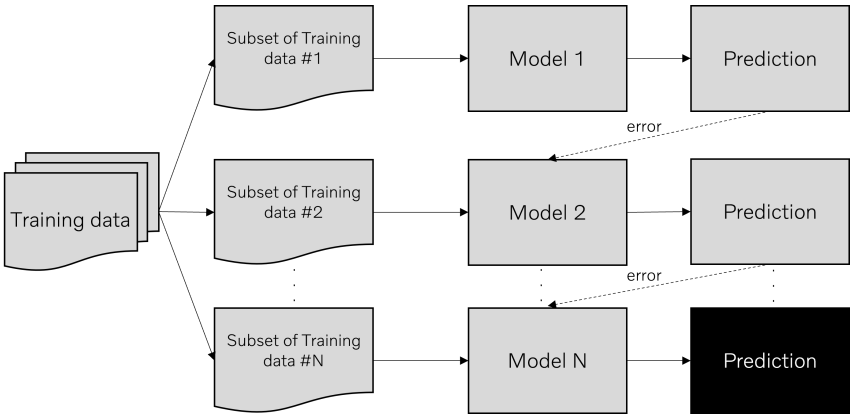


Figure 3.2: Illustration of Gradient Boosting

3.3 Artificial Neural Network

ANNs offer a powerful approach to modeling complex, nonlinear relationships. Unlike traditional regression models, an ANN can capture more complex dependencies, making it suitable for predictions in dynamic environments [29]. A common type of ANN is the feedforward neural network, also known as a Multi-layer perceptron (MLP). MLP networks are widely used in various applications, leveraging a technique called backpropagation for training to optimize model performance [30]. Its key components consist of an input layer, n hidden layers with varying numbers of interconnected neurons, an output layer, and activation functions. In Figure 3.3, the architecture of a general ANN can be seen.

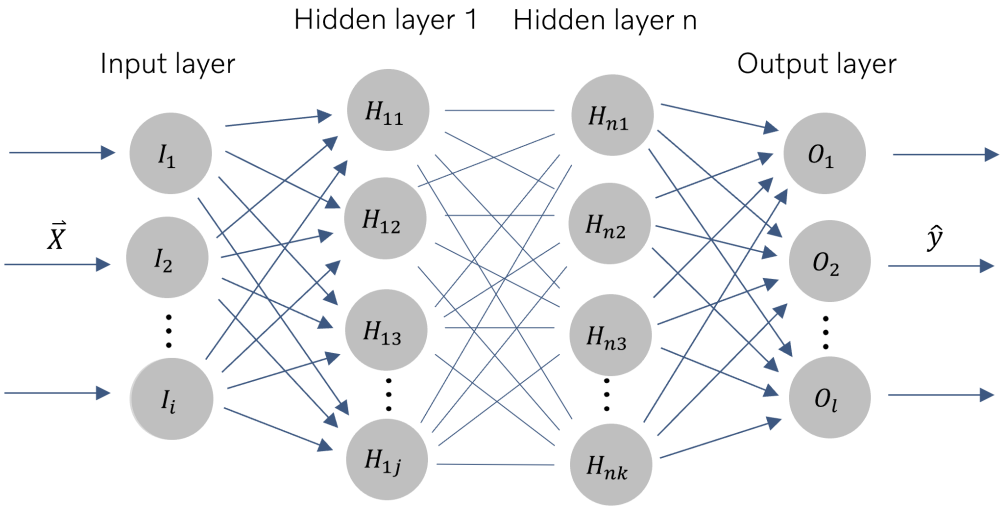


Figure 3.3: Architecture of an ANN. The input nodes are labelled as I_1 to I_i. Neurons in the hidden layers are represented by H_{xy}, where x indicates the hidden layer index and y denotes the neurons position within that layer. The output nodes are labelled as O_1 to O_l.

The input layer serves as the entry point for data into the network. The number and type of inputs depend on the specific problem at hand. Types of inputs can vary, ranging from continuous numerical data to categorical variables, as well as images, text, or audio signals. Each input node corresponds to a distinct attribute relevant to the given problem, often referred to as a feature. Following the input layer are the network's hidden layers, which contain the components commonly referred to as neurons. The purpose of hidden layers is to introduce non-linearity to the model. Suppose the architecture shown in Figure 3.3 has three input nodes and two hidden layers with three neurons each; in that case, a representation of the neuron labelled H11 can be seen in Figure 3.4.

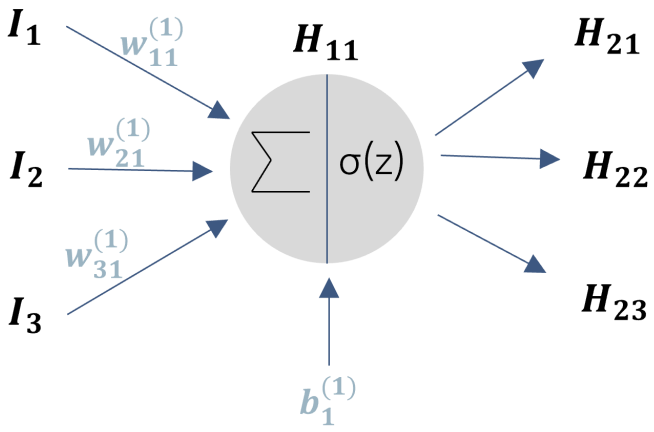


Figure 3.4: Illustration of neuron H_{11}, with reference to the weight notation $w_{ij}^{(l)}$. The subscript ij indicates a connection from neuron i in the previous layer to neuron j in the current layer, while the superscript (l) denotes the layer index. For example, $w_{11}^{(1)}$ represents the weight from input neuron 1 to hidden neuron 1 in layer 1. The same applies to the bias term $b_1^{(1)}$, where the subscript 1 refers to hidden neuron 1 and the superscript (1) indicates that it belongs to the first hidden layer.

A neuron is the fundamental building block of an ANN. It receives one or more inputs, computes their weighted sum, and applies an activation function to transform this sum and produce an output. This output is then passed on to subsequent neurons in the next layer or to an output node.

The weighted sum is calculated by assigning a unique weight to each input, which determines the input's influence on the neuron's output. Additionally, a neuron includes a bias term, which allows the output to be shifted independently of the input values. This bias ensures the model can better fit data, especially when the input values are zero or close to zero. The weighted sum of neuron H11 can be seen in equation 3.5.

$$z = w_{11}^{(1)} I_1 + w_{21}^{(1)} I_2 + w_{31}^{(1)} I_3 + b_1^{(1)} \tag{3.5}$$

The weights assigned to each input, along with the bias of a given neuron, are the parameters that an ANN learns during training. The training process is visualized in Figure 3.5.

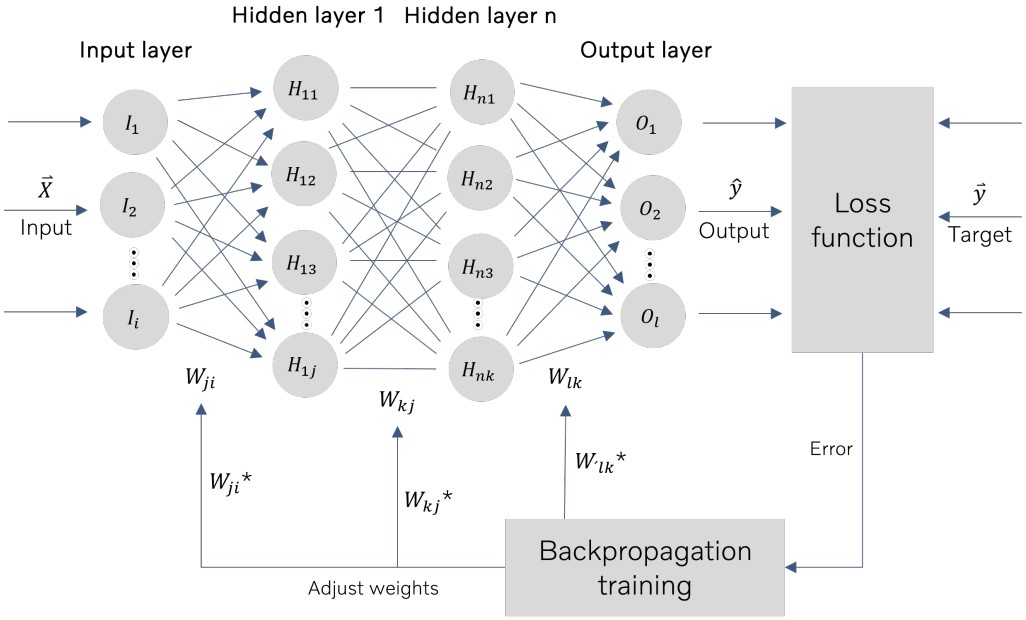


Figure 3.5: Illustration of the learning process with backpropagation to update weights and biases

The process involves adjusting the parameters for all neurons in the network to minimize the error between the network's predictions and the actual target values. This error is often referred to as the loss and is computed using a loss function. This loss is then used in backpropagation to update weights and biases [31]. The role of an activation function and the loss function will be described in the upcoming sections.

3.4 Hyperparameters of ANNs

With the brief description above of how an ANN learns, one might get the impression that it is a universal and simple solution capable of solving all problems. However, this is not the case. Referring back to Figure 3.3, some questions arise: How does one determine the appropriate number of neurons? The number of hidden layers? ANNs are versatile and capable of modelling complex dependencies, but they come with a notable drawback, they require careful tuning of several parameters that significantly impact the model's performance. These parameters are known as hyperparameters. The following subsections will introduce the most relevant hyperparameters for this project. And finally, explain how the most optimal ones can be determined.

3.4.1 Number of Hidden Layers

The number of hidden layers is a crucial hyperparameter of an ANN. As described in the previous section, the purpose of a hidden layer is to introduce non-linearity. In practice, it is the activation function present in the hidden layer that achieves this, but more hidden layers result in more activation functions. The introduction of non-

linearity allows a model to learn complex dependencies and thus solve complex tasks. This means that an increase in the number of hidden layers leads to an increase in the ability to learn complex patterns and increases the accuracy [32]. However, increasing the number of hidden layers increases the total size of the network. This affects both the storage space and the inference time.

3.4.2 Hidden Size

The hidden size refers to the number of neurons within a hidden layer. It is important to note that each hidden layer can have a unique hidden size. This parameter has a similar effect on the network as the number of hidden layers. Increasing the hidden size enhances the network's ability to extract more information from the inputs. This, in turn, can improve accuracy. However, similar to increasing the number of hidden layers, this also increases the network's size.

3.4.3 Activation Functions

Activation functions are mathematical operations applied to the weighted sum of a neuron or an output neuron. They are mostly used in hidden layers to introduce non-linearity, enabling the learning of complex dependencies. Without the activation function, an ANN functions as a linear model, limiting its capabilities. Different activation functions transform the output in various ways. The choice of activation function significantly affects network accuracy, convergence, and generalization to unseen data, making it a critical aspect of designing an ANN [33]. This section will give an overview of the activation functions investigated in this project.

3.4.3.1 Rectified Linear Unit

The Rectified Linear Unit (ReLU) is often considered the state-of-the-art activation function in deep neural networks due to its simplicity and computational efficiency. The function is defined as equation 3.6. It transforms all negative values to zero, while leaving all positive values unchanged.

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.6}$$

The main drawback of ReLU is the dying neuron problem. This occurs when a neuron's weighted sum of inputs repeatedly results in a negative value, causing the neuron to output zero before the ReLU activation function is applied. Consequently, the neuron becomes inactive and stops contributing to the learning process, preventing the model from improving through that neuron [33].

3.4.3.2 Leaky ReLu

Leaky ReLU is a variation of ReLU designed to mitigate the dying neuron problem. The function can be seen in equation 3.7. Instead of mapping all negative values to zero, it introduces a parameter α, typically a small positive value, that defines a

slight negative slope for inputs below zero. This ensures that neurons with negative inputs can still contribute to the learning process.

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{3.7}$$

This variation of ReLU retains the computational efficiency advantage while also addressing the dying neuron problem. However, this comes at the cost of introducing an additional hyperparameter that requires tuning [33].

3.4.3.3 Sigmoid

The sigmoid function is another popular activation function, known for limiting its outputs between 0 and 1. This is particularly beneficial in some instances. Another key characteristic of the sigmoid function is that it is differentiable across its entire range, which is helpful in learning using gradient-based optimization methods. The function is shown below in equation 3.8.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.8}$$

While the sigmoid function offers certain advantages over other activation functions, it also has notable disadvantages. Due to its exponential operation, sigmoid is less computationally efficient compared to simpler activation functions. Additionally, the sigmoid suffers from the vanishing gradient problem. When the input is a large positive or a large negative value, the functions output curve flattens, producing a gradient close to zero. This diminishes the effectiveness of gradient updates and hinders the learning process [34].

3.4.3.4 The Sigmoid Linear Unit

The Sigmoid Linear Unit (SiLU), also known as the swish function, is a smooth, non-monotonic activation function that has shown promising performance in ANN tasks. Unlike ReLU, which is piecewise linear, SiLU provides a smooth transition and avoids hard zero cutoffs. The SiLU activation combines the input with its sigmoid transformation, as shown in equation 3.9.

$$\text{SiLU}(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1 + e^{-x}} \tag{3.9}$$

One of the main advantages of SiLU is its ability to retain small negative values, which can improve the flow of information during training compared to ReLU, which discards all negative values. SiLU is also differentiable everywhere and avoids abrupt changes, which can benefit gradient-based optimization. However, like the sigmoid function, SiLU is computationally more expensive than simpler functions due to the exponential operation [34].

3.4.3.5 Hyperbolic Tangent

Moving on, an activation function with similar characteristics to the sigmoid function is the Hyperbolic Tangent (tanh), described in Equation 3.10. The output range of tanh is limited between -1 and 1, which is known as being zero-centred, one of its key benefits. This property is advantageous because when the output is centred around zero, it does not bias the network toward positive or negative gradients. This balance yields faster convergence, stabilizing the learning process and improving training efficiency [33].

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.10}$$

Just like the sigmoid function, tanh suffers from the vanishing gradient problem and reduced computational efficiency.

3.4.3.6 Linear

A linear activation function maps the input to a linear combination of the input and a constant a, which is most often set to 1. Equation 3.11 showcases the function.

$$f(x) = ax \tag{3.11}$$

While a network consisting solely of linear activation functions can only learn linear relationships, linear activation functions still have their place in an ANN for regression tasks. In a regression task, the goal is to predict a continuous value without restricting the output range, and a linear activation function in the output layer effectively serves this purpose [35].

3.4.4 Loss Functions

A loss function measures how well a model's predictions align with the truth during training. There are several types of loss functions with different areas of application. The choice of loss function depends on the characteristics of the data and the particular task at hand. It compares the predicted values with the actual values and returns a value, known as the loss, that reflects the model's performance. The objective during training is to minimize this loss. The model alters its parameters based on the loss from the previous iteration, emphasizing the importance of choosing a suitable loss function. Monitoring the loss throughout the training process provides insight into how effectively the model is learning the specified task [36]. Below are some commonly used loss functions for regression problems presented.

- **Mean Absolute Error**
  Mean Absolute Error (MAE) is a tool to measure the average absolute difference between the target value and the predicted value. MAE treats all errors

equally, which makes it less sensitive to outliers.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.12}$$

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

- **Mean Squared Error**
  Mean Squared Error (MSE) measures the average of the squared discrepancies between the target value and the predicted value. Unlike MAE, MSE does not treat all errors equally. By squaring the errors, MSE places greater emphasis on larger errors, making it more sensitive to outliers.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.13}$$

- **Huber Loss**
  Huber Loss is a combination of MAE and MSE that is less sensitive to outliers while still maintaining good performance for minor errors. It integrates a threshold parameter, δ, that determines the point at which the loss transitions from quadratic to linear. It is important to note that this introduces an additional hyperparameter to tune.

$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{for } |y_i - \hat{y}_i| \leq \delta \\ \delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta) & \text{for } |y_i - \hat{y}_i| > \delta \end{cases} \tag{3.14}$$

  where δ is the threshold.

- **Robust Loss**
  The robust loss function is defined as a single function, unlike the Huber loss which combines two functions based on a threshold. It provides a smoother transition without conditions. Its partial derivative with respect to the residual is inspired by the sigmoid function [37]. The final form of the Robust loss function is defined as

$$L(y, \hat{y}) = (y_i - \hat{y}_i) + \frac{2}{w}\log(1 + e^{(-w*(y_i - \hat{y}_i))}) - \frac{2}{w}\log 2 \tag{3.15}$$

  where w is a weighting factor. The $\frac{2}{w}\log 2$ term is added to ensure that the loss function evaluates to zero when y is equal to $\hat{y}$

3.4.5 Performance Metrics

Performance metrics serve a similar purpose to loss functions but are used after training is complete. They are metrics, often mathematical functions, that measure how well the model performs on unseen data. Below are some standard performance metrics that can be applied to regression problems. Metrics such as MAE and MSE, which were previously introduced, can also be used as performance metrics.

- **Root Mean Squared Error** (RMSE)
  Root Mean Squared Error (RMSE) measures the average difference between the target value and the predicted value, but it gives more weight to larger errors due to the squaring of differences. This makes RMSE more sensitive to outliers, and a lower RMSE indicates a better model fit.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.16}$$

- **R-Squared** (R^2)
  Rš, also called the coefficient of determination, measures how well the model explains the variation in the target variable. It indicates the proportion of the variance in the target that is captured by the model, with values closer to 1 showing a better fit. However, a higher Rš doesnt always mean the model is perfect or the best predictor. In short, Rš indicates how much of the data's variation is explained by the model compared to guessing the average value.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3.17}$$

  where $\bar{y}$ represents the mean of the target values.

While mathematical performance metrics are essential for evaluating model performance, they do not always provide a complete picture. Their interpretation can be challenging, as the significance of their findings is highly dependent on the type and scale of the predicted data. For instance, an MAE of 1 may be considered either acceptable or unacceptable depending on the context. Consider a case where a model predicts pressure in [KPa] and flow in [L/s], and only a single prediction is being evaluated. An MAE of 1 for pressure could be seen as a small and acceptable error, whereas an MAE of 1 for flow may represent a much more significant deviation. To compensate for this, one can utilise non-mathematical performance metrics that are directly related and applicable to the problem at hand.

3.4.6 Additional Hyperparameters

Apart from the hyperparameters described above, several others were examined in this project. However, since these primarily affect the learning process rather than the network architecture, they are not discussed in detail.

3.4.7 Finding Optimal Hyperparameters

With the most critical hyperparameters explained, the next step is to understand how to find the optimal combinations. As previously discussed, hyperparameters influence model performance in different ways. As an example, selecting a specific

activation function may improve accuracy but could negatively impact computational efficiency. Since many hyperparameter choices involve trade-offs, identifying the most optimal combination is a challenging task.

There are several approaches to hyperparameter tuning, and one of the most basic is simple trial and error. A more structured version of this method is known as grid search. In this approach, a predefined set of possible values is specified for each hyperparameter, and the model is trained and evaluated for every combination of these values. Although computationally intensive, grid search offers a systematic way to explore the hyperparameter space and is particularly effective when the search space is relatively small or constrained. The objective is to identify the combination of parameters that yields the best performance on a validation dataset [38]. The purpose of a validation dataset will be explained in the upcoming section.

It is essential to note, however, that the best-performing hyperparameter set in a grid search is determined solely by prediction accuracy on a specific dataset. As a result, a configuration that performs only slightly worse in terms of accuracy may be disregarded, even if it could be more optimal in a specific deployment scenario, such as one with strict storage or computational efficiency constraints.

3.5 Data Splitting

In ML, it is standard practice to divide the available dataset into separate subsets to ensure that the model is appropriately trained and evaluated. This process typically involves splitting the data into a training set and a test set, enabling the model to be evaluated on data it has not encountered during training. For ANNs, an additional split is often performed to create a validation set.

As a result, a three-way split is commonly employed when training ANNs, consisting of a training set, a validation set, and a test set. The training set is used to fit the model, the validation set is used to tune hyperparameters, evaluate model performance during training, and alter weights and biases, and the test set offers an objective measure of the models performance on unseen data [39].

The proportions used for each subset can vary depending on the size of the dataset. However, a commonly used percentage split is 80/20 when no validation set is required, and 80/10/10 when a validation set is included. Ultimately, the choice of split depends on the total amount of data available, with larger datasets, a smaller percentage of data may be sufficient for validation and testing without compromising reliability.

3.6 Feature Scaling

This is a technique used to standardize a range of variables, often by transforming them to have a mean of 0 and a standard deviation of 1 [40]. This process, also known as standardization, is an essential step in data preprocessing for ML algorithms [41]. Feature scaling, as the name suggests, is often applied to the features, in other

words, the inputs to an ML model. This ensures that all inputs have a similar range, thereby impacting the model with a similar magnitude. The principle of feature scaling can also be applied to the model's outputs. Similarly, the range of inputs can vary, as can the outputs, making this approach applicable to both sets of parameters. Standard feature scaling can be computed using equation 3.18.

$$x_{scale} = \frac{x - \mu}{\sigma} \tag{3.18}$$

where μ is the mean of the training samples and σ is the standard deviation of the training samples [42].

3.7 Correlation Matrix

A correlation matrix is a table that represents the correlation coefficients between variables. The indices of each cell illustrate the correlation between two variables in the table.

3.7.1 The Pearson Correlation Coefficient

The Pearson correlation coefficient (PCC) is commonly used to measure the linear relationship between two datasets. It is calculated by dividing the covariance of the two variables by the product of their standard deviations [43]. Let r_{xy} be the Pearson's correlation coefficient, $(x_1, y_1), ..., (x_n, y_n)$ is the given paired data including of n pairs. Such that r_{xy} is defined as equation 3.19

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.19}$$

where n is a sample size, x_i, y_i are the individual data points with index i-th, and $\bar{x}$ and $\bar{y}$ are the mean of x and y variable respectively [44]. As a result, the PCC value ranges between -1 and 1, where -1 indicates a perfect negative linear correlation, 1 indicates a perfect positive linear correlation, and 0 means no linear correlation. Figure A.1 in the Appendix illustrates the Pearson correlation for various data distributions, including both linear and nonlinear relationships.

3.7.2 Heatmap

A heatmap is a two-dimensional data visualization technique in which color intensity typically represents the magnitude of data points [45]. There are two main types of heatmaps: spatial and grid-based. The former is usually overlaid on a map, while the latter displays magnitudes as colors in a two-dimensional matrix. In data science, data scientists and analysts use grid heatmaps to visualize relationships between data points when analyzing datasets.

3.8 One Hot Encoding

One-hot encoding is a method that converts categorical data into numerical data. It creates new columns for each category, where 1 indicates the presence of that category, and 0 indicates its absence. In regression analysis, this is known as a dummy variable [46].

The benefit of one-hot encoding is that an ML model can process the category information of the input, thereby improving model performance. In addition, it could help eliminate the ordinality problem, which arises when categorical data has an inherent order that could mislead the model if represented numerically. However, it increases the dataset's dimensionality as new columns are generated, leading to higher computational costs and increased model complexity [47].

4

Methodology

The methodology involved conducting a examination of GT-SUITE and the digital twin on which the ML model was to be based. Using the insights gained, a literature review was conducted to explore relevant ML algorithms. This was followed by data exploration and preprocessing, which provided the foundation for developing ML models. Then, the models' performance was validated and evaluated to assess their effectiveness and compare them. As a final step, a method for converting the chosen model into automotive hardware-optimized C code was developed.

4.1 GT-SUITE Analysis

The initial step was to study GT-SUITE in general, with a particular focus on the digital twin that would serve as the basis for the model. The purpose of this step was to gain an understanding of the software's capabilities and the methods for extracting relevant data from the simulation. Simulations were conducted to understand the system's physics, limitations, and the structure of the generated data. Emphasis was also placed on understanding the critical configurations of the digital twin to identify these configurations when evaluating the proposed ML model. While a digital twin can model systems with relative accuracy, it still has its limitations.

4.1.1 GT-SUITE Digital Twin

Due to confidentiality, the actual digital twin of Volvo Cars cannot be disclosed. However, the same generic TMS presented in the Theory chapter is used as an illustration of how the digital twin might look and is shown in Figure 4.1. Also included in the schematic are measuring points for the pressure and flow of two specific components: the pressure before and after the battery pack and the coolant flow through the radiator.

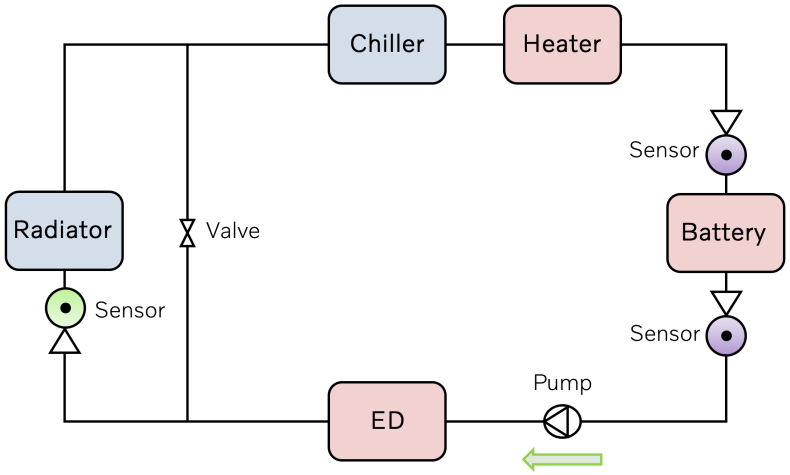


Figure 4.1: Generic TMS. The purple sensors indicate pressure measurement, and the green sensor measures flow.

The simulations conducted in this project are considered adiabatic, meaning no heat is transferred into or out of the system, which results in all components and the coolant maintaining a constant temperature. If this was the TMS, and a simulation was to be conducted on this system, a single simulation would consist of inputs corresponding to the system's configuration and outputs being the measuring points. As briefly mentioned in the Theory chapter, a configuration of this system would include a specific valve setting, pump speed, and a coolant temperature. This results in three inputs and three outputs for this particular simulation. Once the inputs are set and the measuring points are determined, the simulation is run until the measured parameters are considered to have converged, indicating that the system has reached a steady state.

With a clear understanding of the system, its dynamics, and the outputs, the next step involved translating this into an ML problem. The models outputs were defined as the selected KPIs at a specific point or component within the TMS based on a given system configuration. The actual TMS consists of 6 inputs with varying data types and 21 outputs of different characters. The outputs are multiple continuous numerical values, and in ML terms, this is identified as a multi-output regression problem.

4.1.2 Literature Review

A literature review was conducted to assess relevant ML methods. This review provided a foundation for selecting appropriate algorithms and techniques. Emphasis was put on methods capable of multi-output regression. Both traditional regressive ML algorithms and ANNs were researched to identify the most suitable models for this project. The findings of the literature review were presented in the chapter on ML theory.

4.2 Exploration of Data

As previously mentioned, the TMS consists of 6 inputs and 21 outputs. The simulations conducted on the digital twin thus yielded a dataset composed of 17,600 rows and 27 columns, containing both numerical and categorical values. The columns represent the system's inputs, along with the resulting pressure and coolant flow at specific points throughout the circuit. The rows correspond to unique input configurations and their associated outputs, meaning that an ML model would have 17,600 different scenarios available for training and testing. In ML, inputs and outputs are commonly referred to as features and targets. From here on, the inputs will be referred to as features, and the outputs as targets.

4.2.1 Data Distribution

While generating the dataset from simulations, the feature variables were ensured to be uniformly distributed. This is important because uniform distribution helps avoid bias, ensures consistent model performance across the entire range of input values, and improves the model's ability to generalize to new, unseen data. Seen in Figure 4.2 is the distribution of the target variables.

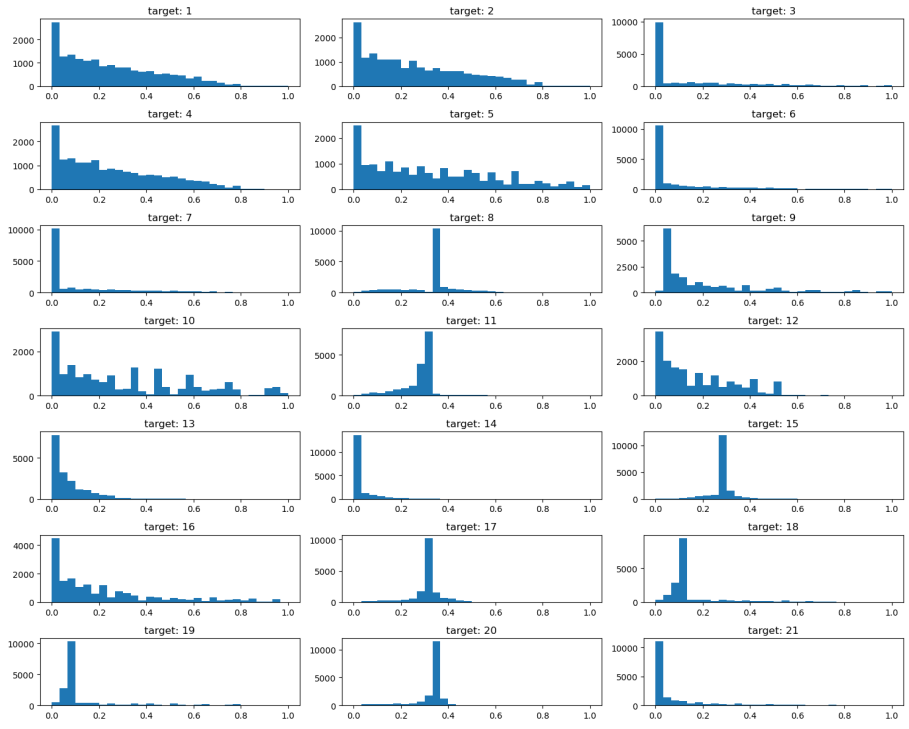


Figure 4.2: Distribution of normalized target variables

What can be observed here is that the target variables are more diverse than the feature variables. This suggests that the targets can result in a broader range of outcomes depending on the input features. Another notable observation is the high concentration of data points within specific bins.

4.2.2 Correlation Analysis

A correlation analysis aids in understanding correlations between variables and helps identify relationships, potential multicollinearity, and feature importance. A correlation matrix computed using PCC of the features can be seen in Figure 4.3.

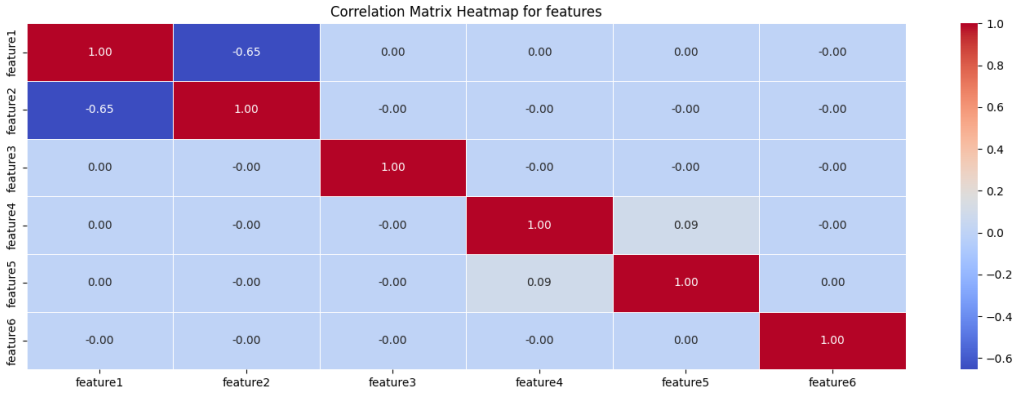


Figure 4.3: Correlation coefficient matrix of the features

What can be noted is a weak correlation among most feature variables, except for a strong correlation between Feature 1 and Feature 2. The reason for this correlation cannot be disclosed, as it would reveal insights into the system. However, it can be stated that the observed correlation is solely based on chance. It does not imply a correlation between these variables. This suggests that multicollinearity is not a major concern with these features. However, feature engineering will play a crucial role in mitigating potential redundancy and improving model performance for the correlated features. The results of the correlational analysis between the feature and target variables are presented in Figure 4.4.

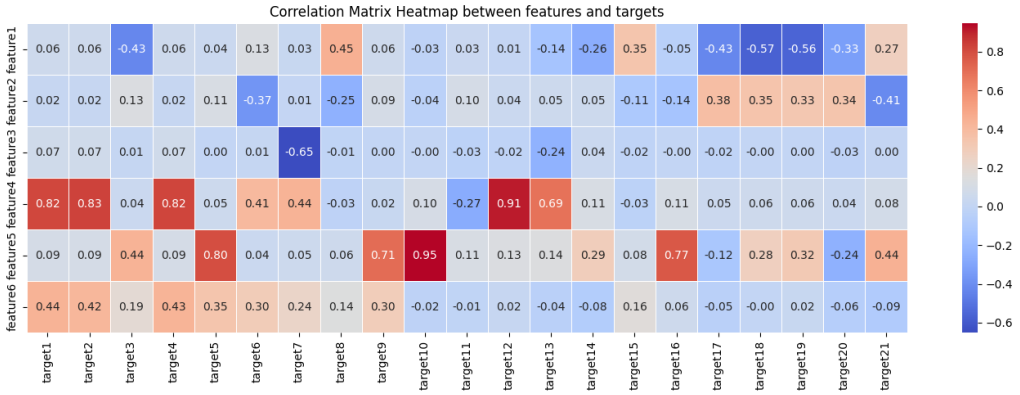


Figure 4.4: Correlation coefficient matrix of the features and targets

From this, we can see that several feature variables, such as Feature 4 and Feature 5, exhibit strong positive correlations with certain target variables, indicating that increases in these feature values are associated with increases in the target values. In the meantime, specific feature-target pairs, such as Feature 3 and Target 7, exhibit strong negative correlations, implying that the higher values of these features correspond to lower target values.

4.3 Data Preprocessing and Preparation

To create an effective model, a crucial step is data preprocessing. Since the model is strictly data-driven, its performance is entirely dependent on the quality of the input data [48]. The preprocessing steps undertaken include several key measures to improve data quality and ensure consistency, the actions taken was guided by the findings from the data exploration.

4.3.1 Preprocessing

Functions were developed to execute the actions stated below. These preprocessing steps were essential to ensure the dataset's reliability and suitability for building an accurate and robust ML model.

- **One Hot Encoding**:
  The categorical feature data were converted into a binary numerical format to enable efficient processing by the model.
- **Removal of Negative Flows**:
  In cases of small flows and those with a value of zero, GT-SUITE may occasionally return a small negative flow. Data entries with negative flow values were identified and removed to prevent unrealistic values from influencing the model.
- **Removal of Small Flows**:
  Flows below a defined threshold were excluded. This threshold was set at $1e^{-3}$ as values below this were deemed to be essentially zero.
- **Unit Conversion**:
  The units of certain variables were adjusted to ensure consistency in measurement units for systems against which the model will be compared.
- **Renaming of Features and Targets**:
  Column names were revised to improve clarity and ease of use for future reference.

4.3.2 Splitting and Scaling of Data

Next, the data was split into a training set and a test set, ensuring the model would be evaluated on unseen data. Two different splits were performed, as the ANN requires a subset of the training set as a validation set. For the ANN models, a three-way split was employed, comprising a training set, a validation set, and a test set. An 80/10/10 split was chosen for the ANNs. For the traditional ML models, a standard 80/20 split was used. Cross-validation was also performed on the training set to ensure model robustness and prevent overfitting by evaluating performance across different folds of the training set.

During the split, the data was randomly shuffled to reduce the bias of specific cases. After splitting, the data was scaled using StandardScaler. To maintain consistent

scaling throughout the training, validation, and testing phases, the scaler was fitted on the training set and then applied to the validation and test sets. This was done to prevent data leakage, ensuring that no information from the training set was leaked to the validation or test set, thus keeping this data unseen by the model.

4.4 Model Development

The next phase involved developing the ML models. This included designing model architectures and tuning parameters and hyperparameters to optimize performance. Both traditional ML models, such as linear regression, and artificial neural networks (ANNs), were explored. Insights from the literature review and the outcomes of data preprocessing guided the model selection.

4.4.1 Traditional Machine Learning models

The problem to be solved is a multi-output regression task, meaning that a single set of input features, in this case, a configuration of the TMS, should produce all 21 KPIs as targets. As a general principle in ML, one should avoid unnecessarily overcomplicating the solution. Therefore, simpler and more interpretable traditional ML models capable of multi-output regression were applied first.

Using the Python library Scikit-learn, models were created for each of the following methods: linear regression, PR with varying degrees, kNN, SVR, decision tree, random forest, and gradient boosting. Scikit-learns GridSearchCV was applied to all models with tunable parameters, that is, all except linear regression, to identify the most suitable parameters for this problem. Since Scikit-learn's standard classes for ML models do not provide direct insight into the training process, a custom function was developed to generate learning curves for each of the 21 KPIs. This function systematically trains the models on progressively larger portions of the dataset, incrementally increasing the dataset size, and evaluates performance at each step. The resulting curves are visualized to allow examination of the training process, helping to identify whether the model is underfitting or overfitting. Furthermore, it provides insight into which KPIs the model struggles to learn and which it learns effectively.

4.4.2 Artificial Neural Network models

Moving on from traditional ML models, ANNs were explored. The principle of starting simple also applies here. Initial experimentation involved manually testing various network architectures using the Python library PyTorch, with results documented to develop an understanding of what hyperparameters and architectures seem promising. As with the traditional models, the training process was monitored and visualized using a custom function. The architecture initially explored was a standard MLP that simultaneously predicts all 21 KPIs, thereby sharing the input information across all neurons in the network. This model will be referred to as the MLP model. Subsequently, other architectures were explored, including an MLP that partially shares information between neurons up to the output layer, where it

branches. The outputs for flow and pressure were separated based on the assumption that these targets exhibit significantly different characteristics. This model will be referred to as the Multi-Task Learning model. A third final model was also developed, fully separating the two output types: one MLP was designed to predict only the flows, and a separate, similar MLP was used to predict only the pressures. The last ANN model will be referred to as the Separated MLP model.

4.4.2.1 Hyperparameter Tuning

After experimenting with initial architectures and gaining insight into what works and what does not, a function for performing a Grid Search was developed. Since manually tuning the hyperparameters of a network is time-consuming and prone to human error, Grid Search was explored as an alternative. A crucial aspect of this approach is ensuring that the combinations evaluated are both relevant and computationally manageable, as the method is time-consuming. To address this, the hidden size was restricted to powers of two. Not only to reduce the number of possible combinations, but also to align with the fact that the final model is intended to run on physical hardware that performs calculations in binary.

Additionally, the hidden size was kept uniform across all hidden layers. Based on insights gained from manual testing and guidance from the literature review, the relevant hyperparameters and their corresponding value ranges were defined. The hyperparameters and their combinations are tabulated in Table 4.1.

Table 4.1: Hyperparameters for Grid Search

Hyperparameter	**Values**	**Description**
Learning Rate	$\{1e^{-2}, 5e^{-3}, 1e^{-3}, 5e^{-4}\}$	Weight update step size
Hidden Layers	{2, 3, 4, 5}	Number of hidden layers
Hidden Size	{32, 64, 128}	Neurons per hidden layer
Activation Function	{ReLU, Leaky ReLU, SiLU, Sigmoid, Tanh}	Non-linear function
Loss Function	{MSE, MAE, Huber}	Model error metric
Optimizer	ADAM	Optimization algorithm
Batch Size	64	Samples per batch

4.5 Model Validation and Evaluation

The next phase involved validating and evaluating the developed models. Validation ensured that the model outputs were consistent with the physics, constraints, and dynamics of the TMS. Furthermore, the evaluation criteria focused on the metrics outlined in the project's aim, specifically, accuracy, storage size, and inference time.

4.5.1 Accuracy

To evaluate the model's accuracy, an automated script was developed to assess several performance metrics simultaneously for all targets. It calculates the MAE, MSE,

RMSE, and R-squared for the trained model on the test set for all 21 targets and generates a report of these metrics. This provides a clear view of which targets the model can predict effectively and which ones it struggles with. However, since mathematical methods are general and not specific to a particular problem, additional methods are needed to assess accuracy in the context of a TMS for a BEV.

Therefore, an interactive plot was developed to investigate critical configurations. The plot is a scatter plot that displays all model predictions on the test set on the y-axis, with the corresponding ground truth or actual target on the x-axis. The data points in the plot are color-coded based on a hybrid accuracy function to determine whether a prediction is satisfactory. The plot is interactive, as hovering over a data point displays the specific configuration of the test set, along with the prediction and the ground truth. This allows for an easy and systematic investigation of the predictions.

4.5.2 Storage Size

Due to the critical role of the VCU, storage size is another important metric. While the exact numerical goal for storage size is unknown at the time of writing this report, it can be concluded that it should be minimized as much as possible while still ensuring that the other metrics remain within acceptable boundaries.

To accurately evaluate the storage size, it would need to be assessed for a model converted into C code, as this is what would be stored on the VCU. Given the need to convert multiple models to C and the large number of models tested, it was decided to evaluate the storage size based on the Python version of the models.

4.5.3 Inference Time

The inference time is a crucial metric due to the constrained environment of the VCU in a BEV. For the proposed model to be used in the vehicle, it must be faster than the hardware's update rate. Since the inference time depends on the hardware used, testing the model requires it to be physically tested on the actual hardware in the vehicle. This presents a challenge, as the models are developed in a programming language that is not compatible with the VCU. To address this, a method was developed to convert the Python models into C code. Since converting Python code into production-ready and hardware-optimized C code is a time-consuming process, this was only performed for the final, best-performing model.

4.5.4 Postprocessing

As the last part of the evaluation process, postprocessing was performed to refine the results. Since the dataset includes nearly all possible configurations the TMS could encounter, and the model was trained on data from a digital twin with inherent simplifications, some configurations proved difficult or unrealistic to predict. Configurations previously identified as problematic during the digital twin analysis were therefore excluded from the final evaluation. This step helps ensure that the reported performance metrics reflect realistic and meaningful system behavior.

4.6 Conversion to C Code

To deploy an ML model in an embedded system, such as the VCU of a BEV, one must ensure it is compatible with the exact hardware in use. In this case, no specific hardware requirements can be disclosed due to confidentiality, but it can be stated that the model should be executable in a C code environment. To achieve this, there are several methods depending on the specific ML algorithm. In the case of this project, only the technique applied using the final chosen model will be explained.

As the chosen model was constructed using Python's PyTorch library, it first needed to be transferred from the Python environment into MATLAB. This was achieved by saving the trained model using PyTorch's Just-In-Time tracing functionality. The model was traced using a representative dummy input and saved as a .pt file. This .pt file was then imported into MATLAB using the Deep Network Designer app, which provides functionality for loading and converting PyTorch models into MATLAB's network format. After verifying the imported network structure and parameters, the model was integrated into a Simulink environment for further processing and testing.

Using MATLABs Embedded Coder toolbox, the Simulink model containing the ANN was then configured for C code generation. Embedded Coder produces optimized C code suitable for deployment in embedded environments.

4.7 Integration into TargetLink

Once the ANN was successfully converted to C code using Embedded Coder, the generated code was integrated into a TargetLink environment. TargetLink, developed by dSPACE, is used for generating production-level embedded C code directly from Simulink models [49]. Integrating a model already in C into software used to create C code may sound a bit strange, but the reason for this is that by incorporating the C code model into TargetLink, an even more optimized and correctly formatted C code can be generated. Furthermore, a model integrated into TargetLink can also be simulated with other software components within the BEV and used to simulate system responses.

To integrate the code, the output from Embedded Coder was encapsulated in TargetLink via an S-Function interface. The TargetLink model was then configured to interact with the ML model block, including the mapping of input and output signals. Validation steps were conducted throughout the integration process to ensure that the model behaved consistently across its PyTorch, MATLAB, and Simulink versions. While having a functional model in TargetLink does not fully guarantee that it would work in practice within the actual BEV, it is the closest possible validation without explicitly testing it in the vehicle.

5

Results

This chapter presents the project's results. The results are organised as follows: first, the results of five traditional ML models, PR, kNN, SVR, XGBoost, and RF, are presented. This is followed by the results of the three ANN models: a standard MLP model, a Multi-Task Learning model, and a separated MLP model.

For each model, the learning curves are shown to illustrate the training progress. Subsequently, performance metrics, specifically the MAE, are visualised in appropriate plots. Finally, the predictions are compared against the actual values through scatter plots. This structure is consistently applied to both traditional ML models and ANN models. However, fewer results are presented for the traditional ML models due to reasons that will be explained in the discussion chapter. All results are based on the same test set of data, and the details of this are tabulated in Table 5.1. These settings are applied to all models during testing to ensure a fair comparison between the different methods.

Table 5.1: Experimental setup for model evaluation.

Parameter	**Value**	**Description**
Random seed	40	Fixed seed for reproducibility
Training set size	80%	Portion of dataset used for training
Test set size	10%	Portion of dataset used for evaluation
Validation set size	10%	Portion of dataset used for tuning ANN models

5.1 Traditional Machine Learning Models

This section presents the results of five traditional ML models: 5th-degree PR, kNN, SVR, XGBoost, and RF Regression. The performance metrics include the average MAE over all targets and MSE for each model. Additionally, predicted versus actual value plots are provided for selected target variables to illustrate model performance further. The learning curves of each model during training are shown in Appendix A.2.

5.1.1 Performance Metrics

Tabulated in Table 5.2 are the normalized average MAE over all 21 targets for predictions generated by five traditional ML algorithms. Among them, the RF model had the largest size, while XGBoost yielded the lowest MAE, indicating the best overall performance.

Table 5.2: Normalized average MAE over all 21 targets for traditional ML algorithms: Polynomial Regression, k-Nearest Neighbor, Support Vector Regression, Random Forest, and XGBoost

Regression Algorithm	MAE	Size (kb)
PR (5th degree)	0.003 326	1474
SVR	0.004 999	2279
kNN	0.006 869	3580
XGBoost	0.001 781	32 341
RF	0.007 520	1 149 123

Moving on, the normalized MAE for Target 1 and Target 21 is presented in Figure 5.1. These two targets were selected for illustration due to their particularly interesting characteristics. The specific reasons for choosing Targets 1 and 21 cannot be disclosed due to confidentiality. Showing all targets was deemed unnecessary, as the selected examples sufficiently illustrate the key insights. It can be observed that kNN exhibits the highest error for Target 1. In contrast, SVR performs poorly on Target 21, despite achieving a relatively low error for Target 1. Interestingly, XGBoost is the best predictor for both Target 1 and Target 21.

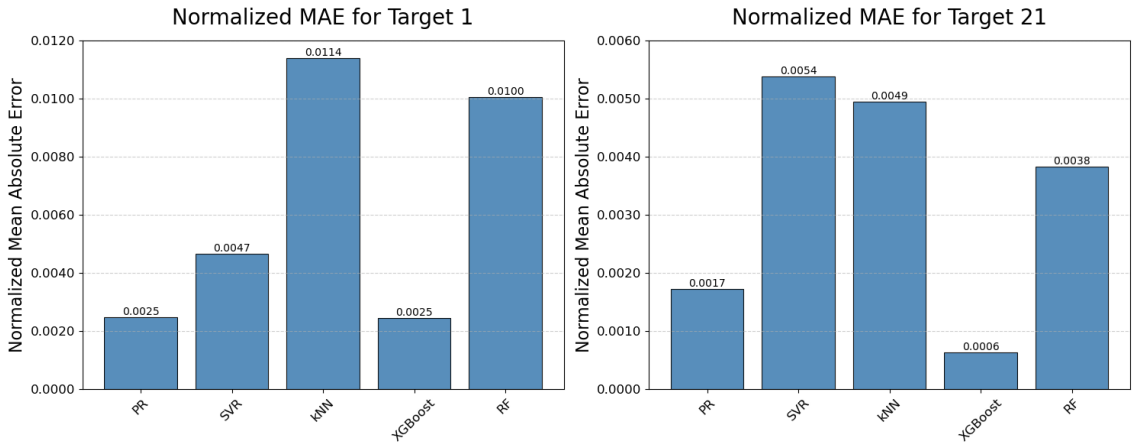


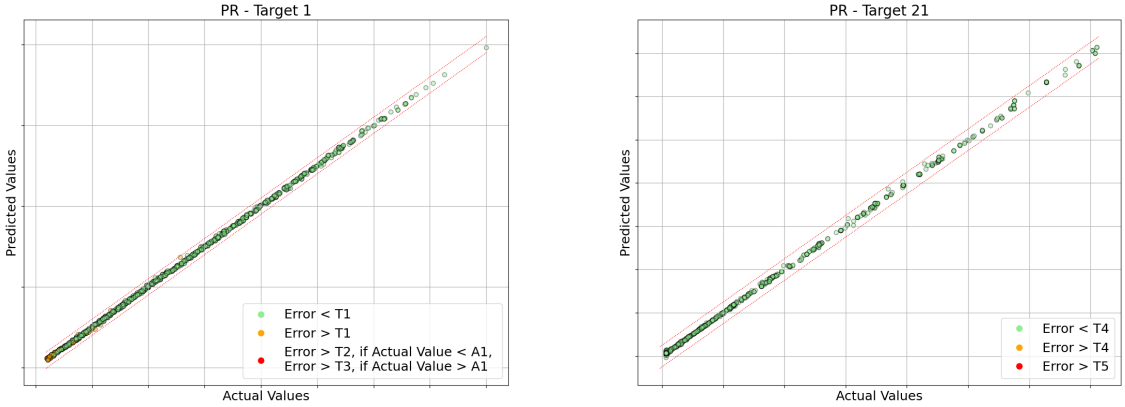
Figure 5.1: Mean Absolute Error for the Target 1 and Target 21 predictions of 5-th degree Polynomial Regression, Random Forest, XGBoost, Support Vector Regression, and k-Nearest Neighbor respectively

5.1.2 Predicted vs Actual Value

In the following section, scatter plots of predicted versus actual values for specific targets are presented. In the ideal case, the predictions align along a diagonal line

where the predicted value equals the actual value. Moving to the right in the x-axis indicates an increase in the numerical value of the actual value, and moving higher up on the y-axis indicates the same but for the predicted value. The x and y values have been removed for confidentiality purposes.

The predictions for Targets 1 and 21 are shown across all evaluated traditional ML algorithms. The two red dotted lines in the figures represent absolute error thresholds. Each data point is color-coded based on its percentage error: red indicates a high error, which means that the error exceeds the threshold, yellow indicates a medium error, which is still satisfactory, and green indicates a low error, which is a satisfactory prediction. A red data point located outside the red dotted lines is considered unacceptable. However, green or yellow points outside the lines are still regarded as satisfactory. And a red dot within the limits of the two red dotted lines can also be considered acceptable. Figure **??** displays the predicted versus actual plots for the 5th-degree PR model.

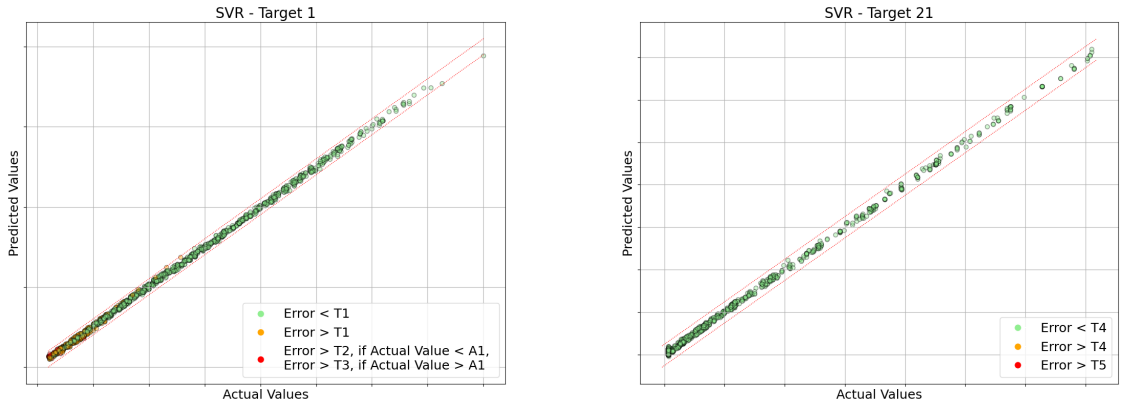


(a) PR Target 1

(b) PR Target 21

Figure 5.2: Prediction vs actual values for two different targets from the 5th degree Polynomial Regression

For Target 1, a density of orange dots is observed at low values, with some scattered points appearing in the mid-value range, indicating moderate prediction errors. In contrast, the model shows better predictive performance for Target 21, as evidenced by the presence of green dots only, suggesting fewer and less severe errors. The prediction vs actual values plot for the SVR model can be seen in Figure 5.3.

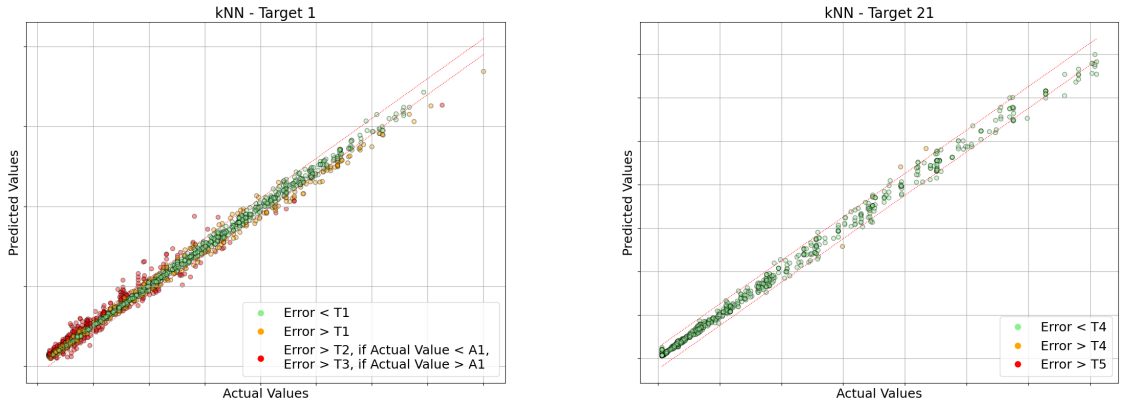


(a) SVR Target 1

(b) SVR Target 21

Figure 5.3: Prediction vs actual values for two different targets from the Support Vector Regression

In the case of Target 1, both medium and high error points are observed, particularly in the lower value range. Similar to the PR model, SVR demonstrates better predictive performance for Target 21. Moving on, seen in Figure 5.4 are the results for kNN regression.

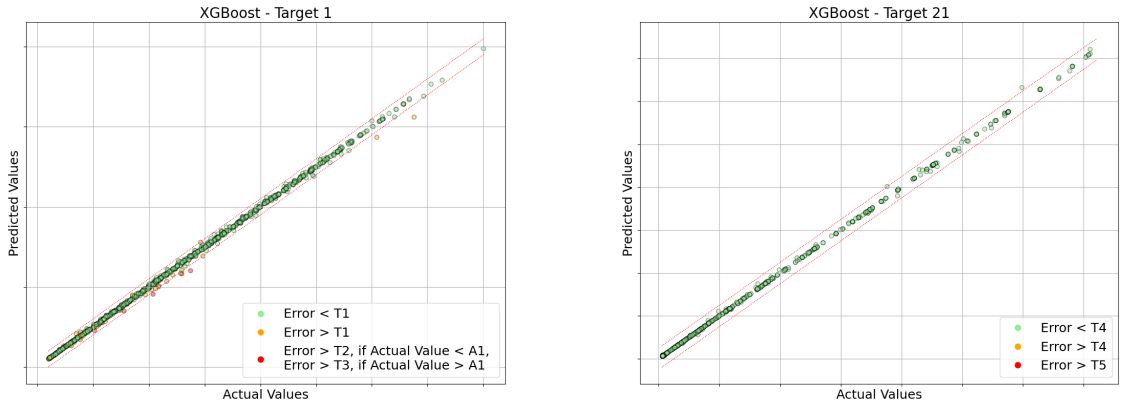


(a) kNN Target 1

(b) kNN Target 21

Figure 5.4: Prediction vs actual values for two different targets from the k-Nearest Neighbors Regression

The prediction results for Target 1 using kNN regression show poor performance, as numerous high-error points are scattered along the line, particularly in the low-to-mid-value range. Additionally, medium-error points can be found throughout the entire range. In contrast, the target 21 predicted with this algorithm has better performance. However, it still has a few points that exceed the absolute error threshold, and the predicted values exhibit a wide range of variation. Figure 5.5 shows the results for the XGBoost model.

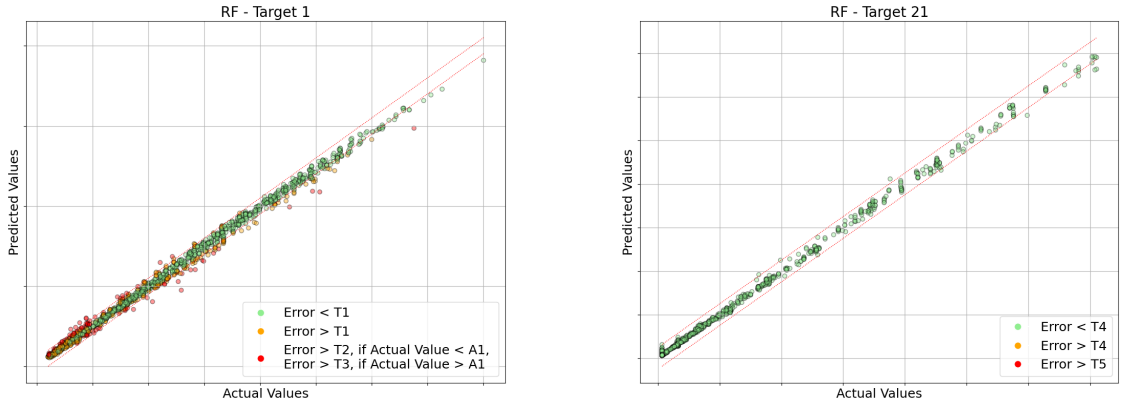


(a) XGBoost Target 1

(b) XGBoost Target 21

Figure 5.5: Prediction vs actual values for two different targets from the XGBoost Regression

For Target 1, several medium and high-error points are observed along the diagonal, with a few exceeding the absolute error threshold. In contrast, the predictions for Target 21 show no medium or high error points, indicating a more accurate performance for that target. However, a few points are over the absolute error threshold. Lastly, the results for the RF model are shown in Figure 5.6.



(a) RF Target 1

(b) RF Target 21

Figure 5.6: Prediction vs actual values for two different targets from the Random Forest Regression

It can be seen that the plot of predicted vs. actual values for Target 1 generated by RF regression has numerous high error points, particularly in the lower value ranges. In contrast, medium error points can be observed along the diagonal. The model demonstrates better performance in predicting Target 21; however, a few points still exceed the absolute error threshold.

5.2 Artificial Neural Network Models

In the following section, the results of the three developed ANN models are presented. First, the learning curves for each model are visualized. Next, violin and bar plots illustrating the MAE for each target are shown. Finally, predicted versus actual value plots, using the same visualization approach as for the traditional models, are included.

5.2.1 Learning Curve

To examine the training process and ensure convergence of the three models, the learning curves for each model are presented. Seen in Figure 5.7 is the learning curve for the MLP.

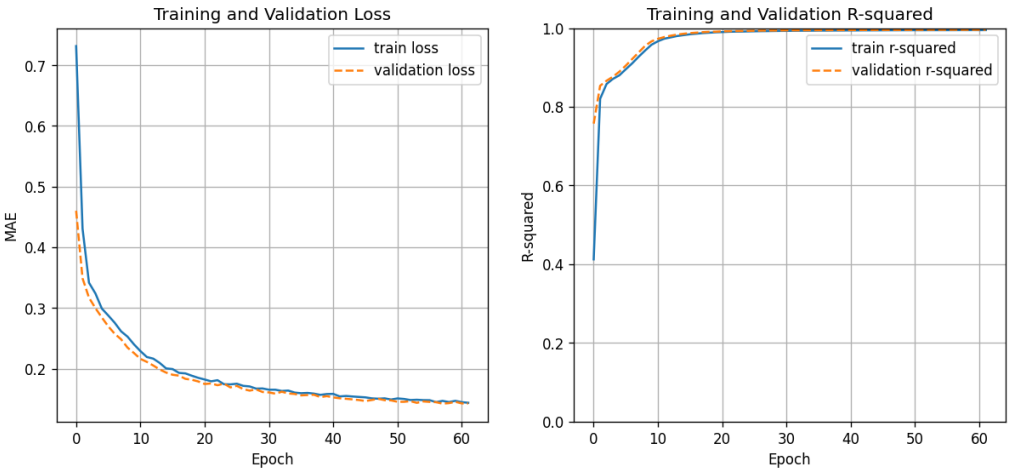


Figure 5.7: Learning curves for MLP model

From the figure, it can be concluded that early stopping was triggered at 61 epochs. The loss curve appears relatively smooth, and the training and validation losses converge to nearly identical values. The R-squared learning curves indicate that the model struggles during the initial epochs but then progresses steadily. Subsequently, Figure 5.8 depicts the learning curves for the Multi-Task Learning model.

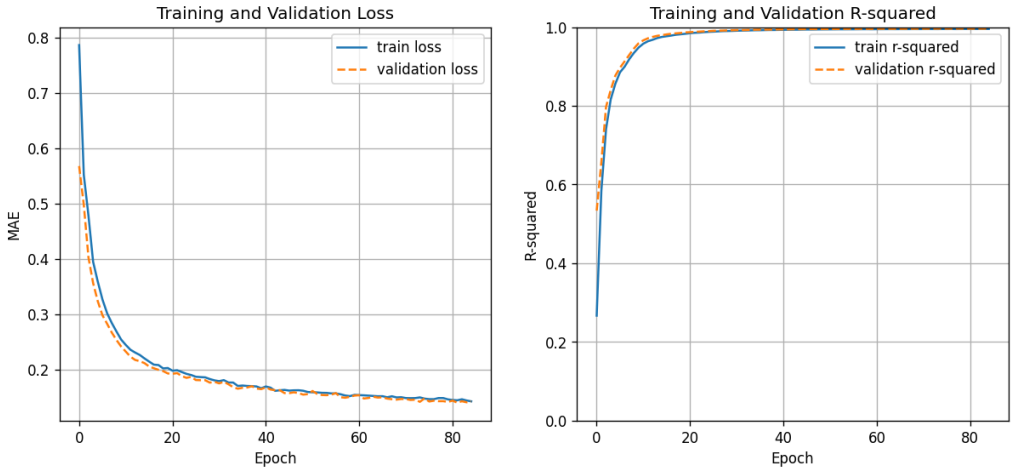


Figure 5.8: Learning curves for Multi-Task Learning model

For this model, the learning process remains smooth throughout all epochs. The training and validation losses converge to nearly identical values, and early stopping is triggered at 82 epochs. Figure 5.9 illustrates the learning curves for the pressure targets of the separated model.

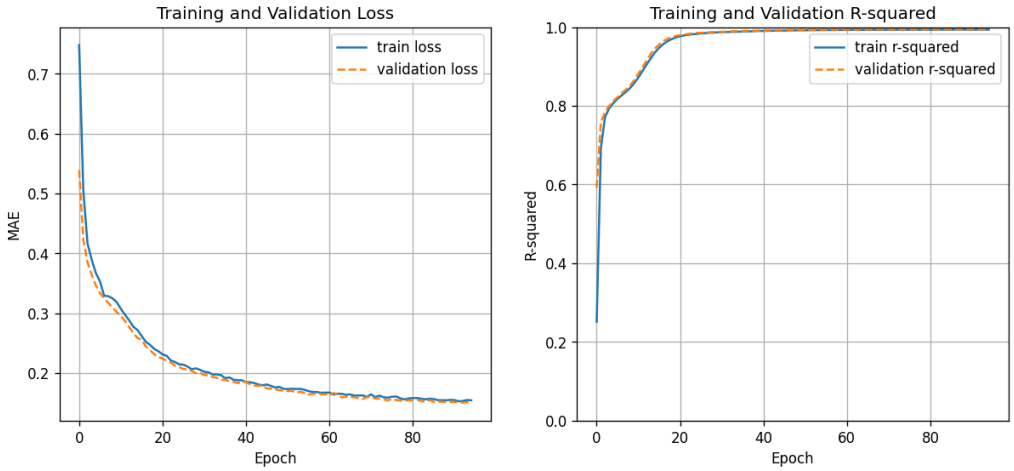


Figure 5.9: Learning curves for pressure targets Separated MLP model

The loss curve for the pressure targets is relatively smooth, but similar struggling behaviour between epochs 0 and 20, as observed in the MLP model, is also evident here. Training continues for a total of 87 epochs. Lastly, Figure 5.10 depicts the learning curves for the pressure targets in the separated model.

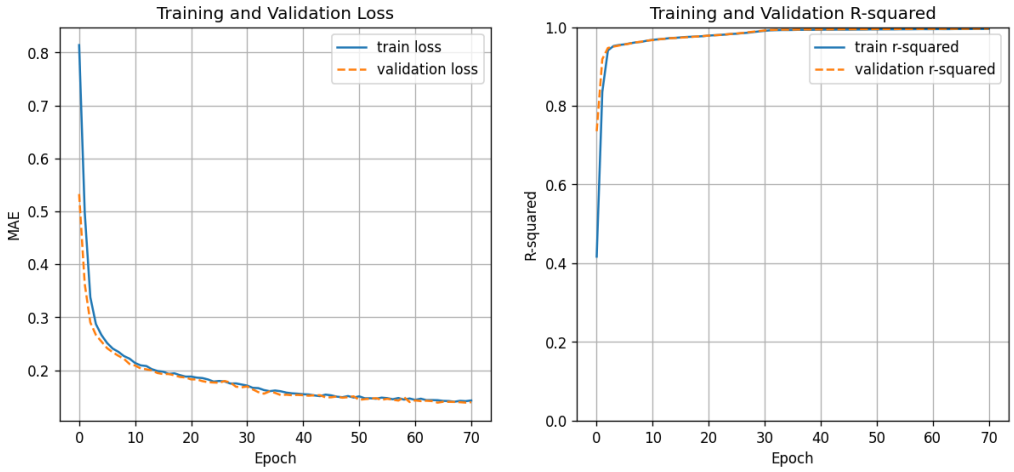


Figure 5.10: Learning curves for flow targets Separated MLP model

Training stops at 70 epochs, and the loss curve remains smooth throughout the process. However, progress slows noticeably between epochs 10 and 30.

5.2.2 Performance Metrics

In this subsection, the performance metrics for the ANN models are presented in the form of MAE. First, in Table 5.3, the overall performance metrics for the three models are tabulated.

Table 5.3: Normalized average MAE over all 21 targets for ANNs: MLP, Multi-Task Learning, and Separated MLP

Regression Algorithm	**MAE**	**Size (kb)**
MLP	0.003 214	41.34
Multi-Task Learning	0.003 638	57.61
Separated MLP	0.003 705	25.42

Furthermore, what will be presented is a violin plot illustrating the distribution of normalized absolute errors for all test set predictions on Targets 1 to 10 for a specific ANN model. This is followed by a bar plot showing the mean of these absolute errors, which represents the MAE. This structure is then repeated for Targets 11 to 21 and all three different ANN models. Figure 5.11 shows the first violin plot of the distribution of normalized outputs for the MLP model across Targets 1 to 10.

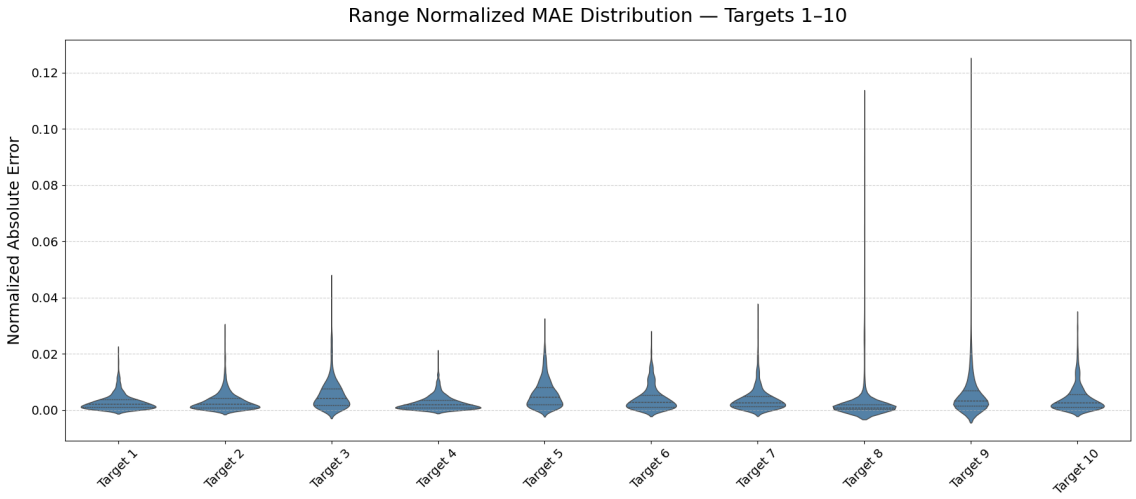


Figure 5.11: Violin plot of normalized outputs for Targets 1 to 10 for the MLP model

It can be noted from the figure that Targets 8 and 9 have outliers with higher errors compared to the other targets. Aside from these two, the error distributions for the remaining targets fall within a low margin. Figure 5.12 shows the same targets and the same test set evaluated using the Multi-Task Learning model.

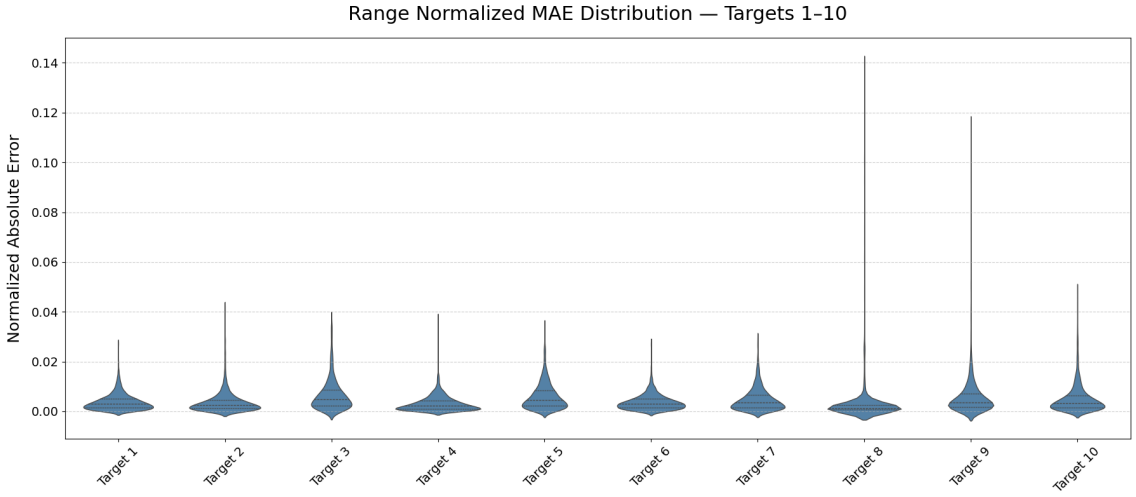


Figure 5.12: Violin plot of Targets 1 to 10 normalized for the Multi-Task Learning model

The results are similar overall. However, the worst predictions for almost all targets, except for Targets 7 and 9, have worsened. Figure 5.13 displays the corresponding results for the separated MLP model.

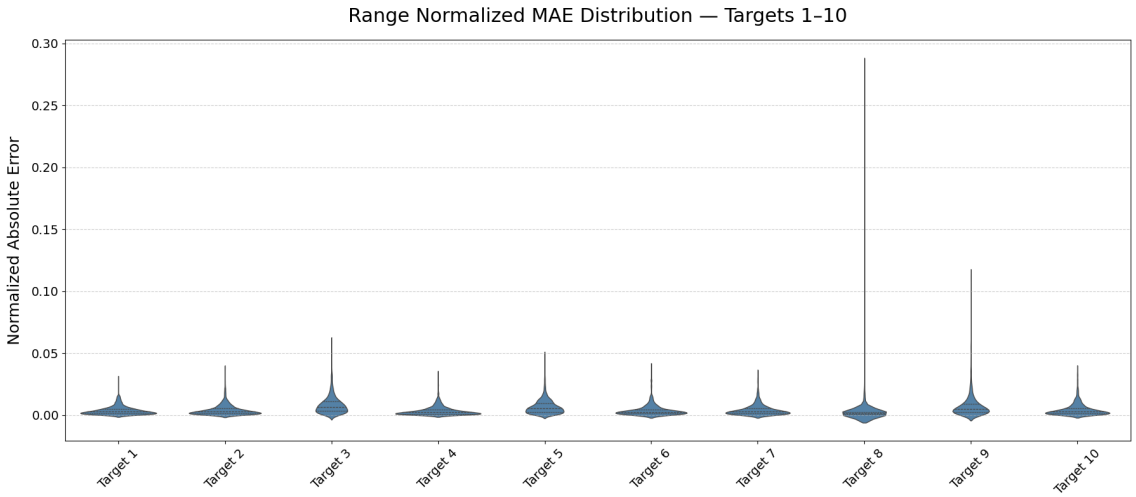


Figure 5.13: Violin plot of Targets 1 to 10 normalized for the separated MLP model

Most notably, the error of the worst prediction for Target 8 has increased. Apart from this, the results for the remaining targets are relatively similar. To better visualize the MAE of the evaluated models, Figure 5.14 presents a bar plot of the MAE for Targets 1 to 10 using the MLP model.

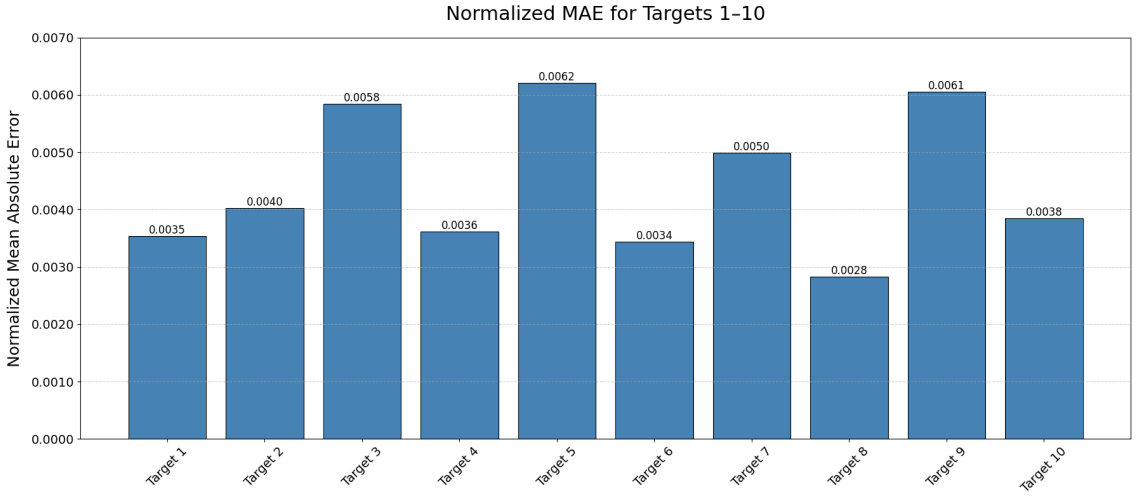


Figure 5.14: Bar plot of normalized outputs for Targets 1 to 10 for the MLP model.

Examining this bar plot, it is evident that the MAE for all targets is low and remains within a narrow range of 0.0034 normalized. The targets with the highest MAE are Targets 3, 5, and 9. Figure 5.15 presents a bar plot with the same structure, but for the Multi-Task Learning model.

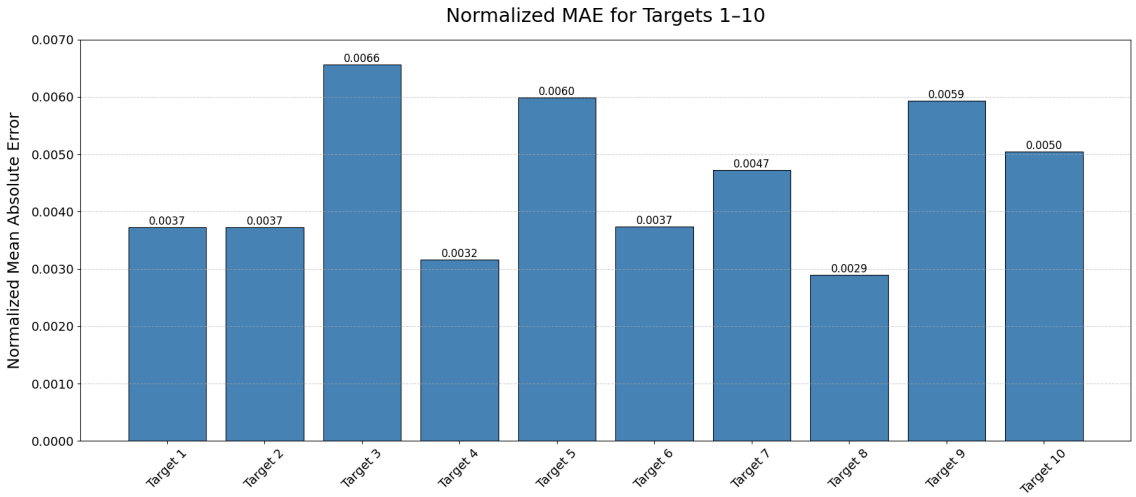


Figure 5.15: Bar plot of Targets 1 to 10 normalized Multi-Task Learning MLP

For the Multi-Task Learning model, the results are again very similar to those of the MLP. The most notable difference is the increase in MAE for Targets 3 and 10. Figure 5.16 depicts the corresponding bar plot for the separated MLP model.

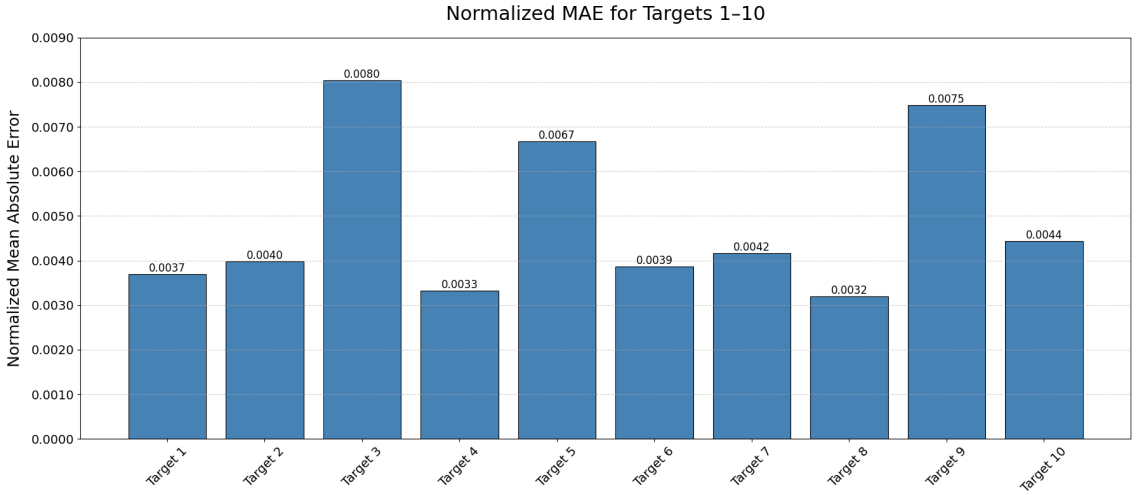


Figure 5.16: Bar plot of Targets 1 to 10 normalized separated MLP

For the final model, there is an apparent increase in error for the previously troublesome Targets 3, 5, and 9. The separated model performs noticeably worse in predicting these targets compared to the other two models. Furthermore, Figure 5.17 visualizes the absolute errors for Targets 11 to 21 using a violin plot.

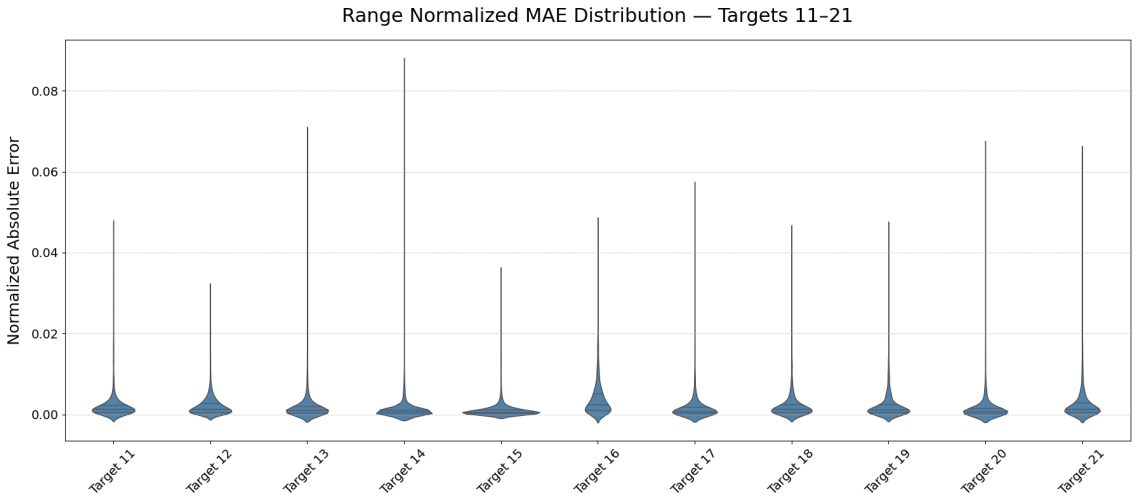


Figure 5.17: Violin plot of normalized absolute errors for Targets 11 to 21 MLP

As seen in this figure, and in contrast to Targets 1 to 10, these targets exhibit a wider range of absolute errors, with Target 14 yielding the highest error. Still, the distribution for all targets is centred close to zero. Aside from this, the range and distribution of predictions for most targets are similar, except for Target 15, which displays a noticeably flat distribution. Figure 5.18 shows the same plot for the Multi-Task Learning model.

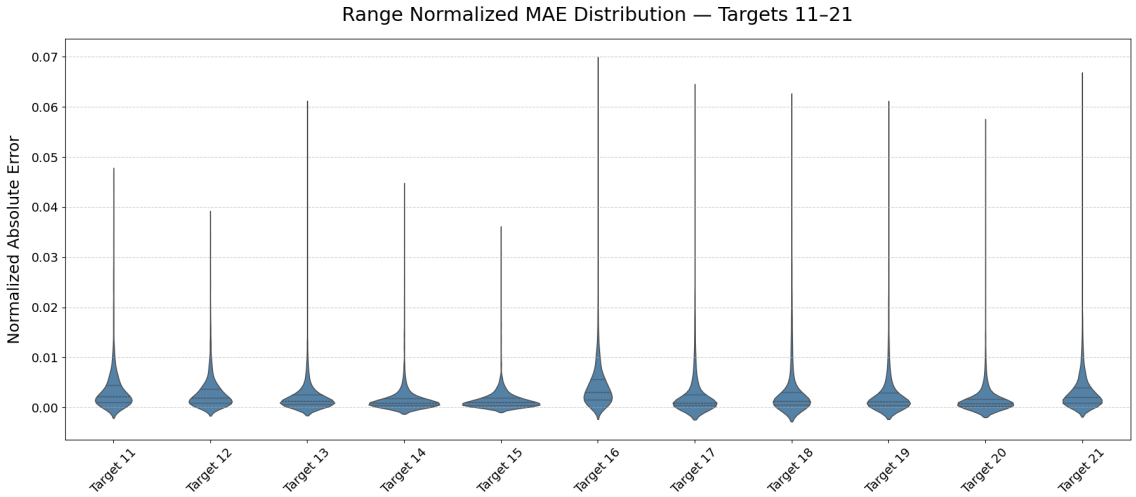


Figure 5.18: Violin plot of normalized absolute errors for Targets 11 to 21 Multi-Task Learning model

This model again produces results similar to those of the MLP, with the key difference being that Target 14 is replaced by Target 16 as the one with the highest error. The distributions for all targets appear to be wider, although this is difficult to assess precisely due to the differing y-axis limits. Figure 5.19 presents the same targets for the final model.

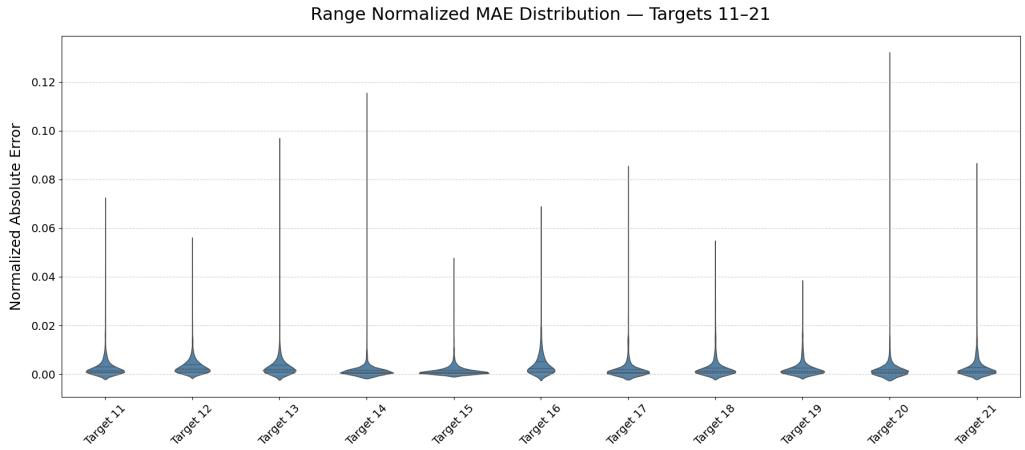


Figure 5.19: Violin plot of normalized absolute errors for Targets 11 to 21 separated MLP

For this model, the range of errors is larger, with Targets 14 and 20 exhibiting the highest values. Subsequently, Figure 5.20 presents a bar plot of the MAE for the MLP model.

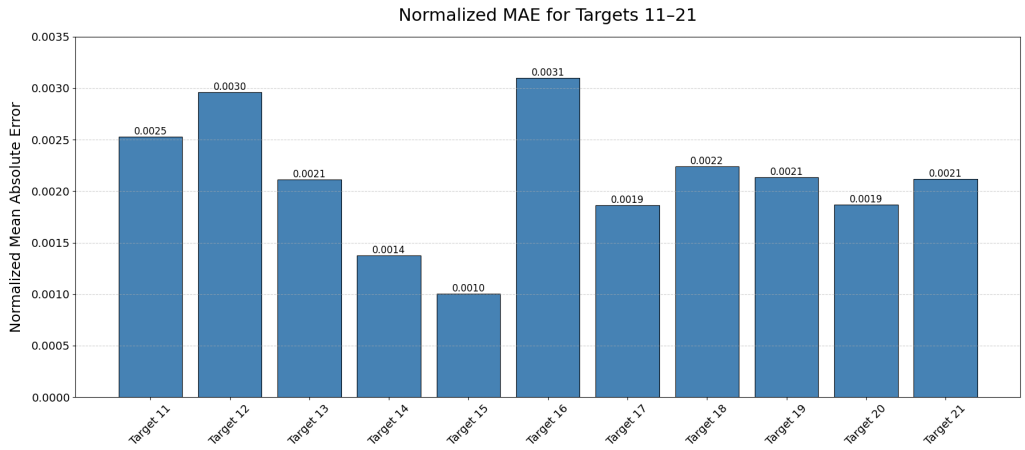


Figure 5.20: Bar plot of normalized outputs for Targets 11 to 21 MLP

It can be observed that the MAE for Targets 11 to 21 is generally lower than that for Targets 1 to 10. The range of MAE across these targets is also smaller, at 0.0021. Targets 12 and 16 exhibit the highest MAE within this group. Moving on, Figure 5.21 presents the bar plot for the next model.

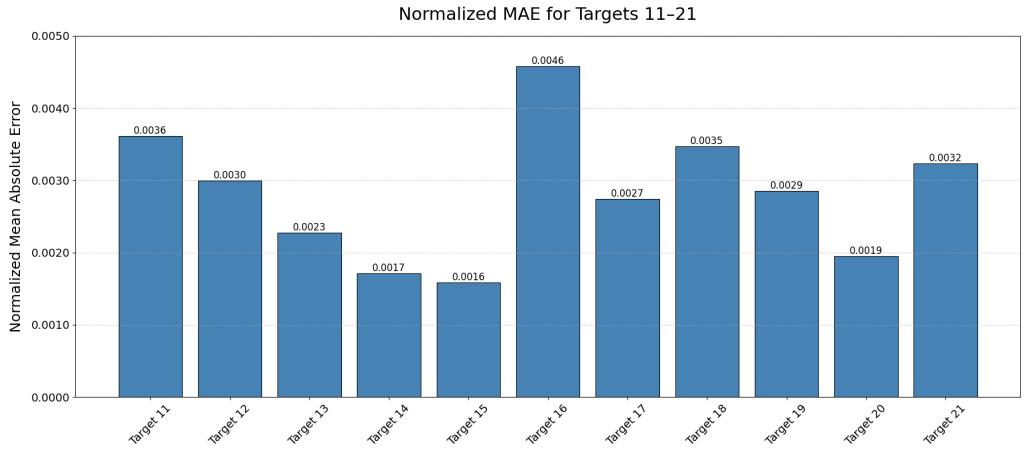


Figure 5.21: Bar plot of Targets 11 to 21 normalized Multi-Task Learning model

This plot confirms the earlier observation regarding the broader distribution of absolute errors in the Multi-Task Learning model compared to the MLP. In this case, the MAE is higher for all targets. Figure 5.22 illustrates the corresponding bar plot for the separated model.

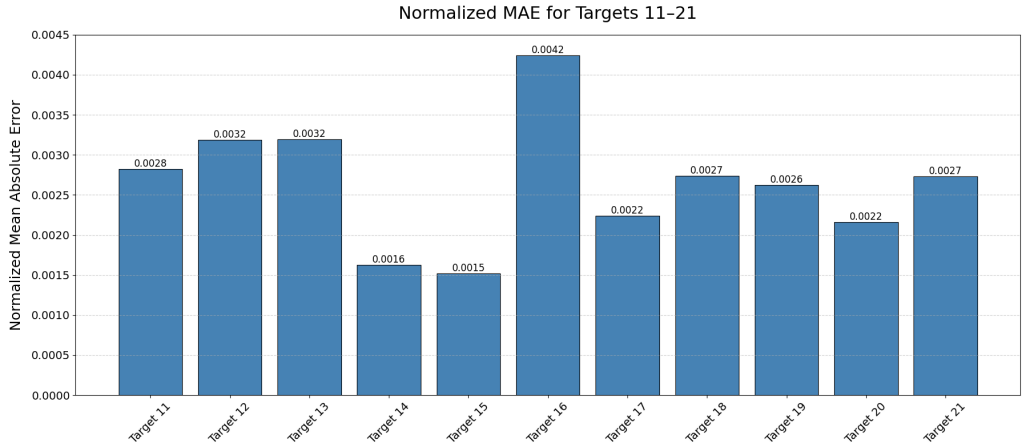


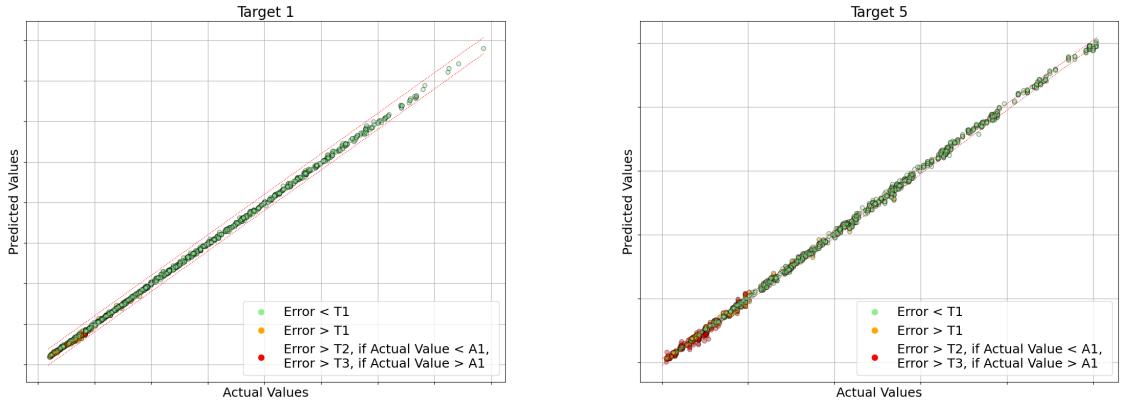
Figure 5.22: Bar plot of Targets 11 to 21 normalized separated MLP

Among these targets, the separated model shows slightly improved performance compared to the Multi-Task Learning model, although it remains worse than the MLP.

5.2.3 Predicted vs Actual Value

As with the traditional models, only a subset of targets is presented in detail due to the large number of targets, and in this case, also due to the similarity of many results. In most cases, the models perform satisfactorily, leaving little to comment on. To keep the result chapter focused and relevant, four targets with distinct characteristics have been selected: Targets 1, 5, 12, and 21. Due to confidentiality, the exact reason behind their selection cannot be disclosed. However, it can be stated that these targets are both essential and interesting, and they exhibit notable

differences in performance across the three ANN models. The results of these four targets are presented model by model, two targets at a time, beginning with the MLP model and Targets 1 and 5, as illustrated in Figure 5.23. The results for the rest of the targets for each respective model can be found in Appendix A.3 - A.4.

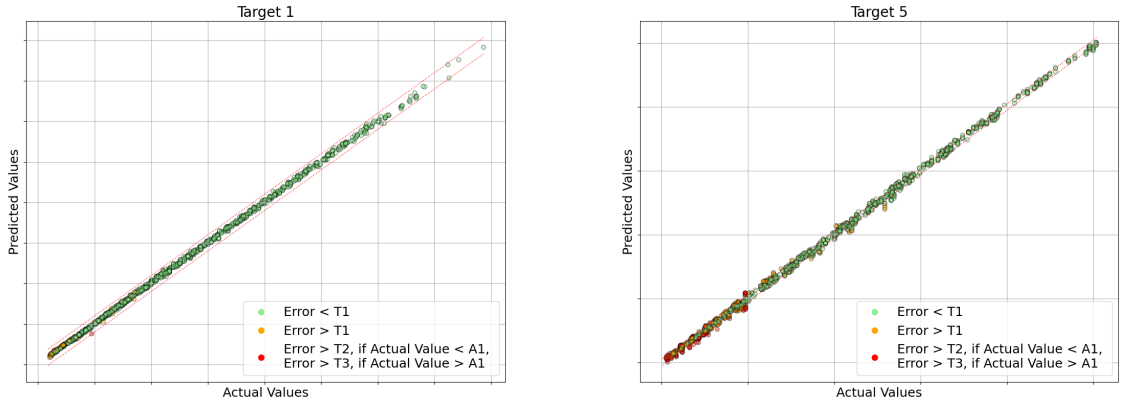


(a) MLP Target 1

(b) MLP Target 5

Figure 5.23: Prediction vs actual values for two different targets from the MLP model

The results for Target 1 using the MLP model are all satisfactory. However, for Target 5, several predictions fall outside both the percentage error threshold and the absolute error threshold. Notably, the non-satisfactory predictions fall within the lower range of numerical values. The same plots for the Multi-Task Learning model are shown in Figure 5.24.



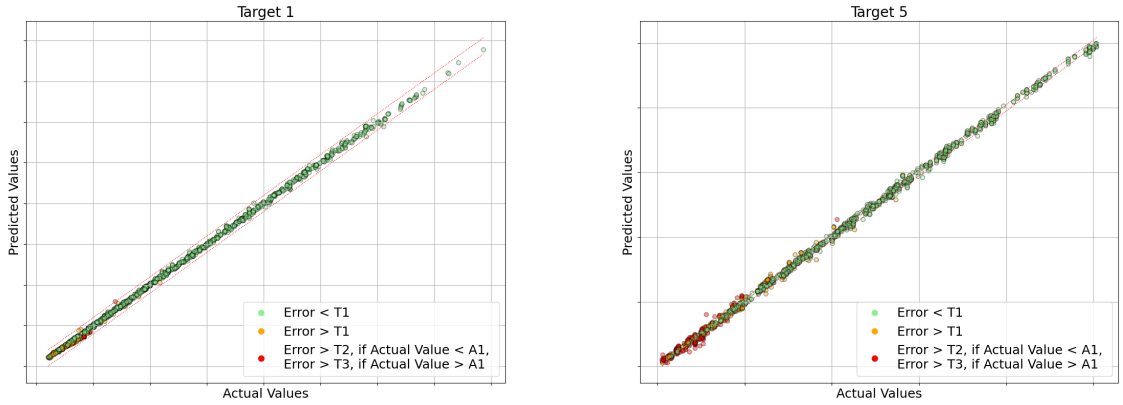
(a) Multi-Task Learning model Target 1

(b) Multi-Task Learning model Target 5

Figure 5.24: Prediction vs actual values for two different targets from the Multi-Task Learning model

The Multi-Task Learning model yields results similar to the MLP for Target 1, with a slight increase in the spread of predictions. This supports the earlier observations from the violin and bar plots. For Target 5, the results are again comparable; the issue with underestimating lower numerical values persists but appears to be less

severe. A broader spread of predictions is also evident in this case. Furthermore, Figure 5.25 illustrates the results for the separated MLP model.

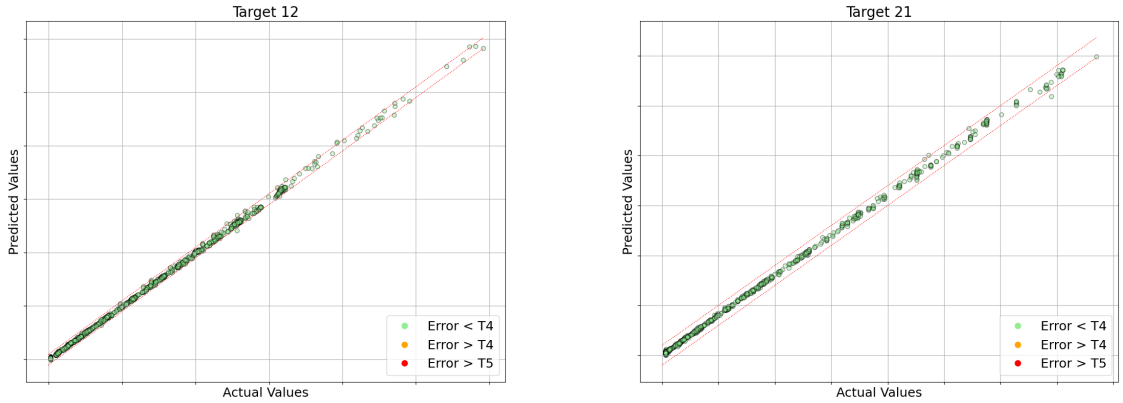


(a) Separated MLP Target 1

(b) Separated MLP Target 5

Figure 5.25: Prediction vs actual values for two different targets from the separated MLP

The results for the separated model are again similar to those of the two previous models. However, the spread is broader for both targets when compared to the MLP, and the issues with lower values persist for Target 5. Targets 12 and 21 for the MLP model are shown next in Figure 5.26.

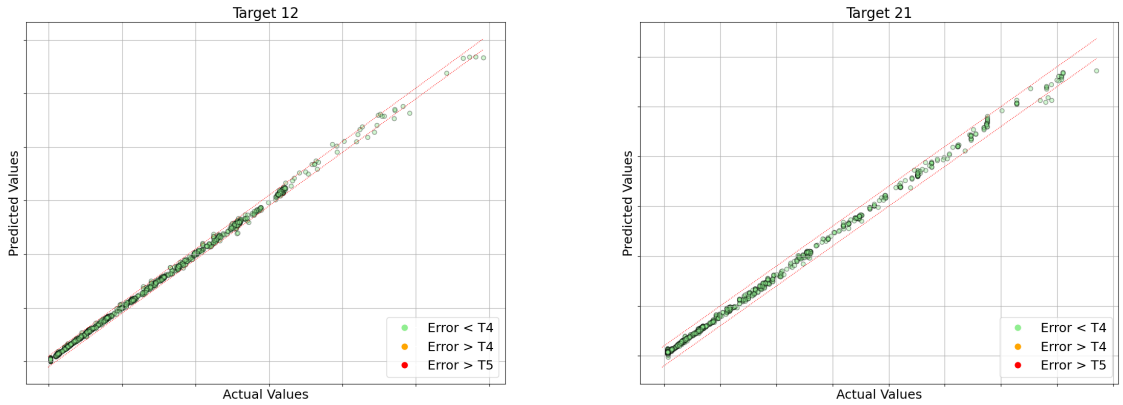


(a) MLP Target 12

(b) MLP Target 21

Figure 5.26: Prediction vs actual values for two different targets from the MLP model

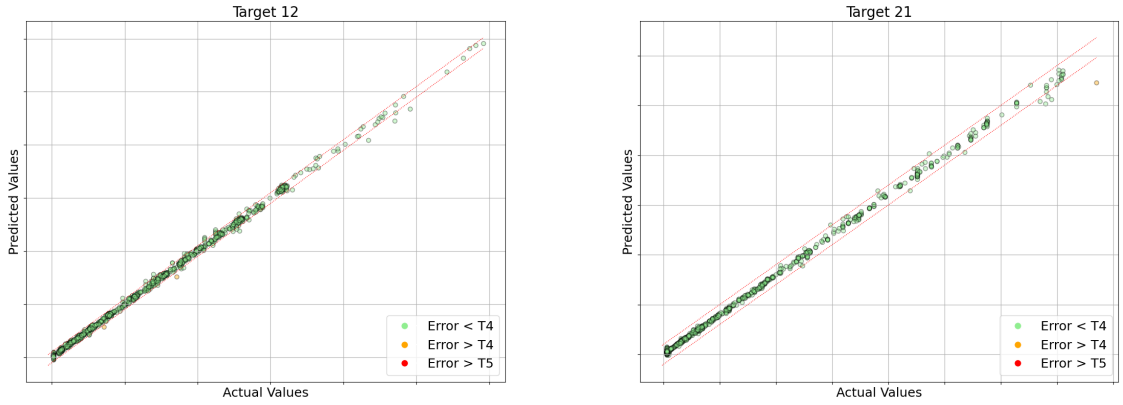
For Targets 12 and 21 using the MLP model, all predictions are satisfactory. Subsequently, the corresponding results for the Multi-Task Learning model are presented in Figure 5.27.



(a) Multi-Task Learning model Target 12 (b) Multi-Task Learning model Target 21

Figure 5.27: Prediction vs actual values for two different targets from the Multi-Task Learning model

Similar to the MLP model, all predictions are satisfactory for both targets. However, the spread of predictions is noticeably larger. Finally, the results for the same targets using the separated model are shown in Figure 5.28.



(a) Separated MLP model Target 12 (b) Separated MLP model Target 21

Figure 5.28: Prediction vs actual values for two different targets from the separated MLP model

Like the two previous models, the predictions for both targets are satisfactory, though the distribution of predictions is more scattered than for the standard MLP model.

5.2.4 Inference Time

As explained in the Methods chapter, accurately measuring the inference time of the proposed model requires evaluation on the specific hardware intended for use in the vehicle. Although this was not feasible within the project's timeframe, a method for estimating inference time in a simulation environment was developed. The configuration settings for the simulation conducted in Simulink are presented

in Table 5.4.

Table 5.4: SIL Simulation Setup in Simulink Environment

Component	**Description**
Simulation Mode	Software-in-the-Loop (SIL)
Solver Type	Fixed-step / Discrete
Step Size	0.01 s
Code Generation Tool	Embedded Coder
Target Language	C
Model Interface	S-Function Wrapper
Execution Platform	x86-64 (Windows64)
Compiler	Microsoft Visual C++

The results of the SIL simulation can be seen in Table 5.5.

Table 5.5: Results from SIL Simulation: Inference Time and Memory Usage

Metric	**Value**	**Unit**
Inference Time	0.07	ms
Memory Usage	2.4	KB

The results of the SIL simulation are promising; however, it should be emphasized that these measurements were obtained from a simulation running on a Windows machine and do not reflect the actual performance that would be observed when running on the VCU.

6

Discussion

The following chapter presents the results of the various ML models developed during the project. Both traditional models and artificial neural networks are compared in terms of predictive performance, model size, and practical feasibility for integration in a BEV environment. In addition, the final model selection is motivated by these findings, and the inference time is briefly evaluated to assess the potential for real-world deployment.

6.1 Traditional Machine Learning Models

The results of the five different traditional ML algorithms consist of PR, SVR, kNN, RF, and XGBoost across multiple targets. Overall, it can be observed from Table 5.2 that XGBoost achieves the lowest MAE, indicating the best overall performance among the models, while RF has the largest model size.

PR showed better performance since it can capture more complex patterns. However, the risk of poor generalization increases with a higher degree. In addition, the model size of PR is correlated with the degree of the polynomial. SVR achieved reasonable prediction accuracy, but it is computationally expensive and slower to train compared to other models, particularly with large datasets. Meanwhile, kNN has a large model size and poor performance because it lacks generalization and is sensitive to local variance. RF produced the largest model, as it generated numerous trees to predict target variables using a bagging technique. Although it is an ensemble learning, it showed high errors in low-value ranges, which may be from insufficient tree diversity. XGBoost performed robustly, which is due to its ability to handle non-linear relationships and prevent a model from overfitting with the ability of regularization, but this comes at the cost of a large model size. The model will be developed separately to predict each target variable, allowing for the prediction of multi-output variables.

Even though performance metrics are essential factors in developing a model to predict KPIs in thermal management, other aspects, including model size and compatibility with deployment in the existing environment, are equally important. In practical settings, it is a challenge to integrate excessively large models into the systems. Given the substantial size of trained models with low prediction error in traditional ML, such as RF or XGBoost, direct embedding of the existing system may not be feasible. This became evident early in the project, which is why the

results for the traditional models are relatively limited. Most of the development and evaluation effort was focused on the ANNs.

6.2 Artificial Neural Network models

The results of all three ANN models are overall impressive. They achieve low normalized MAE for the majority of the targets. Although the exact numerical values are difficult to interpret due to normalization, it is evident that the prediction errors are minor. This is further supported by the prediction-versus-actual value plots, where nearly all predictions on the test set fall within acceptable bounds. Before discussing the presented results, an initial question arises. Is the comparison between the three models fair?

As the storage sizes of the models differ, a direct comparison of MAE and prediction accuracy may be somewhat imbalanced. As shown in Table 5.3, the Separated MLP model has a storage size of 25.42 KB, which is smaller than that of the other two models. While this may initially appear to provide an unfair advantage, it is essential to note that the Separated MLP model generates predictions using two independent models. This design may impact inference time negatively in a production environment.

Additionally, model size is influenced by selecting hidden sizes based on powers of two. For example, increasing the number of neurons per layer from 32 to 64 can significantly increase the total model size. As a result, achieving an exactly fair comparison of storage requirements across architectures is difficult. With these considerations, the evaluation was deemed to maintain a reasonably fair and consistent comparison between models.

Moving on to the discussion about the results. While most of the results have been deemed satisfactory, the violin and bar plots reveal that specific targets appear more challenging for the models. Specifically, Targets 3, 5, 8, 9, and 16 experience higher MAE values, suggesting potential difficulties in prediction. But, a closer look at the prediction vs actual value plots for Targets 9 and 16 (found in the Appendix) shows that the higher MAE does not necessarily indicate poor performance; the predictions remain within the defined acceptable limits. While this does not apply to Targets 3, 5, and 8, the poor performance in these cases can be attributed to the specific range of actual values in which the inaccurate predictions occur. All of the non-satisfactory predictions are concentrated at the lower end of the numerical range, which appears to be more challenging for the models to predict accurately. It should also be noted that these predictions are considered non-satisfactory primarily based on the percentage-based color coding, rather than exceeding the absolute error threshold indicated by the dashed line. One possible explanation for this behavior is the choice of loss function used during model training. While the exact chosen loss function cannot be disclosed, this factor most likely contributed to the performance in the lower value range.

Another vital factor to consider is the range of actual values that the TMS would experience in a real-world driving cycle. It can be noted that the problematic lower

end of the range in the test set represents conditions that are rare in a real-world scenario. While driving, these cases would most likely not appear.

When comparing the three ANN models, the MLP and Multi-Task Learning models show similar performance, and they both outperform the Separated MLP model overall. This is interesting, as the development of the Separated MLP model was based on the hypothesis that the flow and pressure targets have different characteristics and that separating these might yield more accurate results; however, the results indicate the opposite. An ANN model that predicts both the flow and pressure targets simultaneously appears to perform better. A possible explanation of this could be the real-world physical relationship between flow and pressure. The interconnections in a fully connected network, such as the MLP, may, to some degree, capture this relationship, which aids in the predictions. It is also interesting to note that while the MLP and Multi-Task Learning models perform similarly and well, each has its respective pros. The MLP has a narrower range of errors for its predictions overall, but the Multi-Task Learning model outperforms it on lower numerical values.

6.3 Model Selection

When comparing the two categories of ML models, traditional algorithms and ANNs, it is evident that the ANNs outperform the traditional models for this project. This is true in both terms of accuracy and storage size, and especially when combining the two. However, traditional models offer an advantage in that they provide a certain degree of interpretability. As briefly discussed in the Introduction chapter, when applying ML models to physical systems, it is beneficial to have transparency in the decision-making process, particularly in systems such as the TMS of a BEV.

Although ANNs lack transparency, their superior performance and suitability for integration into embedded environments offer a significant advantage that cannot be overlooked. When determining which of the three developed ANN models to proceed with, the choice ultimately came down to the MLP and the Multi-Task Learning model. In this case, the MLP model was selected. This decision was based on its overall narrower spread in predicted versus actual values. While the Multi-Task Learning model performed slightly better at lower numerical values, the narrower spread was considered more important. Furthermore, the MLP model has a much simpler architecture, which makes it more favourable from a reusability and integration standpoint within Volvo Cars.

6.4 Inference Time

Moving on from the accuracy and storage size metrics, the inference time of the selected MLP model was also evaluated. The results appear promising. However, specific performance goals cannot be disclosed due to confidentiality. It should be emphasized once again that these results are based on a simulation and do not reflect the actual inference time that would be observed on the VCU. The real-world

inference time is expected to be longer than the 0.07 ms measured in the simulation; whether this would be acceptable in a production setting remains undetermined and needs further assessment.

7

Conclusion

This chapter presents the main conclusions drawn from the project and reflects on the outcome of the developed machine learning-based virtual sensor for thermal management. It also outlines potential areas for future work that could help improve and further validate the proposed approach.

7.1 Conclusion

In this project, ML techniques were explored to develop a computationally efficient model for predicting KPIs of the TMS of a BEV. The problem was formulated as a multi-output regression task, aiming to predict multiple target variables, including pressure and flow under steady-state conditions. A total of five traditional ML models, PR, SVR, kNN, RF, and XGBoost, were evaluated based on prediction accuracy and model size. Among these, XGBoost provided the best predictive performance but at the cost of a large model size, making integration into the system less feasible. RF resulted in an even larger model, which led to similar difficulties in hardware integration.

In contrast, three ANN models were developed and evaluated: the MLP model, the Separated MLP model, and the Multi-Task Learning model. All three demonstrated strong predictive performance with significantly smaller model sizes compared to the traditional models. The Separated MLP and the Multi-Task Learning model were developed based on the assumption that pressure and flow have different numerical and physical characteristics. However, this did not lead to improved performance compared to the standard MLP. The MLP model was selected due to its narrower error range and simpler architecture, which makes it more suitable for integration into the BEV's embedded systems. While ANN models generally lack the interpretability of traditional approaches, their higher or similar accuracy and smaller storage size makes them suitable for the purpose as a virtual sensor in a BEV.

The purpose of the model is to serve as a virtual sensor in areas of the BEV where physical sensors are absent. By accurately predicting KPIs, the model can provide reliable estimations where needed. This alternative data-driven approach provides a more flexible, modular, and cost-effective solution for thermal management software in BEVs.

In addition to the promising accuracy and storage size, the inference time of the

selected MLP model was also evaluated. The results were satisfactory, with an inference time of 0.07 ms and a memory usage of 2.4 KB. However, again, it should be noted that these measurements were obtained in a simulation environment and may not reflect actual performance on the VCU. Whether the real-world inference time meets the production requirements remains to be determined.

This project demonstrates the viability of using ML models as virtual sensors, showing that accurate and compact solutions can be developed for integration into embedded automotive systems. The results highlight that ML, particularly ANNs, offers a promising approach to modelling the TMS in BEVs. The developed models demonstrate strong predictive performance, confirming the feasibility of ML-based virtual sensors in future TMS softwares and indicating potential for integration into the VCU. Overall, this project illustrates how data-driven methods can complement traditional engineering approaches, paving the way for more efficient and cost-effective TMS software.

7.2 Future Work

While the results of this project are promising, several areas remain open for further exploration to improve the performance and applicability of the proposed virtual sensor model. One important direction is the investigation of custom loss functions tailored to the specific characteristics of the target variables. In particular, low flow values have proven more difficult to predict accurately, and a loss function that emphasizes performance in these regions could improve overall model accuracy and reliability.

Another key area is the integration of the model within the broader system context. Further work is needed to evaluate how the selected MLP model interacts with other components in the TMS. Simulating the model in combination with a controller would allow for assessment of system-level response and control dynamics. In addition, the model should be tested under realistic operating conditions. Simulating a dynamic or transient real-world drive cycle. This would help assess the model's robustness beyond the steady-state scenarios used in this project.

A critical step toward practical deployment is evaluating the real-time inference time and storage requirements on actual vehicle hardware. Although simulation results suggest that the model is efficient, its actual performance on the VCU must be verified to determine whether it meets the constraints of the embedded environment. Related to this is the question of training data. In this project, the model was trained using simulation data generated in GT-SUITE. It would be interesting to explore how the model performs when trained on real-world measurement data from a physical TMS test rig. This could offer a more accurate reflection of system dynamics and further validate the models practicality.

Finally, further tuning of hyperparameters remains a potential area for optimization. While the chosen model already demonstrate strong performance, achieving the highest possible accuracy was not the objective of this project. Performing a more

comprehensive and detailed search could yield additional improvements in accuracy versus storage size and inference time.

These future investigations will be essential for advancing the model from a proof-of-concept stage to a fully deployable component within a BEV's TMS.

Bibliography

[1] International Energy Agency (IEA), *Global ev outlook 2024: Trends in electric cars*, https://www.iea.org/reports/global-ev-outlook-2024/trends-in-electric-cars, Accessed: Dec. 9, 2024.

[2] M. Wawzyniak and A. Wiebelt, "Thermal management for electrified vehicles," *MTZ Worldw*, vol. 77, pp. 38–43, 2016. DOI: 10.1007/s38313-016-0026-1.

[3] A. Olabi, H. M. Maghrabie, O. H. K. Adhari, *et al.*, "Battery thermal management systems: Recent progress and challenges," *International Journal of Thermofluids*, vol. 15, p. 100 171, 2022, ISSN: 2666-2027. DOI: https://doi.org/10.1016/j.ijft.2022.100171. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666202722000350.

[4] J. A. B. Ccoa, B. Strauss, G. Mitic, and A. Lindemann, "Investigation of temperature sensitive electrical parameters for power semiconductors (igbt) in real-time applications," in *PCIM Europe 2014; International Exhibition and Conference for Power Electronics, Intelligent Motion, Renewable Energy and Energy Management*, Nuremberg, Germany, 2014, pp. 1–9.

[5] GTI Systems, *Vehicle thermal management simulation*, https://www.gtisoft.com/vehicle-thermal-management-simulation/, Accessed: Dec. 9, 2024.

[6] Statista, *Artificial intelligence - machine learning worldwide*, https://www.statista.com/outlook/tmo/artificial-intelligence/machine-learning/worldwide, Accessed: Dec. 9, 2024.

[7] K. Garud, J.-W. Han, S.-G. Hwang, and M.-Y. Lee, "Artificial neural network modeling to predict thermal and electrical performances of batteries with direct oil cooling," *Batteries*, vol. 9, p. 559, Nov. 2023. DOI: 10.3390/batteries9110559.

[8] P. Dubey, G. Pulugundla, and A. K. Srouji, "Direct comparison of immersion and cold-plate based cooling for automotive li-ion battery modules," *Energies*, vol. 14, no. 5, 2021, ISSN: 1996-1073. DOI: 10.3390/en14051259. [Online]. Available: https://www.mdpi.com/1996-1073/14/5/1259.

[9] M. Islam, G. Chen, and S. Jin, "An overview of neural network," vol. 5, p. 05, Jun. 2019. DOI: 10.11648/j.ajnna.20190501.12.

[10] A. Al Miaari and H. M. Ali, "Batteries temperature prediction and thermal management using machine learning: An overview," *Energy Reports*, vol. 10, pp. 2277–2305, 2023, ISSN: 2352-4847. DOI: https://doi.org/10.1016/j.egyr.2023.08.043. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S235248472301185X.

[11] *Steady State - an overview | ScienceDirect Topics*. [Online]. Available: https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/steady-state (visited on 02/11/2025).

[12] *GT-Suite*, Mar. 2025. [Online]. Available: https://www.gtisoft.com/gt-suite/.

[13] *What is digital-twin technology? | McKinsey*. [Online]. Available: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-digital-twin-technology (visited on 05/21/2025).

[14] A. Maiorino, C. Cilenti, F. Petruzziello, and C. Aprea, "A review on thermal management of battery packs for electric vehicles," *Applied Thermal Engineering*, vol. 238, p. 122 035, Feb. 2024, ISSN: 1359-4311. DOI: 10.1016/j.applthermaleng.2023.122035. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1359431123020641 (visited on 02/10/2025).

[15] S. Chowdhury, L. Leitzel, M. Zima, *et al.*, "Total Thermal Management of Battery Electric Vehicles (BEVs)," May 2018, pp. 2018–37–0026. DOI: 10.4271/2018-37-0026. [Online]. Available: https://www.sae.org/content/2018-37-0026/ (visited on 02/10/2025).

[16] B. Holcomb, *How Simulation Is Used To Design ICE vs. Battery Electric Vehicle Thermal Management Systems*. [Online]. Available: https://www.gtisoft.com/blog-post/ice-vs-xev-automotive-thermal-management-simulation/ (visited on 02/10/2025).

[17] L. He, H. Jing, Y. Zhang, P. Li, and Z. Gu, "Review of thermal management system for battery electric vehicle," *Journal of Energy Storage*, vol. 59, p. 106 443, Mar. 2023, ISSN: 2352-152X. DOI: 10.1016/j.est.2022.106443. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352152X2202432X (visited on 02/10/2025).

[18] *How does Tesla Model Y Heat Pump Work*. [Online]. Available: https://www.youtube.com/watch?v=Ol9iv23An7I (visited on 02/10/2025).

[19] K. Ogura and M. L. Kolhe, "Battery technologies for electric vehicles," en, in *Electric Vehicles: Prospects and Challenges*, Elsevier, 2017, pp. 139–167, ISBN: 9780128030219. DOI: 10.1016/B978-0-12-803021-9.00004-5. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780128030219000045 (visited on 02/10/2025).

[20] A. Wahl, C. Wellmann, C. Monissen, and J. Andert, "Active temperature control of electric drivetrains for efficiency increase," *Applied Energy*, vol. 338, p. 120 887, May 2023, ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2023.120887. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261923002519 (visited on 02/10/2025).

[21] X. Guan and H. Burton, "Bias-variance tradeoff in machine learning: Theoretical formulation and implications to structural engineering applications," *Structures*, vol. 46, pp. 17–30, 2022, ISSN: 2352-0124. DOI: https://doi.org/10.1016/j.istruc.2022.10.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352012422009018.

[22] C. Bishop, *Pattern Recognition and Machine Learnning*, English. Springer, Aug. 2006, ISBN: 978-0387310732.

[23] *Implementation of Polynomial Regression*, en-US, Oct. 2018. [Online]. Available: https://www.geeksforgeeks.org/python-implementation-of-polynomial-regression/ (visited on 06/05/2025).

[24] *K-Nearest Neighbor(KNN) Algorithm*, en-US, Apr. 2017. [Online]. Available: https://www.geeksforgeeks.org/k-nearest-neighbours/ (visited on 06/05/2025).

[25] A. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, pp. 199–222, Aug. 2004. DOI: 10.1023/B%3ASTCO.0000035301.49549.88.

[26] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification And Regression Trees*, en, 1st ed. Routledge, Oct. 2017, ISBN: 9781315139470. DOI: 10.1201/9781315139470. [Online]. Available: https://www.taylorfrancis.com/books/9781351460491 (visited on 06/05/2025).

[27] *Random Forest Regression in Python*, en-US, Jun. 2019. [Online]. Available: https://www.geeksforgeeks.org/random-forest-regression-in-python/ (visited on 06/05/2025).

[28] J. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, Nov. 2000. DOI: 10.1214/aos/1013203451.

[29] N. G. Paterakis, E. Mocanu, M. Gibescu, B. Stappers, and W. van Alst, "Deep learning versus traditional machine learning methods for aggregated energy demand prediction," in *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, 2017, pp. 1–6. DOI: 10.1109/ISGTEurope.2017.8260289.

[30] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, pp. 579–588, 2009. [Online]. Available: https://www.researchgate.net/publication/228340819_Multilayer_perceptron_and_neural_networks.

[31] R. Rojas, "The backpropagation algorithm," in *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 149–182, ISBN: 978-3-642-61068-4. DOI: 10.1007/978-3-642-61068-4_7. [Online]. Available: https://doi.org/10.1007/978-3-642-61068-4_7.

[32] *Hidden Layer*, May 2019. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning (visited on 03/24/2025).

[33] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, Sep. 2022, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2022.06.111. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222008426 (visited on 03/24/2025).

[34] Y. Chen, L. Li, W. Li, Q. Guo, Z. Du, and Z. Xu, "Chapter 2 - Fundamentals of neural networks," in *AI Computing Systems*, Y. Chen, L. Li, W. Li, Q. Guo, Z. Du, and Z. Xu, Eds., Morgan Kaufmann, Jan. 2024, pp. 17–51, ISBN: 9780323953993. DOI: 10.1016/B978-0-32-395399-3.00008-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323953993000081 (visited on 03/24/2025).

[35] *Activation functions in Neural Networks*, en-US, Jan. 2018. [Online]. Available: https://www.geeksforgeeks.org/activation-functions-neural-networks/ (visited on 03/24/2025).

[36] *Loss Functions in Deep Learning*, en-US, Jul. 2024. [Online]. Available: https://www.geeksforgeeks.org/loss-functions-in-deep-learning/ (visited on 03/18/2025).

[37] L. Sadouk, T. Gadi, and E. H. Essoufi, "Robust Loss Function for Deep Learning Regression with Outliers," en, in *Embedded Systems and Artificial Intelligence*, V. Bhateja, S. C. Satapathy, and H. Satori, Eds., Singapore: Springer, 2020, pp. 359–368, ISBN: 9789811509476. DOI: 10.1007/978-981-15-0947-6_34.

[38] M. H. Hassan, *Using Grid Search For Hyper-Parameter Tuning*, en, Nov. 2023. [Online]. Available: https://medium.com/@hammad.ai/using-grid-search-for-hyper-parameter-tuning-bad6756324cc (visited on 05/21/2025).

[39] D. S. Wizards, *A Guide to Data Splitting in Machine Learning*, en, Nov. 2022. [Online]. Available: https://medium.com/@datasciencewizards/a-guide-to-data-splitting-in-machine-learning-49a959c95fa1 (visited on 05/21/2025).

[40] *Importance of Feature Scaling*, en. [Online]. Available: https://scikit-learn/stable/auto_examples/preprocessing/plot_scaling_importance.html (visited on 03/27/2025).

[41] L. Wang, Z. Zhang, X. Zhang, X. Zhou, P. Wang, and Y. Zheng, "Chapter one - a deep-forest based approach for detecting fraudulent online transaction," in *AI and Cloud Computing*, ser. Advances in Computers, A. R. Hurson and S. Wu, Eds., vol. 120, Elsevier, 2021, pp. 1–38. DOI: https://doi.org/10.1016/bs.adcom.2020.10.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245820300851.

[42] *StandardScaler*, en. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (visited on 03/27/2025).

[43] K. Yeager, *LibGuides: SPSS Tutorials: Pearson Correlation*, en. [Online]. Available: https://libguides.library.kent.edu/SPSS/PearsonCorr (visited on 03/21/2025).

[44] [Online]. Available: https://real-statistics.com/correlation/basic-concepts-correlation/ (visited on 03/21/2025).

[45] L. Wilkinson and M. Friendly, "The History of the Cluster Heat Map," en, *The American Statistician*, vol. 63, no. 2, pp. 179–184, May 2009, ISSN: 0003-1305, 1537-2731. DOI: 10.1198/tas.2009.0033. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1198/tas.2009.0033 (visited on 03/21/2025).

[46] D. B. S. and, "Use of dummy variables in regression equations," *Journal of the American Statistical Association*, vol. 52, no. 280, pp. 548–551, 1957. DOI: 10.1080/01621459.1957.10501412. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1957.10501412. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1957.10501412.

[47] Author not stated, *One Hot Encoding in Machine Learning*, en-US, Jun. 2019. [Online]. Available: https://www.geeksforgeeks.org/ml-one-hot-encoding/ (visited on 03/27/2025).

[48] S. Rao, P. Poojary, J. Somaiya, and P. Mahajan, "A comparative study between various preprocessing techniques for machine learning," *International Journal of Engineering Applied Sciences and Technology*, vol. 5, no. 3, pp. 431–438, 2020, ISSN: 2455-2143.

[49] *TargetLink*, en. [Online]. Available: https://www.dspace.com/en/pub/home/products/sw/pcgs/targetlink.cfm (visited on 05/12/2025).

A

Appendix 1

A.1 Pearson Correlation

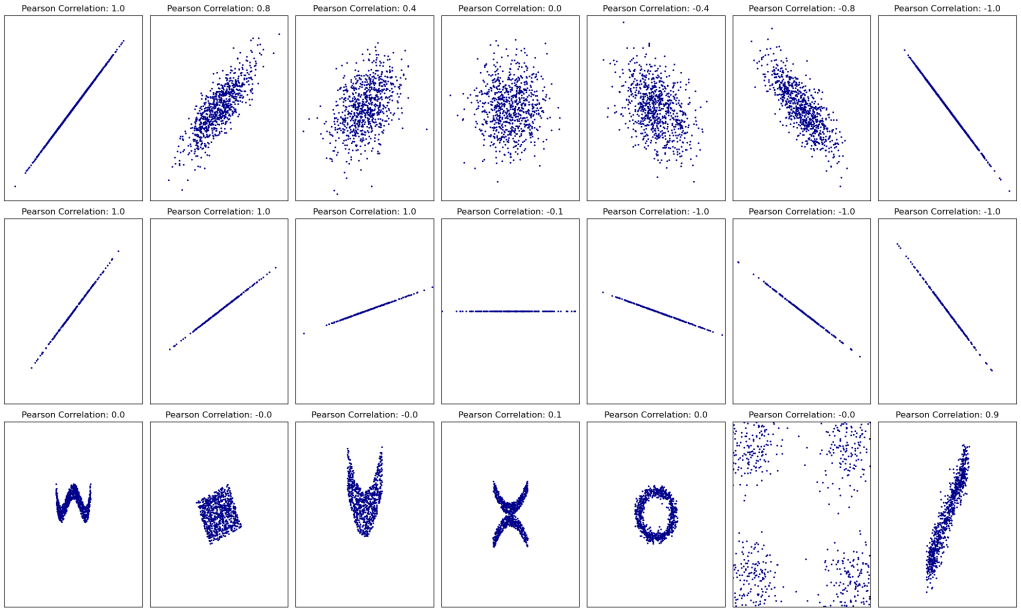


Figure A.1: Pearson correlations

A.2 Traditional machine learning algorithms learning curve

A.2.1 5th degree Polynomial Regression

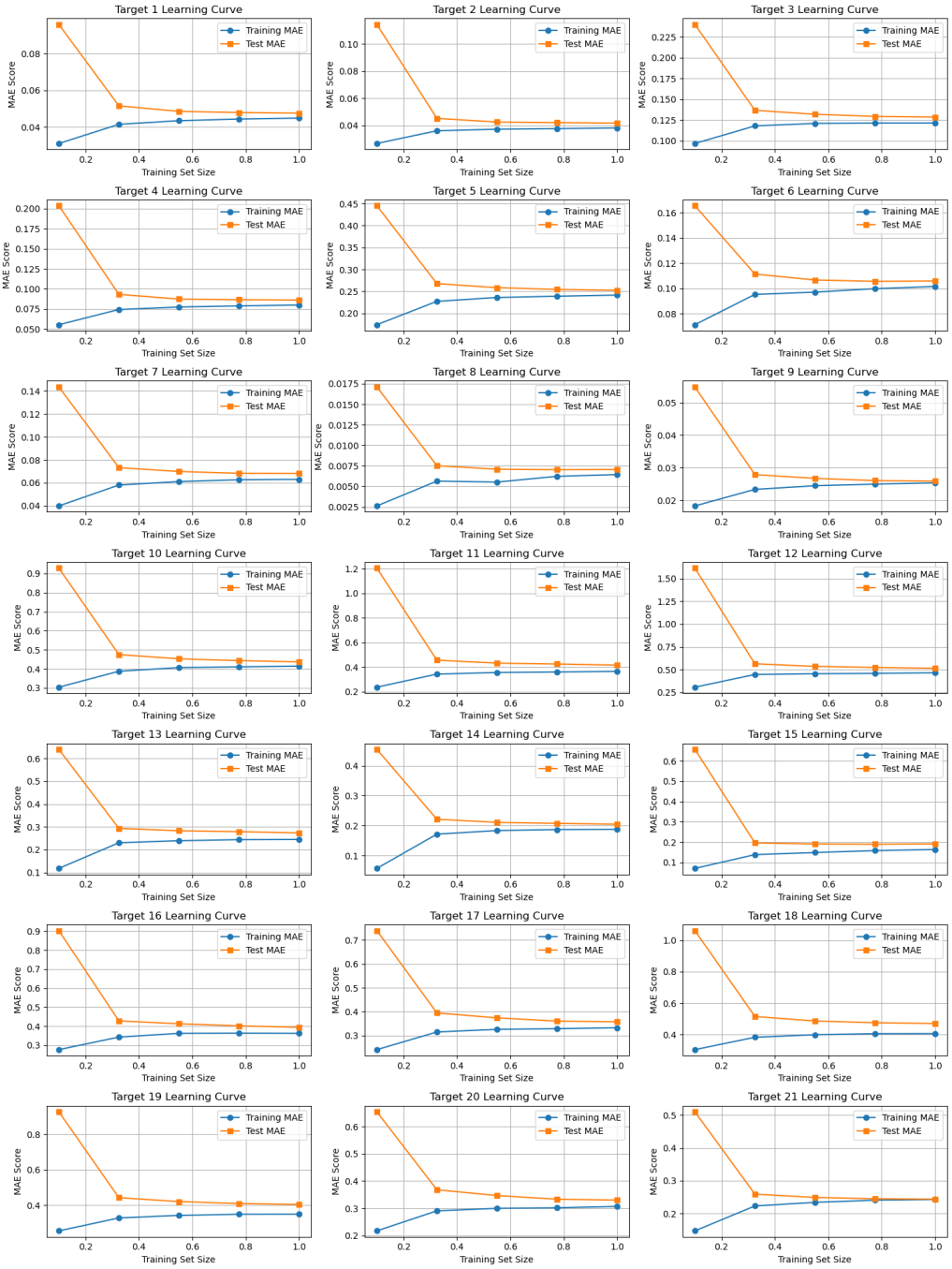


Figure A.2: Polynomial regression learning curve

A.2.2 Support Vector Regression

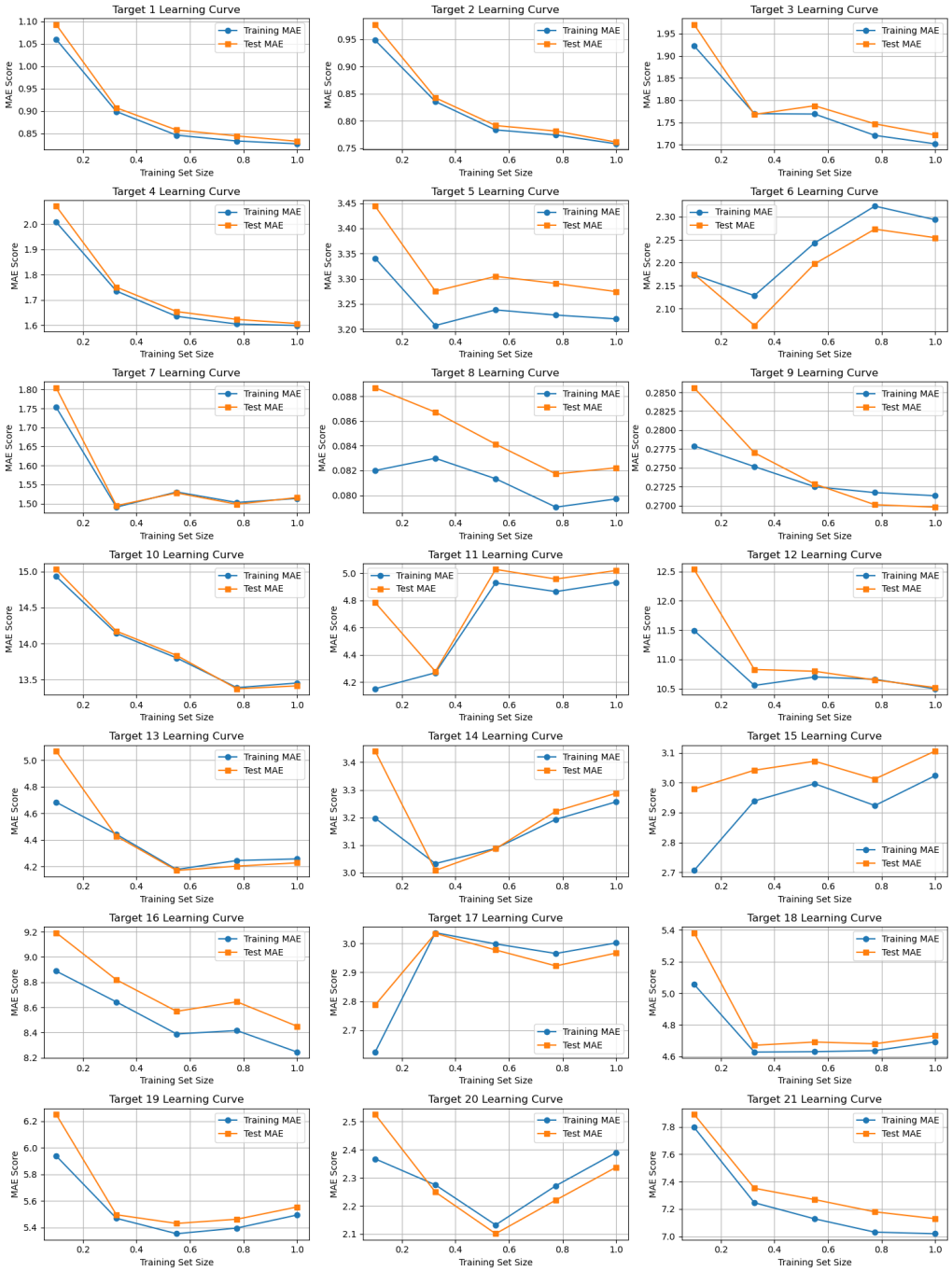


Figure A.3: Support Vector regression learning curve

A.2.3 k-Nearest Neighbor Regression

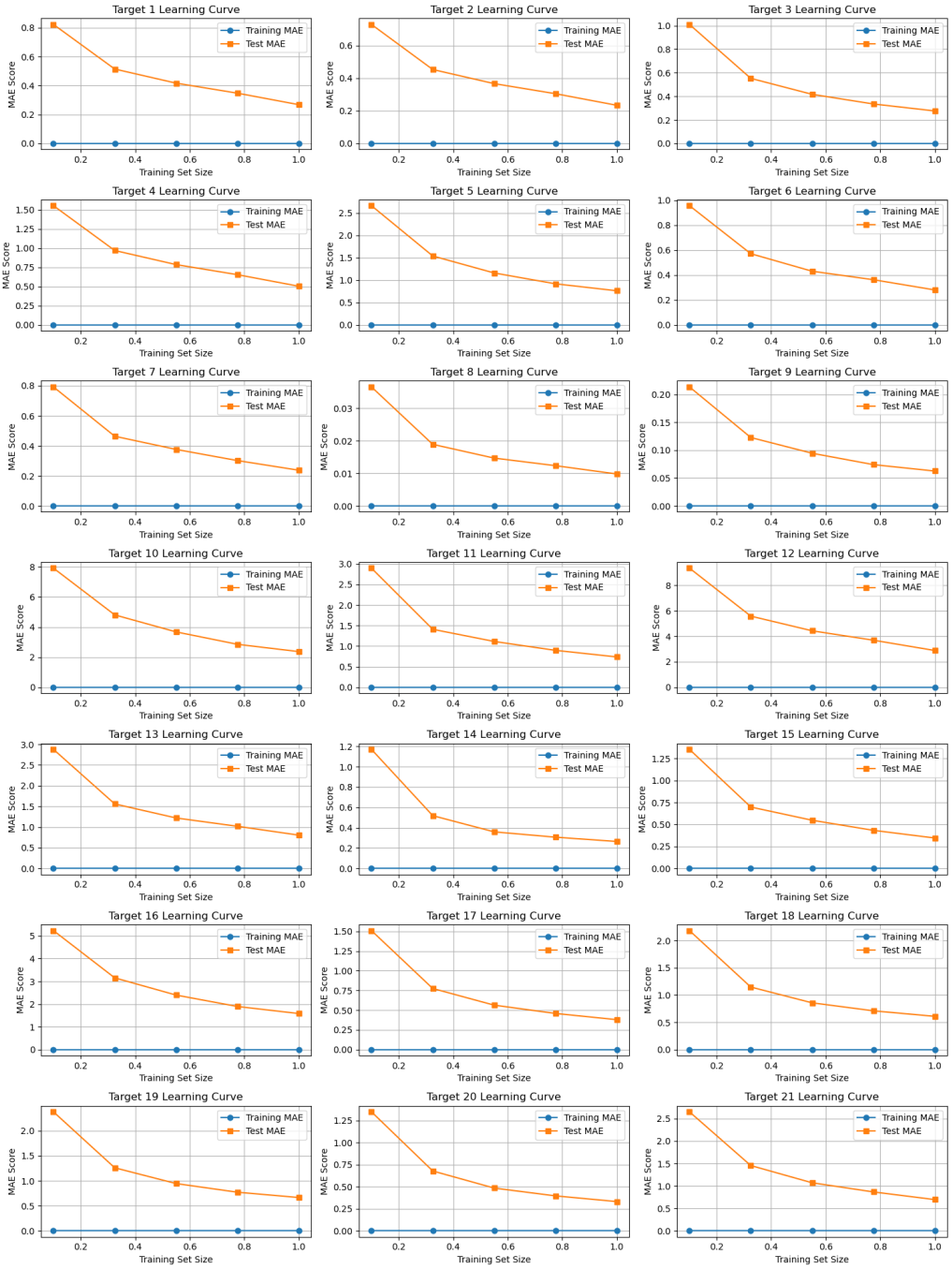


Figure A.4: kNN regression learning curve

A.2.4 XGboost Regression

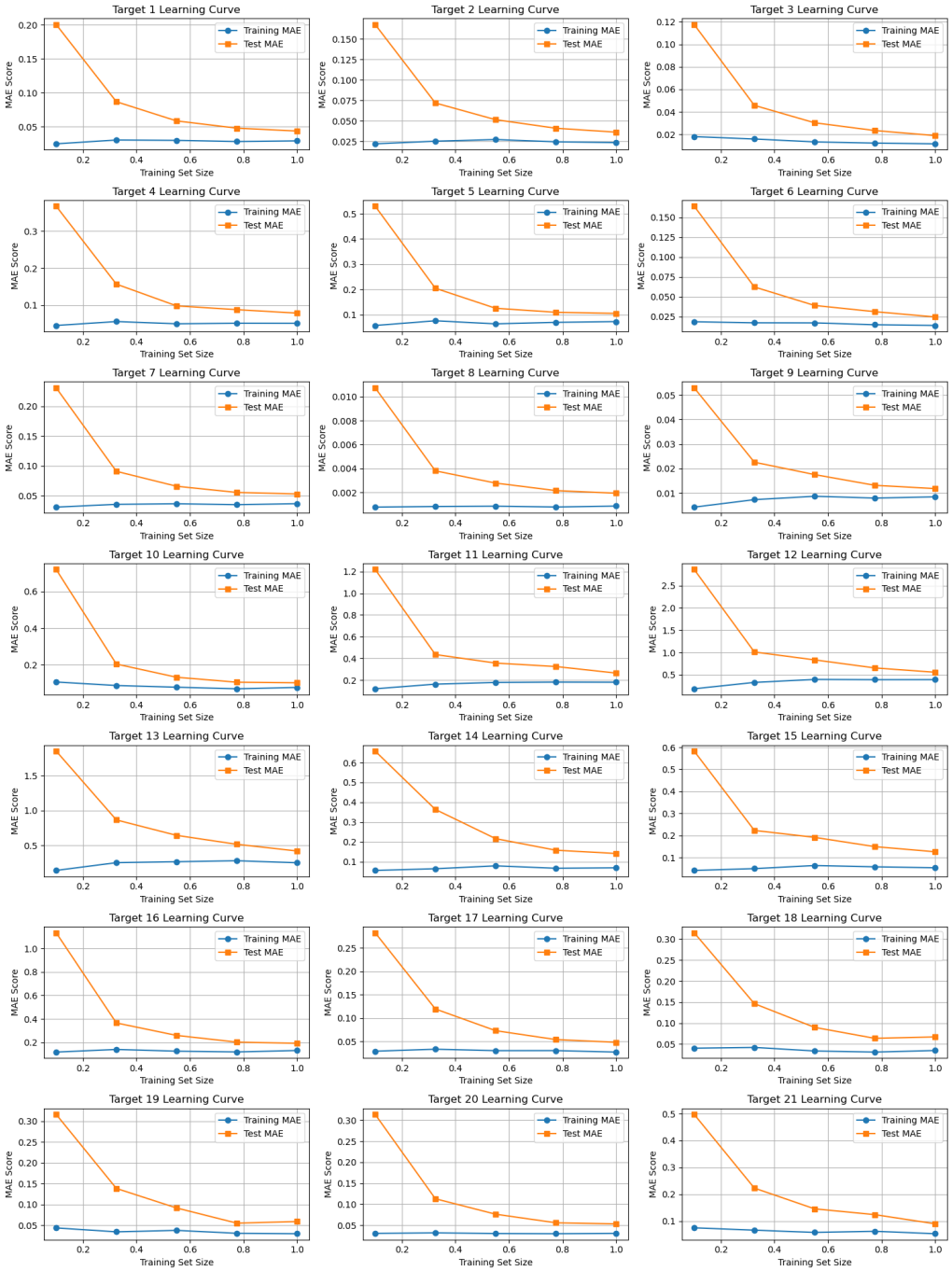


Figure A.5: XGBoost Regression learning curve

A.2.5 Random Forest Regression

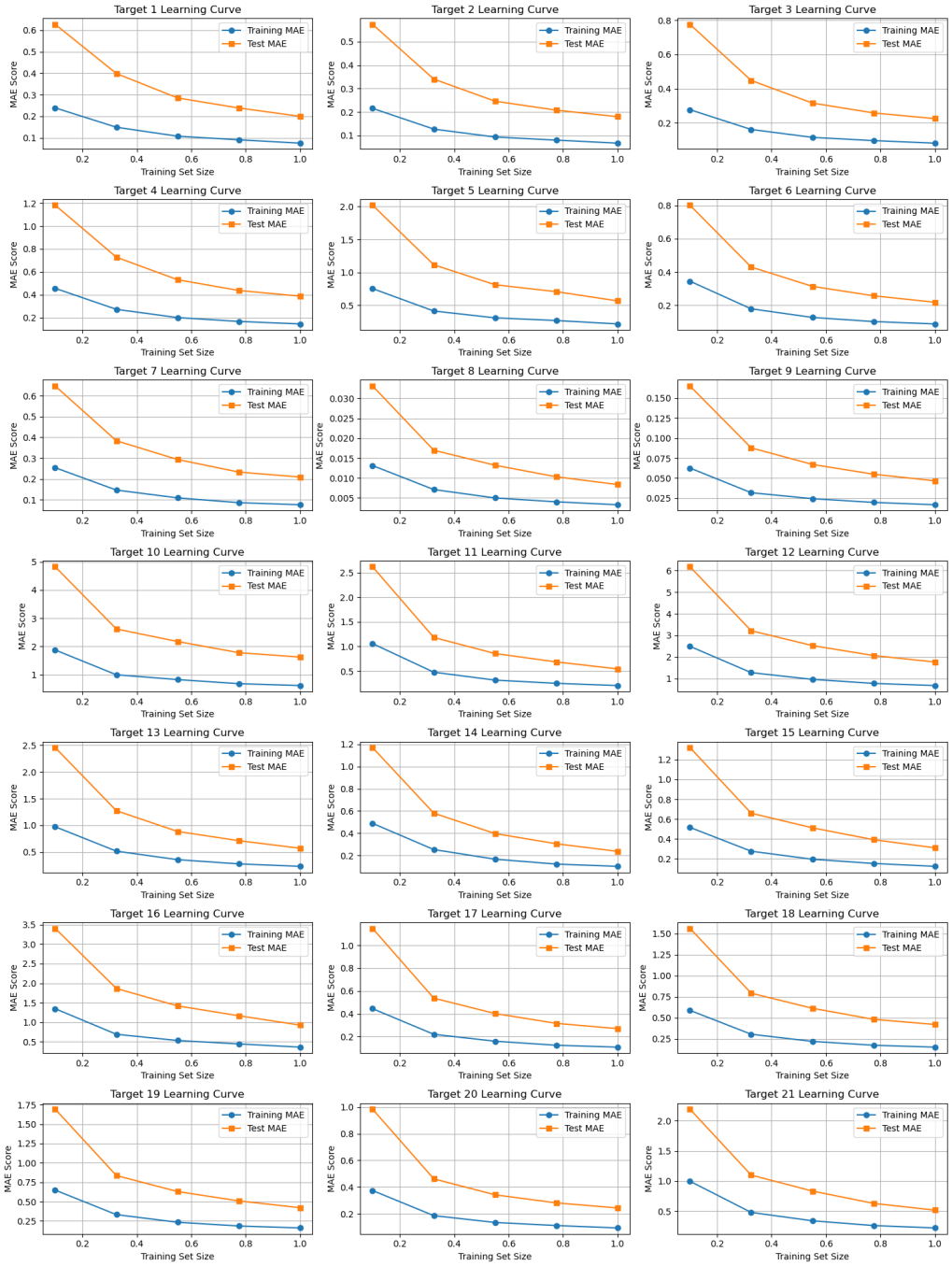


Figure A.6: Random Forest regression learning curve

A.3 Prediction vs Actual Value MLP

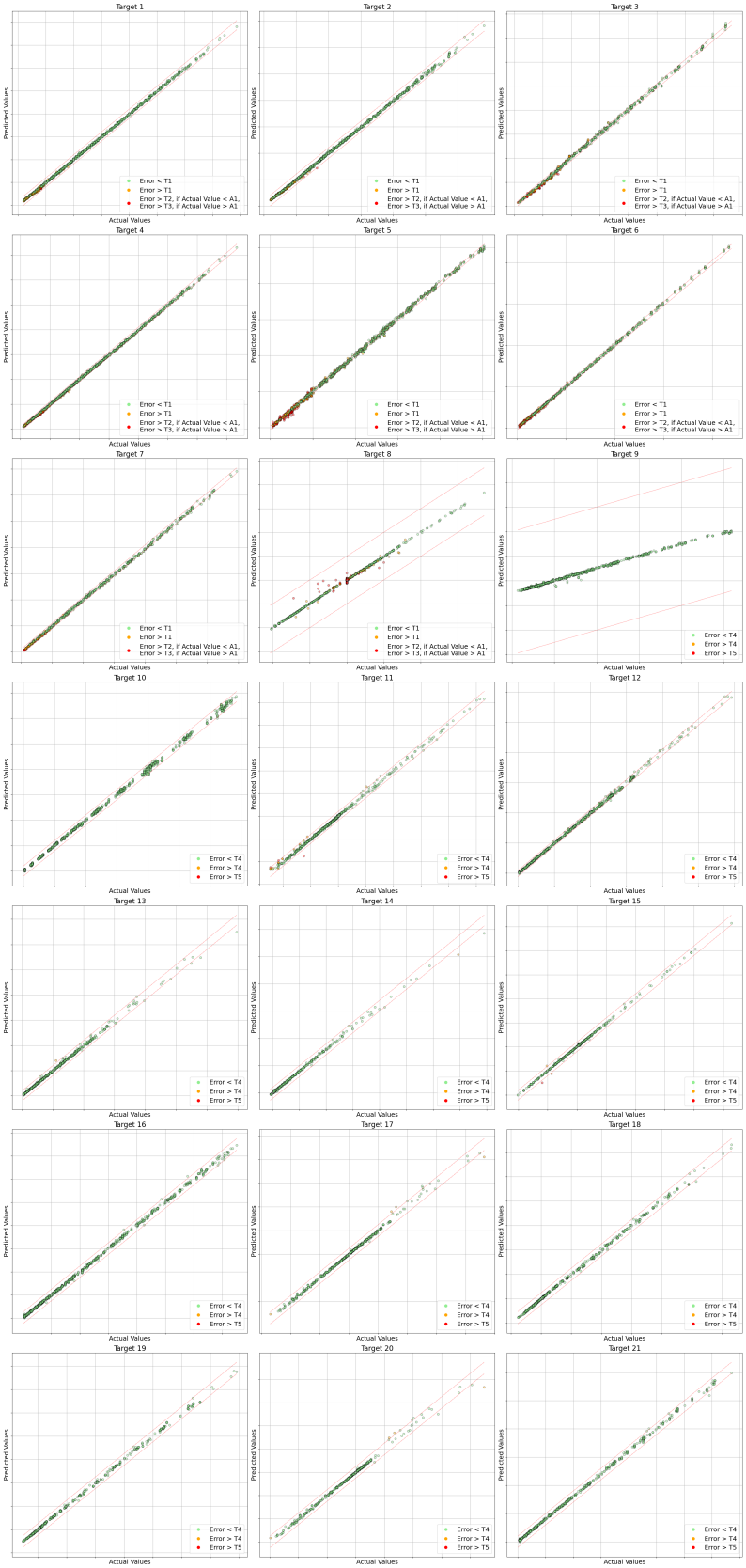


Figure A.7: Prediction vs actual for MLP

A.4 Prediction vs Actual value Multi-Task Learning MLP

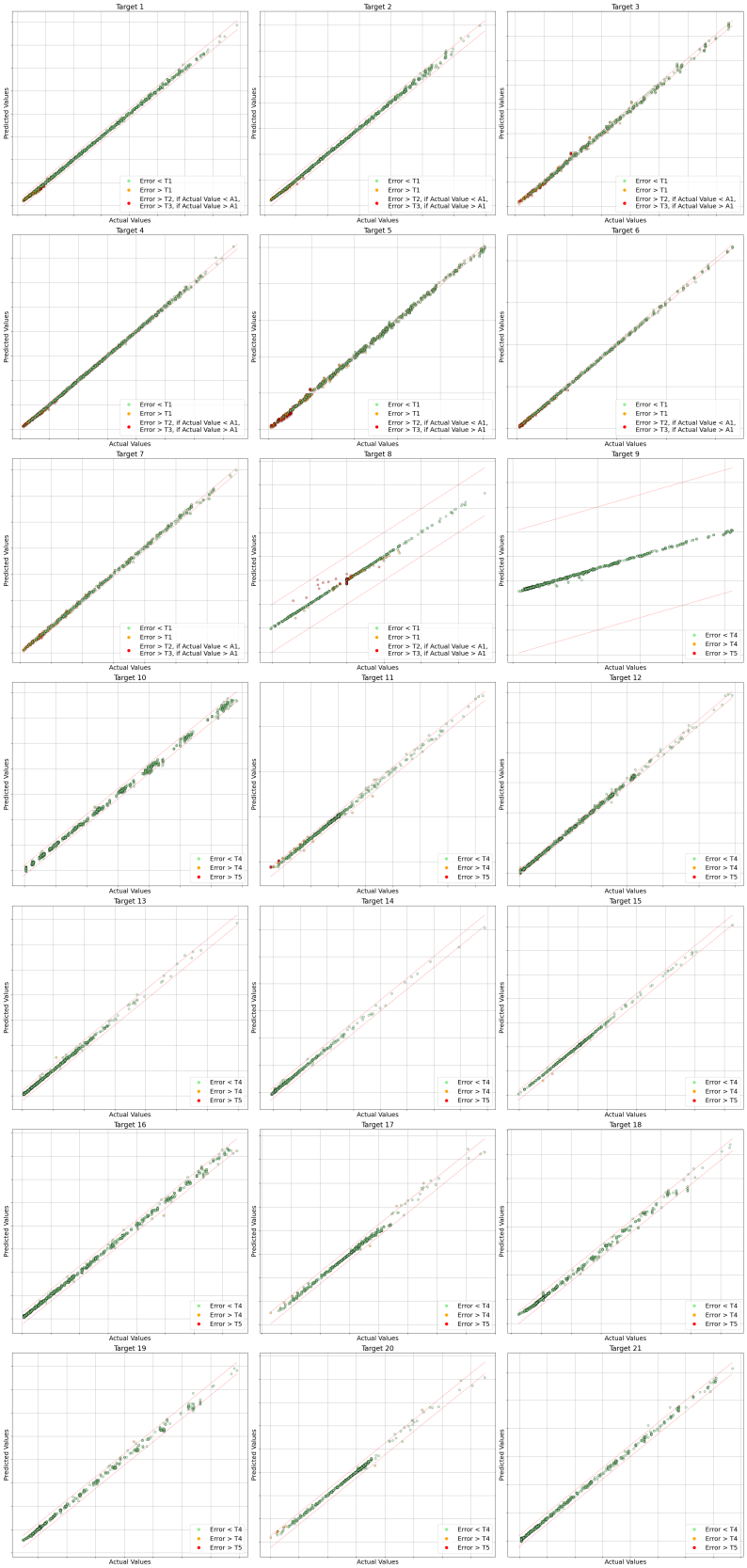


Figure A.8: Prediction vs actual for Multi-Task Learning MLP