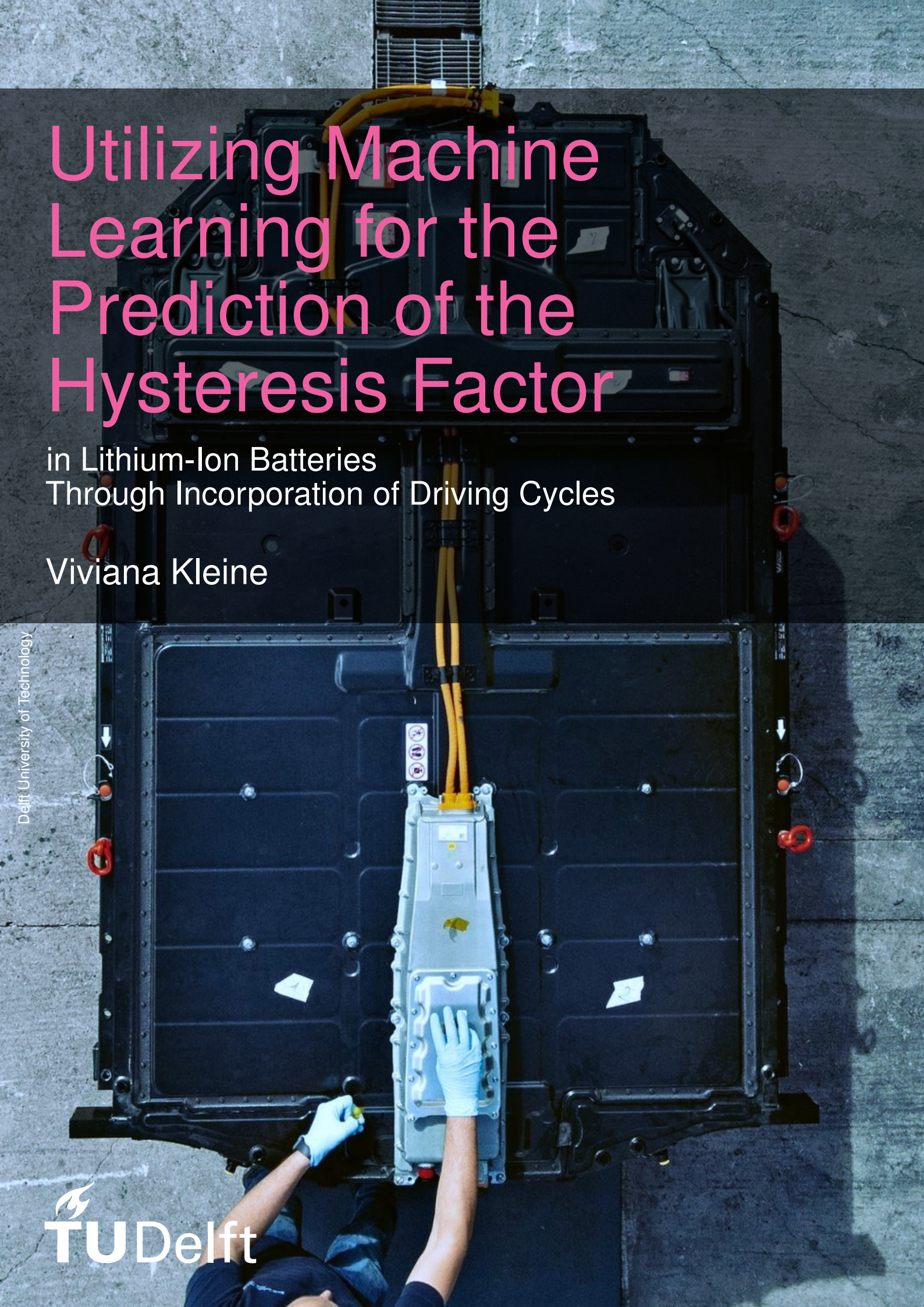
Utilizing Machine Learning for the Prediction of the Hysteresis Factor

in Lithium-Ion Batteries
Through Incorporation of Driving Cycles

Viviana Kleine



Delft University of Technology

TU Delft

Utilizing Machine Learning for the Prediction of the Hysteresis Factor

in Lithium-Ion Batteries Through Incorporation of Driving Cycles

by

Viviana Kleine

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday December 4, 2024 at 03:00 PM.

Student number:	5504953	
Project duration:	February 15, 2022 – December 4, 2024	
Thesis committee:	Prof. dr. P. Palensky,	TU Delft, committee chair
	Dr. J. L. Cremer,	TU Delft, supervisor
	Dr. ir. G. R. Chandra Mouli,	TU Delft, supervisor
	P. Gromotka	Porsche Engineering Services GmbH, daily supervisor

Cover:	Porsche AG / Rafael Kroetz and Luca Santini, Porsche Newsroom.
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Acknowledgments

First and foremost, I would like to extend my gratitude to my daily supervisor, Philipp Gromotka. His unwavering support, insightful guidance, and continuous encouragement throughout the development of this thesis have been invaluable. Philipp introduced me to embedded systems, patiently addressed my numerous questions, and ensured that I gained a comprehensive understanding not only of what was directly related to my thesis but also of the entire vehicle system.

I am also grateful to Thomas Rudolf, who dedicated time to helping me implement my machine learning models and resolve any technical challenges I encountered. Furthermore, I thank Adrian Eisenmann, whose feedback was instrumental in merging the realms of batteries and machine learning.

I would also like to express my appreciation to all the colleagues at Porsche Engineering I had the pleasure of working with for their support, openness, and willingness to help. Being part of such a cohesive and supportive team was a truly rewarding experience.

Moreover, my gratitude goes to my thesis committee at TU Delft, Jochen Cremer, Gautham Ram Chandra Mouli and Peter Palensky, for their constructive feedback and expert advice, which have significantly contributed to the refinement of my work. I am also thankful to the IEPG and DCE&S departments for their practical advice and valuable suggestions.

A special thanks to my friends, family, and my partner for their endless encouragement, patience, and motivation throughout my master's degrees. Your unwavering support has been the cornerstone of my success.

Viviana Kleine
Delft, November 2024

Abstract

The management of electric vehicle (EV) batteries involves accurately estimating their state-of-charge (SOC), which indicates remaining usable energy. Precise SOC estimation extends battery life, increases usable capacity, and enhances vehicle performance by allowing a greater depth of discharge without increasing battery weight or size. Traditional SOC estimation via Coulomb counting accumulates errors and requires correction using the Open-Circuit-Voltage (OCV). Modern silicon graphite anodes exhibit voltage hysteresis, making accurate SOC determination difficult. Introducing a hysteresis factor, ranging from -1 to 1, helps interpret OCV values within this hysteresis region.

This thesis proposes using a black box approach and machine learning models to predict the hysteresis factor for SOC correction more accurately. The research aims to enhance SOC accuracy in real-world EV environments, beyond laboratory settings. Improved SOC estimation through machine learning could significantly advance battery management systems (BMS), yielding both economic and environmental benefits by increasing EV efficiency. The study addresses the existing gaps in SOC estimation for voltage hysteresis, considering the vehicle as a system by incorporating driving cycles, BMS technical limitations, and cell chemistry. By improving EV reliability and efficiency, this research aims to promote broader acceptance and contribute to a sustainable future in personal transportation.

The analysis indicates that the designed model performs well with relevant and comprehensive training data, particularly from test bench data. However, there is room for improvement in generalising across more dynamic driving cycles. Additional measurements are needed at various temperatures within that range to improve the model's performance across a broader temperature range. The study also found that sufficient training data is crucial for optimal performance. Training a single model across multiple vehicle projects was not feasible due to varying cell chemistries. GRU models, where the input time series signals were split into intervals, were the most effective for predicting the hysteresis factor.

It has yet to be verified that the model can be applied to cells with higher silicon content and more pronounced hysteresis. Additionally, the model did account for ageing effects. In the future, other labelling techniques, such as the "stability criterion," could be included to improve the model's performance. This criterion assesses both the short-term and long-term stability of the SOC and capacity, respectively. Using this approach in test fleets to further train the existing models could improve the models significantly. This could lead to better predictions under varied conditions and with different cell types, ultimately improving the model's effectiveness for real-world applications.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
BMS	Battery management system
CART	Classification and regression trees
CPU	Central processing unit
DS	Cell data sheet
ECM	Equivalent circuit model
EIS	Electrochemical impedance spectroscopy
EV	Electric vehicle
GPU	Graphic processing unit
GRU	Gated recurrent unit
GITT	Galvanostatic intermittent titration technique
HEV	Hybrid Electric Vehicle
HPPC	Hybrid Pulse Power Characteristic
LFP	Lithium iron phosphate
Li	Lithium
LMO	Lithium ion manganese oxide
LTO	Lithium titanate oxide
L1	Lasso (regression)
L2	Ridge (regression)
LSTM	Long short-term memory
ML	Machine learning
NCA	Lithium nickel cobalt aluminium oxide
NMC	Lithium nickel manganese cobalt oxide
OCP	Open circuit potential
OCV	Open circuit voltage
PCA	Principal component analysis
RAM	Random access memory
RNN	Recurrent neural network
ROM	Read-only memory
SEI	Solid electrolyte interface
Si	Silicon
SOC	State-of-charge
UDDS	Urban Dynamometer Driving Schedule
WLTP	Worldwide Harmonised Light-Duty Vehicles Test Procedure

Symbols

Symbol	Definition	Unit
a	Factors	[-]
b	Bias	[-]
b_{hn}	Bias vector between hidden and hidden layer at candidate hidden state gate	[-]

Symbol	Definition	Unit
b_{hr}	Bias vector between hidden and hidden layer at reset gate	[-]
b_{hz}	Bias vector between input and hidden layer at candidate hidden state gate	[-]
C_b	Battery capacity	[Ah]
C_n	Nominal capacity	[Ah]
c	Constant	[-]
D	Total number of decision trees	[-]
δ	Dropout variable	[-]
f_d	Function representing the tree v	[-]
G_j	Parameters	[-]
H_j	Parameters	[-]
H_t	Hidden state at time step t	[-]
i	Current	[A]
I_j	Indices of data points corresponding to the jth leaf	[-]
k	Time step	[-]
k_1, k_2	Factors	[-]
m_1, m_2	Factors	[-]
$M(SOC)$	Maximum polarization of the SOC due to hysteresis	[%]
o	Output	[-]
P	Total number of samples	[-]
p	Parameters	[-]
Q	Charge throughput	[A]
$Q_{releasable}$	Releasable charge	[Ah]
q	Function mapping data points to leaves	[-]
r_i	Threshold of i-th operator	[-]
R_t	Reset gate value at time step t	[-]
sgn	Sign function	[-]
SOC	State of charge	[%]
t	Time step	[-]
T	Total number of leaves	[-]
u	Input	[-]
u_{ocp}	Open circuit potential	[V]
v_h	Hysteresis voltage	[V]
v_{OCV}	Open circuit voltage	[V]
w	Weight	[-]
W_{hn}	Weight matrix between hidden and hidden layer at candidate hidden state gate	[-]
W_{hr}	Weight matrix between hidden and hidden layer at reset gate	[-]
W_{hz}	Weight matrix between hidden and hidden layer at update gate	[-]
W_{in}	Weight matrix between input and hidden layer at candidate hidden state gate	[-]
W_{ir}	Weight matrix between input and hidden layer at reset gate	[-]
W_{iz}	Weight matrix between input and hidden layer at update gate	[-]
X_t	Input matrix at time step t	[-]
x_i	Input vector for sample i	[-]
y_i	Label for sample i	[-]
$\hat{y}_i$	Predicted value for sample i	[-]
ξ	Residual	[-]
$\epsilon\rho$	Noise, noise function	[-]

Symbol	Definition	Unit
α, β	Slope parameters	[-]
γ	Factor for the rate of decay	[-]
θ	Electrode stoichiometry	[-]
ψ	Hysteresis factor	[-]
τ	Quantile	[-]
ΔSOC	Change in SOC	[%]
Δv_h	Range of hysteresis voltage	[V]
η_b	Battery efficiency	[-]
ω	Complexity	[-]
$\tilde{H}_t$	Candidate hidden state at time step t	[-]
Z_t	Update gate value at time step t	[-]

List of Figures

List of Tables

1

Introduction

1.1. Background and Motivation

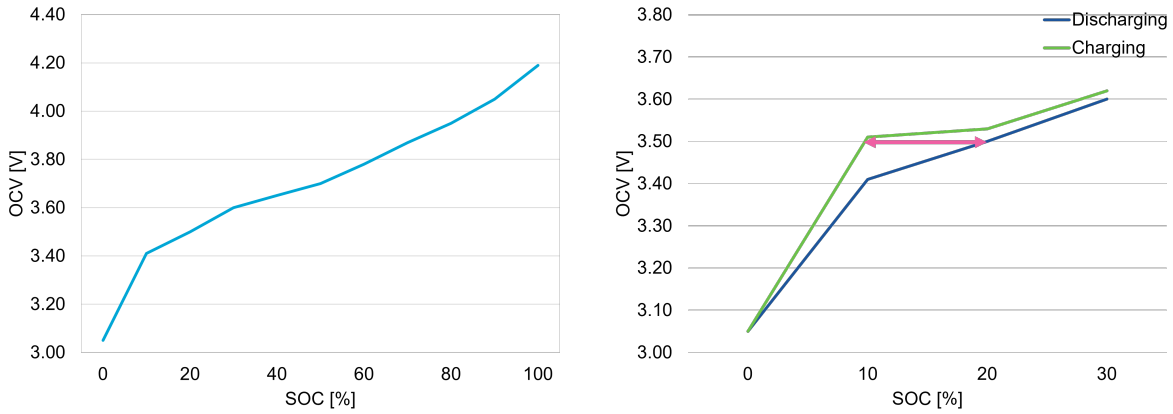
Batteries are one of the key components of electric vehicles (EVs). Central to the management of these batteries is the estimation of their state-of-charge (SOC), which offers an indication of the remaining usable battery energy. This combined with the overall efficiency of the vehicle and user driving behaviour determines the remaining range of the EV. Effective SOC estimation is crucial as it directly influences the performance of electric vehicles. In fact, with a more precise SOC estimation, the SOC boundaries set to protect the battery from ageing can be wider as the required buffer for SOC uncertainty is smaller. This allows for a greater depth of discharge and thus greater usable capacity.

The increased usable capacity of the battery offers several benefits. It can be used to achieve greater range without increasing battery weight or size, or it can be utilised to reduce the overall weight and size of the battery, thus lowering the cost of EVs. Considering the average battery pack price for EV applications in 2022, which was $138 per kWh [1], and assuming a battery capacity of 100 kWh, increasing the depth of discharge by 2% SOC can result in a cost reduction of $276.

If this additional 2% SOC is used to extend the vehicle's range, at an electricity consumption of 19.8 kWh/100 km (comparable to the Porsche Macan Electric), it will result in an increase of approximately 10 km in range. While this may seem insignificant at first glance, original equipment manufacturers (OEMs) are continually striving to improve their range, a critical indicator of vehicle performance for consumers. An additional 10 km may be crucial, as it can increase the range from, for example, 495 km to 505 km, making the latter more appealing to customers ("value attribution" surpassing key thresholds, such as 500 km). Therefore even small boosts in range can make EVs more attractive, especially considering that range anxiety remains a primary concern for potential EV buyers [2].

Traditionally, SOC has been estimated through a method known as Coulomb counting. However, given this method's tendency to accumulate errors over time, a corrective measure after each driving cycle is required to ensure accurate SOC values. A driving cycle refers to a 'fixed schedule of vehicle operation' [3]. It includes both charging and discharging operations, individually or in combination, at various vehicle speeds, including 0 km/h. This correction utilises the Open-Circuit-Voltage (OCV), the equilibrium voltage observed under zero-current conditions. Cell manufacturers provide reference tables that map the OCV to a single SOC value. An example of this is visualised in Figure 1.1a. After completing a driving cycle and allowing the battery sufficient time to relax to meet open-circuit conditions, the measured OCV can be utilised to refine the estimated SOC value.

To improve cell performance, novel cell technologies, specifically silicon graphite anodes, are increasingly being utilised in automotive applications [4]. Due to their high energy and power density, these advanced cells enable longer vehicle ranges and faster charging times. For instance, a silicon lithium-ion cell from the manufacturer Amprius claims to offer a 76% improvement in vehicle range [5]. It is important to note that range is influenced by several other factors beyond cell capacity, including driving behaviour, overall vehicle efficiency, and other external conditions.



(a) Example of an OCV-SOC curve.

(b) Example of an OCV-SOC curve with voltage hysteresis.

Figure 1.1: Examples of OCV-SOC curves.

Despite these advantages, silicon lithium-ion cells have certain drawbacks, one of which is their pronounced voltage hysteresis. This means that the OCV-SOC behaviour changes depending on the direction of charge. While this effect also occurs in other commercially available Li-ion chemistries, it is often much smaller and can usually be neglected. During voltage hysteresis, it is observed that the OCV curve for charging consistently is above the counterpart for discharging, a phenomenon that is clearly illustrated in Figure 1.1b. This disparity leads to difficulty in determining the exact SOC value corresponding to a given OCV value. For instance, when examining Figure 1.1b, it becomes apparent that at an OCV of 3.50mV, the SOC could conceivably range from 9% to 19%, as represented by the pink arrow. This variation is dependent on the cell's most recent history regarding charging and discharging. As the cell manufacturer only provides tables for discharging and charging cycles, and not for partial cycles that include both charging and discharging, accurately correcting SOC becomes challenging. This difficulty arises because the cell's history during realistic driving behaviour often includes partial cycles, such as those resulting from regenerative braking.

To overcome this limitation, a hysteresis factor is introduced. This factor, which can take any value within the continuous range from -1 to 1, offers a quantifiable means of interpreting SOC values within the hysteresis region. A factor of 1, representing the left point on the pink arrow as depicted in Figure 1.1b, corresponds to the SOC according to the charging curve. Conversely, a factor of -1 aligns with the right point of the pink arrow and corresponds to the SOC according to the discharging curve. All values between -1 and 1 are distributed linearly, spanning the space of the pink arrow between the two curves. There are currently not many methods to accurately predict the hysteresis factor in silicon lithium-ion cells that are practically applicable [6]. Existing SOC correction methods work effectively for standard battery chemistries but are less reliable when applied to more extreme cell types, such as those with high silicon content.

This thesis proposes to use a black box machine learning approach to predict the hysteresis factor for SOC correction accurately. The motivation for this research is two-fold. First, there is a need to improve the overall accuracy of SOC estimation. Additionally, this work aims to enable SOC correction for more extreme cell types, such as silicon lithium-ion cells, by developing methods that are not only accurate but also practical for use in real-world EV environments, beyond controlled laboratory settings. The successful implementation of the machine learning models and the improvement in SOC estimation accuracy could lead to significant advancements in battery management systems as better estimates could provide both economic and environmental benefits by improving the overall efficiency of electric vehicles. Furthermore, enabling the use of more extreme cell technologies could lead to significant improvements in vehicle performance or substantial reductions in cell costs.

This research aims to bridge the existing gap in SOC estimation methods in the battery management system (BMS) for voltage hysteresis in silicon graphite cells. Therefore, it will consider the vehicle as a system, taking driving data, technical limitations of the BMS and cell chemistry into consideration when designing the algorithm. By contributing to the reliability and efficiency of EVs, this work aspires to pave the way for broader acceptance and a more sustainable future in personal transportation.

1.2. Conventional Approaches to Model Voltage Hysteresis in SOC Estimation

In prior work, various models have been developed to include hysteresis voltage when estimating the SOC. This section will provide a brief overview of different models that consider voltage hysteresis, followed by a discussion of the required testing for these models and the cell types used in conventional approaches. A more detailed summary of the conventional models can be found in section 2.5.

1.2.1. Models Considering Voltage Hysteresis

Generally, the presented models that consider hysteresis in the open circuit voltage can be divided into three categories: equivalent circuit models, physics-based models and machine learning-based models. The distribution of relevant literature among these categories is presented in Figure 1.2. The works highlighted in pink also take minor loop measurements into account. This means that partial (dis)charging cycles, which do not start at 0% SOC and end at 100% SOC, are also included.

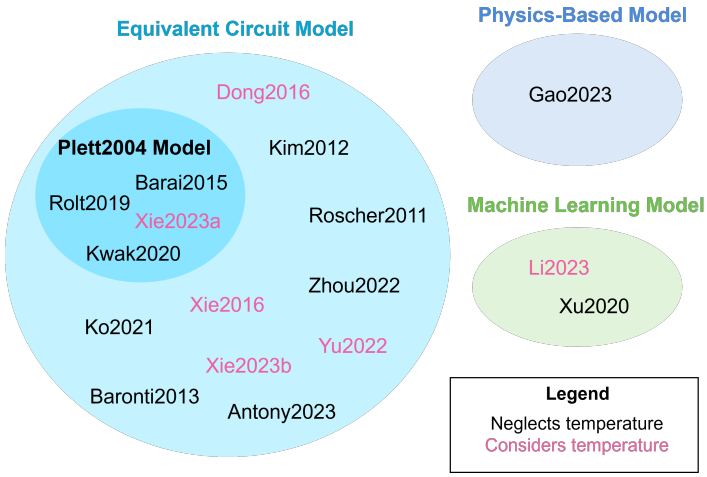


Figure 1.2: Summary of the literature research.

Equivalent circuit models like the Thevenin model incorporate hysteresis through various methods, such as defining a hysteresis factor to blend charge and discharge OCV curves, or using more advanced techniques like the Roscher model, which incorporates additional parameters into the (dis)charging equation to determine the hysteresis factor. Physics-based models focus on modelling the anode's open circuit potential, with Gao et al.'s work isolating the hysteresis factor based on delithiation and lithiation processes. Machine learning models utilize algorithms like random forests or Long-Short Term Memory (LSTM) networks to predict the hysteresis factor, although some approaches, such as those by Li et al., face practical applicability challenges. Each model aims to improve the accuracy of SOC estimations under hysteresis, with varying degrees of complexity and data requirements. A detailed description of each model can be found in section 2.5.

1.2.2. Required Testing for Input Data

According to the surveyed literature, special tests are required for the identification of the hysteresis parameters. Figure 1.3 shows a summary of the findings regarding which tests were used. The majority uses constant current pulse tests. Here the OCV is measured in small SOC steps (e.g. 4% [7]) at a constant current. Between each step is a defined rest period ranging from 1h to 4h. Most did these measurements to obtain the major loop hysteresis voltage. Only some of them, marked in pink also considered the minor loop, by adjusting the tests and starting or ending at different SOC points, rather than completely charging and discharging the cell. Furthermore, these specialised tests are very time-consuming [8] and thus expensive. While the electrochemical impedance spectroscopy does allow to also to learn more about the battery's state than the measured current, temperature and voltage, it is difficult to implement this in an online vehicle [9]. Only a few also considered using driving cycles as their input data. Therefore, it is questionable whether the rest of the models also work well for realistic driving cycles.

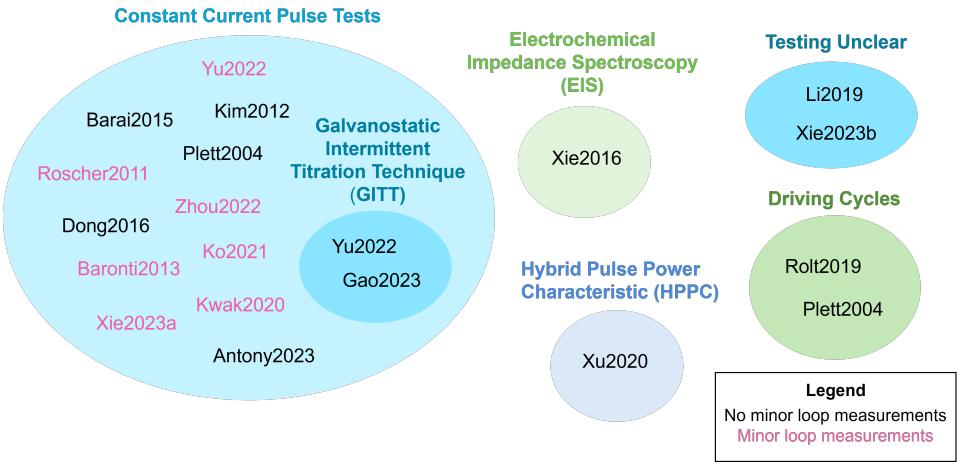


Figure 1.3: Summary of tests done to identify parameters related to the hysteresis phenomenon.

1.2.3. Cell Types Used in Conventional Approaches

Lastly, the conventional models also used different types of cells in their research. In Figure 1.4 the different cathodes (in blue) and anodes (in green) that were used can be seen. While the majority did not specify the anode used, they were likely graphite anodes as those are the most common. Their share in anodes for lithium-ion batteries in 2022 was around 70% [10]. The figure also shows that only very little research has been done regarding voltage hysteresis models for silicon graphite anodes.

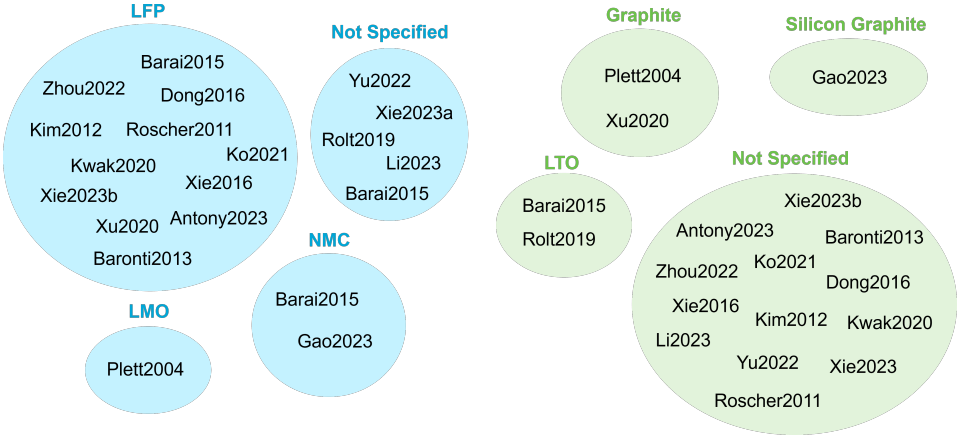


Figure 1.4: Summary of cell types.

1.3. Proposed Approach

This work aims to formulate a black box model to predict the hysteresis factor. It will use available data from vehicle test benches to train on real vehicle data instead of measurement data under lab conditions. This test bench data comprises signals measured by the BMS such as current, voltage and cell temperature. Data for two vehicle projects is available, each project varying in cell chemistry. Moreover, the objective includes identifying a model capable of operating within the BMS, which requires a less complex model due to the BMS's limited computing capabilities. Furthermore, this thesis will address the uncertainty of the SOC estimation. So far, in literature during SOC estimations, the "measured" (actually estimated) SOC was always taken as ground truth. This work will consider the uncertainty that comes with these "ground-truth" SOC labels. To build the black box model, measurement data is properly prepared and then used to train and test the suitable machine learning models. These models are then compared to rate their performance and understand the limitations of each model within this specific application. Finally, the best-performing model is tested for its generalisation capacity.

1.4. Research Questions

There are two main objectives in this work, with each having its respective research questions.

Objective 1: Develop a black box model of the voltage hysteresis effect to determine the hysteresis factor using driving cycles of EVs

1. Can a regression machine learning model accurately predict the hysteresis factor given the computational constraints of a battery management system?
2. Can quantile regression indicate the confidence of the predicted hysteresis factor on a given dataset of driving data?

Objective 2: Perform an evaluation and comparison of different proposed black box models to find the most suitable algorithm to determine the hysteresis factor in terms of accuracy and implementability

3. Which is the best model according to priorly defined evaluation criteria?
4. Can a single algorithm be effectively trained to determine the hysteresis factor across different vehicle projects while maintaining high accuracy?

1.5. Thesis Outline

This thesis first explores the necessary background knowledge in battery technology and machine learning. The chapter on battery technology provides an overview of the functionalities of lithium-ion batteries, discusses silicon lithium-ion batteries, and explains the voltage hysteresis phenomena on a chemical level.

The section on machine learning background covers general concepts, including types of tasks and the pipeline. It then examines different relevant regression models, giving an overview of their principles. A more detailed explanation of the relevant deep learning model is provided due to its complex nature. This section also addresses the origins and impact of label uncertainty, as well as methods for managing it.

Then, the methodology for conducting quantile regression to predict the hysteresis factor is explained. This includes a discussion of the available data, data preprocessing, labelling, and the selection and implementation of suitable machine learning models.

The following chapter presents the results of the case studies. It describes the tests and models, evaluates them, assesses the generalisation capacity of the models, and discusses the feasibility of implementing a battery control unit.

The final chapter comprises the discussion and conclusion. It revisits the problem statement and proposed approach, interprets the results of the thesis, and addresses the research questions. This chapter also highlights the study's limitations and suggests directions for future research.

2

Fundamentals Battery Technology

Batteries are a form of electrochemical energy storage. Lithium-ion (Li-ion) batteries have already been established in automotive applications due to their power and energy density, their rapid decrease in costs over the past decades and the political goals to reduce CO_2. The demand for automotive Li-ion batteries increased more than 680% from 80.7 GWh in 2017 to 550 GWh in 2022 [10].

This chapter aims to provide the fundamental knowledge of battery technology. First, the general functionality of a lithium-ion battery is presented. The chapter then discusses various methodologies for SOC estimation. Subsequently, the role of silicon in Li-ion batteries and its effects on performance are presented. Subsequently, the chapter provides an overview of the hysteresis phenomena observed in batteries, alongside theories proposed to explain its underlying mechanisms. Finally, the conventional methods to model hysteresis are summarized.

2.1. Functionality of a Li-Ion Battery

The operation principle of a Li-ion battery is depicted in Figure 2.1. Typically, a Li-ion battery comprises two electrodes: an anode and a cathode. For simplicity, this work refers to the anode as the electrode where an oxidation reaction occurs during discharging. Typical materials in automotive applications for the electrodes are graphite for the anode [11] and lithium ion manganese oxide (LMO), lithium nickel cobalt aluminium oxide (NCA), lithium nickel manganese cobalt oxide (NMC) and lithium iron phosphate (LFP) cathodes [12] These electrodes are separated by an electrolyte and a separator. The electrolyte functions as a medium to facilitate the movement of ions, whereas the separator prevents short circuits by obstructing the electron flow and enables the flow of ions between the electrodes. Additionally, both electrodes are externally connected to enable the flow of electrons.

During charging, a power supply is connected to the external circuit, thereby forcing the cathode to release electrons. The released electrons move through the external circuit to the anode. Simultaneously, the Li-ions deintercalate from the cathode and move through the electrolyte towards the anode, where they then together with the electron change the oxidation state of the transition metal. With this chemical process, the external electrical energy can be stored in the battery. During discharging, a load is connected to the external circuit, causing electrons to move from the anode to the cathode due to the different electrochemical potentials of the electrodes, thereby releasing electrical energy. Meanwhile, the positively charged Li-ions deintercalate from the anode Li-ions can then move through the electrolyte to be stored in the cathode[13].

2.2. SOC Estimation

The State-of-Charge (SOC) of a battery is one of its defining state variables. Alongside additional system attributes, the SOC serves as an indicator of the battery's residual energy capacity, enabling assessments of its remaining range and overall ageing status. Therefore, errors in SOC estimating can lead to unexpected performance limitations and permanently harm the battery through over- or under-charging [14].

Generally, the SOC is defined as follows [15] with C_b being the actual battery capacity:

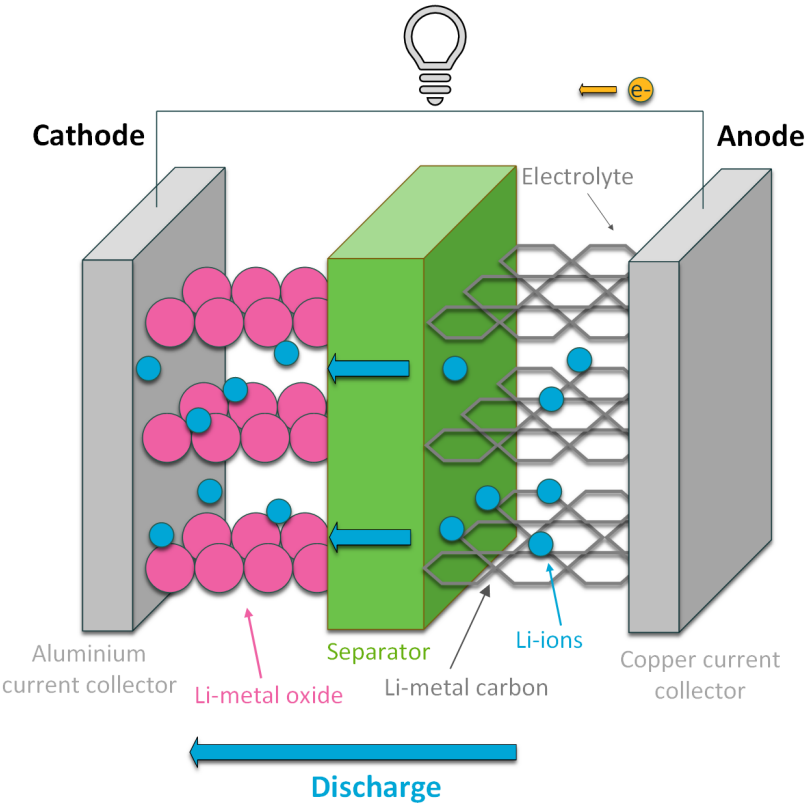


Figure 2.1: Operation principle of discharging a Li-ion battery.

$$SOC = \frac{Q_{releasable}}{C_b} \cdot 100\%. \tag{2.1}$$

Considering that the change in SOC can also be expressed as the fraction of the change in released capacity to the rated capacity, the SOC can also be defined as follows [15]:

$$SOC(t_0 + \Delta t) = SOC(t_0) - \frac{\int_{t_0}^{t_0+\Delta t} i(t)\, dt}{C_b} \cdot 100\%. \tag{2.2}$$

SOC Estimation Using Coulomb Counting
Coulomb counting is a method that can be used to calculate the remaining capacity and thus change in SOC by adding up the charge or discharge throughput over time (see Equation 2.2)[14]. The major drawback of using Coulomb counting is that it accumulates errors over time, whether that is from measurement equipment, errors in Coulombic efficiency (influences C_b) or the neglect of self-discharging processes [16]. This error accumulation is proportional to the duration of operation [14]. Correction is not feasible, as this method operates as an open feedback loop [17]. Additionally, its accuracy depends on the correctness of the initial SOC, $SOC(t_0)$ [14].

SOC Estimating Using Open Circuit Voltage Measurements
In order to increase SOC accuracy, the OCV can be used to correct the SOC after each driving cycle. Current pulse tests can be used to create look-up tables. These tables, typically provided by the cell manufacturer in automotive applications, map each SOC to an OCV value, allowing for SOC correction once the OCV is measured [18]. However, accurate OCV measurements require open circuit conditions, necessitating a relaxation period for the voltage to stabilise at its steady-state value. The duration of this relaxation process can vary significantly depending on the battery chemistry [17]. Consequently, the OCV method cannot continuously estimate SOC in real time. Instead, it serves as a supplementary technique to Coulomb counting, helping to correct its dynamic estimation [17].

2.3. Silicon in Li-Ion Batteries

Silicon has gained significant popularity as an alternative to graphite anodes [10], due to the limited energy density of graphite anodes. In 2022, silicon-graphite anodes comprised approximately 30% of the market share [10]. Numerous automobile manufacturers have announced plans to integrate silicon technology into their batteries [4].

The practical highest achievable lithiated phase of bulk silicon at ambient temperature is $Li_{15}Si_4$, corresponding to a current practical capacity of 3579 mAh/g [19]. In comparison, graphite only offers a practical capacity of around 350 mAh/g [20]. This higher energy density enables a reduction in weight and an increase in range. Additionally, silicon electrodes facilitate faster charging [21]. Silicon is one of the most abundant elements in the Earth's crust and is environmentally friendly, potentially making silicon anode batteries more cost-effective and reducing supply chain issues [22, 19].

Despite their advantages, bulk silicon anodes have not reached widespread adoption. The great disadvantage of bulk silicon is that it pulverises during charging. This is caused by fluctuations in volume, namely a volume expansion and in turn compression of up to 280% during lithium insertion [19]. Due to this pulverisation, the electrodes lose contact with the current collector, lowering the conductivity and the Solid Electrolyte Interface (SEI) accumulates [23] leading to increased internal resistance [24] and thus losing capacity with cycling.

Instead, blended silicon graphite anodes are used. A variation of this is nano-structured silicon. These nanoparticles are dispersed between graphene sheets. They have the advantage that they can be dispersed onto light-weight carbon, thus no additional structures are needed which can increase the weight while simultaneously being stable enough to handle the volume changes [25]. However, due to their graphite content, these electrodes have lower capacities (around 1000-1500 mAh/g under lab conditions [26, 27]).

2.4. Voltage Hysteresis Phenomena

Voltage hysteresis refers to a phenomenon where the potential of an electrode during delithiation is higher than during lithiation at the same stoichiometric lithium content, leading to the cell voltage being higher during charging than during discharging. It is a load-history dependent variation of the OCV curves [28], meaning that the charge and discharge curves of the OCV do not overlap [29]. This impacts the SOC determination as the OCV-SOC relationship is used for SOC correction (OCV method) [29]. The OCV-SOC curves derived from full battery cycles (0% SOC to 100% SOC) are also termed major hysteresis loops [30], while partial cycles result in minor hysteresis loops, which are bounded by the major hysteresis loop [29].

Voltage Hysteresis in Graphite Anodes

This hysteresis phenomenon is visible for electrode materials that undergo a two-phase transition [28], such as lithiated graphite. Due to these two-phase transitions, there are wide potential plateaus during the lithiation/delithiation process as shown in Figure 2.2 [28]. During two-phase transitions, one phase grows at the expense of the other without changing phase compositions. Adding silicon to the graphite anode only enhances this hysteresis phenomenon [6, 31].

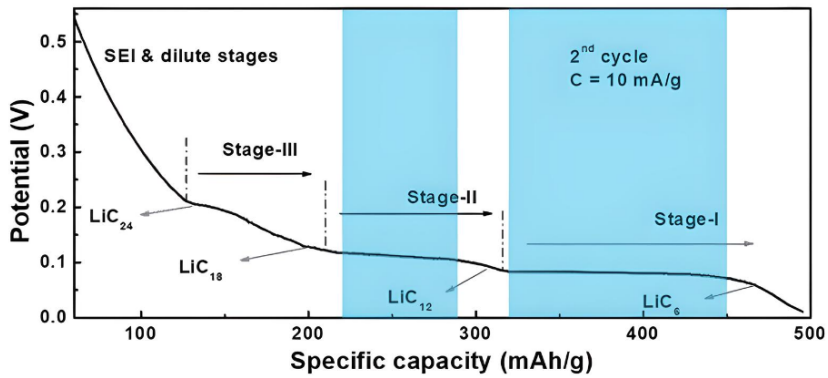


Figure 2.2: Stages of lithium insertion in graphite [32].

Ongoing research aims to understand the underlying causes of hysteresis. One theory is that open circuit voltage (OCV) hysteresis originates from thermodynamic effects. Lithium insertion into particles in the

electrode occurs at different rates. This is because the electric contact to the electrode of particles differs, meaning that particles with better electric contact are more likely to be charged or discharged. In addition, there are differences in the contact with the electrolyte, thus the ionic contact. When lithium is inserted, the concentration of lithium ions in the pores of the electrode decreases since they are absorbed by the active particles. In order to reach particles closer to the inside of the electrode, the ions propagate within the pores, from the surface toward the inside. However, lithium insertion at the surface is more favourable, due to lower ionic conductivity on the inside region. The overall electrode potential considers all individual particles, thus this may differ from the potential of a single particle undergoing lithiation [33, 28]. When considering a lithium-ion insertion, which occurs between two specific phases, this thermodynamic effect implies that although the total number of particles in each phase remains constant during both charging and discharging, the spatial distribution of these particles within the electrode can differ. This difference in distribution leads to varying electrochemical potentials during the charging and discharging processes.

Voltage Hysteresis in Silicon Anodes

There are multiple explanations for the more pronounced hysteresis in cells which contain silicon. Sethuraman et al. [34] stress the correlation between mechanical stress from the volume expansion and voltage hysteresis namely that the compressive pressure lowers the silicon electrode potential while tensile stress elevates it. According to Chevrier et al. [35] the voltage hysteresis can be traced back to the energy required to break the silicon bonds. According to Jiang et al., [36] the pronounced hysteresis results from asymmetric reaction pathways. Specifically, if lithium insertion exceeds a critical threshold—when the lower cutoff voltage drops below 0.05 V vs. Li/Li+—the charging pathway differs significantly from the discharging pathway. During charging, the phase transitions proceed sequentially from amorphous silicon ($a - Si$) to amorphous lithium-silicon alloy ($a - Li_xSi$), and further to the final phase of lithium-rich amorphous silicon ($a - Li_{15}Si_4$). If the voltage threshold is surpassed, an additional phase, $a - Li_{15+\delta}Si_4$, forms at the end. However, during discharging, particles do not revert through the same phases. Instead, they transition from a composite phase of crystalline lithium-rich silicon and excess lithium ($c - Li_{15+\delta}Si_4 + \delta Li$) directly to $a - Li_xSi$, bypassing the intermediate phases encountered during charging.

2.5. Modeling Voltage Hysteresis

This section presents existing models to model voltage hysteresis. Generally, the models can be divided into three categories, equivalent circuit models, physics-based models and machine learning models.

Hysteresis Voltage in Equivalent Circuit Models

There have been multiple attempts to include hysteresis in SOC estimations. One way is to determine the hysteresis factor, ψ, which determines the fraction of the charge and discharge OCV-SOC curve in the actual OCV-SOC curve:

$$u_{OCV}(SOC) = \frac{1+\psi}{2} \cdot u_{OCV,ch}(SOC) + \frac{1-\psi}{2} \cdot u_{OCV,disch}(SOC). \tag{2.3}$$

A hysteresis factor of -1 indicates that the actual OCV is equal to the discharge OCV and 1 signals that it is equal to the charge OCV. There are also derivations from this definition, where the hysteresis factor is defined between 0 and 1 [37], however, the principle remains the same. This hysteresis voltage can then be used in equivalent circuit models, like the Thevenin model depicted in Figure 2.3.

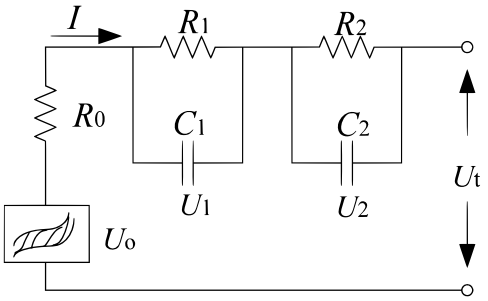


Figure 2.3: Thevenin model [29].

One approach is to assume a hysteresis factor of 0.5, effectively averaging the charge and discharge open-circuit voltages (OCV). This automatically reduces the maximum error due to hysteresis by 50%. However, a method this simple cannot frame the hysteresis effect correctly, leading to significant errors [38].

Another approach is a zero-state hysteresis model, where the hysteresis factor switches between the values 0 and 1, depending on the sign on the current,

$$\theta(t) = \begin{cases} 1 & \text{if } i(t) > \psi, \\ 0 & \text{if } i(t) < -\psi, \\ \psi(t-1) & \text{if } |i(t)| \leq -\epsilon. \end{cases} \tag{2.4}$$

with ϵ being small and positive. However, this model does not consider the history of the current and is thus limited in accuracy [30].

Others have introduced additional parameters for the hysteresis factor, which were determined with parameter identification. One of these models is by Roscher et al. [37] and is defined as follows:

$$\psi(t) = k_1 \cdot \int \frac{m_1 \cdot i(t)}{C_n}\, dt + k_2 \cdot \int \frac{m_2 \cdot i(t)}{C_n}\, dt. \tag{2.5}$$

Note that the hysteresis factor is scaled between 0 and 1. The factor itself, ψ, is defined in Equation 2.5. Here, The hysteresis factor depends on the parameters k_1, k_2, m_1, and m_2, which were previously identified using specialised cell tests. Note that in Equation 2.5, the normalisation factors k_1 and k_2 add up to 1 and the integrals range between 0 and 1. After scaling the hysteresis factor to lie between -1 and 1, it can be applied to the OCV equation shown in Equation 2.3.

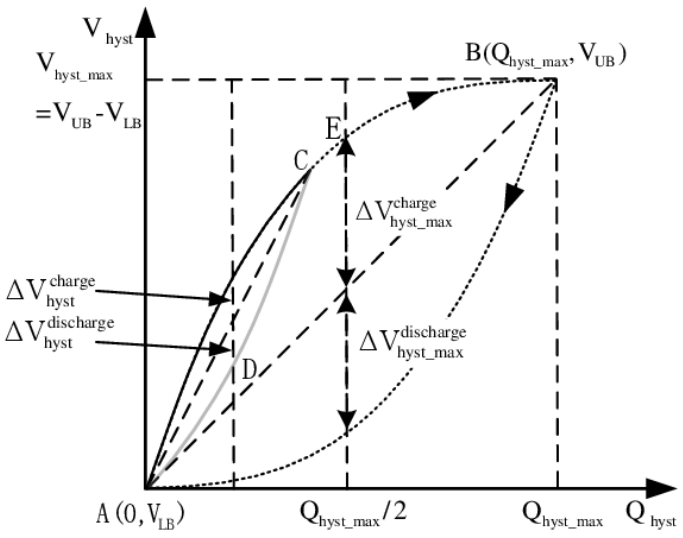


Figure 2.4: Hysteresis model using the relative charge throughput to obtain the minor hysteresis loop according to Xie et al. [39].

Xie et al. [39] also developed a model for determining the OCV under hysteresis, defining the voltage difference that allows the estimation of minor hysteresis loops from the major loops (see Figure 2.4). This approach is similar to Roscher's model [37]. However, instead of defining a hysteresis factor, the transition between major hysteresis loops is determined by the charge throughput Q relative to the battery's capacity $Q_{h,max}$:

$$\begin{aligned} \Delta v_{h,discharge} &= \Delta v_{h,discharge,max} \cdot \frac{Q}{Q_{h,max}}, \\ \Delta v_{h,charge} &= \Delta v_{h,charge,max} \cdot \frac{Q_{max} - Q}{Q_{max}}. \end{aligned} \tag{2.6}$$

They assume the hysteresis voltage depends only on charge throughput and not on temperature.

Ko et al. [40] defined the OCV similarly to Roscher [37] to determine the minor hysteresis loop. However, they describe the charging (α) and discharging (β) loops separately:

$$\begin{aligned} v_{OCV}(SOC, \alpha) &= \alpha \cdot v_{OCV,charge} + (1-\alpha) \cdot v_{OCV,discharge}, \\ v_{OCV}(SOC, \beta) &= \beta \cdot v_{OCV,charge} + (1-\beta) \cdot v_{OCV,discharge}. \end{aligned} \tag{2.7}$$

The slope parameters α and β, which can take any values between α_0 to α_n and β_n to β_0, respectively, are iteratively defined as:

$$\begin{aligned} \alpha_k &= \alpha_0 - \int_0^k \frac{\eta_b(\alpha_n - \alpha n - 1) \cdot i_{b,k,charge}}{(\Delta_n SOC - \Delta_{n-1} SOC) \cdot C_n}\, dt, \\ \beta_k &= \beta_0 - \int_0^k \frac{\eta_b(\beta_n - \beta n - 1) \cdot i_{b,k,discharge}}{(\Delta_n SOC - \Delta_{n-1} SOC) \cdot C_n}\, dt. \end{aligned} \tag{2.8}$$

In this definition, C_n represents the battery capacity, η_b is the battery efficiency, and $i_{b,k,(dis)charge}$ is the charge or discharge current. This model, combined with an extended Kalman filter and a Thevenin model, was used to estimate the SOC. The data for fitting the parameters was obtained through meticulous pulse charge and discharge tests. This hysteresis model was established only for the 20% SOC window, assuming that at higher SOCs, the OCV will always follow the major loops.

Xie et al. [41] adopted the same assumption about the OCV voltage as Roscher [37]. However, they explicitly defined the hysteresis factor as being independent of any parameters to be fitted:

$$\psi_k = \psi_{k-1} - \left(i_k \cdot \Delta T \frac{\eta}{C_n} \cdot \frac{1}{a} \right). \tag{2.9}$$

The hysteresis factor can be directly calculated using only the battery capacity C_n, the current at timestep k i_k, the battery efficiency η, the sampling period ΔT, and the accumulation of SOC a. Their definition, as shown in Equation 2.9, applies to values of ψ between 0 and 1

To include the history of the current, a one-state hysteresis model was proposed by Plett et al. [30]. In this model, the hysteresis factor is not explicitly defined, instead, a hysteresis voltage, which can then be used in an ECM, is defined. The dynamics of the hysteresis voltage are given by the differential equation,

$$\frac{du_H}{dSOC} = \gamma \cdot sgn\left(\frac{dSOC(t)}{dt}\right) \cdot \left(M(SOC, \frac{dSOC(t)}{dt}) - u_H(SOC, t) \right). \tag{2.10}$$

Note, that this differential equation is given in SOC and not time. $M(SOC, \frac{dSOC(t)}{dt})$ is the maximum polarisation caused by hysteresis. The constant γ tunes the rate of decay and needs to be determined with parameter identification. Using this model in combination with a Kalman filter to estimate the SOC did result in a lower estimation error than using the zero-state model [30]. This model has been the basis for other research regarding continuous SOC estimation with ECMs and Kalman filters under consideration of hysteresis [7, 42, 43, 44].

Baronti et al. [45] introduced another model for the hysteresis voltage, where the total OCV voltage is comprised of the average OCV voltage (average of the charge and discharge OCV curve) and the hysteresis voltage defined as

$$\frac{du_H}{dSOC} = \gamma \cdot sgn\left(\frac{dSOC(t)}{dt}\right) \cdot v_H + \gamma \frac{M(SOC)}{2}. \tag{2.11}$$

The model can be seen in Figure 2.5. Here $M(SOC)$ denotes the difference between the charge curve and the discharge curve of the major hysteresis loop.

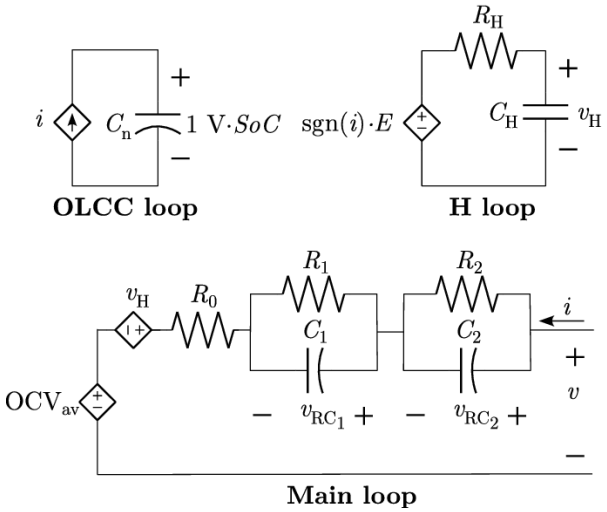


Figure 2.5: ECM for including the hysteresis voltage according to Baronti et al. [45]

Another model for the hysteresis voltage was defined by Dong et al. [46], which is similar to that of Baronti in Equation 2.11 [45], however, with a minus sign in the first term:

$$\frac{du_H}{dSOC} = -\gamma \cdot sgn(\frac{dSOC(t)}{dt}) \cdot v_H + \gamma \frac{M(SOC)}{2}. \tag{2.12}$$

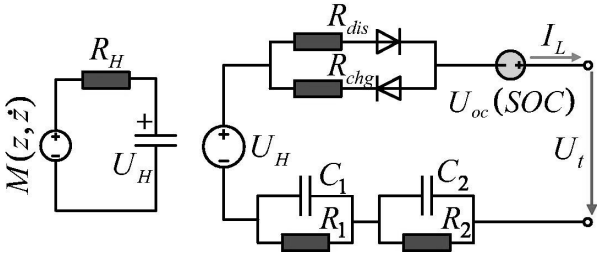


Figure 2.6: ECM for including the hysteresis voltage according to Dong et al [46].

Note that in both models, Dong and Baronti, γ is determined to be greater than 0. The OCV (U_{OC} in Figure 2.6) is a 5th polynomial in dependence of the SOC with parameters that need to be fitted. All parameters of the model, including γ, were identified online with an invariant-imbedding method. The estimated SOC, determined with a multi-state estimator, had an error margin of +/- 2%.

A two-state model was introduced by Kim et al. [47]. They split the OCV voltage in charge and discharge OCV, which was realised by the addition of diodes in the ECM. This split is shown in the ECM in Figure 2.7. In the differential equation for the OCV,

$$\frac{du_{OCV,k}}{dx_k} = \begin{bmatrix} \frac{du_{OCV}(SOC)}{dSOC} & 0 \\ 0 & -1 \end{bmatrix}, \tag{2.13}$$

where x_k is the state vector at step k, which is comprised of the SOC at $k+1$ as well as the voltage difference at step $k+1$. Using an extended Kalman filter based on the ECM, they predicted the OCV. However, their resulting SOC estimation error was up to 5% and the cell temperature was not considered.

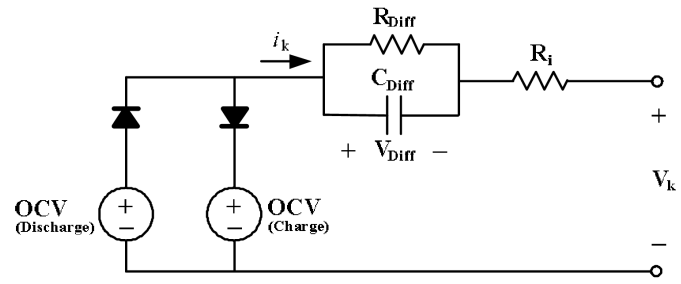


Figure 2.7: ECM of the two-state hysteresis model according to Kim et al. [47].

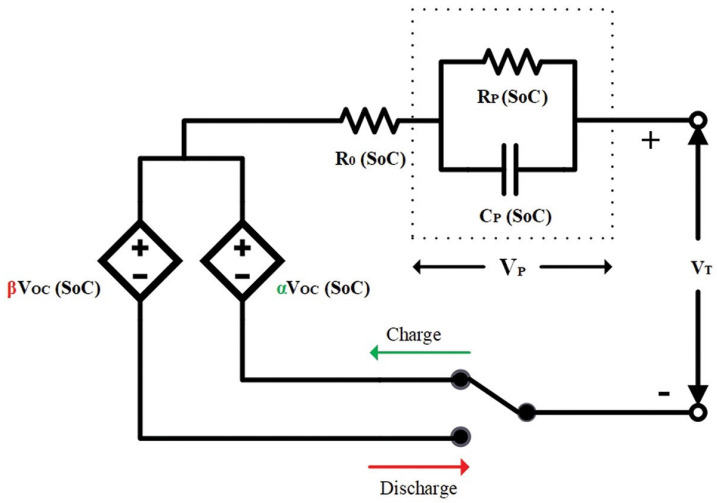


Figure 2.8: ECM of the model used by Antony et al. [48]

Antony et al. [48] used the ECM seen in Figure 2.8. The major loop charge and discharge OCV are multiplied by correction factors α and β, respectively:

$$\begin{aligned} \alpha &= \frac{Q_{charge}}{C_{b,avg}}, \\ \beta &= \frac{Q_{discharge}}{C_{b,avg}}. \end{aligned} \tag{2.14}$$

These factors consist of the charge or discharge throughput divided by the average capacity. Their results show a mean average error of up to 7.5mV.

Zhou et al [29] considered the hysteresis directly in the OCV voltage. They use a Thevenin Model (see Figure 2.3). Their differential equation model is defined as:

$$\frac{du_{OCV}}{dSOC} = \begin{cases} \frac{du_{OCV,charge}}{dSOC} + \eta_b(u_{OCV,charge} - u_{OCV}) & \text{if } \frac{dSOC}{dt} > 0, \\ \frac{du_{OCV,discharge}}{dSOC} + \eta_b(u_{OCV} - u_{OCV,discharge}) & \text{if } \frac{dSOC}{dt} \leq 0. \end{cases} \tag{2.15}$$

The model adjustment coefficient η needs to be parameterised by analyzing a large number of minor and major loop tests. The voltages $u_{OCV,charge}$ and $u_{OCV,discharge}$ refer to the voltage (curve equation) of the major hysteresis loop. In this model, temperature is not considered as an influence on hysteresis.

Yu et al. [49] proposed a superimposed asymmetric hysteresis model. The OCV voltage is modelled by superimposing multiple asymmetric play operators, each noted by the index i. Each play operator is defined as:

$$u_{OCV,i}(k) = \max(a \cdot u^b(k) - r_i, \min(a \cdot u^b(k), u_{OCV,i}(k-1))). \tag{2.16}$$

$u^b(k)$ denotes the input at step k and r_i is the threshold of the i-th operator. a and b are parameters that were found using specialised tests such as Hybrid Pulse Power Characteristic (HPPC) and Galvanostatic Intermittent Titration Technique (GITT). Then the overall OCV voltage can be determined:

$$u_{OCV}(k) = \sum_{i=1}^{n} w_i \cdot u_{OCV,i}(k). \tag{2.17}$$

The weight w_i of each play operator i is then found by minimising the error in the OCV estimation. This led to a maximum error of 17mV.

Physics Based Models
Gao et al. [6] created a physics-based model to determine the half-cell open circuit potential (OCP) of the anode. This is relevant as the OCV is the OCP of the anode subtracted by the OCP cathode and it thus allows to model the hysteresis solely for the anode. They define the anode OCP as follows:

$$u_{OCP}^{anode}(\theta^{anode}) = \psi u_{OCP,de}^{anode}(\theta^{anode}) + (1-\psi) u_{OCP,li}^{anode}(\theta^{anode}), \tag{2.18}$$

where $u_{OCP,de}^{anode}$ is the anode OCP during delithiation and $u_{OCP,li}^{anode}$ during lithiation and with ψ being the hysteresis factor. θ is the electrode stoichiometry. The concept is similar to that of Roscher [37], however applied to only the OCP instead of the OCV. The hysteresis factor ψ was then found by minimising the model estimations of the OCV with the actual OCV under GITT tests (least squares problem). To get estimations, Equation 2.18 was incorporated into a pseudo-2-dimensional model of a battery. Their approach does not consider different temperatures. Depending on the SOH level, their smallest predicted OCV error was 6.182 mV.

Machine Learning Based Methods
Li et al. [50] used a random forest algorithm to determine the OCV used for SOC correction under the effect of hysteresis. They collected OCV-SOC data, to predict the actual SOC/OCV curve (a curve in between the charge and discharge curve). However, the authors themselves declared that their results are not practically applicable.

Xu et al. [8] uses a Long-Short Term Memory model to learn the OCV under hysteresis. They define the hysteresis voltage similarly to Equation 2.3, however, the LSTM then learns to predict the hysteresis factor ψ between 0 and 1. They use the battery current and voltage of each charging counting time as input. The LSTM therefore has an input size of 2 and a size of 5 in their hidden layer. After the linear layer, the output is sent through a sigmoid function to ensure an output value between 0 and 1. However, the model was only trained with the results of 3 HPPC tests in total and the used labels have values of either 0 (discharging) or 1 (charging).

3

Fundamentals Machine Learning

This chapter explores the fundamental principles of machine learning, which is a subset of artificial intelligence. According to Sarker [51], machine learning aims at "using data or past experience to automate analytical model building". This indicates that the ultimate goal is to automatically learn a relationship between input and output with available data or experience without having to explicitly program the rules. Therefore Machine Learning is a very suitable tool for problems with high dimensional data, such as fraud detection or image and speech recognition [52].

This chapter introduces essential machine learning concepts, starting with a brief overview of supervised and unsupervised learning and the machine learning pipeline. Then, it discusses different regression models, including linear regression, XGBoost, and deep learning. The chapter concludes by addressing uncertainty in labels and the use of quantile regression for capturing variability in predictions.

3.1. General Concepts of Machine Learning

This section provides a brief overview of the fundamental concepts in machine learning. It begins with an introduction to the two most relevant types of learning: supervised and unsupervised learning, followed by a description of how a structured machine learning workflow is organised through a machine learning pipeline.

3.1.1. Supervised and Unsupervised Learning

Within machine learning, the type of learning can be classified into supervised learning and unsupervised learning [53], with other types existing but excluded from this discussion due to their irrelevance to this work. In supervised learning, the objective is to learn a function that accurately maps input to output using a training set of data where input-output pairs are given. The respective output of each input is also referred to as label [54]. Typically, supervised tasks are regression (prediction of a continuous numerical value based on the input vector [53]) or classification problems [55] (prediction of a value out of a set of discreet values [53]). If no labels are given, such as in clustering tasks, unsupervised learning is applied [55]. Weakly supervised learning can be considered a mixed form between supervised and unsupervised learning. Usually, weakly supervised learning is separated into three different categories [56]:

- **Incomplete supervision**: The majority of the available training data is unlabelled and only a small subset of labelled data is available. A reason for this can be, for example, high costs to obtain labels, such as labelling that needs to be done manually.
- **Inexact supervision**: In this case, only course-grain labels are available as it is not possible to determine the exact ground-truth value.
- **Inaccurate supervision**: Here, some labels which are given might not be accurate and, thus do not hold ground-truth values. This can be due to errors in the labelling process.

3.1.2. Machine Learning Pipeline

The machine learning pipeline is an automated workflow of preparing, training, testing and validating machine learning models and is part of academic and industry best practices for reproducibility.

The following steps are an essential part of the machine learning pipeline [57]:

1. **Data preprocessing:** This step includes the collection of sufficient data to ensure generalization within the scope of the model's application. This can be done, for example, using measurements or available databases. Data preprocessing also includes the cleaning of data, as there might be missing values values, or the data might be inconsistent. Moreover, the data preprocessing step includes the transformation of data. Examples include the normalisation or standardisation of data, which is necessary because different features may have varying scales and machine learning frameworks require normalised or standardised feature inputs [58]. At last, the data has to be split into a training, test and validation set.
2. **Feature design:** This step includes feature engineering. It means that new features can be created for the model or existing features can be transformed. It also involves feature selection; identifying which features are relevant to the model. Moreover, this also includes feature extraction, which describes the process of transforming raw data, for example, data which is in Hex decimal into numerical features. This increases the performance of the model as the model often performs better than with the raw data [59].
3. **Model development:** The model development includes the selection of the model, which is discussed in more detail in subsection 4.6.2. It also includes the training and validation of the model. This can be done either offline, by providing a batch of training data beforehand, which will then be used all at once to train the model, or online, when the full dataset is not available at the beginning and the best predictor will be updated as more data becomes available. In this work offline learning is applied, as all the measurement data is already available. Lastly, all relevant hyperparameters of the chosen model should be optimised [57].

3.2. Regression Models

Regression models are designed to map the input variables to continuous output variables [60]. These models can vary in complexity from rather simple linear regression models to gradient boosting models to neural networks. This section will provide a concise overview of these three distinct models, delving into their assumptions, their capabilities, and their limitations.

3.2.1. Linear Regression Model

The objective of linear regression is to construct a hyperplane that best approximates a set of scattered data points. This dataset is formally defined as a set of tuples $(x_p, y_p)_{p=1}^{P}$, where each x_p is a column vector with a length of N of sample p, which is the number of features. Linear regression assumes linearity between the inputs x_p and the labels y_p By employing the equation for a hyperplane, the associated loss function can be written as the minimisation of the squared differences between the hyperplane's predictions and the actual labels y_p. This least squares cost function is defined as [61]:

$$\min_{b,w} \sum_{p=1}^{P} (\dot{x}_p^T w - y_p)^2. \tag{3.1}$$

With $\dot{x}_p$ defined as encompassing the unit vector as the first element, and then the input vectors x_1 to x_P, the minimisation of this convex function is achieved through the solution of w^* [61],

$$w* = (\sum_{p=1}^{P} \dot{x_p} \dot{x_p}^T)^{-1} \sum_{p=1}^{P} \dot{x_p}^T y_p. \tag{3.2}$$

However, as solving for w^* is numerically expensive, the problem is typically approached with gradient descent. This iterative technique starts with an initial weight matrix w^0 and progresses by repeatedly adjusting w as can be seen in Equation 3.3, where α is the step length. With these iterative steps, this method gets closer to

a stationary point, thus a (local or global) minimum of the objective function g. The weight matrix at iteration k, denoted as w^k, is updated until the procedure either reaches a predefined number of iterations or the norm of the gradient $\nabla g(w^k)$ becomes small enough [61]:

$$w^k = w^{k-1} - \alpha_k \nabla g(w^{k-1}). \tag{3.3}$$

The step length α_k is crucial as it must be small enough to prevent overlooking the minimum, yet sufficiently large to ensure the computational feasibility of the method. Nevertheless, it is important to note that gradient descent does not assure convergence to the global minimum [61].

3.2.2. Extreme Gradient Boosting Model (XGBoost)

XGBoost is a library designed to optimise and extend gradient boosting for supervised learning problems [62]. It builds upon the gradient boosting algorithm [63] and uses decision tree ensembles, specifically regression and classification trees (CART). A decision tree consists of nodes, leaves ("endpoints" of the tree) and branches, aiming to determine a classification rule to divide the training data into homogeneous subsets. At each node, a test based on the sample's features is done. Each outgoing branch of the node represents a possible test outcome, leading to another node or leaf. Leaves indicate which subset the input belongs to and thus do not have any outgoing branches [64].

Unlike class-based decision trees, CARTs contain a scalar score in the leaves. In practice, multiple trees are used to improve performance, with the overall score being the sum of scores from individual trees, defined as follows:

$$\hat{y}_i = \sum_{d=1}^{D} f_d(x_i), \tag{3.4}$$

where $\hat{y}_i$ is the predicted value for sample i, D is the number of trees, and f_d is a function representing each tree [62]. The model is trained by optimising an objective function defined as [62],

$$obj(\Theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{d=1}^{D} \omega(f_d), \tag{3.5}$$

where $l(y_i, \hat{y}_i)$ is the loss function and $\omega(f_d)$ is the complexity function (for details see section A.1) which serves as regularisation.

XGBoost uses an additive strategy to learn the tree functions. Instead of learning all trees simultaneously, the trees are learned one by one, with each tree improving upon the previous one (for more details see section A.1), where $\hat{y}_i^{(t)}$ describes the output at time step t [62]:

$$\hat{y}_i^{(t)} = \sum_{d=1}^{t} f_d(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i). \tag{3.6}$$

The optimal tree structure is found by evaluating the gain of splitting leaves. The gain,

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma, \tag{3.7}$$

comprises the score of the left new leaf, the right new leaf and the original leaf (for details see section A.1). If the gain is negative, the leaf is not split [62].

3.2.3. Deep Learning Model

Deep learning, a subset of machine learning, is deployed when conventional machine learning architectures fall short of capturing the intricacies of a problem. This is particularly evident in tasks such as recognising speech or recognising objects, where the algorithm has to learn a function in a high-dimensional space [53].

Their architecture, which mimics the cognitive processes of the human brain, enables them to represent exceedingly complex functions by employing an increasing number of layers [53]. When the amount of data increases, deep learning demonstrates higher efficiency compared to traditional machine learning models [51].

The following things need to be considered before designing a model for deep learning [51]

- **Data dependency**: Deep learning often performs badly when the available dataset is too small. Therefore a lot of data is needed to achieve the desired performance
- **Hardware dependencies**: Deep learning requires high computational power, making GPUs more suitable than CPUs due to their efficiency for these tasks.
- **Feature engineering process**: Extracting features from available raw data often takes less time and effort than traditional ML models
- **Model training and execution time**: Deep learning training is slower due to many parameters but quicker in testing compared to traditional ML models.
- **Interpretability**: Deep learning lacks transparency as a black box model, making result traceability difficult compared to more interpretable traditional models.

Feed Forward Neural Network

The feed-forward model derives its name from the flow of information from the input to in-between computations towards the output with no feedback connections from the output back to the input or the in-between computations [53]. There are three types of layers, namely the input layer, the hidden layer and the output layer, each consisting of neurons. Every neuron of one layer is connected to every neuron of the next layer, making the network fully connected. While there is only one input and output layer, there can be multiple hidden layers and choosing the number of hidden layers, thus deciding on the depth of the network requires experience and tuning [65].

In addition, there are other attributes of the network's architecture that have to be considered, namely the number of layers and the width of each layer, so the amount of neurons also referred to as hidden units per layer. These two attributes, depth and width also interact. Deep networks need fewer units per layer and fewer parameters to reach the same network capacity but are in turn more difficult to optimise. The optimal architecture must be chosen by manually adjusting according to the error of the validation set [53].

Another consideration is the type of hidden unit. The type refers to the activation function g of the neuron. An illustration of this can be seen in Figure A.1. This function makes it possible for the network to also learn non-linear functions. The most common types of units are listed in section A.2 [53].

The learning process requires two fundamental mechanisms, forward propagation and backpropagation, to minimise the loss function effectively. Forward propagation involves the transmission of initial data from the input through the hidden layers to generate the output. Conversely, backpropagation entails the reversal of this information flow, where the computed loss from the output stage is relayed backwards throughout the network and thus enables the computation of gradients. These are instrumental for optimising connection weights via algorithms such as Stochastic Gradient Descent during the network's training phase [53].

Recurrent Neural Network

Using recurrent neural networks (RNNs) has become a common practice when processing sequential input data as they are designed for this input data [66]. The three different design patterns for the architecture of RNNs can be found in section A.3.

As explained for feed-forward networks, RNNs also use backpropagation, referred to as backpropagation in time, as backpropagation is done on the unfolded graphs (examples of which can be seen in Figure A.2)[67].

RNNs face two major challenges: The first one is exploding or vanishing gradients. When gradients propagate through many layers, they typically vanish or, in rare cases, explode [68]. RNNs repeatedly compose the same function [53]:

$$h^{(t)} = W^T * h^{(t-1)} = (W^t)^T * h^{(0)}. \tag{3.8}$$

Note that this is a simplified representation without non-linear activation functions or x inputs. This nesting leads to highly non-linear behaviour. This can be further simplified by representing W as an eigenvalue decomposition with an orthogonal Q matrix [53]:

$$h^{(t)} = Q^T * \Lambda^t * Q h^{(0)}. \tag{3.9}$$

Eigenvalues Λ raised to the power of t show that values less than 1 approach 0 as t increases, while values greater than 1 grow rapidly. As illustrated in Equation 3.9, only elements of $h^{(0)}$ aligned with the largest eigenvector significantly influence $h^{(t)}$ [53].

The second challenge is a direct consequence of the first: The weights given to long-term interactions are exponentially smaller than those given to short-term interactions [68]. Consequently, the gradient of long-term interactions is exponentially smaller than that of short-term interactions, resulting in a slower learning process for long-term dependencies [53].

Gated RNNs Gated RNNs are a special form of RNNs designed to address the challenges of exploding/-vanishing gradients and the small weights for long-term interactions. Their core idea is to create pathways through time with stable derivatives. The weights of these pathways can vary at each time step, allowing the network to keep prior information and selectively forget it when no longer useful [53, 8]. Gated RNNs excel in learning when to forget [53]. There are two notable types of gated RNNs, namely long short-term memory and gated recurrent units.

Long Short-term Memory (LSTM) In a long short-term memory (LSTM) network, self-loops create paths through time with stable gradients. The weight of the self-loop is gated, controlled by another hidden unit, and thus not fixed. An illustration of an LSTM cell is shown in Figure 3.1. LSTMs contain three gates: the input gate determines if an input feature is accumulated into the state; the forget gate controls the self-loop weight; and the output gate controls the cell's output. All gates use a sigmoid non-linearity, the input unit, however, can use any squashing non-linearity. LSTMs have demonstrated strong performance in learning long-term dependencies [53].

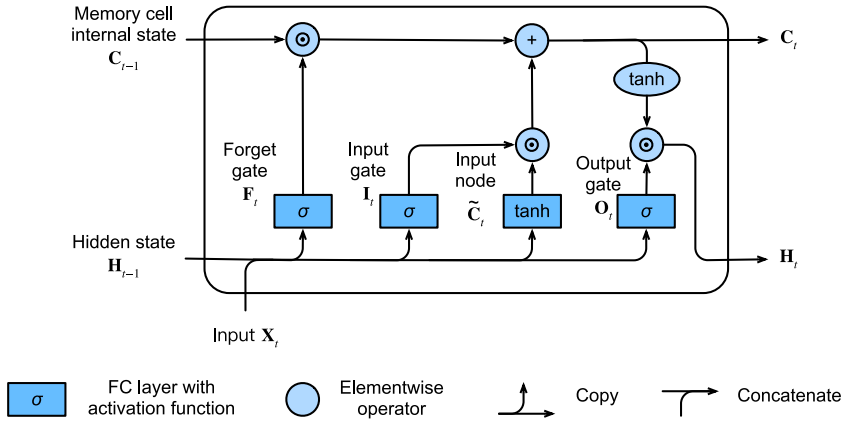


Figure 3.1: Illustration of a LSTM cell [69].

Gated Recurrent Units (GRU) Gated recurrent units are similar to LSTMs in terms of functionality. However, unlike LSTMs, there are only two gating units instead of three (see Figure 3.2). The update gates act as leaky integrators that can choose to copy, ignore or replace the current state with a new target state [69]. These gates help capture the long-term dependencies [53] The reset gates control which part of the state influences the next target state. Compared to LSTMs, GRUs have the advantage of being computationally faster [69].

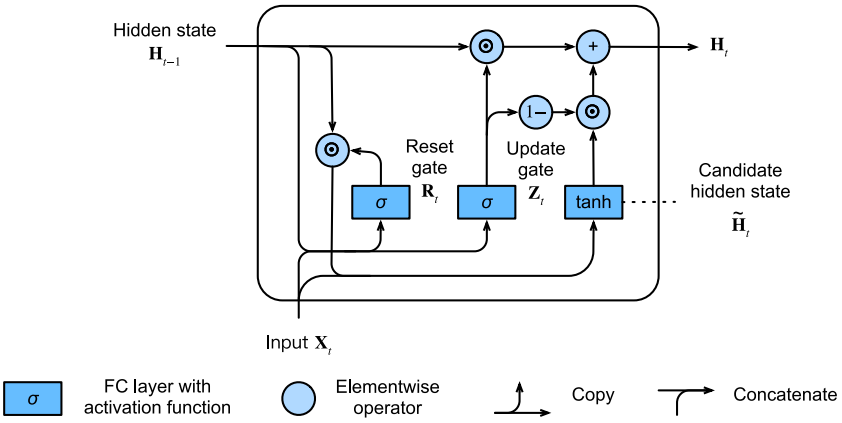


Figure 3.2: Illustration of a GRU cell [69].

The mathematical formulation of a GRU for time step t is as follows [70]:

$$R_t = \sigma(W_{ir} \cdot X_t + W_{hr} \cdot H_{t-1} + b_{hr}) \tag{3.10}$$
$$Z_t = \sigma(W_{iz} \cdot X_t + W_{hz} \cdot H_{t-1} + b_{hz}) \tag{3.11}$$
$$\tilde{H}_t = \tanh(W_{in} \cdot X_t + b_{in} + R_t \circ (W_{hn} \cdot H_{t-1}) + b_{hn}) \tag{3.12}$$
$$H_t = (1 - Z_t) \circ \tilde{H}_t + Z_t \circ H_{t-1} \tag{3.13}$$
$$\tag{3.14}$$

Note that the first two equations describe the formulations for the two gates, the reset gate R_t and the update gate Z_t. The last two equations then define how to determine the candidate hidden state $\tilde{H}_t$ and the hidden state H_t. X_t is the input vector with dimensions (input size, 1). Thus the weight matrices which describe the weights between the input and hidden layer (W_{ig} with g being the respective gate/hidden state candidate gate), have dimensions (hidden size, input size). If it is the first GRU layer in the network, the input size corresponds to the size of features, similarly, the input size for the second layer (denoted by $l = 2$) would correspond to the hidden size of the first layer. Hence, X_t in Equation 3.10 to Equation 3.12 would be $H_t^{(l-1)} \cdot \delta^{l-1}$. The dropout variable δ^{l-1} is a Bernoulli random variable which is zero with the probability defined in the parameter dropout during the initialisation of the model.

The weights connecting the hidden layers are denoted by W_{hg} and have dimensions (hidden size, hidden size). b_{hg} and b_{in} denote the bias vector. The Hadamard product is represented by the symbol $\circ$, while σ denotes the sigmoid function.

3.3. Managing Uncertainty in Labels

In real-world scenarios, the quality of training data is often imperfect due to the difficulty of obtaining reliable and high-quality data. Poor data quality may arise from unknown features, unknown feature values, or incomplete, incorrect, or inconclusive data [71]. In order to develop a robust machine learning mode, these uncertainties have to be managed.

Uncertainty in machine learning can be categorised into aleatoric uncertainty and epidemistic uncertainty. Aleatoric uncertainty also referred to as statistical uncertainty arises from the inherent variability or randomness in the data, including noise that cannot be reduced by obtaining additional data. In contrast, epistemic uncertainty, also known as systematic uncertainty, stems from a lack of knowledge that can be reduced through the acquisition of more data or information[72].

Imperfect learning occurs when the quality of the training data is poor, or the quantity of training samples is insufficient [71]. If labels are not available for every sample, resulting in a dataset that contains a certain percentage of labelled and a certain percentage of unlabelled data, this is referred to as semi-supervised learning [73]. If the labels are noisy and their quality is compromised, then this is referred to as weakly supervised learning [56].

There are two types of noisy labels in regression data. The first one is referred to as additive noise and can be described by,

$$y = y_{True} + \epsilon, \tag{3.15}$$

where y_{True} is the ground truth label and ϵ is the noise drawn from a normal distribution and thus independent of the input [74].

The second type is referred to as instance-dependent noise and can be defined as,

$$\tilde{y} = \rho(x), \tag{3.16}$$

with x being the input and ρ being a noise function which depends on the input [74].

There has been research on addressing the challenges posed by noisy data. Chang et al. [71] aimed to improve the performance of machine learning models that train with noisy input data by imitating the human learning process during the training phase. This is done by changing the way imperfect data is represented, namely by introducing a certainty weight that denotes the reliability of each sample based either on statistical information or assignments by experts.

Hartono et al. [75] proposed a neural network ensemble algorithm that learns to exclude erroneous samples and effectively handles training sets containing such errors.

Bekker et al [76] introduced an additional layer in neural networks, allowing the network to learn not only its parameters but also the noise distribution for a classification problem. They assume that ground-truth labels are superimposed with noise generated by an unknown noise channel. By modelling the transformation from true labels to noisy labels, the network learns the associated parameters.

Wang et al. [77] developed a deep neural network architecture that improves the performance of models that are trained on noisy labels and tested on clean labels. The application is a sentiment classifier, which contains two specific layers, one of which is a noise transition layer to handle the input noise, similar to Bekker et. al [76], and one to predict which ones are clean labels.

Another approach is to exclude the imperfect labels during preprocessing. However, this may not always be possible [78].

There are three common approaches to learning with noisy labels. The first approach involves employing a robust loss function that includes regularisation. The second approach involves estimating the noise by assuming a probability distribution function. However, this method requires a sufficient number of ground-truth labels for validation or assumptions about the data and information about the noise. The third approach is noise correction. Initially, the model learns simple relationships and predicts accurately. By generating pseudo-labels, such as probabilistic labels, and training on them, it is suggested that this can correct the noisy labels [79].

Imperfect data is not the only issue that can impact an algorithm's performance; problems in the model training process, such as underfitting, overfitting, insufficient training data, incorrect model selection, and suboptimal hyperparameters, can also reduce performance. Various techniques can be applied to address these issues. A few common examples are listed below.

- **Data augmentation**: This technique helps address issues related to insufficient training data or unevenly distributed data. By transforming the input data (e.g., rotating or reflecting images), a more abundant and diverse training dataset can be created. This approach is often utilised in image classification [80].

- **Lasso (L1) and Ridge Regression (L2)**: These techniques are used to avoid overfitting, where the model learns the training data and its details too well, resulting in poor generalisation on the test set. Both methods minimise the model's complexity alongside the loss. In Lasso regression (L1), this is achieved by adding a penalty term representing the absolute value of the sum of coefficients, regulated by the hyperparameter lambda, which can reduce certain feature weights to zero and thus perform feature selection. Lasso regression effectively eliminates multicollinear features entirely. In Ridge regression (L2), the penalty term is the squared sum of coefficients without performing feature selection [81].
- **Early stopping**: This method also prevents overfitting by stopping the model training process early, namely when the validation error starts increasing again or remains constant. This ensures that the model does not overlearn the details of the training data [81].
- **Hyperparameter tuning**: This process optimises the model parameters, which are crucial for the learning stage. Several methods can implement this, with randomised search being one example. In this method, parameter configurations are generated from a given parameter range, and the configuration with the lowest validation error is selected [82].

3.4. Quantile Regression

Quantile regression can be used to quantify the aleatoric uncertainty in a model's prediction [83], thereby estimating the level of confidence in the prediction. Instead of predicting a singular value, quantile regression allows for the prediction of an interval of the respective value.

Thus, quantile regression aims to determine conditional quantile functions based on covariates (features). Unlike least squares regression, which estimates the conditional mean, quantile regression estimates the conditional quantile [84]. By predicting an interval rather than a singular value, quantile regression provides a way to quantify the uncertainty in a model's predictions. This is particularly useful for managing the various uncertainties encountered in the machine learning pipeline, from uncertainties in the data, such as noisy labels, to uncertainties in the model itself.

A quantile refers to a segment of a dataset. For example, in a dataset, the 0.5 quantile (median) divides the data in a way that half the points are above and half are below the quantile. Similarly, the 0.1 quantile indicates that 10% of the data points are below this value and 90% are above. In general, the τ-th quantile means that a proportion τ of the dataset is smaller, and a proportion $(1-\tau)$ is larger [84].

The loss function for quantile regression minimises the sum of absolute residuals. This is achieved by ensuring an equal number of positive and negative residuals. Asymmetrically weighting the absolute residuals enables the prediction of other quantiles. The optimisation problem can be defined as follows:

$$min \sum \rho_\tau(y_i - \xi). \tag{3.17}$$

The function ρ_τ (see Figure 3.3) takes the weighted absolute value of $y_i - \xi$ [84].

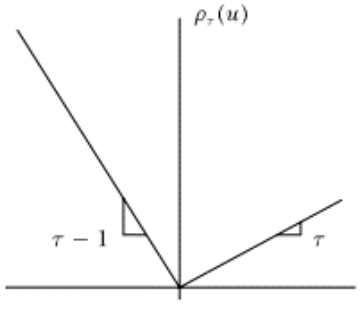


Figure 3.3: Tilted absolute value function ρ to obtain the τ-th quantile [84].

Similarly, the conditional quantiles can be defined, where $\xi(x_i, \beta)$ describes the residual as a parametric function depending on the feature x_i and the parameter β [84]:

$$min \sum \rho(y_i - \xi(x_i, \beta)). \tag{3.18}$$

Quantile regression provides a robust method for managing the aleatoric uncertainties in the data. By quantifying the uncertainty, the predictions become more informative and robust to the imperfections in the data. However, quantile regression does not handle the uncertainties arising from model selection, parameter selection, or missing knowledge (epistemic uncertainty). It is only a data-driven technique.

4

Quantile Regression in the Prediction of the Hysteresis Factor

This chapter will examine the methodology used to predict the hysteresis factor using quantile regression. It will start by introducing the different approaches to the problem and discussing the availability of data. Afterwards, the preprocessing of input is explained in greater detail, mentioning the preprocessing pipeline, feature selection and the transformation to the actual input data. Then, the distribution of the available data is presented. Next, the data labelling process is discussed, followed by a section on the relevant machine learning models. This section explains which models were chosen and the reasoning behind their selection. It will further explain the implementation of the chosen models, how the size of the models was estimated and how the model input data was tuned.

4.1. Approaches to Problem Solving

Different approaches are taken to solve the problem of determining the hysteresis factor based on driving profiles. Each methodology is distinguished by the method of hysteresis factor determination, the form of the input data, and the machine learning model employed. An overview of the approaches and the respective resulting case studies is provided in Table 4.1.

Since the SOC correction only occurs after the cell voltage has stabilised, the hysteresis factor only needs to be determined at that point. Therefore, intermittent determination of the hysteresis factor might suffice. Alternatively, the hysteresis factor can be determined quasi-continuously. This involves determining the factor in regular intervals e.g., every minute.

ID	**CS01**	**CS02**	**CS03**	**CS04**	**CS05**	**CS06**
Case Study	**Attribute-Based Prediction with Simple Models**	**Time Series Prediction with Neural Networks**	**Interval Delta Prediction with Simple Models**	**Interval Prediction with Simple Models**	**Interval Prediction with Neural Networks**	**Autoregressive Interval Prediction with Neural Networks**
Determination of hysteresis factor	Intermitted		Quasi continuous			
Form of input data	Time series attributes (2D)	Time series (3D)	Time intervals (2D)	Time intervals (2D)	Time intervals (3D)	Time intervals (3D)
Example input for signal i(t)	$\max_{t \in T} i(t)$	$i(t), t \in T$	$i(nT_s)$	$i(nT_s)$	$i(t),$ $nTs \leq t < (n+1)T_s$	$i(t),$ $nTs \leq t < (n+1)T_s$
Type of model	Simple model	Neural network model	Simple model	Simple model	Neural network model	Neural network model

Table 4.1: Different approaches and resulting case studies.

The input data can take various forms:

1. Time series attributes, which summarise the time series.
2. The entire time series
3. Intervals resulting from splitting the time series

More details can be found in subsection 4.3.3. At last, the approaches differ in the type of model that is used. subsection 4.6.1 will elaborate on the model choice in greater detail.

4.2. Availability of Data

The data used to train and test the models is taken from a test bench. The test bench enables testing on a battery system level in a dedicated climate chamber, ensuring that measurements can be executed with precision. For each test case, the starting conditions, such as the start SOC or the temperature are fixed. Furthermore, each test description also provides the testing process. This could for example be that the battery should be charged with a current of 1C until the SOC reaches 90%. It is important to know that while a lot of different test cases are available, not all of them are suitable as training or test data. For a test case to be considered, the cells need to be able to relax within the measurement (thus zero-current for a certain period). Whether this relaxation time is included or not is specified in the test description and can be verified by reading one of the measured signals from the BMS. This signal switches to true when the relaxation conditions, indicate the start of SOC correction.

In these test bench measurements, an existing algorithm is used in the BMS to correct the SOC. This algorithm determines the hysteresis factor based on the charge throughput and a manually calibrated parameter. However, this algorithm also has a margin for error and thus can also correct the SOC with a +/- 3% SOC precision.

4.3. Preprocessing of Input Data

In this section, the preprocessing pipeline will be introduced. Following this, the feature selection will be explained in greater detail. In addition, information will be given about the transformation of the input samples to 1-dimensional and 2-dimensional samples. At last an overview of the distribution of data is given.

4.3.1. Preprocessing Pipeline

The first step of the pipeline is extracting the signals from the raw data. This raw data is available in an mf4 file format [85], which can be read in Python using the asammdf library [86]. Using a predefined dictionary which maps the names of the signals to the actual signals for each vehicle project, the relevant signals can be extracted. It is important to note that even within a single-vehicle project, a signal may be known by different names due to evolving naming conventions. This is addressed by iteratively searching through a list of known names for each signal. If signals cannot be found or recreated, the measurement is skipped and no sample is generated.

Due to varying sampling times of the signals, their number of elements differs. To ensure that they have cohesive timestamps, the desired sampling rate is set. New timestamps are then generated, starting at zero and extending to the end of the original measurements. The sampling rate is adopted from the signal "measuring" the hysteresis factor, namely 10 Hz. These timestamps are used to linearly interpolate the signals.

The next step is the conversion of signals, if necessary, such as converting voltage signals from mV to V. This ensures uniformity in units, whether the signals are used directly or as input to calculate more complex features.

Certain signals might not always be available in all measurements. In this case, the missing signals are, if possible, generated. For example, the current and time signals can be used to calculate the net integrated current. If generating a signal is not feasible, a warning is given and the measurement as a whole is skipped due to missing signals.

In the next step, the extracted signals are segmented. One split begins either at the very start of the measurement or after the SOC correction is completed (if a measurement starts with a relaxation period) and

it ends when the next SOC correction occurs. If the current in one split is 0 for the whole time, the split is ignored. This process is illustrated in Figure 4.1.

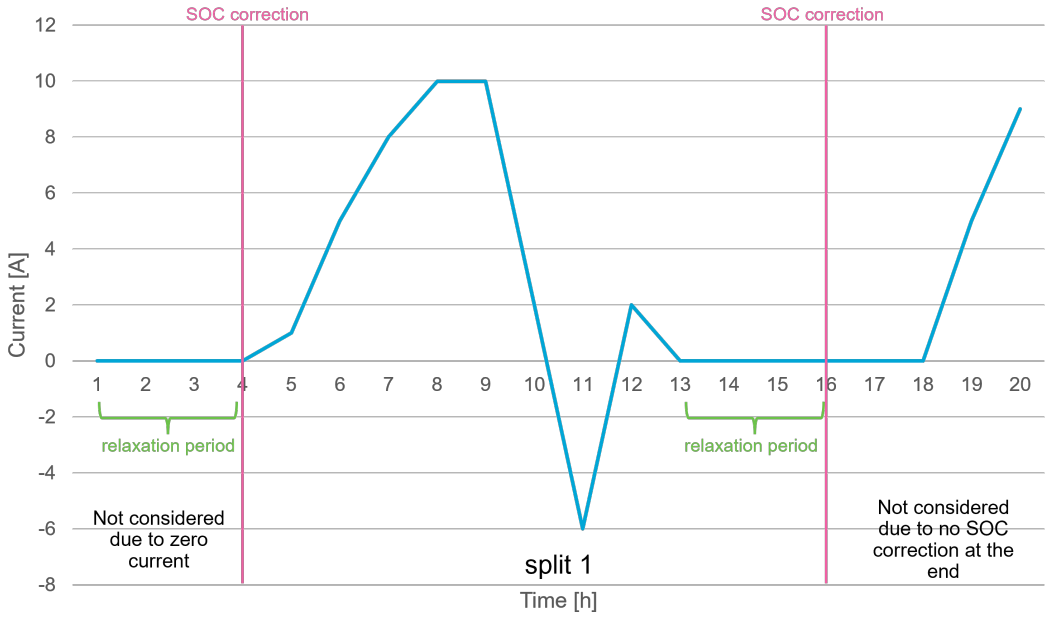


Figure 4.1: Illustration of the segmentation of measurements into samples.

After segmenting the measurements, each of the splits can be labelled. The data of each split is then saved in a Parquet format [87], which minimises memory usage. The entire preprocessing pipeline is depicted in Figure 4.2.

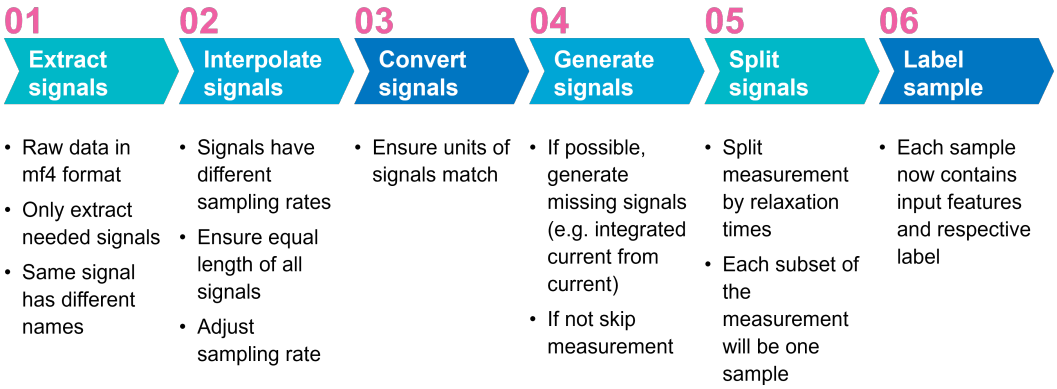


Figure 4.2: Overview of the preprocessing pipeline.

Model/ Feature	Previous hysteresis factor	Current	Capacity	SOC	OCV hysteresis curves	OCV voltage
Zero-state model (Plett)	x	x				
Roscher model		x	x			
Xie model (2016)		x			x	
Ko model		x	x	x		
Xie model (2023)	x	x	x	x		
One-State model (Plett)				x	x	
Baronti model				x	x	
Dong model				x	x	
Two-state model (Kim)				x		x
Antony model		x	x			
Zhou model				x	x	x
Yu model						x
Gao model					x	x
Li model				x		x
Xu model		x				x

Table 4.2: Relevant features according to different hysteresis models.

4.3.2. Feature Selection

To identify the relevant features influencing the hysteresis factor, a literature review was conducted, and internal experts at Porsche Engineering were consulted.

An overview of the introduced models and their respective signals that are part of the models can be found in Table 4.2. Furthermore, it should be noted that a lot of the models used further artificial parameters, which were identified with methods such as HTTP tests. Therefore, these parameters, and thus the models, indirectly depend on the current and SOC), as they were used to determine the parameters. Additionally, because of the definition of Coulomb counting, the SOC is also influenced by the current. From Table 4.2 it becomes evident that current is a significant factor and cell voltage may also play an important role.

Features influenced by the hysteresis factor and thus the current algorithm, such as SOC and capacity, were not considered, to minimise the influence of the existing algorithm. As the OCV hysteresis curves in the literature models were only used when the model predicted the OCV voltage (serving as a starting point for subsequent calculations of the change in voltage), they were not considered as this work determines the hysteresis factor directly. Instead, the previous hysteresis factor was considered. "Previous" refers to the hysteresis factor determined at the end of the last measurement, indicating the hysteresis factor at the beginning of the ongoing measurement.

Additionally, as some models from the literature indirectly considered temperature by performing parameter identification under different temperature conditions (see Figure 1.2), cell temperature was also included as a feature.

Therefore, the following signals were considered as potential features:

- Current
- Cell voltage
- Cell temperature
- Previous hysteresis factor

Note that current refers to the battery current as this should be approximately equal to the current going through the cell. Furthermore, the cell voltage and cell temperature measurements are taken from the first cell of the battery pack.

4.3.3. Transformation to Model Input Data

Each Parquet file from the preprocessing pipeline contains the time series corresponding to a specific measurement. Note that each measurement may differ in length. If the active period of a measurement, so the pre-relaxation period, was under 10 s, the measurement was not considered. Depending on the problem

approach (see Table 4.1) taken, the model input format varies. The following subsections will explain the transformation process in greater detail.

Once the input data is transformed into the right format, all features are linearly normalised to be in a range between -1 and 1 using a MinMax normalisation. For the two-dimensional input data, the scikit-learn MinMaxScalar [88] was used. For three-dimensional input data, a custom scaler based on the MinMaxScaler was utilised to accommodate the normalisation of the entire time series for each sample and feature. Scaling accelerates convergence [89] and features with different scales, such as the hysteresis factor which naturally cannot exceed 1 and the integrated current which is significantly larger can be regarded as equally important by the models [90].

Time Series Attributes

This approach is specifically for simpler models. Since these models are capable of processing only one-dimensional features and not entire time series, the idea is to describe the time series using specific attributes. Python libraries like TSfresh [91] can be used to extract time series characteristics. However, the extracted characteristics were limited for two main reasons: an excessive number of features and suitability to run on an electronic control unit.

As these time series characteristics must be determined for every relevant signal, the number of total features increases not only with the number of characteristics but also with the number of signals. Furthermore, not all characteristics are suitable for BMS implementation. Many characteristics require variables, such as the standard deviation of the time series, which can only be determined if the entire time series is available. Given that the battery management system does not have a lot of RAM, long time series, especially those with a high sampling rate, cannot be stored. Instead, it is better to capture these variables using a counter, such as the mean, where one counter sums all values and another counter keeps track of the number of values during runtime.

The final chosen characteristics can be seen in Table 4.3. Note that x_1 refers to the first element of the time series and n is the last active element of the time series thus the last element before the start of the relaxation period. $\tilde{n}$ refers to the last point of the entire time series, thus the time step immediately before SOC correction. As different time horizons can be chosen (e.g. the whole time series, the last 10 active minutes, etc.) the start point m can vary.

Time Series

As neural networks can handle two-dimensional input features such as time series, it was possible to supply the complete time series to the model. However, the time series resulting from preprocessing could not immediately be used due to their high sampling rate which would result in too many timesteps for the network to process and for the BMS to save. Furthermore, the training process required the input to have the same number of time steps for each sample.

One option was to standardise the length of all time series, for instance, by considering only the last 10 active minutes, as illustrated in Figure 4.3a. Alternatively, different sampling rates could be employed, meaning that the interval between timesteps would be longer for longer measurements and shorter for shorter measurements, as depicted in Figure 4.3b. Both options result in the same number of elements per sample. All features are listed in section A.4.

(a) Option 1 of time series features: All samples (blue) have the same length.

(b) Option 2 of time series features: All samples (blue) may have different lengths

Figure 4.3: Options to have the same number of time steps with measurements of different lengths.

Name	Mathematical definition
Minimum value	$\min_{i=m,\dots,n} x_i$
Maximum value	$\max_{i=m,\dots,n} x_i$
Absolute minimum value	$\min_{i=m,\dots,n} \lvert x_i \rvert$
Absolute maximum value	$\max_{i=m,\dots,n} \lvert x_i \rvert$
Sum of absolute changes	$\sum_{i=m}^{n-1} \lvert x_{i+1} - x_i \rvert$
Mean of changes	$\frac{1}{n-m-1} \sum_{i=m}^{n-1} x_{i+1} - x_i$
Mean of absolute changes	$\frac{1}{n-m-1} \sum_{i=m}^{n-1} \lvert x_{i+1} - x_i \rvert$
Absolute energy	$\sum_{i=m}^{n} x_i^2$
Sum of values	$\sum_{i=m}^{n} x_i$
Mean	$\frac{1}{n-m} \sum_{i=m}^{n} x_i$
Complexity	$\sqrt{\sum_{i=m}^{n-1} (x_{i+1} - x_i)}$
Sum of positive values	$\sum_{i=m}^{n} x_i \, , where \; x_i > 0$
Sum of negative values	$\sum_{i=m}^{n} x_i \, , where \; x_i < 0$
Mean of positive values	$\frac{1}{n-m} \sum_{i=m}^{n} x_i \, , where \; x_i > 0$
Mean of negative values	$\frac{1}{n-m} \sum_{i=m}^{n} x_i \, , where \; x_i < 0$
Number of zero elements	$\sum_{i=m}^{n} \mathbb{I}(x_i = 0)$
Root mean square	$\sqrt{\sum_{i=m}^{n} x_i}$
First value	x_1
Last value	$x_{\tilde{n}}$
Previous hysteresis factor	φ_1

Table 4.3: Time series attributes and their mathematical formulation.

Time Intervals
Another approach is to split the whole time series into time intervals. For the simple models, which require two-dimensional features (one dimension for samples and one for features), the interval between $t-1$ and t can be described by the signal values at t, and the change in the signal and the average of the signal within the interval. Additionally, if the time horizon expands beyond the interval, previous periods can also be considered. This is shown in Figure 4.4a.

When supplying data to neural networks capable of handling time series input, the data can be structured in three dimensions (time, feature and sample). In this case, it is possible to input the entire time series between $t-1$ and t. In addition to selecting the interval size, it is also possible to adjust the sampling rate of the signals within that interval. Given that each sample is more complex (containing an entire time series) than in the two-dimensional interval approach, past intervals are not directly considered. If a 5-minute interval and the preceding interval are to be considered, the interval size can be set to 10 minutes. An illustration of this, with a sampling rate of 8 elements per interval, is provided in Figure 4.4b. An extensive list, explicitly listing all features used for the interval approach can be found in section A.4. As each interval can be seen as a sample, each interval also requires its own label.

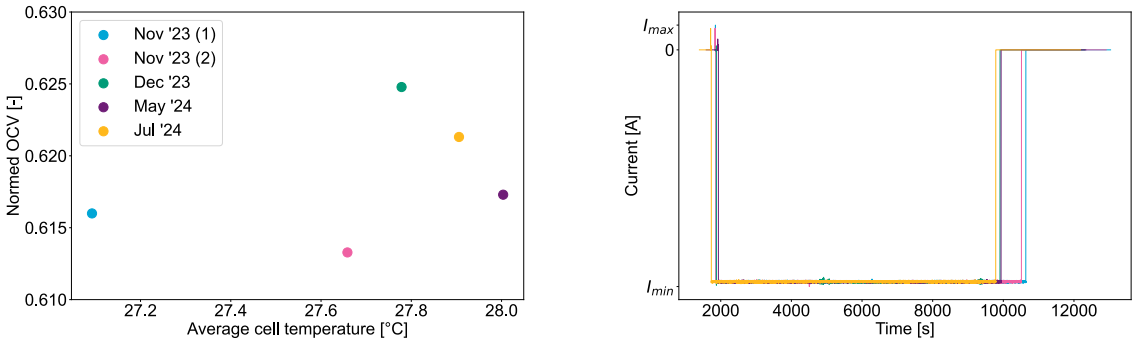
(a) Time interval for two-dimensional input.

(b) Time interval for three-dimensional input.

Figure 4.4: Difference between time interval approach for two-dimensional and three-dimensional input.

4.4. Distribution of Available Data

After properly preprocessing the input data, it was possible to examine the data distribution to assess its diversity. It is important to note, that there is a predefined set of test cases as mentioned in section 4.2. For the initial training, a vehicle project with 10 months of testing data was selected, referred to as "vehicle project A". The same type of test may be done multiple times, sometimes even in the same month. While the current profile and other parameters are similar there are minor differences. Figure 4.5 provides an example of this.



(a) Temperature vs open-circuit voltage (normed).

(b) Current profiles.

Figure 4.5: Same test type in different measurements, denoted by the month the measurement was recorded. The legend in (a) indicates which colour corresponds to which month.

Figure 4.5 shows five measurements, all of the same test type, denoted by the month in which the measurement was taken. Note that the first two measurements are from the same month. Figure 4.5a illustrates minor variations in temperature between the months, with the temperature ranging from 27.1°C to 28°C. The OCV was normalised based on the maximum and minimum values specified for the cell. Figure 4.5b displays the current profiles applied in the measurements. As seen, the measurements differ in length and start of the current pulse. While the overall current profile remains consistent, the measurements are shifted in time, each with a different pre-pulse period and some containing a small pulse in the opposite direction at the start.

It was decided to utilise all available data, even if some measurements are similar, to ensure sufficient training data and to account for the subtle differences in the various measurements of the same test.

Figure 4.6 illustrates the lengths of each measurement, including the inactive period, known as the relaxation time. The measurement lengths vary significantly, ranging from just a few hours to over 200 hours. The average length is 30.42 hours. Note that these lengths correspond to the active period and the relaxation period. Long measurements result from multiple pulses with rest periods in between. However, relaxation conditions are not met during these intermittent rest periods but are only met at the end of the measurement.

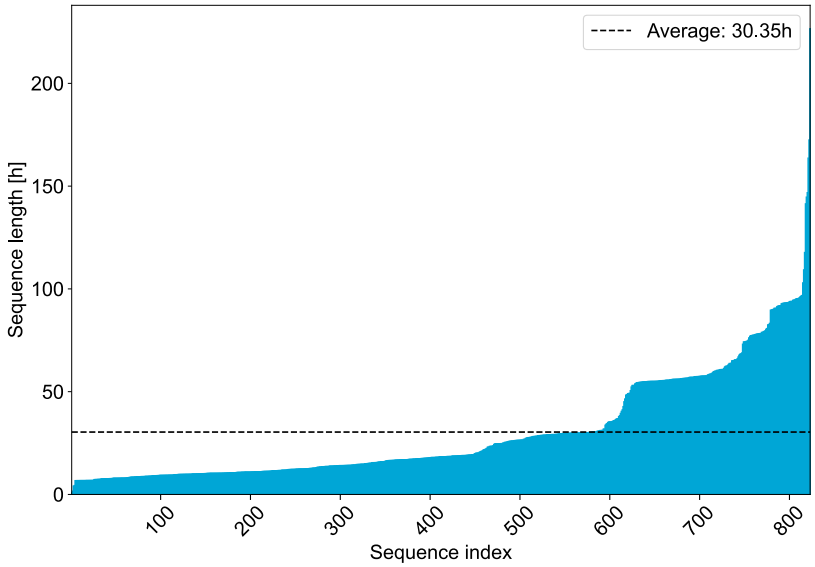


Figure 4.6: Distribution of sequence lengths of measurements.

Figure 4.7 shows the distribution of the average cell temperature versus the measured OCV scaled according to the minimum and maximum values (cut-off voltages) specified for the cell.

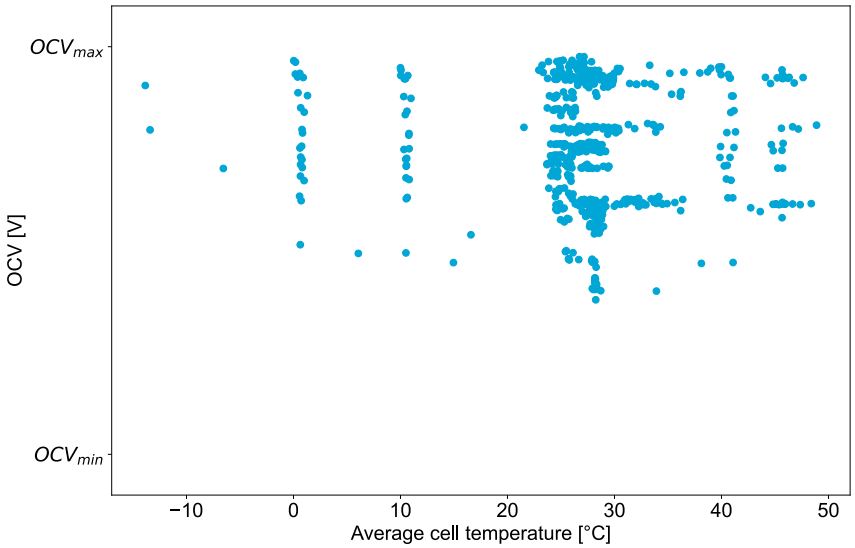


Figure 4.7: Average cell temperature vs normed OCV.

There are relatively few measurements near the minimum cut-off voltage. The cut-off voltages vary with temperature, and in this case, the lowest cut-off voltage, defined for high temperatures, is used. At 20°C, the lower cut-off voltage would be higher. It can be seen that the majority of measurements fall within a temperature range of 20°C to 35°C. This range also corresponds to the predominant use case for typical customer applications. For this reason, it was decided to limit the samples used for the initial training to this temperature range.

Figure 4.8 displays the minimum and maximum charging current of each measurement. Since each measurement includes a relaxation period, the minimum current will be negative, representing the discharging current, while the maximum current will be positive, corresponding to the charging current. The horizontal black line indicates a maximum current of 0A, meaning that measurements on this line only have a negative current and are thus discharging experiments. Similarly, the black vertical line indicates a minimum current of 0, which means that measurements on this line are charging experiments. There is a significant number of charging experiments. Approximately 49% of the measurements lie on the charging line, while 30% lie on the discharging line. The majority of charging experiments reach a maximum current of about 150A, which corresponds to a charging power of up to 120kW. Measurements exceeding -800A in minimum current indicate high power use, such as fast acceleration.

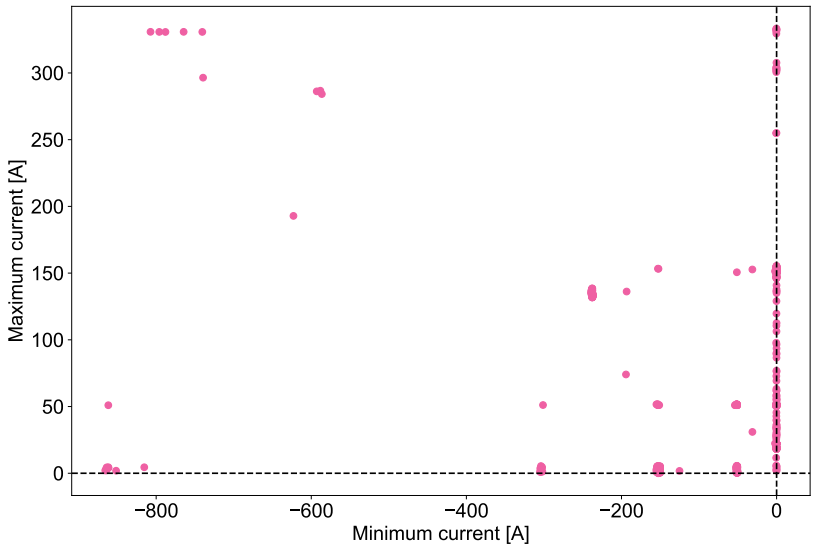
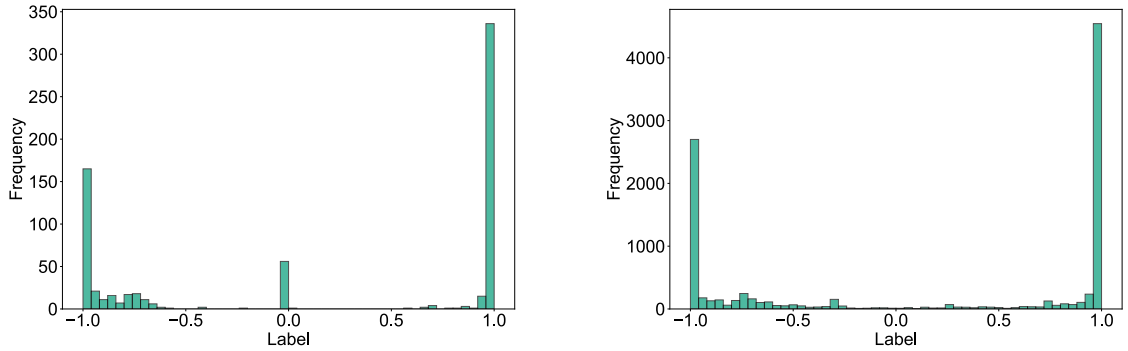


Figure 4.8: Minimum and maximum current of measurements with charge-only and discharge-only measurements indicated by the black dotted lines.

Figure 4.9a displays a bar plot of the labels if each measurement is taken as a sample (as done in the time series attributes and time series approaches). The majority of labels are clustered around the extremes of -1 and 1. This is logical given that most test cases end with either significant charging or discharging, which drives the hysteresis factor to these extreme values. In real-world scenarios, the hysteresis factor is also likely to be at these points. However, the intermediary space must be covered as well. For instance, in a test case where the vehicle undergoes regenerative braking and then immediately parks, resulting in the measurement ending with neither extreme charge nor discharge.

If the measurements are further divided into intervals (as done in the time interval approach), the majority of labels still cluster around the extremes, as shown in Figure 4.9b. However, there are now more samples available in the intermediary space. Note that Figure 4.9b was created for a dataset where each interval has a length of 10min. These plots also demonstrate that the time interval approach yields a significantly higher number of measurements.

More information about the second available vehicle project, vehicle project B, can be found in section A.5.



(a) Each measurement is one sample.

(b) Each measurement interval is one sample.

Figure 4.9: Distribution of labels for different approaches.

4.5. Labelling

Since the algorithm's objective is to predict the hysteresis factor, each sample must be labelled with the "true" hysteresis factor. However, as the hysteresis factor is defined by the charge and discharge OCV curve (SOC/OCV look-up) given by the cell manufacturer, the measured OCV and the estimated SOC, the ground truth hysteresis factor is unknown. There are two different methods of obtaining labels for the hysteresis factor: Domain knowledge labelling and current algorithm labelling.

4.5.1. Uncertainty Considerations

Samples cannot simply be labelled without considering the uncertainty of the SOC. As mentioned in section 1.3, the SOC cannot be measured since it is not a physical quantity itself but rather the result of accumulated charge and discharge in relation to the capacity of the cell. Given an accurately determined cell capacity, minimal measurement equipment errors, and relatively short measurement durations, Coulomb counting can provide a reasonable approximation of the true SOC. However, these conditions are not always met outside of a laboratory setting. Furthermore, for cells exhibiting voltage hysteresis determining the true SOC after relaxation of the cell using the SOC/OCV look-up tables is not feasible as these are not unique for each OCV value anymore. Therefore the aim was to find the best labels possible, recognizing that these are not ground truth values. Instead, the labels are assumed to contain instance-dependent noise. Therefore this is a problem of inaccurate supervision. Two methods of labelling are used. For samples eligible for domain knowledge labelling, the noise is expected to be minimal to non-existent and for samples relying on current algorithm labelling, the noise level is higher, influenced, for instance, by the length of the measurement.

4.5.2. Domain Knowledge Labelling

In this case, the label is determined based on known characteristics of the signals. If during one measurement the cell is either exclusively charged or exclusively discharged for at least 70% delta-SOC, the hysteresis factor can be definitively labelled as 1 or -1. This is because a hysteresis factor of 1 corresponds to the value of the charging curve (which was obtained by incrementally charging the battery) and -1 corresponds to the discharging curve (which was obtained by incrementally discharging the battery). The minimum charging hub of 70% delta SOC was set based on prior hysteresis measurements of the cells done by Porsche. These measurements analysed the amount of charge necessary to transition from the charge curve to the discharge curve and vice versa.

4.5.3. Current Algorithm Labelling

An existing algorithm, which determines a hysteresis factor based on the current throughput is currently in use in the BMS. Within this algorithm a parameter is used that must be calibrated for every vehicle project, thereby making the process time-consuming.

By using the signal that corresponds to the continuously determined hysteresis factor, a value between -1 and 1 that is sampled every 0.1 s, it is possible to obtain the label for the samples.

4.6. Machine Learning Models

This section provides a detailed discussion of the machine learning models employed in this work. First, it will elaborate on the selection process, explaining the reasoning behind the chosen models. Then, the implementation details for each model are described in depth. Following this, the estimation of the model size is addressed. Finally, the tuning of the input data will be discussed.

4.6.1. Selection of Suitable Models

Due to the time constraints of this work, it was decided to compare three different types of models. The models were categorised into two categories, the "simple models" referring to models which can only process two-dimensional inputs, and the neural network models, which can process three-dimensional inputs.

For the simple models, various options are considered such as linear regression models, polynomial regression models, (extreme) gradient boosting models decision tree models and random forests. After careful consideration, a linear regression model and an XGBoost model were chosen. The linear regression model is advantageous due to its simplicity, which ensures feasibility for implementation in the battery management system (BMS), where computational power is limited. Furthermore, the determined weights are straightforward to interpret, which in turn makes it easy to identify the most relevant features. An existing implementation of quantile regression in the scikit-learn package further supports its use. Generally, a linear regression model provides a robust baseline.

The second selected model is the XGBoost model. This model typically performs well with larger datasets which is necessary for the "time interval" approach. Unlike the linear regression model which assumes linearity in its parameters, XGBoost can also learn non-linear relations, allowing for the modelling of more complex interactions. Moreover, there is also an existing implementation of quantile regression for this model.

Finally, a more complex model, a neural network, was chosen. Given the nature of time series data as input, models such as a GRU or an LSTM are intuitively suitable. A GRU was chosen for this work. GRUs have fewer parameters to train compared to LSTMs because the input and forget gates are combined. This also leads to faster training time. The reduced number of parameters is also beneficial for implementation in the BMS, considering its memory constraints.

4.6.2. General Implementation Considerations

Before training the models, the available samples must be split into a training set, a validation set (used for tuning) and a test set (used for evaluating the tuned model). In this work, an 80/10/10 split was chosen.

Special attention was paid to the splitting of the "time interval" datasets to maintain the integrity of individual measurements. It was essential to keep the intervals of the same measurement together and know the order of intervals. This avoids the fragmentation of the measurement, where one interval of it might be in the test set and one might be in the train set.

Preserving this chronological context is beneficial for subsequent testing phases. There are two different approaches for testing: a non-autoregressive approach and an autoregressive approach. In non-autoregressive testing, each interval within a measurement is tested independently which means the resulting hysteresis factor from one interval does not influence the next. This means that the feature "previous hysteresis factor" is given by the test dataset. In contrast, the autoregressive approach is more realistic for practical implementation in the BMS. The hysteresis factor determined in the previous interval serves as the feature "previous hysteresis factor" for the subsequent interval.

Given that measurements vary in length and, consequently, the number of time intervals, the 80/10/10 split refers to the division of measurements, not individual samples. For the other approaches, each sample corresponds to a measurement and the splitting process is thus more straightforward.

In all models, the 0.05, 0.5 and 0.95 quantiles were determined. This allows the analysis of the upper and lower tail of the distribution as well as the median value. The median value, in particular, is most useful for implementation in the BMS, as it can be a good approximation of the hysteresis factor.

Parameter	Range
L1 regularization	[0, 0.001, …, 0.999, 1]

Table 4.4: Parameters to be tuned and their respective tuning range for the linear regression model.

Parameter	Range
Learning rate	[0.001, 0.005, 0.01, 0.05, 0.08, 0.1]
Maximum depth of tree	[3, 4, 5]
Minimum child weight	[1, 2, 3, 4]
Subsample (ratio of shuffled samples used to train model)	[0.6, 0.7, 0.8, 0.9, 1]
Colsample by tree (ratio of shuffled features to be considered)	[0.7, 0.8, 0.9]
L1 regularization	[0.1, 1, 10]
L2 regularization	[0.1, 1, 10]

Table 4.5: Parameters to be tuned and their respective tuning range for the XGBoost model.

4.6.3. Implementation of Quantile Regression in a Linear Regression Model and XGBoost

For both simple models, existing implementations of quantile regression were used as a foundation. This section will outline the procedures for training, tuning, and testing these models.

Implementation Training Routine

For the linear model, scikit-learn's QuantileRegressor [92] was used. This is a linear regressor which aims to minimise the pinball loss for a given quantile α. Additionally, L1 regularization is inherently built into this model. For each quantile, a separate model has to be trained. After initializing the QuantileRegressor with optimal parameters found during tuning and the respective quantile, the regressor can be fitted to the training dataset.

The same is done for the XGBoost model [93]. However, the training data needs to be presented as a QuantileDMatrix object. Similarly to the linear regression model, the booster can then be trained for the respective quantile using the optimal parameters.

Implementation Tuning

The tuning of the simple models was done using scikit-learn's RandomizedSearchCV class [82]. Since linear regression only has one parameter to be tuned, namely the L1 regularization, only a single parameter was optimised. The parameter grid was defined between 0 and 1 in increments of 0.001 (see Table 4.4). Given that the linear regression model is fairly slow with large datasets the number of parameter settings to be sampled was set to 10. The default cross-validation strategy of 5 was chosen, meaning the input data is split into 5 and in each iteration, one of these five folds is chosen as a validation set. The performance metric is then the average over the results of all of these 5 combinations. Note that for tuning the chronological order of intervals in the time interval approach was disregarded.

For the XGBoost model, more parameters could be tuned. Table 4.5 lists each parameter and its respective range. After initializing the booster for the respective quantile, with 2000 boosting rounds and an early stopping criterion of 10 rounds, the random search is done over the parameter grid. This early stopping criterion was also later used when training.

It is important to note that this tuning method may not guarantee optimal results due to its non-deterministic nature. The minimal validation loss is identified using non-autoregressive testing due to time limitations. This is particularly relevant for time interval methods as for the remaining approaches, non-autoregressive and

autoregressive testing yield comparable results.

Implementation for a Custom Testing Routine
For the "time attributes" and "time series" approach, the model was evaluated post-training by determining the 0.05, 0.5, and 0.95 quantiles and calculating the corresponding pinball loss. Additionally, the predicted median was plotted against the actual labels for each sample. Note that in these approaches, each sample corresponds to one measurement.

For the time interval approach, a custom testing routine was designed to also consider the chronology of samples belonging to the same measurement. First, the model was tested non-autoregressively. This means the test routine iterated through all samples of a measurement, predicting the quantiles using the respective trained regressors, and accumulating the overall loss before averaging it. The quantiles were then plotted over time. This process was repeated for all measurements in the test dataset.

The autoregressive test routine iterated through all samples of one measurement, accumulating the loss. In this process, the resulting hysteresis factor of the previous interval was used to overwrite the feature "previous hysteresis factor" for the next interval. As before, the results for each quantile were plotted over time.

4.6.4. Implementation of Quantile Regression in a Gated Recurrent Unit Model

The following subsection explains the implementation of the quantile regression model in a GRU. It differentiates between a non-autoregressive training approach and an autoregressive training approach, addresses the model tuning, and describes the testing process. The GRU model was implemented using the neural network library PyTorch because it offers flexible tools to build neural network architectures.

Implementation Of A Non-autoregressive Training Routine
To train the GRU model a loss function must be specified. As there are no ready-to-use implementations of the quantile regression available in PyTorch itself, the TensorFlow implementation [94] of the pinball loss was modified to fit into the PyTorch framework. The implementation of Equation 3.18 is as follows:

$$L = \frac{1}{N}\frac{1}{|Q|}\sum_{i=1}^{N}\sum_{\alpha\in Q}\max\left\{\alpha\cdot(y_{\text{true},i}-y_{\text{pred},i}),(\alpha-1)\cdot(y_{\text{true},i}-y_{\text{pred},i})\right\}. \tag{4.1}$$

Note that α represents the tensor of quantiles to be determined with dimensions (number of quantiles,). y_{true} is the tensor containing the labels with shape (batch size, 1) and y_{pred} is the tensor containing the predictions made by the model with dimensions (batch size, number of quantiles). N represents the total number of batches and Q describes the set of quantiles to be determined. Therefore the overall loss L is determined by summing the loss for each batch element and quantile.

After defining the loss function, the rest of the training loop can be implemented. The optimised parameters for the model are loaded and the model is initialised. The model consists of a certain number of GRU layers and a linear output layer. Finally, the output of the model is transformed using a tanh function to ensure predictions remain within the boundaries of -1 and 1. The source code for the model can be found in section C.1. Furthermore, the criterion needs to be specified using the custom pinball loss function from Equation 4.1 and the optimiser must be initialised with the corresponding optimal learning rate. Adam was chosen as the optimiser due to its efficiency especially when dealing with large datasets [95]. The training and validation datasets are put into DataLoaders, which split the data into batches. When the batches are loaded from the DataLoader it is also ensured that their order is randomised. The batches contain a fixed number of samples.

For each epoch and each training batch, the model is trained in a forward pass. Next, the loss is computed using the predictions from the forward pass. Then, the gradients of all optimised parameters are reset. In the next step, the backward pass is executed, determining the gradients of the loss with respect to the model parameters. The final step is to update the parameters based on those that minimise the loss. For each epoch, the model is also evaluated on the validation set.

This training approach is referred to in this work as non-autoregressive training because each sample is treated individually. When the input data is in the form of time intervals, this ultimately leads to each interval being shuffled. Consequently, the model is not trained consistently on complete measurements which would

involve a certain number of intervals in a specific order. Additionally, the intervals do not influence one another as they are treated completely independently.

Implementation of an Autoregressive Training Routine

If the input is in the form of time intervals, the non-autoregressive training may cause issues for the following reasons. In real-life applications, for certain time intervals, the hysteresis factor would be predicted. This factor is then used as an input for the next interval since the previous hysteresis factor is a feature. Thus, errors might propagate, especially if the model is trained in a way that does not account for the fact that previous errors affect the loss of later intervals.

To overcome this, an autoregressive training routine was implemented based on the general idea of loss and gradient accumulation. Typically gradient accumulation is used when computational sources are limited, as accumulation over multiple batches has the same effect as taking greater batch sizes, which requires more memory [96]. In this case, the idea is to accumulate the gradients and loss over an entire measurement, thus only resetting the gradients and updating the model parameters at the end of a measurement instead of after each interval. This approach aims at maintaining the connection between intervals of the same measurement, rather than reducing memory usage.

Similar to non-autoregressive training, the model must be initialised with the optimal parameters, and the criterion and optimiser must be set before actual training. The first major difference lies in the training DataLoader. In non-autoregressive training, samples were randomly placed into batches, which the model then trained on. In this case, as illustrated in Figure 4.10, each batch is ensured to contain intervals from a certain number of measurements (determined by the batch size). Furthermore, to maintain consistency of a measurement, the intervals of the measurement are in their original order (see blue boxes in Figure 4.10). During training, the algorithm iterates through the intervals of a batch, starting with the first interval and ending with the last. All measurements within a batch must have the same length to enable parallel processing. Therefore, shorter measurements are padded with zero values (see the dark blue box in Figure 4.10). The corresponding labels for these padded intervals are set to -2, a value that is outside the range of possible hysteresis factor values. Based on the labels, a mask is created for each batch. If the label is outside the possible range (i.e. -2), the mask ensures that these padded intervals are not considered for the total loss of the measurement.

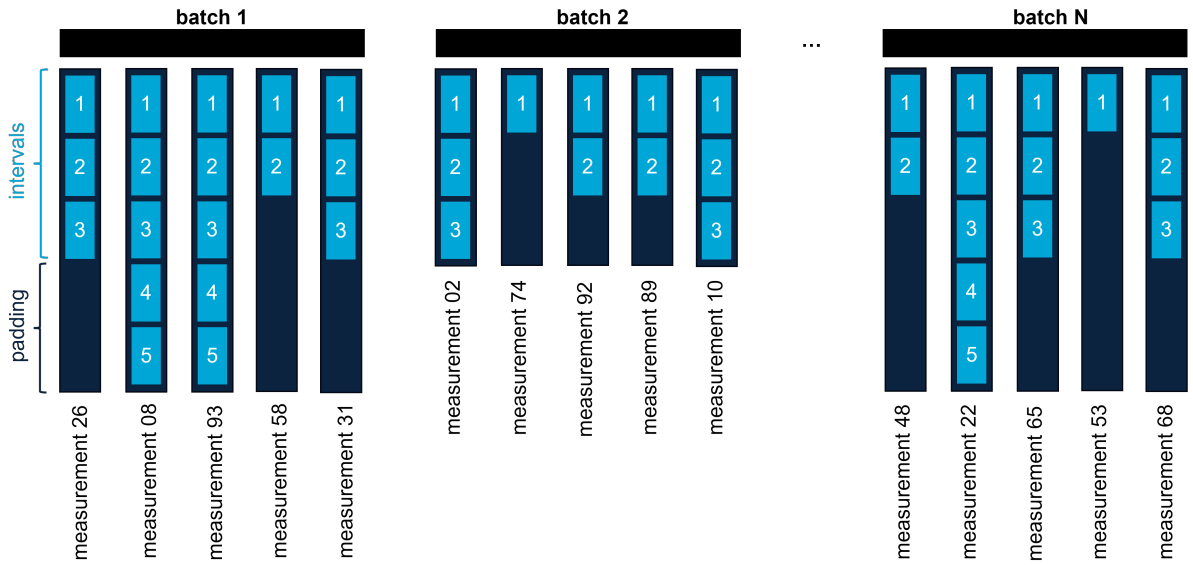


Figure 4.10: Structure of batches for autoregressive training.

For each batch, the training algorithm iterates through the intervals (for multiple measurements in parallel). Similar to non-autoregressive training the forward pass is executed. The loss for the interval is then calculated using the mask, the labels, the predictions from the forward pass, and the length of the actual measurement (excluding padding) to normalise the interval loss. The measurement loss, which accumulates the total loss over the whole measurement is updated and used to do backpropagation. Finally, the predicted hysteresis

Parameter	Range
Number of layers	[1, 2]
Hidden size	[2, 4, 8, 16, 32, 64]
Learning rate	[1e-4,...,1e-1]
Batch size	[8, 32, 128, 256]

Table 4.6: Parameters to be tuned and their respective tuning range for the GRU model.

factor is used in the next iteration to overwrite the "previous hysteresis factor" feature in the input. If the end of a measurement is reached, the model parameters are updated and the gradients and loss are reset.

An illustration of the difference between auto-regressive and non-autoregressive training is shown in Figure 4.11. The figure demonstrates how the two training approaches differ in terms of when the model is updated and how, in the autoregressive approach, the intervals are interconnected.

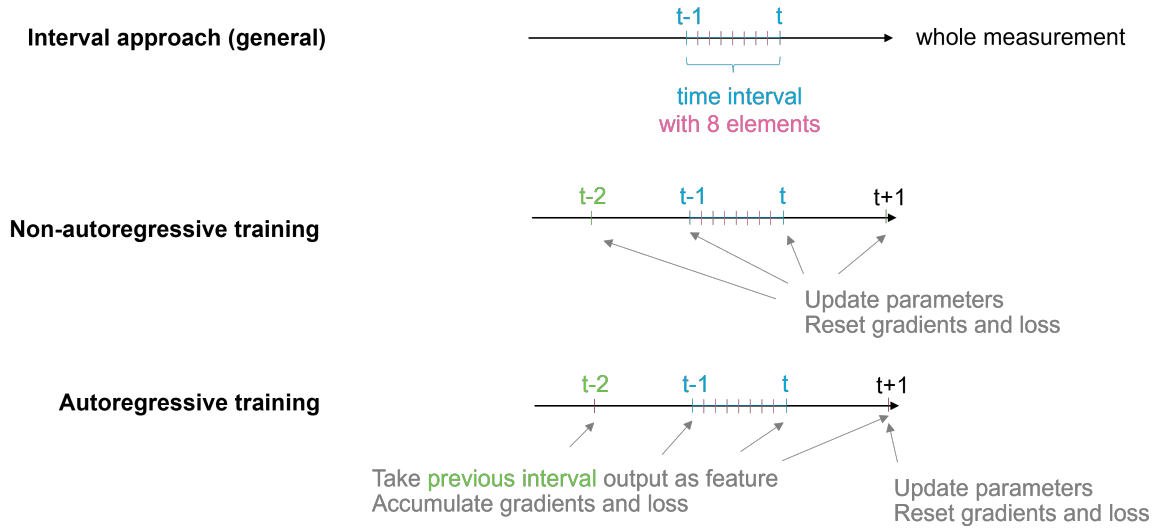


Figure 4.11: Comparison of non-autoregressive and autoregressive training.

Tuning of the GRU

To tune the PyTorch GRU the Ray Tune tool was used. The framework was adapted from the example given by PyTorch [97]. Table 4.6 displays the parameters of the GRU to be tuned. Prior to the actual tuning, the ASHAScheduler was initialised to specify that the loss should be minimised. This scheduler helps with the allocation of computational resources during tuning by terminating, pausing and closing trials as well as modifying the hyperparameters of an active trial [98]. The number of tested configurations was set to 25 and the number of epochs in the test runs was limited to 10 to ensure time-efficiency.

Do note that this tuning approach may not yield optimal results as it is not a deterministic process. Also, the optimal results are found by checking the minimal validation loss which is obtained with non-autoregressive testing (this matters for the time interval approaches, for the other approaches non-autoregressive testing and autoregressive testing are the same).

Implementation Of Testing Routine

The implemented testing procedures were done similarly to the simple models. In the case of the series approach, the model was used to make predictions for the test set. These predictions were then used to determine the test loss and to plot the median values against the labels.

For the interval approach, both non-autoregressive and autoregressive testing routines were implemented. The actual implementation is very similar to that of the simple models, with the primary distinction being that the training data is now three-dimensional rather than two-dimensional.

4.6.5. Estimating Required Memory

Given the importance of memory requirements in evaluating the feasibility of implementing the models in the BMS, this section explains how these requirements were determined. Note that it is assumed that the models are trained offline, before implementation in the BMS. Consequently, the memory needed is only that required to obtain the hysteresis factor during testing, where the test sample size is one, corresponding to a single BMS receiving signals from the vehicle. It is assumed that each parameter and variable has a size of 4 bytes (32-floating point). Also, note that the mathematical operations used to determine intermediate results are saved in the cache and thus do not consume RAM.

Linear Regression Model

Estimating the size and memory requirements for the linear regression model during vehicle application is relatively straightforward. According to Equation 3.1, the hysteresis factor is determined by,

$$\psi = \omega_0 + x_1 \cdot \omega_1 + x_2 \cdot \omega_2 + ... + x_N \cdot \omega_{N_{features}}. \tag{4.2}$$

Note that x_i is the value for feature i. The trained weights w_i can be stored in ROM, as they are once trained and subsequently applied:

$$\text{ROM} = 4\text{byte} \cdot N_{features}. \tag{4.3}$$

The required RAM has to store the actual output ψ and the inputs x_i, thus N+1 objects:

$$\text{RAM} = 4\text{byte} \cdot (N_{features} + 1). \tag{4.4}$$

XGBoost Model

The prediction of the XGBoost model is defined as,

$$\psi = \hat{\psi}_{init} + \eta \cdot r\hat{e}s_1 + \ldots + \eta \cdot r\hat{e}s_n \tag{4.5}$$

Here $r\hat{e}s_i$ represents the predicted residual of tree i and η denotes the learning rate. Each tree comprises a certain number of nodes N_{nodes} (which are used to compare the input of feature x, with a threshold, thereby enabling traversal through the tree). Additionally, each tree has a number of leaves N_{leaves}. The number of leaves and nodes for each tree can be extracted using the built-in dumping function of XGBoost [99].

All static parameters can be stored in ROM, including the initial hysteresis factor $\hat{\psi}_{init}$, the learning rate as well as the node thresholds and the leaf values for every tree:

$$\text{ROM} = 4\text{byte} \cdot (2 + \sum_{i=1}^{N_{trees}} N_{i,leaves} + N_{i,nodes}). \tag{4.6}$$

In RAM, the predicted residuals $r\hat{e}s_i$ for each tree need to be stored. Moreover, the inputs for the trees which consist of $N_{features}$ elements and the final prediction of the hysteresis factor ψ must also be stored:

$$\text{RAM} = 4\text{byte} \cdot (N_{trees} + N_{features} + 1). \tag{4.7}$$

Form of input data	Attributes	Time series	Time intervals	
Dimension	2D	3D	2D	3D
Frequency ***(how often ψ is determined)***	once	once	Every 10s Every 1min Every 5min Every 10min	Every 10s Every 1min Every 5min Every 10min
Time horizon ***(how far to look back)***	Everything 10min 5min 1min	Everything 10min 5min 1min	Current period Last period Last 2 periods	Current period
Sampling rate ***(how are signals sampled)***	Original rate	10 elements 60 elements 120 elements 240 elements	Original rate	10 elements 60 elements 120 elements 240 elements

Table 4.7: Aspects of input data to modify.

GRU Model

The size of the GRU models was estimated using the library torchinfo [100]. This library provides a model summary which estimates the required parameter size, corresponding to the ROM estimation. Furthermore, it determines the input size and the memory needed for the forward and backward pass of a given input. For this purpose, a test tensor was formulated simulating a single sample (as it would be in a BMS). These two estimates can then be summed up for the RAM estimation. It is assumed that the memory required for the backpropagation is small enough to be neglected since only the forward pass is relevant for the test process.

4.6.6. Tuning Input Data

In addition to tuning the machine learning model parameters, the input data can also be modified and optimised. The various possibilities for different approaches are presented in Table 4.7. The first row describes the frequency, indicating how often the hysteresis factor should be determined. For the time attribute and time series approach, this is not tunable as the hysteresis factor is only determined once, namely at the end of the measurement, before the relaxation period. For the time interval approaches a range between every 10 s to every 600 s was chosen.

The second row describes the considered time horizon. For the first two approaches, this can vary from a minute up to the entire time series. For the interval approaches, the time horizon is directly connected to the frequency; for example, determining the hysteresis factor every 10 minutes results in a time window of 10 minutes. For the relatively less complex two-dimensional approach, the previous and the previous two time periods can be considered, meaning that the maximum time horizon here is 30 minutes. This was not done for the three-dimensional approach due to the increased complexity associated with using the entire time series as input. Thus, the time horizon for the three-dimensional approach is limited to 10 minutes.

The frequency and time horizon ranges were considered using the following considerations:

1. The hysteresis factor only needs to be determined once per measurement (corresponding to a driving cycle), specifically after the relaxation period when the SOC is corrected. From a practical vehicle standpoint, there is no need for higher frequencies. However, due to the BMS's memory limitations, it is infeasible to store long time series, especially at a high sampling rate. Therefore, shorter intervals result in shorter time series that need to be stored.
2. There is no definitive answer from the literature regarding the required time horizon to determine the hysteresis factor. While the majority of papers do not specify the exact time horizon (often referring to "recent" history), it is very likely that the considered time horizon is short. As the cell is relaxed between the current pulses, which would mark the end of a measurement in this work, it is assumed that the active periods are rather short.
3. It is also very likely that the specific time horizon is directly connected to the exact cell chemistry, such as the silicon content in the anode. Therefore, the time horizon needs to be specified by the cell

manufacturer, which is currently not the case.

4. Schmitt et al. [101] discusses the memory requirements of their model in greater detail. In their model, the memory (and the change in memory) depends on the rate of change of SOC. Large (dis)charge currents fill the memory more quickly resulting in a shorter time horizon to be regarded compared to small (dis)charge currents where the time horizon would be longer. However, no concrete numbers can be extracted from that paper. Additionally, since the framework used in this work works with time as a dimension and not the change in SOC as in Schmitt's framework, a specific time value is required.

The last row in the table describes the sampling rate. For the two-dimensional models, the original rate after preprocessing is used, which corresponds to 10 Hz (the original sampling rate of the hysteresis factor signal). Resampling is unnecessary as the input will be transformed into attributes anyway. For the three-dimensional input approaches, where entire time series are fed into the model, resampling is needed to accommodate longer time series. The maximum number of elements per time series can range between 10 and 240 elements. This range should enable the GRU to operate efficiently (also considering that the size of the model and thus the required memory increases with the number of time elements) while simultaneously being detailed enough to represent the signal's progression accurately.

5

Results

This chapter presents the results of the different case studies. First, it gives an overview of all the case studies and explains the evaluation criteria used. It then investigates each case study, presenting the respective results. After analysing the first six case studies, the results will be used to evaluate the proposed models. Once this evaluation is complete and the most suitable model is selected, an additional case study will examine the generalisation capacity of the chosen model.

5.1. Overview Case Studies

An overview of all case studies can be found in Table 5.1. Note that the first six case studies aim to identify the best model, while Case Study 7 focuses on evaluating the generalisation capacity of the selected model. Thus Case Study 7 is only conducted after comparing the previous case studies.

ID	CS01	CS02	CS03	CS04	CS05	CS06	CS07
Case Study	**Attribute-Based Prediction with Simple Models**	**Time Series Prediction with Neural Networks**	**Interval Delta Prediction with Simple Models**	**Interval Prediction with Simple Models**	**Interval Prediction with Neural Networks**	**Autoregressive Interval Prediction with Neural Networks**	**Generalisation Capability Assessment**
Determination of hysteresis factor	Intermitted		Quasi continuous				tbd
Form of input data	Time series attributes (2D)	Time series (3D)	Time intervals (2D)	Time intervals (2D)	Time intervals (3D)	Time intervals (3D)	tbd
Example input for signal i(t)	$\max_{t \in T} i(t)$	$i(t), t \in T$	$i(nT_s)$	$i(nT_s)$	$i(t),$ $nTs \leq t < (n+1)T_s$	$i(t),$ $nTs \leq t < (n+1)T_s$	tbd
Type of model	Simple model	Neural network model	Simple model	Simple model	Neural network model	Neural network model	tbd

Table 5.1: Overview of all case studies.

All case studies were conducted on a 64-bit Windows computer equipped with an Intel(R) Core(TM) i9-10920X CPU operating at a base speed of 2.5GHz. The computer has 128GB of DIMM RAM and two NVIDIA RTX A5000 GPUs, each with a total GPU memory of 87.55GB. All models were implemented in Python using the framework from the libraries scikit-learn, XGBoost, and PyTorch. The specific Python libraries and their respective versions used can be found in Table B.1. To ensure reproducibility, the seed was set to 0 for all random processes (e.g., splitting datasets into train/test/validation sets, initialising the hidden state of the GRU models, etc.).

5.2. Evaluation Criteria

To evaluate the results of the different methods, two main aspects were considered: accuracy of the model prediction and required memory usage.

The training time was not considered because model variations that require excessive training time were

excluded due to time constraints. This factor is also less critical, as the models are assumed to be trained offline before being implemented in the BMS and used in a vehicle.

The amount and type of required input data were not directly considered. They are included indirectly as it is indirectly part of the accuracy score, the pinball loss. If a model requires more data, it will be apparent through a higher pinball loss. All training data was obtained from a test bench, making it the only type of training data available, thus preventing an evaluation of different training data types.

If the actual implementation of the algorithm in the BMS was within the project's scope, execution time during live operation could be measured. However, since this was not included, execution time was not considered a criterion.

The accuracy of the prediction was measured using the pinball loss (see Equation 4.1) of the test set. A pinball loss of 0.01 was set as the baseline to normalise the achieved pinball losses of the models. Therefore, a model crossing this threshold would lead to a negative pinball score thus lowering the overall score. Accuracy was deemed the most critical element, and thus, a weight of 70% was assigned, selected to ensure that only highly accurate models achieve a high score.

The other crucial aspect is memory usage due to the computational constraints of the BMS. The aim is to disqualify models with very high accuracy but infeasibly high memory usage for BMS implementation. Ideally, memory usage should be as low as possible since various algorithms are implemented in the BMS, which compete for computational resources. The memory usage is split into ROM usage and RAM usage. ROM refers to non-volatile memory used for permanent storage, while RAM refers to volatile memory used for temporary storage of data that the system accesses while executing the algorithm.

As a reference for the BMS, a microcontroller with a maximum ROM of 10MB and RAM of approximately 1.5MB was used. Note that vehicle projects vary in the type of controller used and the amount of memory allocated for other algorithms. Additionally, there is ongoing development to use dedicated controllers with greater computational power for more complex algorithms (such as ML algorithms) that are not safety-relevant. If a function is safety-relevant, the controller has to fulfil certain requirements which ultimately limit its computational resources. This work assumes the algorithm should be implemented in one of the regular controllers with limited computational power. It is assumed that 3% of the total ROM (0.3MB) would be available for the algorithm and that 7% of total RAM (0.1MB) would be available. Based on this, the ROM and RAM requirements of each model are normalised.

The final score is then calculated as follows:

$$\text{score} = 0.7 \cdot \frac{0.01 - L_{test}}{0.01} + 0.15 \cdot \frac{0.3 - r_{ROM}}{0.3} + 0.15 \cdot \frac{0.1 - r_{RAM}}{0.1} \tag{5.1}$$

L_{test} denotes the pinball loss of the test set and r_{ROM} and r_{RAM} denote the required ROM and RAM according to the model size estimations subsection 4.6.5.

5.3. Case Study 1: Attribute-Based Prediction with Simple Models

5.3.1. Description

In this case study, the input data was in the form of time series attributes, resulting in a two-dimensional matrix with one dimension for the samples and the other for the features. This matrix was then used as input for a linear regression (LinReg) model and an XGBoost model. A list of the input features can be found in Table 4.3. The features listed there were extracted for the current, cell voltage and cell temperature signal. These features were extracted from the current, cell voltage, and cell temperature signals. Due to the substantial number of features generated for each signal, and considering that only approximately 600 samples were available, the number of features needed to be reduced to mitigate the suspected high collinearity in the input matrix. An example would be that the average of all temperature values is the same as the average of all positive temperature values since the temperature range was set between 20 ∘C and 35∘C. To address this, feature selection was performed using SelectKBest [102] which identified the most significant features, thereby eliminating redundant or non-important ones. As a scoring function the "f-regression" [103] was used, which calculates the F-statistic and p-value for the cross-correlation between each feature and label. Alternatively, a dimensionality reduction with principal component analysis (PCA) [104] was done. PCA is a statistical technique that reduces data dimensionality by transforming the original features into

a smaller set of uncorrelated components that capture most of the data's variance [105]. There are four different configurations for the input data (see Table 4.7). The time horizon, which specifies the period over which the features are calculated, varied across the entire time series ("all") and the last 10 minutes (600 s), 5 minutes (300 s), or 1 minute (60 s). Each configuration is denoted by its respective time horizon.

5.3.2. Results

K Most Important Features

After adjusting the input data to include only the top five features (for details see section B.2), the linear regression (purple) and XGBoost(dark blue) models were run. The tuned hyperparameters can be found in section B.2. Figure 5.1 illustrates the pinball loss for different configurations of both the XGBoost and linear regression models. It can be observed that the XGBoost model slightly outperformed the linear regression model. However, for both models, the pinball loss is greater than the threshold set at 0.01. In section B.2 a comparison of the required ROM and RAM can be found, both of which are unproblematic.

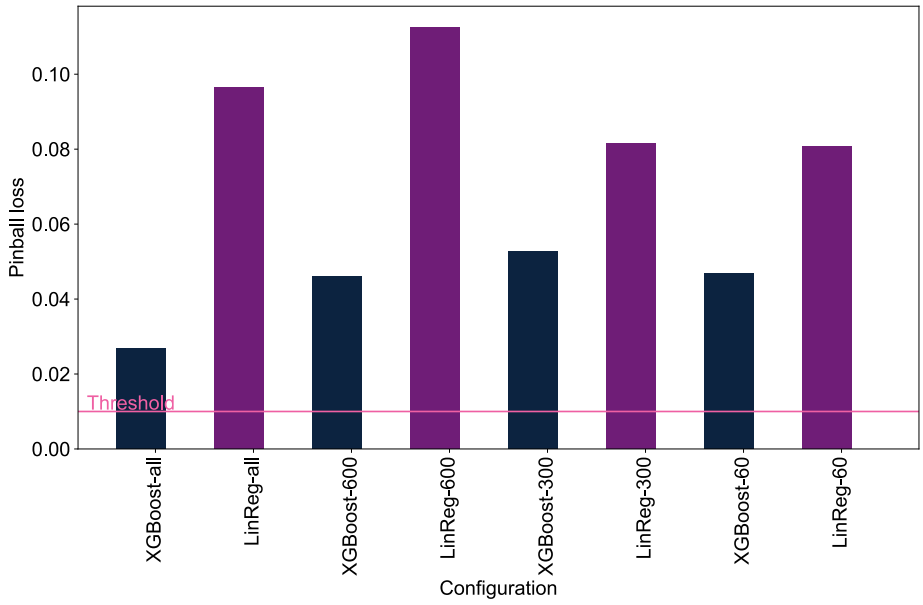


Figure 5.1: Pinball losses for different configurations in Case Study 1 after selecting the most important features.

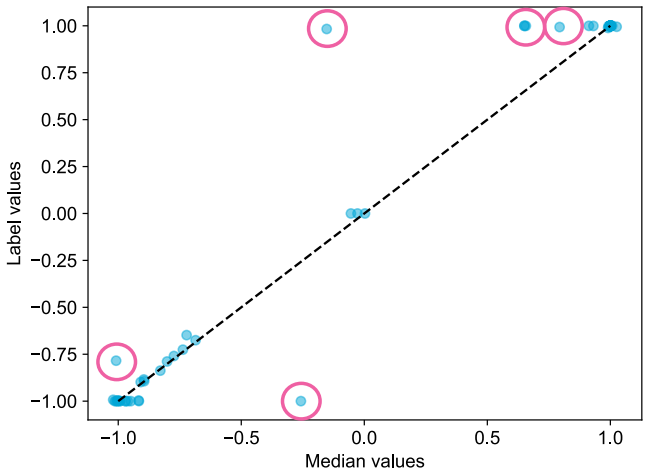


Figure 5.2: Predicted median vs. label for the "all" configuration in the XGBoost model in Case Study 1

Figure 5.2 presents a comparison between the predicted median and the actual labels for the best-performing configuration, namely the "all" configuration of the XGBoost model. Ideally, all points are on the dotted line. There are notable outliers (pink) around the hysteresis factor upper and lower boundaries. Due to the small test set size (around 60 samples), a few outliers will significantly increase the average of the pinball loss. Furthermore, it can be seen that the models do not predict the hysteresis factor accurately. This is likely because the chosen features do not describe the chemical processes in the cell well enough. The time series data is too simplified to effectively learn the relationship with the training data. Consequently, this case study will investigate PCA for reducing the time series to attributes.

Principal Component Analysis

The selection of the number of components N, so features, was determined by the cumulative sum of explained variance, ensuring that N is chosen such that the N features account for 80% of the variance. This implies that the new features, N, should collectively capture 80% of the original dataset's structure while maintaining that each feature is uncorrelated with the others. This threshold was not set higher as the amount of features should be limited due to the small dataset with around 600 samples.

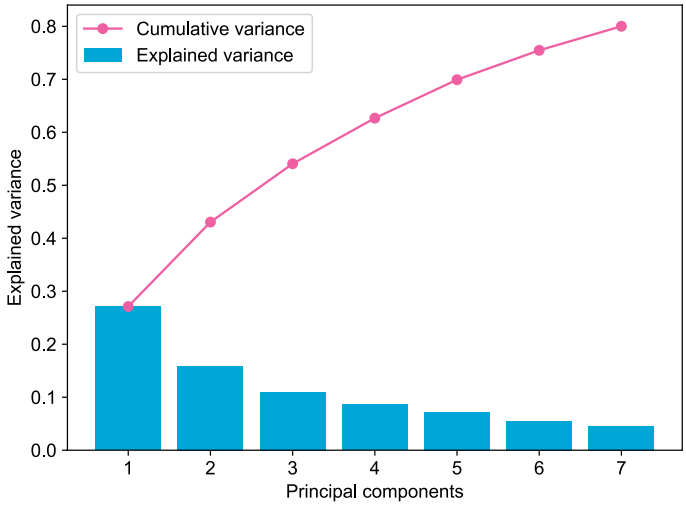


Figure 5.3: The extracted principal components and their respective explained variance for the 300 s configuration.

Figure 5.3 presents a plot of the new features, denoted by a number between 1 and 7, and their respective explained variances for one example configuration. The pink line represents the cumulative sum of the explained variances. It is evident that the first three features already account for over 50% of the variance.

Using this new input with the reduced features, the linear regression model and XGBoost model were run. The details of the optimal parameters can also be found in section B.2. The results for the pinball loss for the models across all configurations are illustrated in Figure 5.4. Like before, all configurations cross the threshold set at 0.01. While the performance of the linear regression model is similar to that of the K most important features, the XGBoost performance degraded, thus suggesting that this method is more unsuitable for XGBoost. Note, that both approaches, K features and PCA, do not meet the accuracy standard.

Figure 5.5 depicts the predicted median versus the actual labels. Note that this plot is from the 300 s configuration which performed best (see Figure 5.4). As already seen in Figure 5.4, the pinball losses for PCA are higher due to more outliers (pink). In this case, the outliers are present not only at the boundaries, 1 and -1, but also in between. The results for memory requirements can be found in section B.2. However, both ROM and RAM were below their thresholds.

The chosen features are not sufficiently descriptive. Similarly to previous observations, the model fails to effectively learn the relationship between input and output, thereby rendering the attributes approach infeasible for the available data.

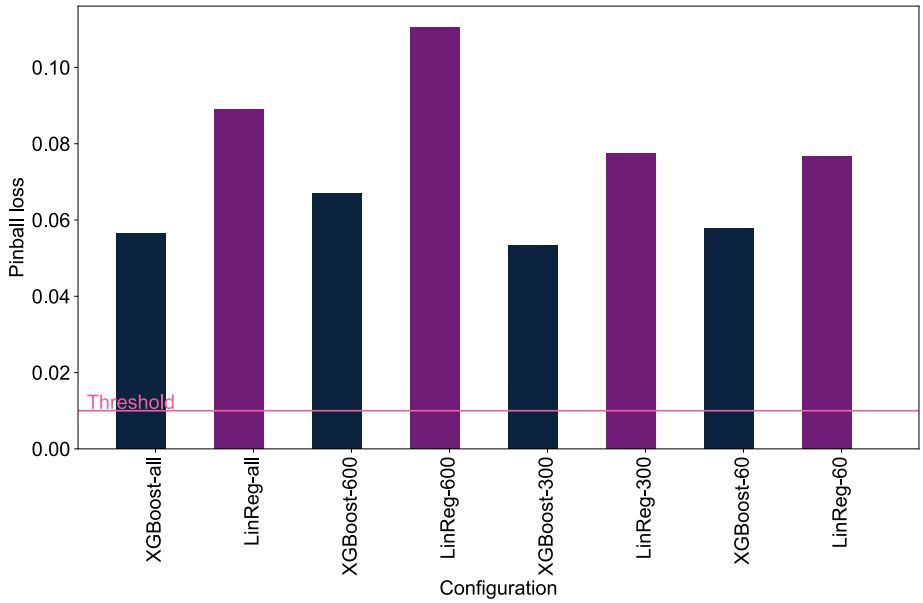


Figure 5.4: Pinball losses for different configurations in Case Study 1 after performing PCA.

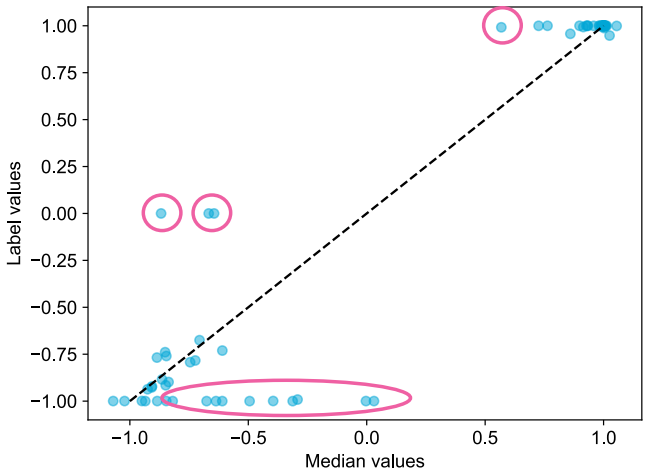


Figure 5.5: Predicted median vs. label for the 300 s configuration in the XGBoost model in Case Study 1

5.4. Case Study 2: Time Series Prediction with Neural Networks

5.4.1. Description

In this case study, the input was in the form of time series resulting in a three-dimensional structure: one dimension for the sample, one for the feature, and one for the time steps. The used features can be found in Table A.1. This input data was then used to train a GRU model. The third column of Table 4.7 shows the different possible input configurations. As shown, the hysteresis factor is determined only once per measurement, specifically right before the SOC correction. The considered time horizon can vary from the entire time series ("all") to merely the last active minute. Active means that the cell has not started relaxing yet. The number of elements per time series varies between 240 and 10 elements. A naming convention was established where the digit before "-" indicates the time horizon and the the digit after denotes the number of elements. For example, 60-240 corresponds to a configuration with the last active 60 s, where each time series has 240 elements.

5.4.2. Results

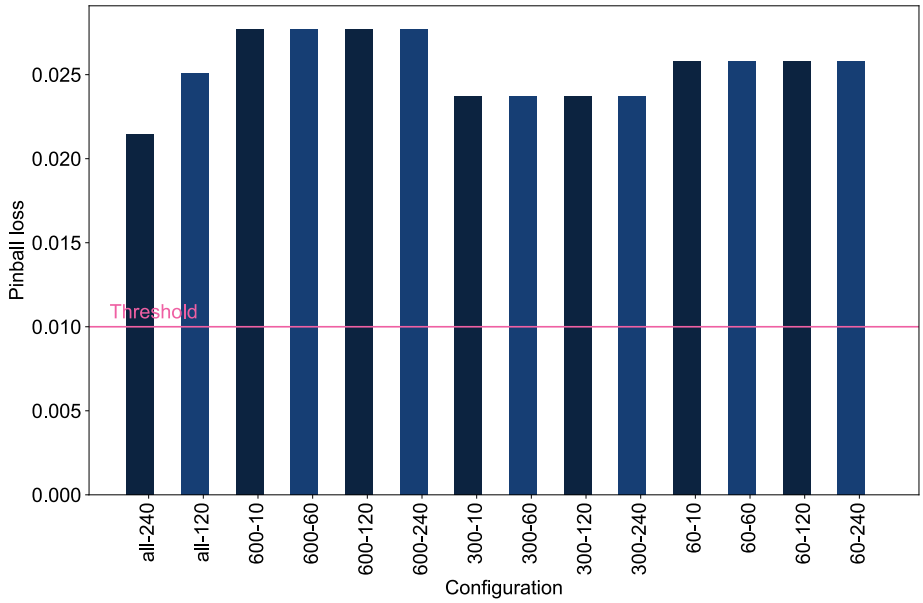


Figure 5.6: Pinball losses for different configurations in Case Study 2.

The tuned parameters for each configuration can be found in section B.3. The pinball loss results are presented in Figure 5.6. The configuration using the entire time series with 120 elements achieved the lowest pinball loss and thus performs best in this category. Interestingly, within the same time horizon of the last active minutes results in the same pinball loss, regardless of the number of time steps. This is likely because the models have the same parameters (see section B.3). This could also be caused by the low number of samples and the model not being able to learn the relation properly. Therefore, it would not be able to learn the differences arising from different sampling rates. The fact that the all-120 and all-240 configurations do not have the same test loss could be attributed to the overall length of the time series being consistent in other configurations (600 s, 300 s or 60 s). In these two cases, the time series differ in length (see Figure 4.6). Regardless of the configuration, the pinball loss exceeds the set threshold of 0.01.

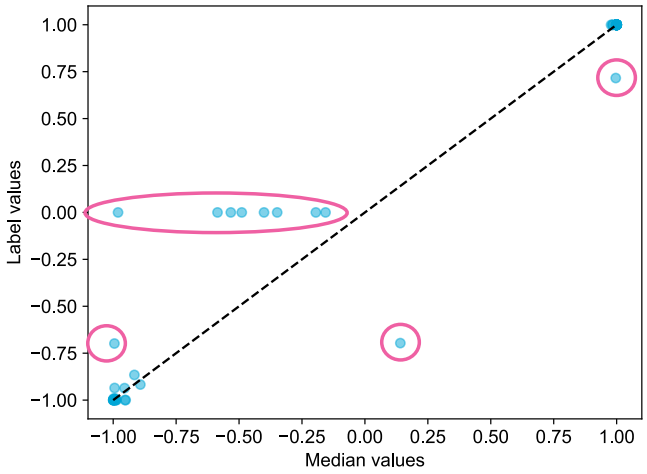


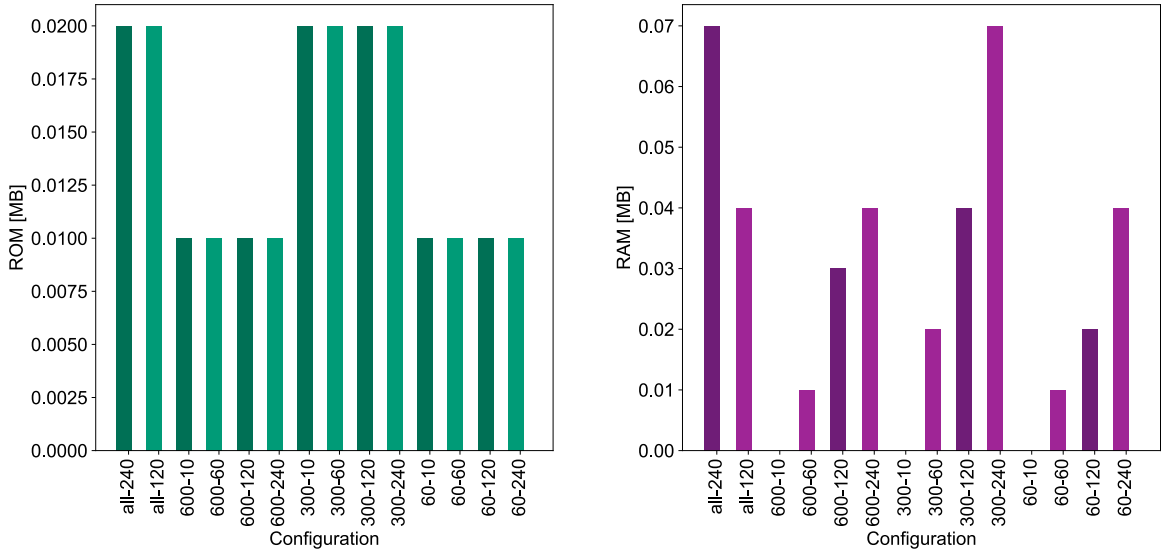
Figure 5.7: Predicted median vs label for the all-120 configuration in Case Study 2.

It should be noted that even though the all-120 configuration scores best, the pinball loss is still relatively high. This is illustrated in Figure 5.7, which shows significant outliers (pink), especially for labels that are

not at -1 or 1. This is likely due to the distribution of labels (see Figure 4.9a). The model had more data to learn the extreme points and less to learn labels that lie between the boundaries. Additionally, the number of samples is likely too small to train the model adequately. More samples, especially those with labels between the extremes, would be required to improve performance.

The memory usage for ROM and RAM is shown in Figure 5.8a and Figure 5.8b. It should be noted that if the input size is very small (e.g., 10 time steps), the RAM usage is less than 0.01 and is thus depicted as 0 due to the precision of the torchinfo summary output. The configurations with a time horizon of 300 s and those that consider the entire time series ("all") have the largest ROM usage. As ROM scales with the model parameters and these configurations have a greater hidden size (32 instead of 16), they require more ROM.

RAM usage scales, among other factors, with the number of time steps, as this increases the input size and, consequently, the memory required for the forward pass. Therefore it is evident that the more timesteps are taken into account, the greater the required RAM. Both RAM and ROM are well within the computational limits of the BMS.



(a) Required ROM.

(b) Required RAM.

Figure 5.8: Memory usage for different configurations in Case Study 2.

5.5. Case Study 3: Interval Delta Prediction with Simple Models

5.5.1. Description

In this case study, rather than predicting the absolute value of the hysteresis factor, the change in the hysteresis factor is predicted. This is achieved using the interval approach, where the time series is divided into smaller samples. This approach is first applied to an XGBoost model due to its higher performance in Case Study 1. Each label is calculated by subtracting the hysteresis factor of the previous interval from that of the current interval. Unlike other case studies, in this case, the previous hysteresis factor was not included in the features, thus eliminating the feedback loop (see autoregressive training and testing in subsection 4.6.3 and subsection 4.6.4) that would usually occur in the interval approach. The complete list of features and their mathematical formulations can be found in Table A.2. In the tested input configuration, the hysteresis factor was determined every 60 s, and the past two time periods were considered as well. The hyperparameters can be found in section B.4.

5.5.2. Results

The achieved average pinball loss of this trained model was 0.2364 with a required ROM of 0.18 MB and RAM of 0.01 MB. The loss was calculated by determining the hysteresis factor for each interval by adding the predicted change in the hysteresis factor to the previous hysteresis factor. Note that apart from the first interval, the previous hysteresis factor is based on the previous delta predictions. This makes the

loss comparable with that of the other case studies and corresponds to how the BMS would determine the hysteresis factor. The high loss is further reflected in the comparison between the predicted median hysteresis factor (obtained using the same method) and the actual hysteresis factor, as illustrated in section B.4.

Figure 5.9 presents an example measurement for this case study, with time on the x-axis and the hysteresis factor on the y-axis. This plot was generated by reconstructing the measurement from its interval predictions, applying the previously mentioned method to derive the absolute hysteresis factor from the change in the hysteresis factor.

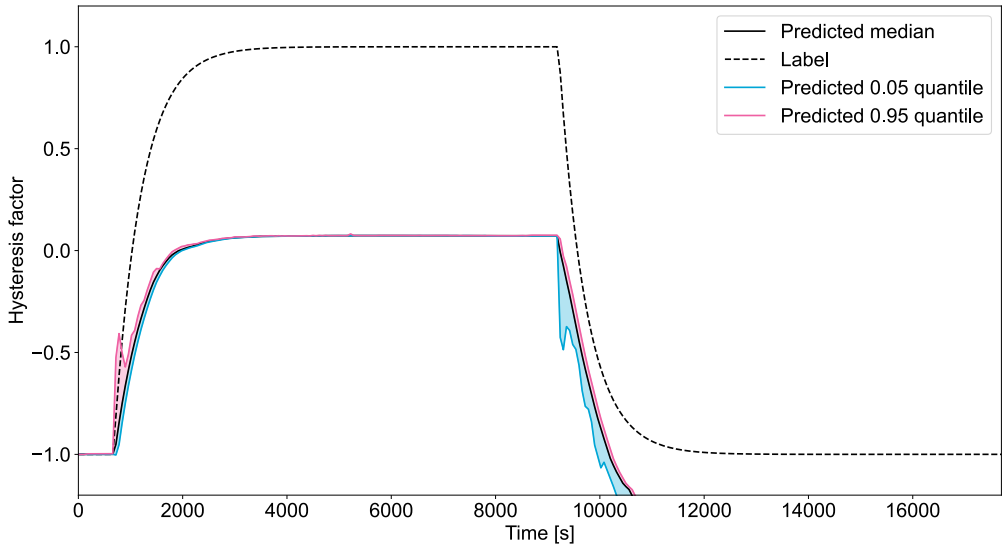


Figure 5.9: Example reconstructed measurement for Case Study 3.

The plot demonstrates that, over time, the predicted quantiles significantly deviate from the actual labels. This deviation is particularly pronounced when the hysteresis factor changes, corresponding to a label between but not including -1 and 1, likely due to fewer labels in this range. Early errors in delta predictions accumulate over time. In this case, the incorrect prediction of a more moderate increase in the beginning resulted in a consistent offset throughout the measurement.

This case study indicates that the previous hysteresis factor could be crucial to predict the change in the hysteresis factor, suggesting that the determination of the hysteresis factor is an autoregressive problem. Without including the previous hysteresis factor as input, the model appears to "blindly" predict changes in the hysteresis factor, disregarding its boundaries of -1 and 1, which can be seen in the second half of the measurement in Figure 5.9. This will remain a problem of this approach, thus no other input configuration or types of models were tested.

5.6. Case Study 4: Interval Prediction with Simple Models

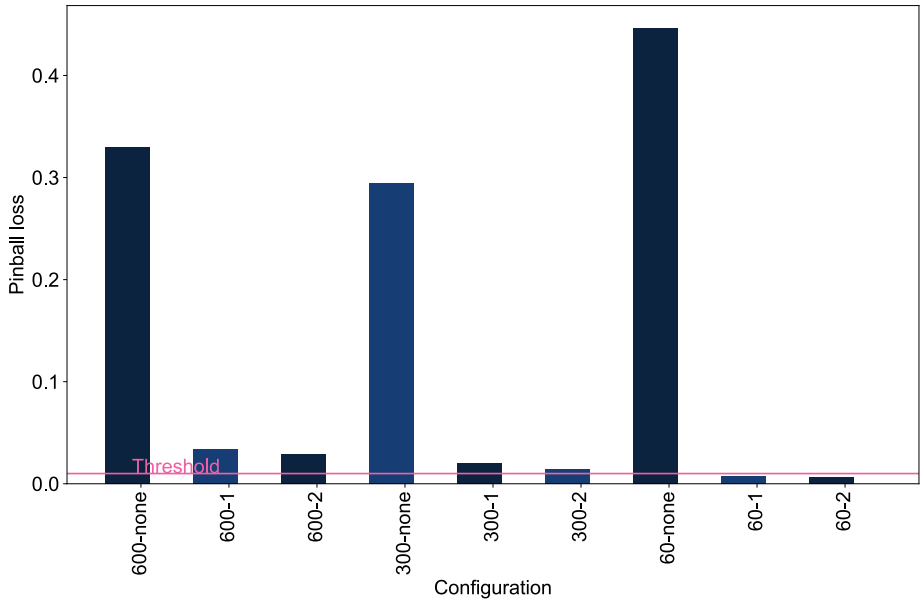
5.6.1. Description

In this case study, the performance of the simple models with the time interval approach was evaluated. Unlike the previous case study, where the change of the hysteresis factor was used as a label, the labels here represent the actual hysteresis factor at the end of each interval. The input data is two-dimensional, thus the time series are reduced to one-dimensional features. The list of features and their mathematical formulation can be found Table A.3. Note that these features were determined for the current, cell voltage and cell temperature signal. The input configuration (see Table 4.7) can vary based on the length of the period during which the hysteresis factor is determined, ranging from every 10 s (XGBoost) or 60 s (linear regression) to every 600 s, as well as the length of the time horizon. The linear regression model was not tested for a 10 s configuration due to the high number of samples of this configuration and the long training time. The time horizon varies from considering only the current interval to including up to the last two intervals.

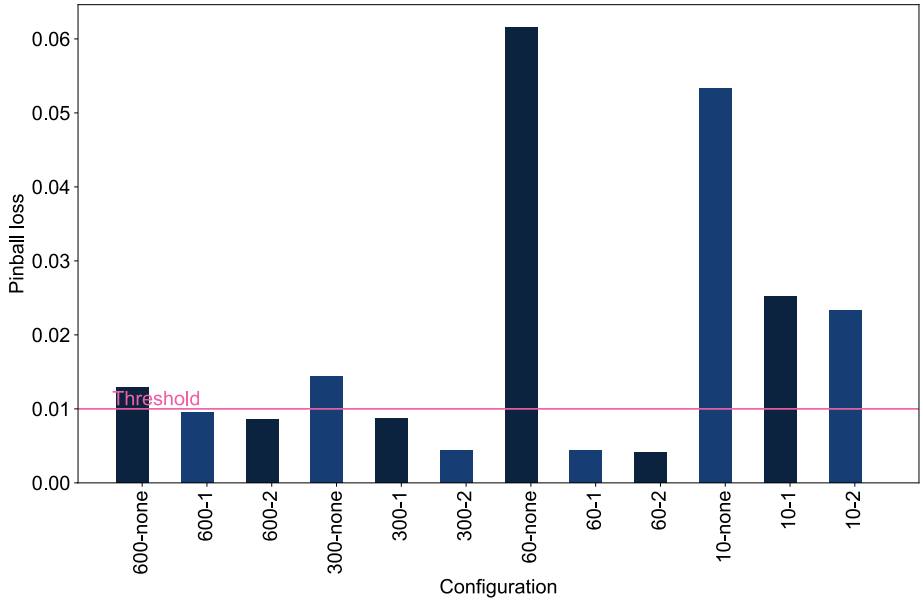
A naming convention was introduced where the number before the "-" indicates the period, and the number after the "-" indicates the time horizon. Note that a time horizon of none corresponds to only the current interval being considered, thus not including any past intervals.

5.6.2. Results

The results for the pinball loss for different configurations for the linear regression model can be seen in Figure 5.10a. Similarly, the pinball loss for various configurations of the XGBoost model is displayed in Figure 5.10b. For both models, the pinball loss decreases when past intervals are taken into account. Specifically, the reduction in pinball loss when considering the past two intervals as opposed to just the last interval is significantly smaller for the linear regression model.



(a) Pinball losses for different configurations for the linear regression model in Case Study 4.



(b) Pinball losses for different configurations for the XGBoost model in Case Study 4.

Figure 5.10: Pinball losses for different configurations for the linear regression and XGBoost models in Case Study 4.

It is also apparent that when past intervals are considered, the pinball loss is below the set threshold of 0.01, thus making the models feasible in terms of accuracy. Notably, this does not apply to the configurations with only a 10 s interval (only done for the XGBoost model). This could be either due to insufficient tuning of the corresponding models or due to the interval size being insufficient to capture the hysteresis phenomenon. There is also a peak in pinball loss for the 60-none configuration which is most likely due to hyperparameters (specifically the lambda parameter) of this model configuration.

The lowest pinball loss for both models could be achieved when considering the last two intervals and with an interval size of 60 s. This suggests that a time horizon of in total 180 s improves the model predictions when compared to a time horizon of merely 30 s (10-2 configuration). Additionally, the worse performance in the 300-2 and 600-2 configurations might indicate that greater periods of determining the hysteresis factor may be unsuitable as too many details would get lost when transforming the time series to one-dimensional features. Moreover, it can be observed that the XGBoost model generally outperforms the linear regression model, suggesting that the linear model cannot effectively learn the complex relationship between input and output.

Figure 5.11 shows the predicted median versus the label for the XGBoost 60-2 configuration, which achieved the lowest loss. While some significant outliers (pink) are present, the overall performance surpasses the one observed when considering the time attributes in Case Study 1, as shown in Figure 5.5. An example reconstructed measurement for the XGBoost 60-2 configuration can be found in section B.5, which verifies the increase in performance compared to Case Study 3.

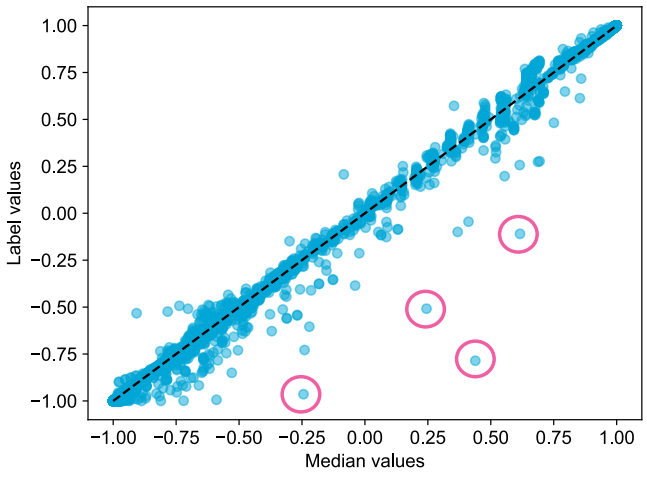


Figure 5.11: Predicted median vs label for the 60-2 configuration (XGBoost) in Case Study 4.

5.7. Case Study 5: Interval Prediction with Neural Networks

5.7.1. Description

In this case study, similar to Case Study 4, the interval approach is used. However, a GRU model was trained instead of a simple model. The input features, namely the current, voltage and temperature signal and their respective derivatives as well as the previous hysteresis factor were in the form of time series. The complete list of the features can be found in Table A.4. The input configuration could vary based on the frequency of hysteresis factor determination, i.e., how often the hysteresis factor is determined, ranging from every 10 seconds to every 10 minutes. Additionally, the sampling rate of the time series could be adjusted. This rate varied between 10 elements per time series and 240 elements per time series. For reference, see also Table 4.7. A naming convention was introduced where the number before "-" indicates the period and the number after "-" indicates the sampling rate. Each configuration was first run for 100 epochs.

5.7.2. Results

The tuned parameters, including the optimal number of epochs, can be found in section B.6. The different pinball losses for the configurations can be found in Figure 5.12. The pinball loss ranges from 0.1 to approximately 0.003, achieving the lowest pinball loss of all the case studies so far. Looking at Figure 5.12 it is difficult to find a clear pattern a clear pattern between pinball loss and input configuration. Generally, the models perform best with a time horizon of 600 s. The lowest pinball loss was achieved with a sampling rate of 240 elements per time series.

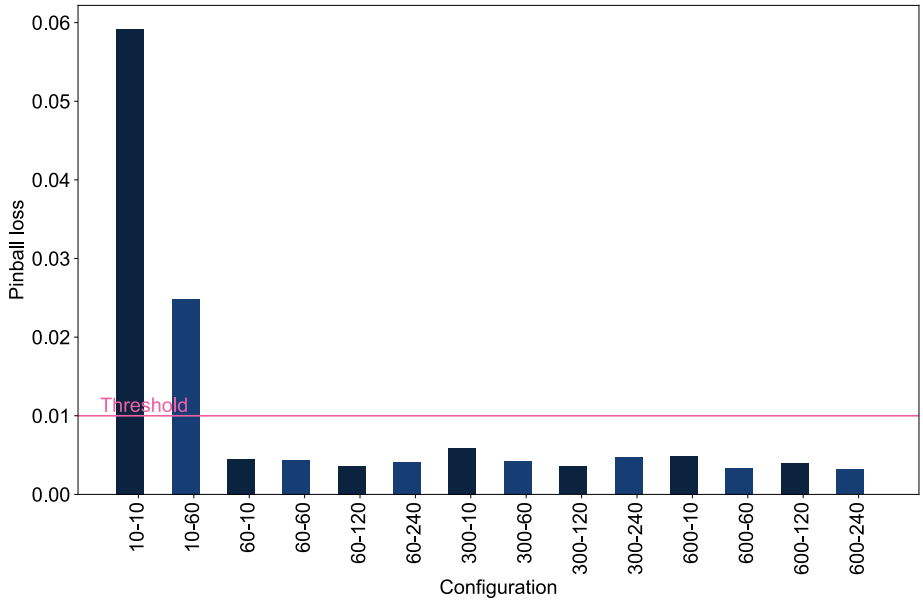


Figure 5.12: Pinball losses for different configurations in Case Study 5.

This may suggest that shorter time horizons do not allow the model to adequately learn the details of the input signals necessary to predict the hysteresis factor. It is noteworthy that this configuration performs the best despite having the smallest sample size, approximately 10000 samples. However, since the second lowest loss was achieved with a 300-second horizon and the shortest sampling period of 10 elements per time series, it is not possible to conclusively state that more or fewer elements per time series lead to better or worse results. Small differences between configurations, like the 600-240 and 300-10 configuration can also be explained by non-optimal tuning or could be based on the test set itself, i.e. that for this particular test set, which is used on both configurations, the 600-240 configuration marginally scores better, while it may be the other way around with another test set.

The predicted median for the 600-240 configuration is shown in Figure 5.13. There are greater outliers (pink) when the label is between -0.5 and 0.5. This could be due to fewer samples with labels within this area since the majority of samples are approximately -1 and 1 (see Figure 4.9b).

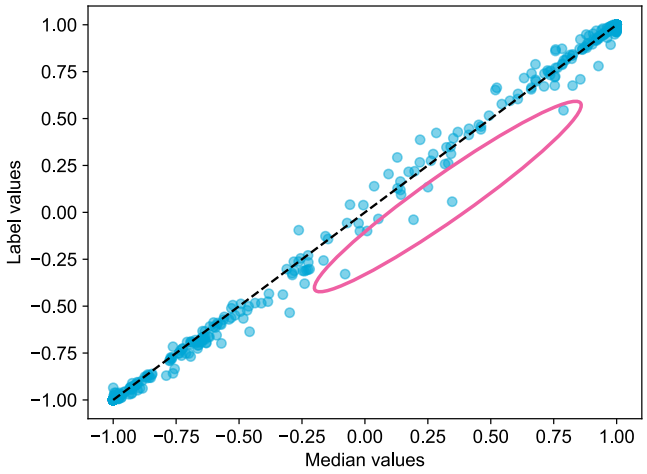


Figure 5.13: Predicted median vs label for the 600-240 configuration in Case Study 5.

An example of a measurement reconstructed from intervals, with the previous interval prediction used as input for the next interval, is shown in Figure 5.14. The model correctly predicts the progress of the hysteresis factor. Although there are minor deviations from the actual label (e.g., from 13000 s onward), these remain within an acceptable range. The figure also shows that the quantiles enclose the actual label, except for a brief period of around 16500 s. Additionally, the upper and lower quantiles are very close to one another when the hysteresis factor is -1 for 8000 s, indicating that the model can predict the extreme points (-1) well.

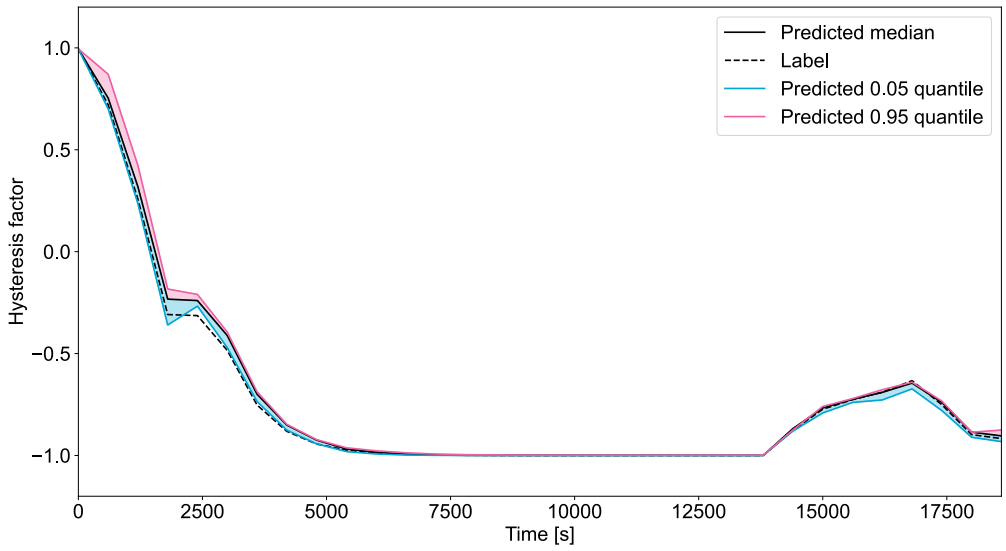
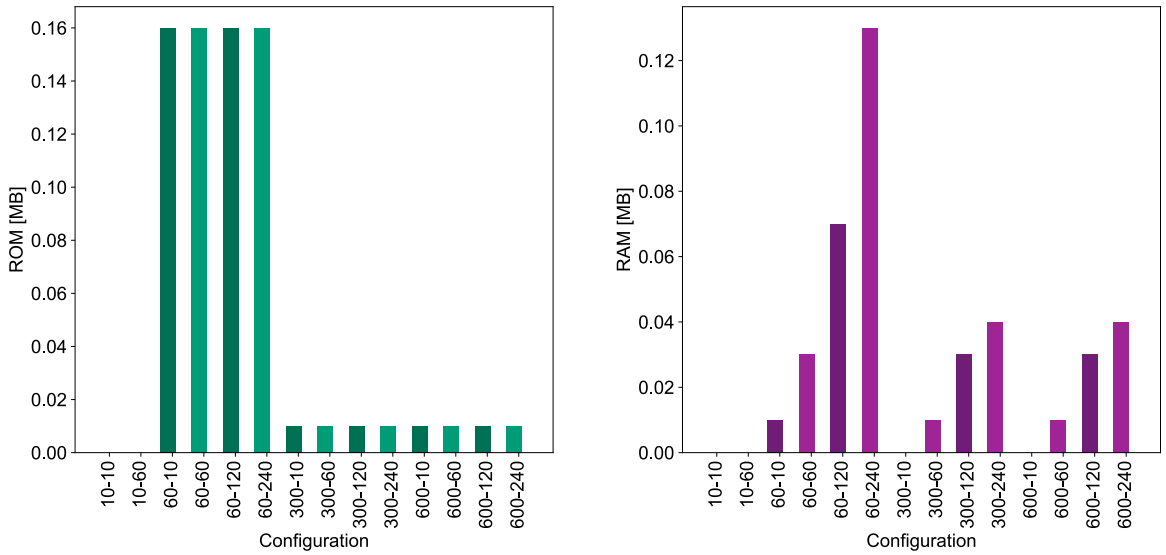


Figure 5.14: Example reconstructed measurement for the 600-240 configuration in Case Study 5.

The required memory usage is depicted in Figure 5.15a and Figure 5.15b. The required ROM is the same for models with the same parameters (hidden size and GRU layers). The 600-x and 300-x configurations have a required ROM of 0.01 MB, which is below the critical ROM threshold of 0.3. Also, the other configurations fall below this threshold. Figure 5.15b illustrates the influence of the sampling rate on required RAM. As explained in section 5.4, the required RAM scales with the number of time steps. The RAM value for the 60-2 configuration is above the threshold of 0.1 MB. However, the other configurations are unproblematic. It is worth noting that, due to the resolution of the torchinfo summary function, memory usage below 0.01 MB is shown as 0 (as observed in the 10-x configurations).



(a) Required ROM.

(b) Required RAM.

Figure 5.15: Memory usage for different configurations in Case Study 5.

5.8. Case Study 6: Autoregressive Interval Prediction with Neural Networks

5.8.1. Description

The last tested model was a GRU model with the input being formatted as intervals similar to Case Study 5. However, the difference lies in the training methodology. Unlike in Case Study 5, the different intervals of a measurement remained connected throughout the training process. This is described in greater detail in subsection 4.6.4. This approach aimed to determine whether the prediction error could be reduced by training the model in a way that acknowledges the influence of earlier errors on later intervals. In the previous case study it could be seen that the test error was smaller for non-autoregressive testing than for autoregressive testing (see comparison of autoregressive and non-autoregressive training in subsection B.9.3). Therefore, this was done to see if autoregressive training would improve this.

Due to the extended training time for this training approach, only one configuration was tested, with parameters taken from the tuned parameters of the corresponding model in Case Study 5. As observed there, the input configuration with a time horizon of 600 s and a sampling rate of 240 elements per time series resulted in the best results. Note that the batch size was increased to 128 samples per batch and the learning rate was adjusted from 0.0013 to 0.01 to account for the slower training.

5.8.2. Results

The achieved test loss after 100 epochs was 0.041, which is worse than the one achieved in Case Study 5. After analysing the train and validation loss over the number of seen batches, shown in Figure 5.16, it becomes apparent that the model might not have had enough epochs to train adequately.

Therefore, the model was retrained with 1000 epochs, resulting in a test loss of 0.0047. Although this loss is still higher than the loss achieved in Case Study 5 (0.0031), it represents a significant improvement compared to the previous experiment. The corresponding comparison between the label and the predicted median can be found in section B.7.

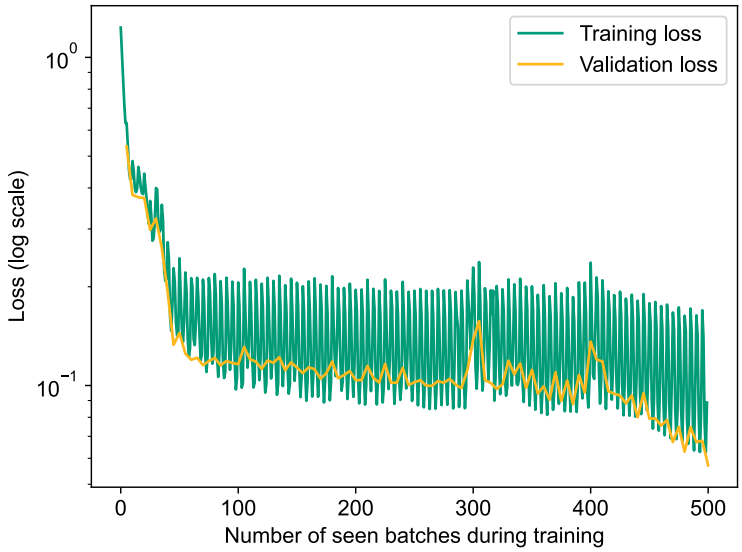


Figure 5.16: Training and validation loss over number of seen batches for 100 epochs in Case Study 6.

Additionally, when examining the predictions for a single measurement as displayed in Figure 5.17, it can be observed that the upper and lower quantiles are slightly wider compared to the results of Case Study 5 (Figure 5.14), which is in line with the higher test loss in this case study.

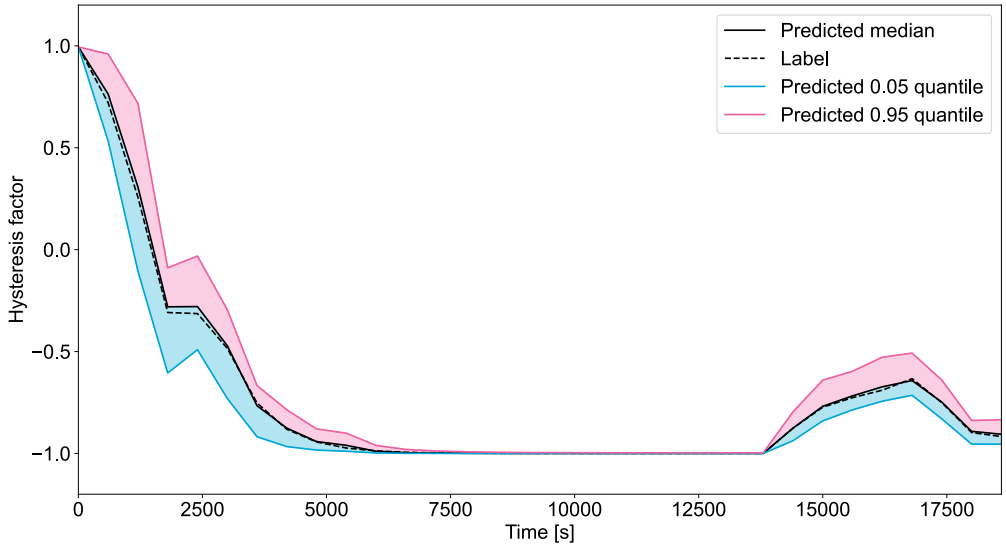


Figure 5.17: Example reconstructed measurement for Case Study 6.

The performance in terms of loss of this model is slightly worse than that of Case Study 5. This could be due to numerous reasons. In this training approach, the optimiser step, so the parameter update, is only done at the end of a measurement rather than for every sample. This results in fewer optimiser steps being taken. If the learning rate is too small, the optimiser steps might be too small to get to the minimum of the loss function. On the other hand, if the learning rate is not too large, the optimiser might skip the minimum. Therefore, the learning rate could be further tuned to find an optimal value. However, due to the long training time with many epochs, this was beyond the scope of this work. Fewer optimiser steps also lead to this training approach

requiring more epochs than in Case Study 5. Further increasing the number of epochs could potentially decrease the training loss as well. The progress of the train and validation for 1000 epochs can be seen in section B.7. It is also possible that for autoregressive training, the GRU model, this input configuration and the respective parameters might not be optimal. However, due to the prolonged training time, other models as well as separate tuning of the input configuration and model parameters were not feasible.

5.9. Evaluation of the Models

The best model configuration of each case study is compared in terms of pinball loss, ROM and RAM in Figure 5.18 and Figure 5.19. It becomes clear that the interval approach (Case Study 4, 5 and 6) worked best. This suggests that for this problem, where the input is in the form of long time series (»1h) time series, it is a feasible strategy to split these time series into smaller intervals before processing them. This approach also has the advantage of providing more samples for training, which is particularly beneficial for more complex models like the GRU and available measurements are limited.

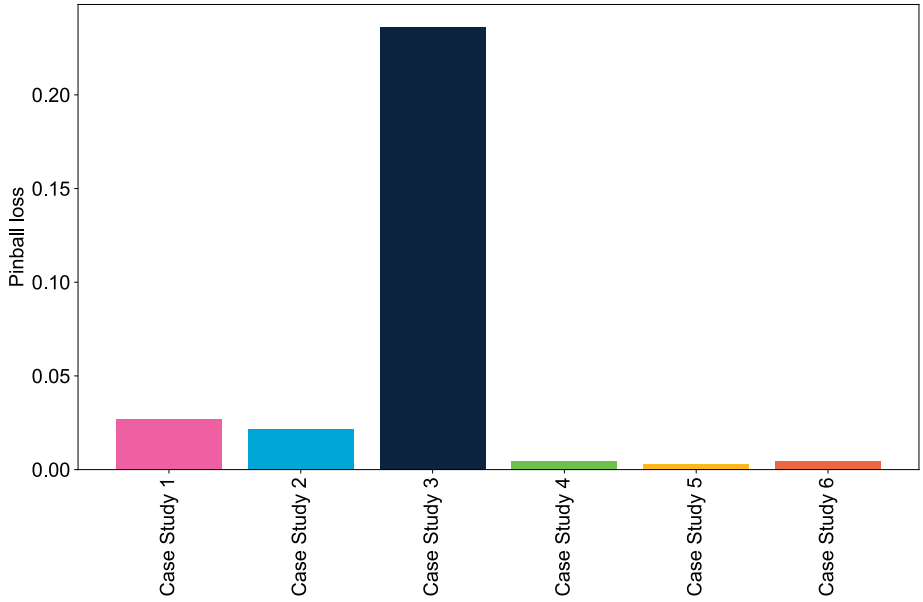
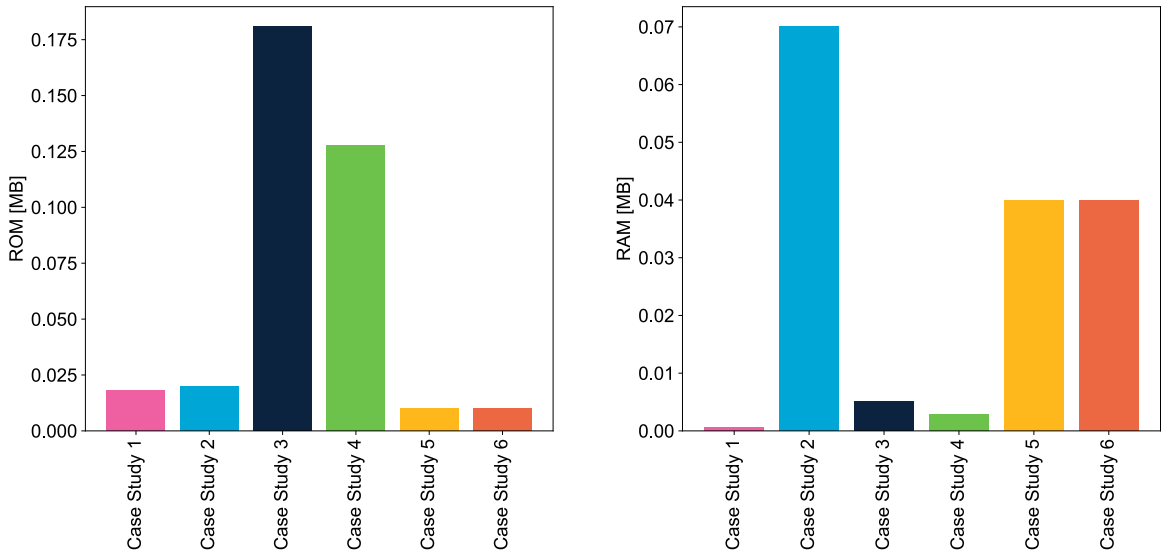


Figure 5.18: Comparison of the achieved pinball loss for the best configurations of each case study.

It can be seen that Case Study 5 is at the forefront in terms of both pinball loss and memory requirements. To further quantify this, the score (Equation 5.1) was determined for the best configuration of each case study and is depicted in Figure 5.20. According to this Case Study 5 performs best and thus makes it the most suitable candidate for the next steps.

To compare the case studies beyond the best configuration, the same plots for the average across all configurations can be found in section B.8. It becomes clear that Case Study 5, on average, scores higher than all other case studies (which also have multiple configurations), indicating that its performance is less susceptible to the configuration of the input data and it generally predicts the hysteresis factor more accurately.



(a) Required ROM.

(b) Required RAM.

Figure 5.19: Comparison of memory usage for the best configurations of each case study.

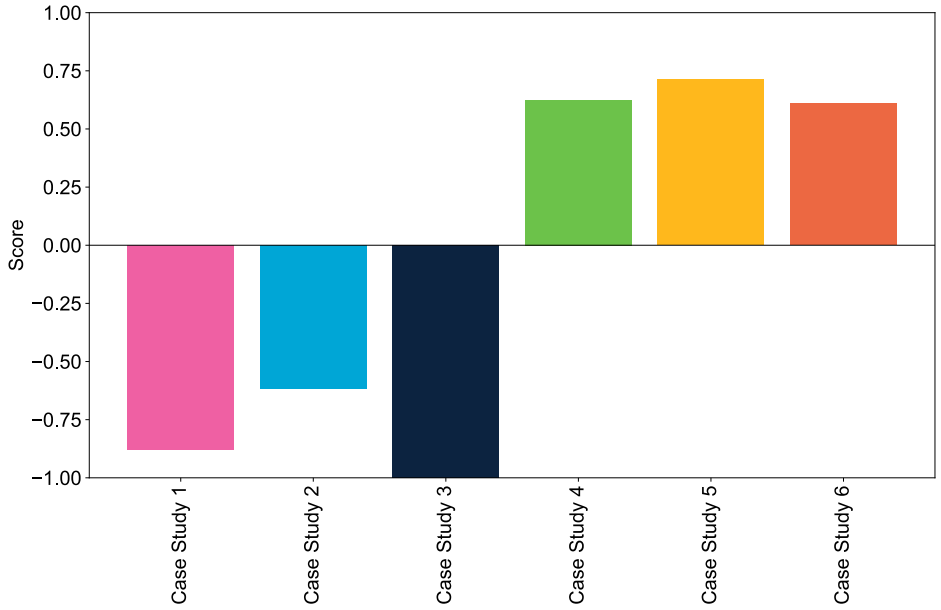


Figure 5.20: Comparison of the achieved score for the best configurations of each case study.

To provide further insight into the robustness of this choice, a sensitivity analysis was done to check whether the weights of the score would influence which model was picked. Three different scenarios were regarded. The base scenario describes the loss being weighted at 70% and the memory being weighted at 30%. The memory was not separately split because ROM and RAM should be equally important regardless of the scenario. The lower scenario is when the loss is weighted at only 50% and the upper scenario is when the score is almost exclusively weighted just on the loss, namely at 95%. The results of the sensitivity analysis for case studies 4 to 6 can be seen in Figure 5.21. The y-axis depicts the derivation in loss (absolute number) compared to the base scenario. Note that the first three case studies are not depicted as their scores will remain negative due to their high pinball loss. It can be seen that the order remains the same regardless of the scenarios. A detailed overview of all case studies and their percentage change for each scenario compared to the base scenario can be seen in Table B.7.

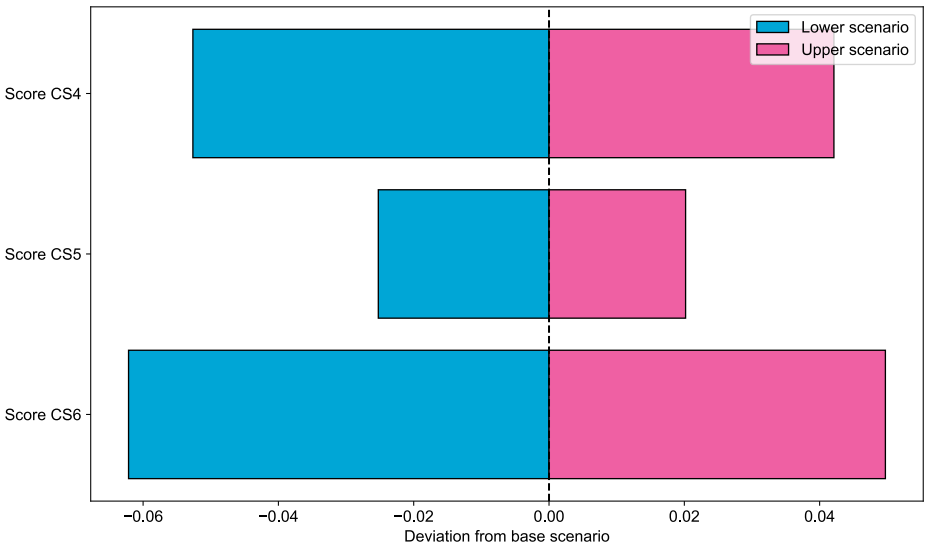


Figure 5.21: Comparison of different scenarios regarding the weight of the pinball loss.

Figure 5.21 also shows that if the pinball loss is weighted less the score reduces compared to the base scenario, while it increases if it is weighted more, this is likely because the normed loss is greater than the normed memory requirement (thus reducing the loss and increasing the normed memory requirement by the same % will in total still lead to reduction of the score).

Note that due to the limited tuning rounds and the boundaries set by the model input tuning, the selected model and configuration may not necessarily be the best possible configuration. However, given the constraints formulated in this work, it is the most suitable one.

5.10. Case Study 7: Generalisation Capability Assessment

5.10.1. Description

In this case study, the previously selected model from Case Study 5 is utilised and assessed for its generalisation capability which refers to the ability of a model to learn general rules from specific data and apply them to new, unseen data. Without generalisation, machine learning models would merely memorise data, limiting their prediction ability. It is important to note that generalisation capacity was not a selection criterion in the initial model selection, as the requirements for assessing this capacity were beyond the scope of this work. Consequently, the chosen model may not exhibit optimal performance in terms of generalisation. Moreover, it is assumed that the core characteristics of the problem remain unchanged thus no reconfiguration of the input data format is needed. The generalisation capability is evaluated across three specific aspects:

1. Vehicle model generalisation: How well does the selected model perform on a different vehicle project?
2. Driving cycle generalisation: How well does the selected model perform on new driving cycles?
3. Temperature range generalisation: How well does the model perform for data outside of the initial set temperature range?

5.10.2. Results: Vehicle Model Generalisation

So far, models have been trained using data from vehicle project A. Now, vehicle project B is introduced and utilised for testing. It is important to note that both vehicle project A and vehicle project B are fully electric vehicles. However, the type of cell used in each project differs. Consequently, the cell chemistry and electrical cell specifications may vary slightly. For both vehicle projects, test bench data is available, containing mostly similar test types (i.e., driving cycles) as those present in dataset A, but in different quantities and with slight variations in details.

The case study can be divided into three approaches:

1. Testing dataset B with a model trained solely on dataset A
2. Testing datasets A and B with a model trained on both datasets.
3. Testing dataset B with a model trained solely on dataset B

The aim is to determine which of these approaches performs best in terms of pinball loss.

Testing B With a Model Trained Solely on A
After preparing the data in a similar manner to vehicle project A, using the same scaler, the pinball loss results were higher than in Case Study 5, with a pinball loss of 0.04055 (averaging to 0.0219 when considering the respective loss of testing with model A). Detailed analysis of the measurements identified two potential issues:

1. **Unseen tests:** In vehicle project B, certain tests were conducted that were not performed for vehicle project A. Consequently, the model has not encountered these types of tests during training. In a specific measurement, where this is the case (shown in Figure B.14a), an initial faulty prediction of the hysteresis factor seems to cause a significant deviation between prediction and label. When adjusting the testing routine to correct the hysteresis factor to the "true" value at the start of every interval, the model's performance improves notably and can better follow the curve (Figure B.14b).
2. **Different cell chemistry:** The bad initial prediction and other slightly off predictions could also result from the different cell chemistry in vehicle project B. Currently, the data of B is normalised like the data of A, even though the maximum and minimum voltage of each cell differs. Therefore, even for the same test, the voltage and current profiles may appear different.

To overcome the issues mentioned in point two, the normalisation strategy was changed and adapted, so that each cell was normed according to its cell datasheet. More details about this can be found in subsection B.9.2.

In Figure 5.22, it is evident that the performance remains poor, with a test loss for vehicle project B of about 0.0649 (corresponding to an average loss for vehicle projects A and B of 0.0338). This result is actually worse after individual normalisation. While the considerations make sense from a physical perspective, the processes within the neural network are not interpretable, meaning the model might use the signals in a way that is not logical to humans with background knowledge of cell chemistry. Furthermore, when normalising with the datasheet, current, integrated current, and voltage are set according to the minimum and maximum of the datasheet, which might mean that the full span of possible values is not captured in the training dataset. This results in a range of values between -0.5 and 0.8 instead of the ideal -1 to 1.

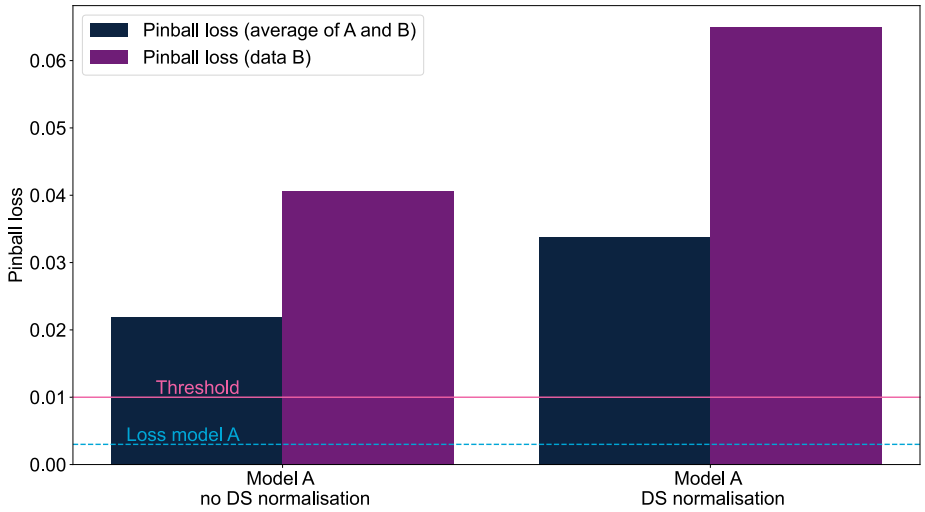


Figure 5.22: Comparison of the pinball loss for different methods of using the model, trained on vehicle project A, on vehicle project B.

As mentioned before, not all test types from vehicle project B have been conducted on vehicle project A. As a result, the trained model might encounter driving profiles it has not seen in its training dataset.

Another, likely the most significant factor contributing to the decreased performance, is the different nature of the cell in vehicle project B. The previously trained model does not work well since it was trained on a different cell (cell A). Therefore, the next step will be to assess the performance in terms of loss when the model is trained on both vehicle projects.

Training the Model With Both Vehicle Projects
In this case, the model was retrained using both vehicle projects. Three different configurations were tested:

1. Completely retrain a new model without considering the cell datasheets (DS), using only the respective minimum and maximum values in the combined dataset during normalisation.
2. Completely retrain a new model considering the cell datasheets (DS) and the respective minimum and maximum values in the combined dataset during normalisation.

Figure 5.23 illustrates the pinball loss for different normalisation approaches. These include not normalising according to the cell datasheet (reusing the initial scalar from vehicle project A) and normalising according to the cell datasheet. For the latter, the scalar is created from the merged dataset (with minimum and maximum values based on both datasets). It should be noted that the results presented have been fine-tuned (see subsection B.9.1) and reflect the optimal number of epochs determined by the validation loss (for details see Figure B.9.3).

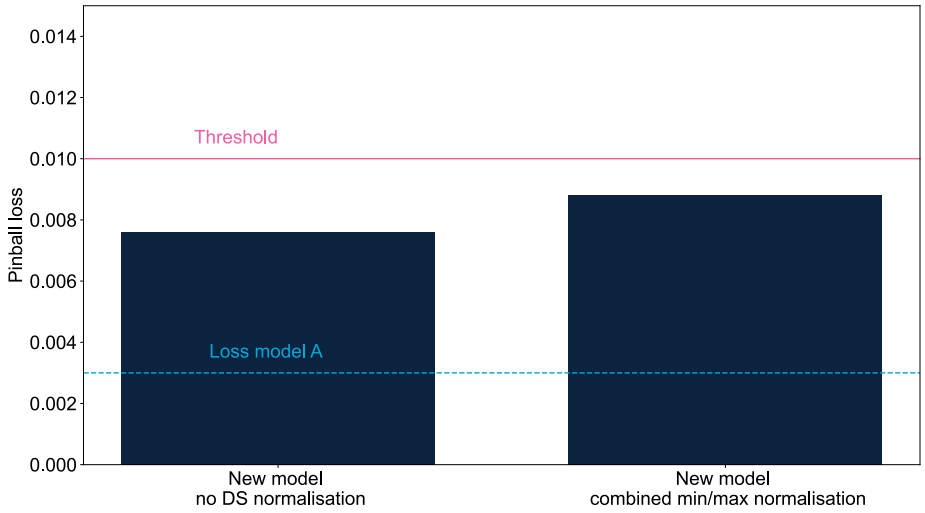


Figure 5.23: Comparison of the pinball loss for different methods of using both projects to train the model.

In Figure 5.23, the loss for model A (blue dotted line) represents the loss obtained when testing model A with its own test data. The average value is depicted due to minor differences in the losses between using cell datasheet normalisation and not using it. In all scenarios, the test data included data from both vehicle projects.

It is evident that the lowest loss (approximately 0.0076) was achieved when the data was normalised not using the cell datasheet. However, the difference between normalising with the datasheet and without it is minimal, namely 0.0076 and 0.0088. This difference is likely due to the current, integrated current and voltage being normalised according to the minimum and maximum values in the cell datasheet, which do not span the full range (as explained at the beginning of this section).

Training each Vehicle Project With Its Own Model
In this approach, it was investigated whether performance would improve if a model was trained specifically for vehicle project B. Three different configurations were tested:

1. Further training model A with data from project B (normalised according to the datasheet).
2. Creating a new model based on data from vehicle project B (normalised only according to the min/max of signals).
3. Creating a new model based on data from vehicle project B (normalised according to the cell datasheet).

To further train the model, the initial model A (where the input was normalised according to the cell datasheet of A, as well as the minimum and maximum of signals from dataset A) was used. The data from project B was prepared in a similar fashion, utilising cell datasheet B and normalising the other signals according to the minimum and maximum of dataset A. Various learning rates were tested. Ideally, if the projects are somewhat similar, a small learning rate should be used to avoid "unlearning" what was previously learned with project A. Conversely, if the projects differ significantly, larger learning rates would be necessary. In this case, a learning rate of 0.001 (within the same range as the initial learning rate) was employed, as it yielded the lowest pinball loss. Additionally, it was varied whether the entire model A was retrained or if the existing layers of model A were frozen and only a linear or GRU layer on top was trained. Further details, including the tuned hyperparameters of model B, can be found in section C.2, section C.3 Figure B.9.3.

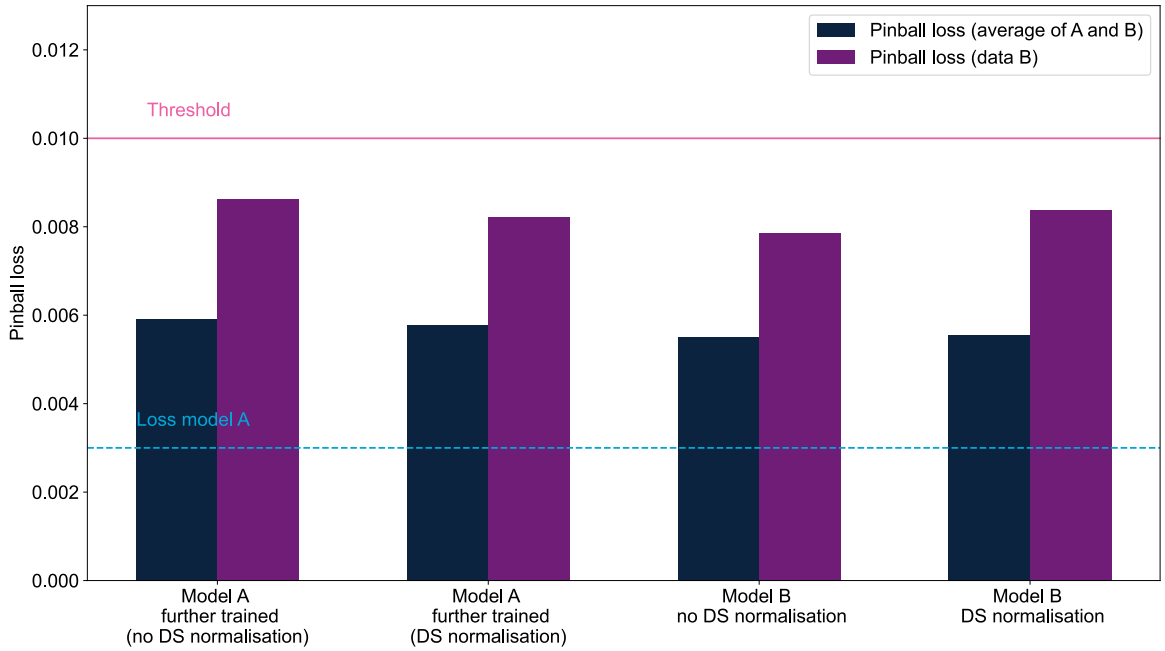


Figure 5.24: Comparison of the pinball loss for different methods of using training a separate model for vehicle project A.

Figure 5.24 illustrates the pinball loss for the three different configurations. The purple bar represents the loss for project B only, while the dark blue bar denotes the average loss for both projects. All four models achieve pinball losses below the threshold set at 0.01. The pinball loss for data B (purple) varies only slightly, with the highest loss occurring when the model is further trained with model A. Given that the learning rate and the number of epochs used to achieve this test loss (the lowest loss among the options tested) are quite high (learning rate around 0.001 and 89 epochs), this may indicate that the two cells are too different to be trained with the same model. Additionally, it is evident that for all model options, the loss for data B is significantly higher than the loss achieved for data A (dotted blue line). This discrepancy could be attributed to the availability of data, as less than 70% of the data for project A was available for project B. Therefore, it is expected that if more data were available, the model performance would also improve.

5.10.3. Results: Driving Cycle Generalisation

To test the hypothesis that the model generally does not perform well with unseen types of driving cycles, vehicle measurements from the calibration engineers were utilised. These measurements correspond to various driving cycles, ranging from high-speed driving cycles at the test ring in Nardo, Italy, to driving cycles between locations in Germany. It is important to note that some of these measurements have an average temperature outside the temperature range with which the model was initially trained.

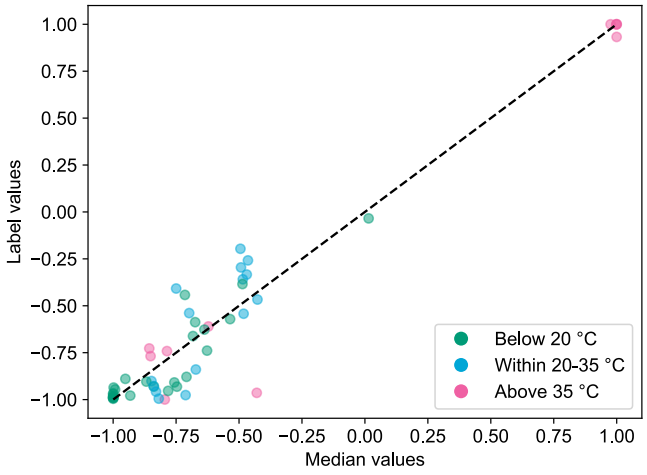


Figure 5.25: Predicted median vs. label for the calibration measurements.

In Figure 5.25, it is evident that the GRU model with the 600-240 configuration does not perform well on vehicle measurements, irrespective of whether the temperature is within the original training range or not. The average pinball loss is 0.03807, with the highest loss being 0.14757. The poor performance cannot be solely attributed to samples with temperatures outside the initial training range, as indicated by the green and pink markers in Figure 5.25. Notably, when the starting hysteresis factor deviates significantly from the subsequent hysteresis factor (anticipated after 600 s according to the chosen interval size), the predictions tend to drift from the start. This observation raises the question of whether the model choice, despite achieving the lowest loss in Case Study 5, is truly optimal, particularly for shorter, more dynamic driving cycles. The larger interval sizes (e.g. 600 s) might be problematic, as the hysteresis factor can change dramatically within that time frame.

To illustrate this, the "measured" hysteresis factor used for labelling is depicted in Figure 5.26 with the original sampling period of 0.1 s. Note that this measurement corresponds only to the currently implemented algorithm and is not a physical quantity. The blue markers indicate where the 600 s intervals would end and start. It can be seen that, for the first interval, the hysteresis factor decreases from an initial value of approximately -0.75 to -0.93.

First, the 60-120 model, which has a 10 times smaller interval size and resulted in only a slightly higher loss (0.003567 compared to 0.003176) in Case Study 5, was used to test the new driving cycles. This resulted in an average loss of 0.02760 with a maximum loss of 0.06263. Although the overall loss is slightly lower, the average loss remains above the threshold. Interestingly, the highest loss is significantly reduced (from 0.14757 with the 600 s interval to 0.06263). The 300-120 configuration, i.e. an interval size of 300 s, which also performed well in Case Study 5, was also tested, yielding results comparable to the 600 configurations (average loss of 0.04718 with a maximum loss of 0.15808). A comparison of all three models using the same measurement is provided in subsection B.9.4.

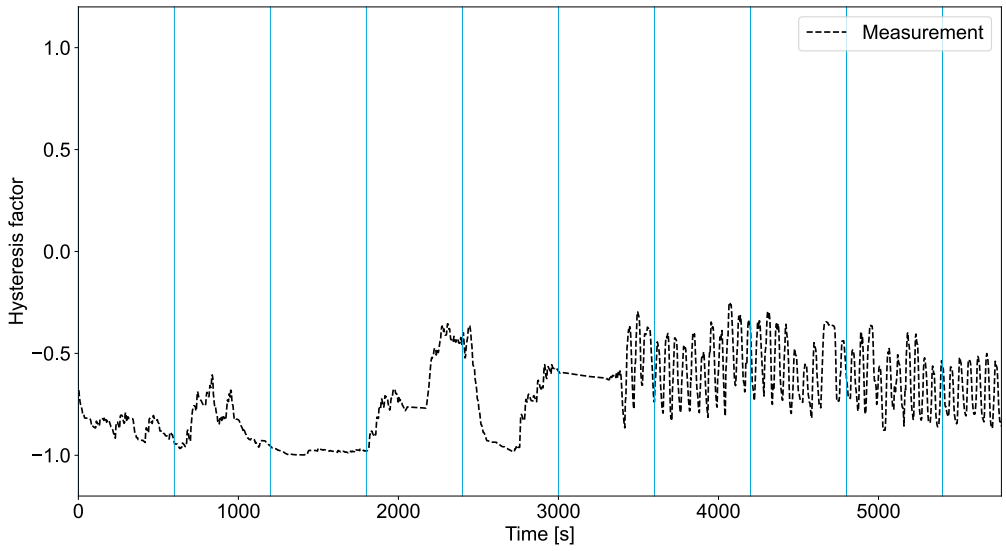


Figure 5.26: Measured hysteresis factor (black) and 600 s intervals (in blue).

These results suggest that using a smaller interval size might be beneficial when more dynamic measurements are available in the dataset. However, Case Study 5 would need to be repeated with a suitable, more diverse dataset to confirm this.

In general, while the 60-120 configuration performs better, especially in reducing large pinball losses in measurements, the achieved loss for many measurements still exceeds the threshold of 0.01 (as indicated by the average loss). This suggests that regardless of the model configuration (e.g., smaller or larger interval size), a dataset with more diverse and dynamically changing driving profiles is needed. Most of the test bench tests are only-charge or only-discharge tests, and if a more dynamic driving cycle is included, it is usually the same set of a few standardised driving cycles.

Additionally, it is possible that the algorithm used to label the data performs worse under very dynamic conditions. After observing measurements such as Figure 5.26, it is questionable whether the hysteresis factor would actually have such steep slopes, especially in the second half of the measurement starting around 3700 s. The main issue with the machine learning model seems to be that the slopes are not predicted correctly. The predicted gradient appears flatter than the actual label. To test this hypothesis, more in-depth measurements from the cell would be needed to analyse the gradient behaviour in dynamic situations, independent of the currently implemented algorithm which was used for labelling.

5.10.4. Results: Temperature Range Generalisation

In the previous section, there were many samples that were outside the original trained temperature range, so it was decided to also examine the effect of temperature on the model's performance. To do this, measurements from project A were taken, specifically those that were previously excluded due to being outside the temperature range. Generally, these measurements have similar driving cycles as the initial training set. This indicates that the datasets are comparable, however, conducted at different temperatures.

First, it was verified whether the currently trained GRU model (600-240 configuration) could be applied to temperatures outside the 20-35$\circ$C range. The resulting label vs. median plot is shown in Figure 5.27. The overall average test loss in this case is 0.04144, which is more than 10 times higher than the previously achieved test loss with samples within the range. This confirms that the initially trained model is sensitive to changes in temperature. Therefore, in practice, it would not be recommended to apply the initially trained model to situations outside of the temperature range.

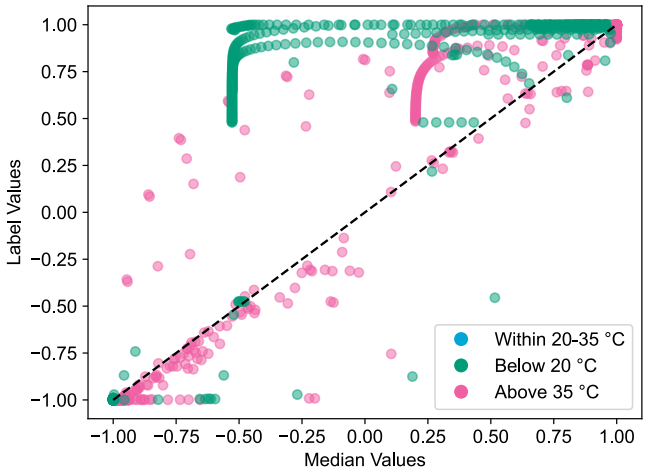


Figure 5.27: Predicted median vs. label testing the out-of-temperature-range measurements with model A.

Next, it was evaluated whether the model could be retrained to include all samples, regardless of temperature. The tuned hyperparameters are detailed in subsection B.9.5. The predicted median vs. label plot is shown in Figure 5.28. The performance does decrease compared to the training where only temperatures in the 20-35°C range were considered, with the pinball loss increasing from 0.003176 to 0.00472. Notably, the greatest outliers originate from samples within the 20-35°C range. This could also be due to the split of training and test data; if certain types of driving cycles were not included in the training set but only in the test set due to random selection.

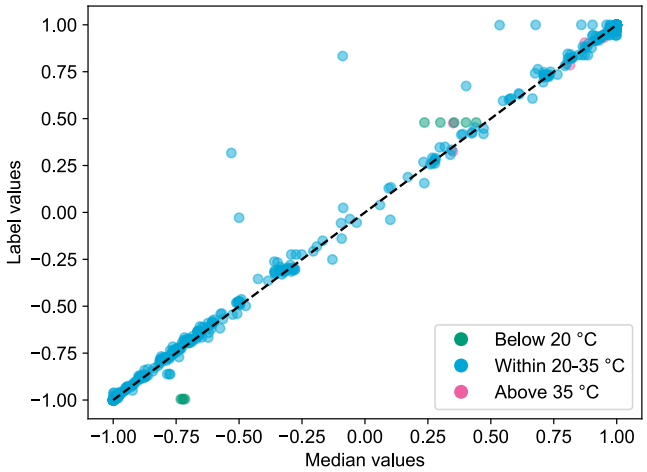


Figure 5.28: Predicted median vs. label for retraining the model with all temperature samples.

Moreover, it appears that samples below 20°C (in green) tend to have greater errors (distance to the ideal diagonal line) than samples above 35°C (in pink). It is important to note that during the training, validation, and test split, it was ensured that samples in each temperature bin (green, blue, and pink in Figure 5.28) were equally distributed among the train, test, and validation sets. Further tuning of the learning rate and the number of epochs could potentially enhance the performance of the retrained model A. Additionally, the model could be improved by providing more samples outside of the initial temperature range to achieve a more even data distribution.

6

Discussion and Conclusion

This chapter analyses and interprets the research findings and revisits the problem statement formulated in chapter 1. The insights derived from the results address the initially posed research questions regarding the development and evaluation of various black box models to determine the hysteresis factor from driving cycles. The discussion highlights the effectiveness and practical feasibility of these models, emphasising their accuracy and implementability. Additionally, the limitations encountered during the research are acknowledged, and an outlook on potential future studies and applications based on the findings is given.

6.1. Problem Statement and Proposed Approach

The SOC estimation of batteries is crucial for battery management in EVs as it directly affects vehicle performance and possible range. More precise SOC estimation allows for decreased safety margins in SOC boundaries, enhancing battery usability by permitting greater depth of discharge. This results in increased range or reduced battery size and weight, potentially lowering EV costs. Traditional SOC estimation methods, like Coulomb counting, lack accuracy therefore corrective measures based on OCV are necessary. However, novel silicon graphite anodes exhibit significant voltage hysteresis and complicate OCV-based corrections. This thesis uses machine learning models to address SOC estimation challenges posed by voltage hysteresis. The objective is to formulate a data-driven model for hysteresis factor prediction, accounting for the technical limitations of BMS and vehicle use cases. Improvements in SOC estimation and enabling the use of new cell types, such as silicon graphite anodes, are expected to yield both economic and environmental benefits, advancing the efficiency and acceptance of EVs.

This thesis formulates a black box model to predict the hysteresis factor using driving cycles. The data taken from multiple vehicle projects includes signals measured by the BMS such as current, voltage, and cell temperature. The objective is to identify a model that can operate within the computational constraints of the BMS. Additionally, the thesis addresses the uncertainty in SOC estimation, acknowledging that existing literature often assumes the "measured" SOC as ground truth without considering its inherent uncertainty. Through preparation, training, testing, and comparison of various machine learning models, the study aims to evaluate their performance and limitations.

6.2. Answering Research Questions

6.2.1. Objective 1: Develop a black box model of the voltage hysteresis effect to determine the hysteresis factor using driving cycles of EVs

Can a regression machine learning model accurately predict the hysteresis factor given the computational constraints of a battery management system?

This research question aims to assess the viability of using machine learning regression models to predict the hysteresis factor within the operating limits of a battery management system. Various models of differing complexity were designed, ranging from linear regression to XGBoost to a gated recurrent unit (GRU) neural network. Three general approaches were explored: transforming the initial input signal into attributes, maintaining them as time series, and splitting them into smaller intervals. The last approach offered the most

promising results as it led to the lowest pinball loss and thus the highest accuracy.

While some models demonstrated the capability to achieve the memory requirements, not all of the model configurations were successful. This indicates that not only the model itself but also the configuration of input data plays a critical role. For instance, albeit being the most complex model tested, the GRU models generally performed well in terms of memory requirements. However, if the hyperparameter of the models, such as the number of hidden units or number of hidden layers is too high, the memory thresholds would be exceeded. Similarly, the number of features (for Case Study 1) and the number of considered time steps (for Case Study 2, 5 and 6), both of which result from the input configuration, are directly linked to the required memory.

Can quantile regression indicate the confidence of the predicted hysteresis factor on a given dataset of driving data?
This research question addresses the uncertainty of the labels and whether it is possible to effectively manage it with quantile regression. If the implemented algorithm used for labelling does not introduce variation in the data, the resulting quantiles will not aid in determining confidence levels. In such cases, the resulting quantiles will differ from the median only due to errors in the model itself due to incorrect quantile predictions.

To illustrate this further, consider a hypothetical scenario where the feature space is one-dimensional with Feature 1. If the currently implemented algorithm, which was used for labelling, predicts the hysteresis factor consistently for Feature 1, there should be minimal noise in the input data for the machine learning algorithm to learn from. However, due to the higher dimensionality of the feature space, particularly given that depending on the case study the input signals are time-series, it is assumed that there is noise in the input data. This assumption is further supported by the current algorithm's inconsistent performance, meaning that its accuracy fluctuates and only the maximum error is known. This noise can then be considered through the use of quantile regression, which provides a means to examine the variability and uncertainty within the predicted values.

Based on observations from the reconstructed measurements in Case Study 5, the most successful among the case studies examined, the quantile range is relatively narrow, suggesting limited variability in the input data. As previously noted, a larger gap between the lower and upper quantiles may also indicate model inaccuracy for these two quantiles. The limited range aligns with the expected maximum error of the current algorithm, which is within acceptable bounds. Consequently, it is questionable whether quantile regression offers significant practical benefits in this context.

Ideally, quantiles could help assess the confidence level in the predicted hysteresis factor. For instance, if the upper and lower quantiles are widely separated, this could suggest high noise levels at the given operating point, indicating that the available labels are not adequately representative. In such cases, the information provided by the quantiles might guide a decision to not adjust the SOC and instead wait for the next driving cycle.

6.2.2. Objective 2: Perform an evaluation and comparison of different proposed black box models to find the most suitable algorithm to determine the hysteresis factor in terms of accuracy and implementability

Which is the best model according to priorly defined evaluation criteria?
The aim of the research question is to establish criteria through discussions with stakeholders that provide a basis for comparing designed models. The selected model can subsequently be utilised for the following research question, which focuses on the comparison of vehicle models.

The selected criteria consist of two components: pinball loss, which measures the accuracy of the model's predictions, and memory requirements, divided into ROM and RAM. For both components, thresholds were established to normalise the achieved values. Specifically, the pinball loss was normalised to a maximum of 0.01. If the achieved loss exceeds this threshold, it would contribute as a negative addend in the calculation of the total score. The threshold value of 0.01 was chosen based on experiments, as loss values above this threshold correspond to a level of accuracy infeasible for practical use. The thresholds for ROM and RAM were set at 0.3 MB and 0.1 MB, respectively. These values are set based on a real battery control unit and an estimation of the available free memory for the algorithm.

The configuration of the input data played a significant role in the model's performance. Specifically, whether

the input is in the form of attributes, the entire time series or time intervals, was critical. In this study, the interval-based approach consistently yielded the best results across different types of models. The details of the input configuration, such as the interval size and the sampling rate of the input signal, also proved to be important for the simple models, but less important for the GRU models. The comparison of case studies showed that a GRU model, with interval input, performed best. In this case, the best configuration was that of an interval length of 600s and 240 elements per time series. Do note that the differences in loss between the configurations were comparatively small.

The memory requirements were considered secondary, as the differences between models in terms of memory usage were less significant compared to the differences in loss performance.

Finally, a sensitivity analysis was conducted. The analysis confirmed the previous model choice across all scenarios, including the base scenario, one where the loss was weighted more than 70%, and one where it was weighted less.

Can a single algorithm be effectively trained to determine the hysteresis factor across different vehicle projects while maintaining high accuracy?
This question aims to investigate whether a single model can be trained and subsequently applied across various vehicle projects that utilise different cell chemistries. The primary advantage of such a model would be the elimination of the need for retraining, thereby potentially extending its applicability to vehicle projects where acquiring training data and labels is particularly challenging.

However, Case Study 7, which involved two different vehicle models, revealed that the performance of a universally trained model is significantly compromised. This performance reduction is likely due to the variations in cell chemistries, which influence the determination of the hysteresis factor. While the approach of training a separate model for each vehicle project worked best, the approach of further training an existing model for a new vehicle project resulted in a comparable pinball loss.

Given the data-driven nature of the employed method, it is understandable that differing datasets would affect the model's generalisation capability. Furthermore, extensive testing on a wide variety of cell types would be required to confidently assert the universal applicability of the model. In this study, only two cell types were examined, which is insufficient to generalise the findings across different cell chemistries.

6.3. Interpretation of Results

The analysis reveals that the black box model performs effectively when the training data is relevant and comprehensive. Specifically, the model trained with test bench data showed robust performance for tests conducted on the test bench (evident in Case Study 5). Previous studies employing machine learning to model hysteresis, such as those by Li et al. and Xu et al. [50, 8], did not sufficiently verify model performance outside laboratory settings. In contrast, this work is based on driving cycles, which provide a more realistic framework. While others have used driving cycles in their validation processes [42, 30], they still rely on constant current pulse tests to calibrate parameters [106], using driving cycles only for additional validation. This study, however, directly employs driving cycles to train the model parameters, offering a key advantage from an industry perspective. The required measurement data is already available, as it is used to calibrate other algorithms, eliminating the need for dedicated test benches for constant current pulse tests. Therefore, this approach provides a practical alternative to conventional Kalman filters, provided that labelling the training samples is feasible (e.g., through predictions from another algorithm or using the "stability criterion", which is further discussed in section 6.5).

In this work, the model's ability to generalise across different driving cycles needs improvement, as highlighted in Case Study 7. This raises the question of whether models that have not been validated using driving cycles (see Figure 1.4) can generalise sufficiently for practical use in vehicles. Even studies that do validate using driving cycles [42, 30] often rely on a single driving cycle, such as the "Hill Route" from the First Group Millbrook Fuel Economy Trial [42], where the authors acknowledge that their error may increase with more dynamic cycles, or the urban dynamometer driving schedule (UDDS) cycle [30]. While these cycles cover light-duty electric vehicle driving conditions, this work encompasses a broader range of driving cycles, better reflecting real-world conditions and sports car-specific behaviour (i.e., non-urban driving).

Moreover, these previous studies [42, 30] focus on (mild) hybrid electric vehicles (HEVs), which have smaller battery capacities compared to fully electric vehicles (EVs), potentially reducing the impact of hysteresis.

Additionally, HEVs exhibit different charging and discharging patterns during driving cycles, as the internal combustion engine can recharge the battery during operation.

To improve model performance across a wider range of conditions, this work identifies the need for additional measurements, particularly at extreme temperatures. Case Study 7 also emphasizes the importance of sufficient training data, as demonstrated by the poorer performance of the model trained on vehicle project B compared to vehicle project A, which had more than 1.5 times the amount of measurement data.

This study also found that training a single model across multiple vehicle projects did not yield the same performance as models trained separately for each project, likely due to differences in cell chemistries. This finding aligns with Barai et al. [7], who reported significant variations in hysteresis behaviour across different cell types. The most promising approach for predicting the hysteresis factor using BMS signals was identified as a GRU model, combined with splitting the initial time series signals into intervals.

From the industrial perspective is it interesting that the memory requirements suggest that simple machine learning models could be implemented in the BMS. However, the decision to use machine learning for hysteresis factor prediction hinges on the availability of training data and the effort required to obtain it. Currently, the machine learning-based hysteresis factor prediction does not outperform the existing algorithm. Looking forward, with potential advancements in extreme cell designs necessary to stay competitive in the EV market, the calibration limits of the current algorithm may be reached. Machine learning could serve as a viable alternative, necessitating a dedicated test bench for data generation and a shift from the current algorithm to Coulomb counting. This shift would require minimised current error, a correct start SOC, and a well-learned capacity—feasible in a test bench setting across various driving cycles and temperatures. Furthermore, real driving cycles should also be provided to the model, which may be much more difficult to label. This is further discussed in the limitations and outlook.

These advanced cell designs are crucial to achieving greater ranges and faster charging which are key expectations for high-performance vehicles. Moreover, mature extreme cell designs could allow for smaller, cost-saving batteries while maintaining the same range.

For society, the potential for greater vehicle ranges addresses range anxiety—a significant barrier to EV adoption, especially in regions lacking charging infrastructure. While Porsche vehicles optimise performance, other Volkswagen Group vehicles might prioritise cost reduction. Should the cost savings from smaller batteries be passed on to consumers, EVs could become more attractive. Thus, extreme cell designs could foster greater EV adoption, supporting the mobility transition necessary to reduce greenhouse gas emissions. Accurate hysteresis factor prediction is also vital for precise state-of-health estimation, which is crucial for the second-hand market. In the US and European Union, the state-of-health estimation must have an accuracy of up to 5%, obliging manufacturers to meet this standard for market eligibility [107, 108].

6.4. Limitations

6.4.1. Label and Data Dependency

To ensure optimal performance, it is necessary that the labels used in the model are accurate (enough). It is important to note, that the designed model can only be as good as its labels. Therefore, if a currently already implemented algorithm is used, the model's prediction cannot be better than the algorithm's prediction. However, what this designed black box model can deliver is additional information such as the lower and upper quantiles.

The necessity of sufficient training data was evident in this study; the model showed better performance with a total of 800 measurements (approximately 12,000 samples in vehicle project A) compared to 500 measurements (around 7,000 samples in vehicle project B). Additionally, it is important to recognize that this methodology relies on the OCV curves for charging and discharging provided by the cell manufacturer, which have been assumed to be accurate. Notably, both tested vehicle models incorporate only a negligible amount of silicon in their cells. Therefore, expanding the input dataset to include cells with higher silicon content is necessary to validate whether this methodology works for these cell types as well.

6.4.2. Driving Cycles

The designed black box model performs exceptionally well with driving cycles that were included during the training phase. Given that the tests were predefined, they showed high accuracy with charging and discharging cycles, as well as some mixed charge-discharge driving cycles. However, performance may degrade with other driving cycles. The model has been trained exclusively on measurements taken at average temperatures between 20 and 35◦C and it was shown that the performance decreases when training on the full range of temperatures, likely due to fewer measurements being available outside of this range.

6.4.3. Hyperparameter Tuning and Model Selection Considerations

The tuning process involved stochastically chosen parameters, limited to a maximum of 25 configurations. For the XGBoost model, early stopping was implemented after 10 rounds, while the GRU model was run for either 10 or 30 epochs, depending on whether it was for initial tuning or fine-tuning. Most importantly, the tuning process selected the best parameters based on validation loss. However, due to time constraints, the validation loss was based on a non-regressive loss, which is especially significant for the time interval approaches. Given more time or greater computational resources, it would be beneficial to adapt the tuning functions to utilise a custom validation loss calculation that accounts for the connections between intervals, where the last prediction serves as input for the next prediction.

Further improvement in tuning could be achieved through a separate study employing a more robust hyperparameter tuning strategy, such as Bayesian Optimisation, which may enhance the model's performance. During model selection, it was assumed that the model excelling in terms of both loss and memory efficiency would be the most suitable for vehicle comparison. However, this assumption may not hold universally and should therefore be considered cautiously.

6.5. Outlook

6.5.1. Variety in the Dataset

To improve the model's performance, it is essential to diversify the input data. This may include data from a designated test bench that conducts current pulse tests. While these profiles may not closely resemble typical customer scenarios, they yield more accurate Coulomb counting SOC measurements. Additionally, incorporating data from prototype vehicles, such as test results from calibration engineers, would be beneficial. From an industry standpoint, this data is as readily accessible as the test bench data. Since the selected interval approach, thus the continuous determination of the hysteresis factor does not require a relaxation period at the end of the measurement, the absence of relaxation periods in these vehicle measurements is not an issue. However, it is crucial to ensure that these vehicle measurements are sufficiently "clean" to be used as training data. This means that the calibration engineers may not change any of the essential parameters. Thus, preparing this data might require more expert involvement to filter out the unusable measurements (due to parameters being set to "wrong" values for certain test cases).

In the work, only measurements were considered where the cell can also relax. As mentioned above, it is also possible to include measurements without relaxation periods in the interval approach. This increases the size of the training dataset as more test bench measurements, and thus a greater variety of driving cycles, are suitable for training. This mix of data should provide a diverse input for the machine learning model to learn from.

6.5.2. Selection of Suitable Measurements and More Extreme Cell Chemistries

This methodology so far has been shown to work on cells with a very low amount of silicon and thus also a small amount of hysteresis. It would be necessary to verify if the methodology also holds with cells with greater hysteresis. A greater amount of hysteresis might also cause issues in one of the prerequisites for the methodology though, namely the labelling. The greater the hysteresis, the more difficult it may be to find "good-enough" labels to use. Unfortunately, this work indicated that it is not possible to use a model trained on another vehicle project and apply it to a new one. If this was the case, no labels for new vehicle projects would be required.

6.5.3. Alternative Labelling and Test Fleet Training

An alternative labelling approach is the "stability criterion," which has two main components: a long-term and a short-term component. The short-term component assesses the stability of the SOC. Depending on the duration of the active period, a qualitative decision can be made regarding the plausibility of the corrected SOC. For instance, it is questionable if a 7% correction is made to the SOC after only a 5-minute active period, which would mean that the Coulomb counting result is 7% off. To apply this method effectively, appropriate thresholds must be established in advance, and it is critical to account for measurement conditions (e.g., active period length, confidence in initial SOC, etc.).

The second component is the stability of the battery's capacity. Capacity is estimated only when the SOC change during the driving cycle is sufficiently large. The BMS re-estimates the current capacity based on the corrected SOC. Over time, a gradual decrease in capacity due to ageing effects is expected. Inaccuracies in the SOC can lead to errors in the capacity estimation, often observable as irregular fluctuations in capacity over time. Examples of this are an increase in capacity and a sudden, drastic decrease.

Since this labelling method is less quantifiable than the one based on the current algorithm, a practical strategy would be to initially train a baseline model using the current algorithm, even if it is not fully accurate. This model could then be implemented in a test fleet, where it would be further tuned using the stability criterion, which can be used even with more extreme cell types. This step is essential to ensure the algorithm can be used in the production software and thus be implemented in customer vehicles.

6.5.4. Model Complexity and Computational Considerations

In the future, it might also be possible to train more complex models due to developments in control units in the car. For instance, putting non-safety relevant functions on another control unit, which has more computational power, or establishing a connection with a remote PC, doing the calculations there and then sending the results back to the car. This would probably be problematic for the interval approach though due to timing constraints and potential connectivity issues.

6.5.5. Ageing

It would also be interesting, if enough data is available, to train the model also considering ageing. The effect of ageing on hysteresis is still an ongoing research topic. One possible approach for incorporating ageing effects is to introduce an ageing factor that modifies the charge and discharge curves originally provided for new cells by the manufacturer. This approach, however, assumes that ageing does not influence the hysteresis factor itself, such as the rate at which the cell transitions between charge and discharge curves.

6.5.6. Autoregressive Training

Autoregressive training was done in Case Study 6 in order to include the autoregressive nature of the problem in the training process. In this work, this approach was not further followed due to limited time and because the results did not indicate a strong improvement. However, it could be seen that when the model trained on project A was applied to project B, an initial wrong prediction offsets later predictions throughout the whole measurements Therefore, it could be interesting to check if an autoregressive model, after tuning it a bit more in terms of learning rate and number of epochs, may have better generalisation capacity because it might be better at stopping these "drift-offs".

References

[1] Veronika Henze. *Lithium-ion Battery Pack Prices Rise for First Time to an Average of $151/kWh*. Dec. 2022. URL: https://about.bnef.com/blog/lithium-ion-battery-pack-prices-rise-for-first-time-to-an-average-of-151-kwh/.

[2] Menaka Samant et al. *EY Mobility Consumer Index 2022 study*. Tech. rep. EY, 2022.

[3] Tim J. Barlow et al. *A reference book of driving cycles for use in the measurement of road vehicle emissions*. Tech. rep. Transport Research Laboratory, 2009.

[4] Andrew Evers. *Why Porsche, Mercedes and GM are betting on silicon-anode batteries*. 2022.

[5] Amprius. *SiMaxxTM Enables Superior Energy Density for High-Performance Applications*. URL: https://amprius.com/technology/.

[6] Yizhao Gao et al. "Determination of half-cell open-circuit potential curve of silicon-graphite in a physics-based model for lithium-ion batteries". In: *Applied Energy* 349 (Nov. 2023). ISSN: 03062619. DOI: 10.1016/j.apenergy.2023.121621.

[7] Anup Barai et al. "A study of the open circuit voltage characterization technique and hysteresis assessment of lithium-ion cells". In: *Journal of Power Sources* 295 (July 2015), pp. 99–107. ISSN: 03787753. DOI: 10.1016/j.jpowsour.2015.06.140.

[8] Zhicheng Xu et al. "Improving the state of charge estimation of reused lithium-ion batteries by abating hysteresis using machine learning technique". In: *Journal of Energy Storage* 32 (Dec. 2020). ISSN: 2352152X. DOI: 10.1016/j.est.2020.101678.

[9] Lin Wang et al. "Application of electrochemical impedance spectroscopy in battery management system: State of charge estimation for aging batteries". In: *Journal of Energy Storage* 57 (Jan. 2023). ISSN: 2352152X. DOI: 10.1016/j.est.2022.106275.

[10] International Energy Agency (IEA). *Global EV Outlook 2023: Catching up with climate ambitions*. Tech. rep. 2023. URL: www.iea.org.

[11] Hao Zhang et al. *Graphite as anode materials: Fundamental mechanism, recent progress and advances*. Apr. 2021. DOI: 10.1016/j.ensm.2020.12.027.

[12] Yuan-Li Ding et al. "Automotive Li-Ion Batteries: Current Status and Future Perspectives". In: *Electrochemical Energy Reviews* 2.1 (2019), pp. 1–28. DOI: 10.1007/s41918-018-0022-z.

[13] Da Deng. "Li-ion batteries: Basics, progress, and challenges". In: *Energy Science and Engineering* 3.5 (Sept. 2015), pp. 385–418. ISSN: 20500505. DOI: 10.1002/ese3.95.

[14] Kong Soon Ng et al. "Enhanced coulomb counting method for estimating state-of-charge and state-of-health of lithium-ion batteries". In: *Applied Energy* 86.9 (Sept. 2009), pp. 1506–1511. ISSN: 0306-2619. DOI: 10.1016/J.APENERGY.2008.11.021.

[15] Shijie Tong, Matthew P. Klein, and Jae Wan Park. "On-line optimization of battery open circuit voltage for improved state-of-charge and state-of-health estimation". In: *Journal of Power Sources* 293 (June 2015), pp. 416–428. ISSN: 03787753. DOI: 10.1016/j.jpowsour.2015.03.157.

[16] Tarun Huria et al. "High Fidelity Electrical Model with Thermal Dependence for Characterization and Simula-tion of High Power Lithium Battery Cells". In: *2012 IEEE International Electric Vehicle Conference*. 2012, pp. 1–8. DOI: 10.1109/IEVC.2012.6183271.

[17] Lei Pei, Rengui Lu, and Chunbo Zhu. "Relaxation model of the open-circuit voltage for state-of-charge estimation in lithium-ion batteries". In: *IET Electrical Systems in Transportation* 3.4 (2013), pp. 112–117. ISSN: 20429738. DOI: 10.1049/iet-est.2013.0020.

[18] Hao Yang et al. "Online parameters identification and state of charge estimation for lithium-ion capacitor based on improved Cubature Kalman filter". In: *Journal of Energy Storage* 24 (Aug. 2019). ISSN: 2352152X. DOI: 10.1016/j.est.2019.100810.

[19] William McSweeney, Hugh Geaney, and Colm O'Dwyer. "Metal-assisted chemical etching of silicon and the behavior of nanoscale silicon materials as Li-ion battery anodes". In: *Nano Research* 8.5 (May 2015), pp. 1395–1442. ISSN: 19980000. DOI: 10.1007/s12274-014-0659-9.

[20] Haipeng Zhao et al. "Modification of natural graphite for lithium ion batteries". In: *Solid State Sciences* 10.5 (May 2008), pp. 612–617. ISSN: 12932558. DOI: 10.1016/j.solidstatesciences.2007.10.017.

[21] Jun Lee et al. "Silicon Anode: A Perspective on Fast Charging Lithium-Ion Battery". In: *Inorganics* 11.5 (May 2023). ISSN: 23046740. DOI: 10.3390/inorganics11050182.

[22] Prachi Patel. *The Age of Silicon Is Here... for Batteries*. 2023. URL: https://spectrum.ieee.org/silicon-anode-battery.

[23] Yang He et al. "Progressive growth of the solid–electrolyte interphase towards the Si anode interior causes capacity fading". In: *Nature Nanotechnology* 16.10 (Oct. 2021), pp. 1113–1120. ISSN: 17483395. DOI: 10.1038/s41565-021-00947-8.

[24] Jinglei Lei et al. "Characterization of SEI Layers on LiMn2O4 Cathodes with In Situ Spectroscopic Ellipsometry". In: *Journal of the Electrochemical Society* 152.4 (2005). ISSN: 00134651. DOI: 10.1149/1.1867652.

[25] Jeong K. Lee et al. "Silicon nanoparticles-graphene paper composites for Li ion battery anodes". In: *Chemical Communications* 46.12 (2010), pp. 2025–2027. ISSN: 13597345. DOI: 10.1039/b919738a.

[26] Rui Huang and Jing Zhu. "Silicon nanowire array films as advanced anode materials for lithium-ion batteries". In: *Materials Chemistry and Physics* 121.3 (June 2010), pp. 519–522. ISSN: 02540584. DOI: 10.1016/j.matchemphys.2010.02.017.

[27] Candace K. Chan et al. "Solution-grown silicon nanowires for lithium-ion battery anodes". In: *ACS Nano* 4.3 (Mar. 2010), pp. 1443–1450. ISSN: 19360851. DOI: 10.1021/nn901409q.

[28] Michael A. Roscher, Oliver Bohlen, and Jens Vetter. "OCV Hysteresis in Li-Ion Batteries including Two-Phase Transition Materials". In: *International Journal of Electrochemistry* 2011 (2011), pp. 1–6. DOI: 10.4061/2011/984320.

[29] Wenlu Zhou et al. "SOC Estimation Based on Hysteresis Characteristics of Lithium Iron Phosphate Battery". In: *Machines* 10.8 (Aug. 2022). ISSN: 20751702. DOI: 10.3390/machines10080658.

[30] Gregory L. Plett. "Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs - Part 2. Modeling and identification". In: *Journal of Power Sources* 134.2 (Aug. 2004), pp. 262–276. ISSN: 03787753. DOI: 10.1016/j.jpowsour.2004.02.032.

[31] Philip Kargl et al. "Investigation of voltage and expansion hysteresis of Si-alloy-C/NMC622 pouch cells using dilatometry". In: *Journal of Power Sources* 548 (Nov. 2022). ISSN: 03787753. DOI: 10.1016/j.jpowsour.2022.232042.

[32] Qiang Liu et al. "Kinetically Determined Phase Transition from Stage II (LiC12) to Stage i (LiC6) in a Graphite Anode for Li-Ion Batteries". In: *Journal of Physical Chemistry Letters* 9.18 (Sept. 2018), pp. 5567–5573. ISSN: 19487185. DOI: 10.1021/acs.jpclett.8b02750.

[33] Wolfgang Dreyer et al. "The thermodynamic origin of hysteresis in insertion batteries". In: *Nature Materials* 9.5 (2010), pp. 448–453. ISSN: 14764660. DOI: 10.1038/nmat2730.

[34] Vijay A. Sethuraman et al. "In Situ Measurements of Stress-Potential Coupling in Lithiated Silicon". In: *Journal of The Electrochemical Society* 157.11 (2010), A1253. ISSN: 00134651. DOI: 10.1149/1.3489378.

[35] Vincent L. Chevrier and Jeff R. Dahn. "First Principles Studies of Disordered Lithiated Silicon". In: *Journal of The Electrochemical Society* 157.4 (2010), A392. ISSN: 00134651. DOI: 10.1149/1.3294772.

[36] Yang Jiang et al. "Voltage Hysteresis Model for Silicon Electrodes for Lithium Ion Batteries, Including Multi-Step Phase Transformations, Crystallization and Amorphization". In: *Journal of The Electrochemical Society* 167.13 (Oct. 2020), p. 130533. ISSN: 0013-4651. DOI: 10.1149/1945-7111/abbbba.

[37] Michael A. Roscher and Dirk Uwe Sauer. "Dynamic electric behavior and open-circuit-voltage modeling of LiFePO 4-based lithium ion secondary batteries". In: *Journal of Power Sources* 196.1 (Jan. 2011), pp. 331–336. ISSN: 03787753. DOI: 10.1016/j.jpowsour.2010.06.098.

[38] Loic Lavigne et al. “Lithium-ion batteries aging motinoring througth open circuit voltage (OCV) curve modelling and adjustment”. In: *ICINCO 2016 - Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics*. Vol. 1. SciTePress, 2016, pp. 57–67. ISBN: 9789897581984. DOI: 10.5220/0005961400570067.

[39] Jiale Xie et al. “Estimating the state-of-charge of lithium-ion batteries using an H-infinity observer with consideration of the hysteresis characteristic”. In: *Journal of Power Electronics* 16.2 (Mar. 2016), pp. 643–653. ISSN: 15982092. DOI: 10.6113/JPE.2016.16.2.643.

[40] Younghwi Ko and Woojin Choi. “A new soc estimation for lfp batteries: Application in a 10 ah cell (hw 38120 l/s) as a hysteresis case study”. In: *Electronics (Switzerland)* 10.6 (Mar. 2021), pp. 1–14. ISSN: 20799292. DOI: 10.3390/electronics10060705.

[41] Shuyu Xie et al. “State-of-Charge Estimation of Lithium-Ion Battery Based on an Improved Dual-Polarization Model”. In: *Energy Technology* 11.4 (Apr. 2023). ISSN: 21944296. DOI: 10.1002/ente.202201364.

[42] Ryan Rolt et al. “Full battery pack Modelling: An electrical Sub-Model using an EECM for HEV applications”. In: *SAE Technical Papers*. Vol. 2019-April. April. SAE International, Apr. 2019. DOI: 10.4271/2019-01-1203.

[43] Yanxin Xie et al. “Improved lumped electrical characteristic modeling and adaptive forgetting factor recursive least squares-linearized particle swarm optimization full-parameter identification strategy for lithium-ion batteries considering the hysteresis component effect”. In: *Journal of Energy Storage* 67 (Sept. 2023). ISSN: 2352152X. DOI: 10.1016/j.est.2023.107597.

[44] Minkyu Kwak et al. “Parameter Identification and SOC Estimation of a Battery under the Hysteresis Effect”. In: *IEEE Transactions on Industrial Electronics* 67.11 (Nov. 2020), pp. 9758–9767. ISSN: 15579948. DOI: 10.1109/TIE.2019.2956394.

[45] Federico Baronti et al. “Experimental analysis of open-circuit voltage hysteresis in lithium-iron-phosphate batteries”. In: *IECON Proceedings (Industrial Electronics Conference)* (2013), pp. 6728–6733. DOI: 10.1109/IECON.2013.6700246.

[46] Guangzhong Dong et al. “Online state of charge estimation and open circuit voltage hysteresis modeling of LiFePO4 battery using invariant imbedding method”. In: *Applied Energy* 162 (Jan. 2016), pp. 163–171. ISSN: 03062619. DOI: 10.1016/j.apenergy.2015.10.092.

[47] Jonghoon Kim et al. “OCV Hysteresis Effect-based SOC Estimation in Extended Kalman Filter Algorithm for a LiFePO 4 /C Cell”. In: *2012 IEEE International Electric Vehicle Conference, IEVC 2012*. 2012, pp. 1–5. DOI: 10.1109/IEVC.2012.6183174.

[48] A. Johnson Antony and Kamakshy Selvajyothi. “A comparative performance analysis of electrical equivalent circuit models with the hysteresis effect of lithium iron phosphate batteries”. In: *International Journal of Green Energy* (2023). ISSN: 15435083. DOI: 10.1080/15435075.2023.2258216.

[49] Peng Yu et al. “Study of hysteresis voltage state dependence in lithium-ion battery and a novel asymmetric hysteresis modeling”. In: *Journal of Energy Storage* 51 (July 2022). ISSN: 2352152X. DOI: 10.1016/j.est.2022.104492.

[50] Wenyun Li et al. “Exploring the Hysteresis Effect in SOC Estimation of Li-ion Batteries”. In: *Journal of Physics: Conference Series*. Vol. 2456. 1. Institute of Physics, 2023. DOI: 10.1088/1742-6596/2456/1/012023.

[51] Iqbal H. Sarker. “Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions”. In: *SN Computer Science* 2.6 (Nov. 2021), pp. 1–20. ISSN: 26618907. DOI: 10.1007/S42979-021-00815-1/FIGURES/11.

[52] Christian Janiesch, Patrick Zschech, and Kai Heinrich. “Machine learning and deep learning”. In: *Electronic Markets* 31.3 (Sept. 2021), pp. 685–695. ISSN: 14228890. DOI: 10.1007/S12525-021-00475-2/TABLES/2.

[53] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: https://www.deeplearningbook.org/.

[54] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Elsevier, 2012. ISBN: 978-0-12-381479-1. DOI: 10.1016/C2009-0-61819-5.

[55] Iqbal H. Sarker. “Machine Learning: Algorithms, Real-World Applications and Research Directions”. In: *SN Computer Science* 2.3 (May 2021), pp. 1–21. ISSN: 26618907. DOI: 10.1007/S42979-021-00592-X/FIGURES/11.

[56] Zhi Hua Zhou. “A brief introduction to weakly supervised learning”. In: *National Science Review* 5.1 (Jan. 2018), pp. 44–53. ISSN: 2053714X. DOI: 10.1093/nsr/nwx106.

[57] IBM. *What Is a Machine Learning Pipeline? | IBM*. URL: https://www.ibm.com/topics/machine-learning-pipeline.

[58] *6.3. Preprocessing data — scikit-learn 1.4.1 documentation*. URL: https://scikit-learn.org/stable/modules/preprocessing.html.

[59] *Feature Extraction Explained - MATLAB & Simulink*. URL: https://de.mathworks.com/discovery/feature-extraction.html.

[60] Vladimir Berikov and Alexander Litvinenko. “Solving weakly supervised regression problem using low-rank manifold regularization”. In: *ArXiv* abs/2104.06548 (Apr. 2021). URL: http://arxiv.org/abs/2104.06548.

[61] Jeremy Watt, Reza Borhani, and Aggelos Katsaggelos. *Machine Learning Refined: Foundations, Algorithms, and Applications*. 2nd ed. Cambridge University Press, 2020. DOI: 10.1017/9781108690935.

[62] XGBoost. *Introduction to Boosted Trees*. URL: https://xgboost.readthedocs.io/en/stable/tutorials/model.html.

[63] Jerome H. Friedman. “Greedy Function Approximation: A Gradient Boosting Machine”. In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. ISSN: 00905364,21688966.

[64] John R. Quinlan. “Induction of Decision Trees”. In: *Machine Learning* 1 (1986), pp. 81–106. DOI: https://doi.org/10.1007/BF00116251.

[65] Daniel Svozil, Vladimir Kvasnieka, and Jie Pospichal. “Introduction to multi-layer feed-forward neural networks”. In: *Chemometrics and Intelligent Laboratory Systems* 39 (1997), pp. 43–62. DOI: 10.1016/S0169-7439(97)00061-0.

[66] Larry Medsker and Lakhmi C. Jain. *Recurrent Neural Network: Design and Applications*. 1st ed. CRC Press, Inc., 1999. ISBN: 0849371813. DOI: https://doi.org/10.1201/9781003040620.

[67] Paul J. Werbos. “Backpropagation Through Time: What It Does and How to Do It”. In: *Proceedings of the IEEE* 78.10 (1990). ISSN: 15582256. DOI: 10.1109/5.58337.

[68] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training Recurrent Neural Networks”. In: *ArXiv* (Nov. 2012). DOI: https://doi.org/10.48550/arXiv.1211.5063. URL: http://arxiv.org/abs/1211.5063.

[69] Aston Zhang et al. *Dive into Deep Learning*. Cambridge University Press, 2023. DOI: https://doi.org/10.48550/arXiv.2106.11342. URL: https://D2L.ai.

[70] PyTorch. *GRU*. URL: https://pytorch.org/docs/stable/generated/torch.nn.GRU.html.

[71] Kuo-Chin Chang, Tzung-Pei Hong, and Shian-Shyong Tseng. “Machine Learning by Imitating Human Learning”. In: *Minds and Machines* 6.2 (May 1996), pp. 203–228. ISSN: 1572-8641. DOI: 10.1007/BF00391286. URL: https://doi.org/10.1007/BF00391286.

[72] Eyke Hüllermeier and Willem Waegeman. “Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods”. In: *Machine Learning* 110.3 (Mar. 2021), pp. 457–506. ISSN: 15730565. DOI: 10.1007/s10994-021-05946-3.

[73] Xiaojin Goldberg. “Introduction to semi-supervised learning”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6 (June 2009), pp. 1–116. ISSN: 19394608. DOI: 10.2200/S00196ED1V01Y200906AIM006.

[74] Hwanjun Song et al. “Learning from Noisy Labels with Deep Neural Networks: A Survey”. In: *IEEE Transactions on Neural Networks and Learning Systems* 34 (2023), pp. 8135–8153. URL: https://github.com/songhwanjun/Awesome-Noisy-Labels..

[75] Pitoyo Hartono and Shuji Hashimoto. “Learning from imperfect data”. In: *Applied Soft Computing Journal* 7.1 (Jan. 2007), pp. 353–363. ISSN: 15684946. DOI: 10.1016/j.asoc.2005.07.005.

[76] Alan Joseph Bekker and Jacob Goldberger. “Training deep neural-networks based on unreliable labels”. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2016, pp. 2682–2686. DOI: 10.1109/ICASSP.2016.7472164.

[77] Hao Wang et al. *Learning with Noisy Labels for Sentence-level Sentiment Classification*. Tech. rep. 2019, pp. 6286–6292.

[78] Uri Alon et al. “Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays”. In: *Proceedings of the National Academy of Sciences of the United States of America* 96.12 (1999). ISSN: 00278424. DOI: 10.1073/pnas.96.12.6745.

[79] Hao Chen et al. “Imprecise Label Learning: A Unified Framework for Learning with Various Imprecise Label Configurations”. In: *ArXiv* (May 2023). DOI: https://doi.org/10.48550/arXiv.2305.12715Focustolearnmore. URL: http://arxiv.org/abs/2305.12715.

[80] Agnieszka Mikołajczyk and Michał Grochowski. “Data augmentation for improving deep learning in image classification problem”. In: *2018 International Interdisciplinary PhD Workshop, IIPhDW 2018*. Institute of Electrical and Electronics Engineers Inc., June 2018, pp. 117–122. ISBN: 9781538661437. DOI: 10.1109/IIPHDW.2018.8388338.

[81] Jacob Murel and Eda Kavlakoglu. *What is regularization?* 2023. URL: https://www.ibm.com/topics/regularization.

[82] scikit-learn. *RandomizedSearchCV*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html.

[83] Haleh Akrami et al. “Beta quantile regression for robust estimation of uncertainty in the presence of outliers”. In: *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2024, pp. 7480–7484. ISBN: 2309.07374v1. DOI: 10.1109/ICASSP48485.2024.10445867.

[84] Roger Koenker and Kevin F Hallock. *Quantile Regression*. Tech. rep. 4. 2001, pp. 143–156.

[85] ASAM. *ASAM MDF*. URL: https://www.asam.net/standards/detail/mdf/wiki/.

[86] *asammdf's documentation*. URL: https://asammdf.readthedocs.io/en/latest/intro.html#contributing-support.

[87] Apache Software Foundation. *Reading and Writing the Apache Parquet Format*. URL: https://arrow.apache.org/docs/python/parquet.html.

[88] scikit-learn. *MinMaxScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[89] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *ArXiv* abs/1502.03167 (Feb. 2015). DOI: https://doi.org/10.48550/arXiv.1502.03167.

[90] Vinod Sharma. “A Study on Data Scaling Methods for Machine Learning”. In: *International Journal for Global Academic & Scientific Research* 1.1 (Feb. 2022). DOI: 10.55938/ijgasr.v1i1.4.

[91] Maximilian Christ et al. *tsfresh*. URL: https://tsfresh.readthedocs.io/en/latest/index.html.

[92] scikit-learn. *QuantileRegressor*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.QuantileRegressor.html.

[93] XGBoost. *Quantile Regression*. URL: https://xgboost.readthedocs.io/en/latest/python/examples/quantile_regression.html.

[94] TensorFlow. *tfa.losses.pinball_loss*. URL: https://www.tensorflow.org/addons/api_docs/python/tfa/losses/pinball_loss.

[95] Diederik P Kingma and Jimmy Lei Ba. “Adam: A Method for Stochastic Optimization”. In: *ArXiv* (2014). DOI: https://doi.org/10.48550/arXiv.1412.6980.

[96] Zimeng Huang et al. “Measuring the Impact of Gradient Accumulation on Cloud-based Distributed Training”. In: *Proceedings - 23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2023*. Institute of Electrical and Electronics Engineers Inc., 2023, pp. 344–354. DOI: 10.1109/CCGrid57682.2023.00040.

[97] PyTorch. *Hyperparameter tuning with Ray Tune*. URL: https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html.

[98] Ray Tune. *Tune Trial Schedulers (tune.schedulers)*. URL: https://docs.ray.io/en/latest/tune/api/schedulers.html.

[99] XGBoost. *Introduction to Model IO*. URL: https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html.

[100] PyPI. *torchinfo*. URL: https://pypi.org/project/torchinfo/.

[101] Jakob Schmitt, Ivo Horstkötter, and Bernard Bäker. "A novel approach for modelling voltage hysteresis in lithium-ion batteries demonstrated for silicon graphite anodes: Comparative evaluation against established Preisach and Plett model". In: *Journal of Power Sources Advances* 26 (Apr. 2024). ISSN: 26662485. DOI: 10.1016/j.powera.2024.100139.

[102] scikit-learn. *SelectKBest*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html.

[103] scikit-learn. *f_regression*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html#sklearn.feature_selection.f_regression.

[104] scikit-learn. *PCA*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[105] Michael Greenacre et al. "Principal component analysis". In: *Nature Reviews Methods Primers* 2.1 (2022). ISSN: 26628449. DOI: 10.1038/s43586-022-00184-w.

[106] Gregory L. Plett. "Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs - Part 3. State and parameter estimation". In: *Journal of Power Sources* 134.2 (Aug. 2004), pp. 277–292. ISSN: 03787753. DOI: 10.1016/j.jpowsour.2004.02.033.

[107] Code of Federal Regulations. *§ 86.1815-27 Battery-related requirements for battery electric vehicles and plug-in hybrid electric vehicles*. 2024. URL: https://www.ecfr.gov/current/title-40/chapter-I/subchapter-C/part-86/subpart-S/section-86.1815-27.

[108] United Nations Economic Commission for Europe. *United Nations Global Technical Regulation No. 22*. 2023. URL: https://unece.org/sites/default/files/2023-01/ECE_TRANS_180a22e.pdf.

A

Details on Machine Learning Fundamentals and Methodology

A.1. XGBoost Details on Mathematical Formulation

The complexity function f_d contains the tree structure as well as the leave scores, which will be needed to determine the prediction $\hat{y}_i$ [62]:

$$f_d(x) = w_{q(x)}, w \in R^T, q : R^d \to 1, 2, ..., T, \tag{A.1}$$

where the vector w represents leaf scores, q represents a function mapping data points to the corresponding leaves, and T represents the total number of leaves [62].

The tree f_d is chosen as the one that minimises the objective function (based on Equation 3.5) at time step t using the defined outcome at the respective time step [62]:

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \omega(f_d) + c, c \in \mathbb{R}. \tag{A.2}$$

Minimising the objective function Equation A.2 for a given structure $q(x)$ and the complexity defined as

$$\omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{A.3}$$

results in the optimal value for the leave values w,

$$w_j^* = -\frac{G_j}{H_j + \lambda}, \tag{A.4}$$

and the corresponding value of the objective function,

$$obj^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T, \tag{A.5}$$

which assesses the performance of the mapping function $q(x)$.

The parameters G_j and H_j with I_j containing the indices of data points belonging to the j-th leaf [62] are defined as:

$$G_j = \sum_{i \in I_j} \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$
$$H_j = \sum_{i \in I_j} \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \tag{A.6}$$

A.2. Deep Learning Model Activation Functions

The principle of the activation function is depicted in Figure A.1. The three most common types of activation functions are:

- **Rectified linear units (ReLU) with activation function** $g(z) = max\{0, z\}$: Being similar to linear units, these functions are easy to optimise. However, their gradient-based learning is constrained to instances with zero activation. Nonetheless, the generalisations, such as absolute value rectification, leaky ReLU, or parametric ReLU, can yield gradients across the entire domain, ensuring learning capability in all scenarios [53].
- **Logistic sigmoid with activation function** $g(z) = \sigma(z)$: These functions were used mostly before the adoption of rectified linear units. They tend to saturate across a wide range of their domain and exhibit significant sensitivity when the input variable approaches zero. Generally, their use is discouraged because their widespread saturation makes gradient-based learning difficult [53].
- **Hyperbolic tangent with activation function** $g(z) = tanh(z)$: Like logistic sigmoid functions, these hyperbolic tangent functions were also mostly used before the adaption of rectified linear units. They are related to logistic sigmoid because of the relation $tanh(z) = 2\sigma(2z) - 1$, however, they typically perform better than the logistic sigmoid functions [53].

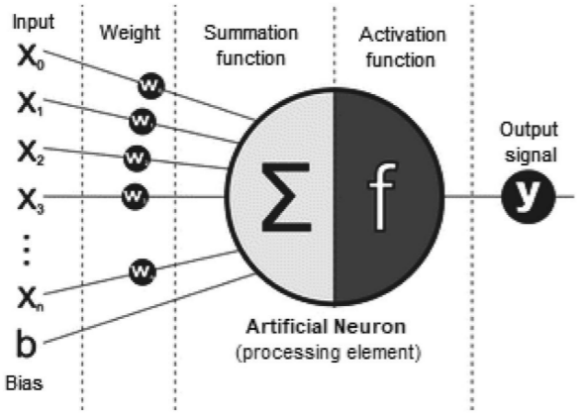
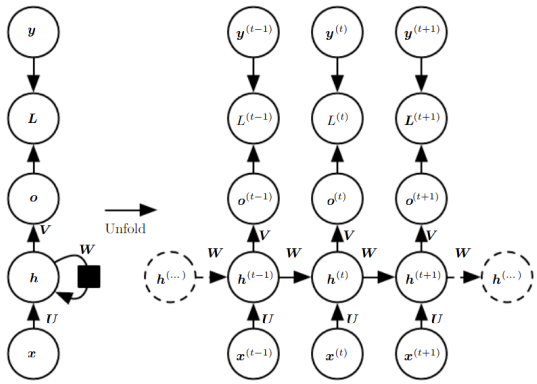


Figure A.1: Illustration of the activation of an artificial neuron [51].

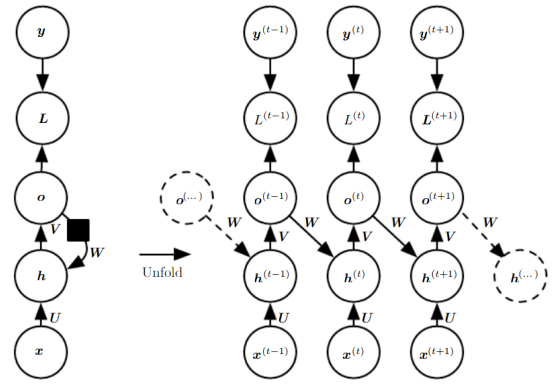
A.3. Architectures of RNNs

There are three different design patterns for the architecture of RNNs [53]:

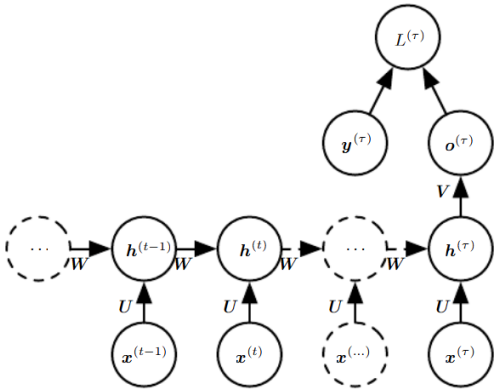
- **RNN that produces an output at each time step and has recurrent connections between hidden units Figure A.2a)**: Time sequence x is transformed to output o, which, in this case, is also a sequence. The training label y is then used to determine the loss L. The weight matrix U parameterises the connections between input and hidden layer and the connections between hidden layers are defined by weight matrix W. Lastly, the inter-layer connections from the hidden layer to the output are characterised by the weight matrix V.
- **RNN that produces an output at each time step and has recurrent connections between output at one time step to hidden units at next time step (Figure A.2b)**: In this configuration, the architecture focuses on transmitting only the output, o, to the subsequent layer. This design implies a limitation in the RNN, as o may not adequately represent all the state information required if the label or training target must encompass all the state's attributes. Despite this limitation, the benefit of this RNN structure is that it is a more straightforward training process due to the decoupling of temporal dependencies. Consequently, it allows for parallel training, meaning for each time step the gradient can be calculated separately.
- **RNN that produces an output for the entire sequence and has recurrent connections between hidden units (Figure A.2c)**: In this architecture, the model does not compute the output o and the corresponding loss L at each time step. Like in the first RNN design, the weight matrix W parameterises the connections between hidden layers, streamlining the process.



(a) Unfolded graph of an RNN that produces an output at each time step and has recurrent connections between hidden units.



(b) Unfolded graph of an RNN that produces an output at each time step and has recurrent connections between output at one time step to hidden units at the next time step.



(c) Unfolded graph of an RNN that produces an output for the entire sequence and has recurrent connections between hidden units.

Figure A.2: Various architectures of RNNs [53].

A.4. Features of the Models

This section discusses the features used in various models. Table A.1 outlines the features used for the time series approach. The variables t_{start} and t_{end} represent the initial and final time steps considered, marking the beginning and end of the active period. Each feature is a time series.

Signal	Feature
Current	Signal between t_{start} and t_{end}
	First derivative of signal between t_{start} and t_{end}
	Second derivative of signal between t_{start} and t_{end}
	Third derivative of signal between t_{start} and t_{end}
Integrated current	Signal between t_{start} and t_{end}
Cell voltage	Signal between t_{start} and t_{end}
	First derivative of signal between t_{start} and t_{end}
	Second derivative of signal between t_{start} and t_{end}
	Third derivative of signal between t_{start} and t_{end}
Cell temperature	Signal between t_{start} and t_{end}
	First derivative of signal between t_{start} and t_{end}
	Second derivative of signal between t_{start} and t_{end}
	Third derivative of signal between t_{start} and t_{end}
Hysteresis factor	Value at t_{start}

Table A.1: Features used for the three-dimensional input in the time series approach (Case Study 2).

Table A.2 lists the features utilised in the interval approach for predicting the change (delta) of the hysteresis factor. Each feature is a scalar value.

Signal	Feature
Current	Value at t
	Difference between value at $t-1$ and t
	Average between $t-1$ and t
Integrated current	Value at t
Cell voltage	Value at t
	Difference between value at $t-1$ and t
	Average between $t-1$ and t
Cell temperature	Value at t
	Difference between value at $t-1$ and t
	Average between $t-1$ and t

Table A.2: Features used for the two-dimensional input in the time interval approach when predicting the change in the hysteresis factor (Case Study 3).

Table A.3 lists the features employed in the interval approach for predicting the hysteresis factor. Each feature is represented as a scalar value. Notably, unlike Table A.2, the previous hysteresis factor is included as a feature in this approach.

Signal	Feature
Current	Value at t
	Difference between value at $t-1$ and t
	Average between $t-1$ and t
Integrated current	Value at t
Cell voltage	Value at t
	Difference between value at $t-1$ and t
	Average between $t-1$ and t
Cell temperature	Value at t
	Difference between value at $t-1$ and t
	Average between $t-1$ and t
Hysteresis factor	Value at $t-1$

Table A.3: Features used for the two-dimensional input in the time interval approach (Case Study 4).

Table A.4 illustrates the features used in the interval approach. Each feature is presented as a time series. The variables $t-1$ and t denote the start and end of each interval, respectively.

Signal	Feature
Current	Signal between $t-1$ and t
	First derivative of signal between $t-1$ and t
	Second derivative of signal between $t-1$ and t
	Third derivative of signal between $t-1$ and t
Integrated current	Signal between $t-1$ and t
Cell voltage	Signal between $t-1$ and t
	First derivative of signal between $t-1$ and t
	Second derivative of signal between $t-1$ and t
	Third derivative of signal between $t-1$ and t
Cell temperature	Signal between $t-1$ and t
	First derivative of signal between $t-1$ and t
	Second derivative of signal between $t-1$ and t
	Third derivative of signal between $t-1$ and t
Hysteresis factor	Value at $t-1$

Table A.4: Features used for the three-dimensional input in the time interval approach (Case Study 5 and 6).

A.5. Data Distribution of Vehicle Project B

This section discusses the data distribution of the second vehicle project, vehicle project B. Figure A.3 presents the distribution of the sequence lengths for the measurements associated with vehicle project B. It is observable that, in contrast to vehicle project A, the average sequence length is significantly extended, measuring 47.4 hours as opposed to 30.35 hours.

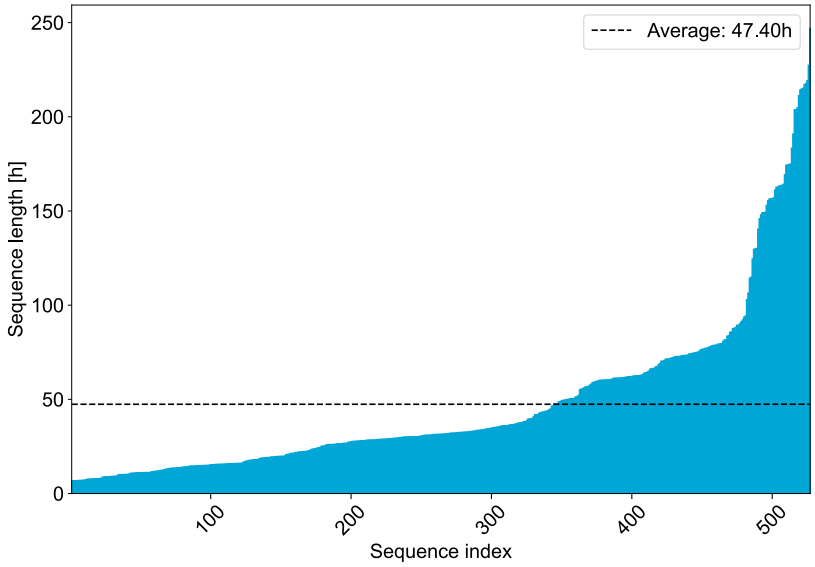


Figure A.3: Distribution of sequence lengths of measurements for vehicle project B.

Figure A.4 illustrates the average cell temperature plotted against the normalised open-circuit voltage (OCV). It is important to note that, due to the differing cell types, the minimum and maximum OCV values are different to those of vehicle project A. The figure indicates that, similar to vehicle project A, the majority of measurements fall within the range of 20 to 35°C

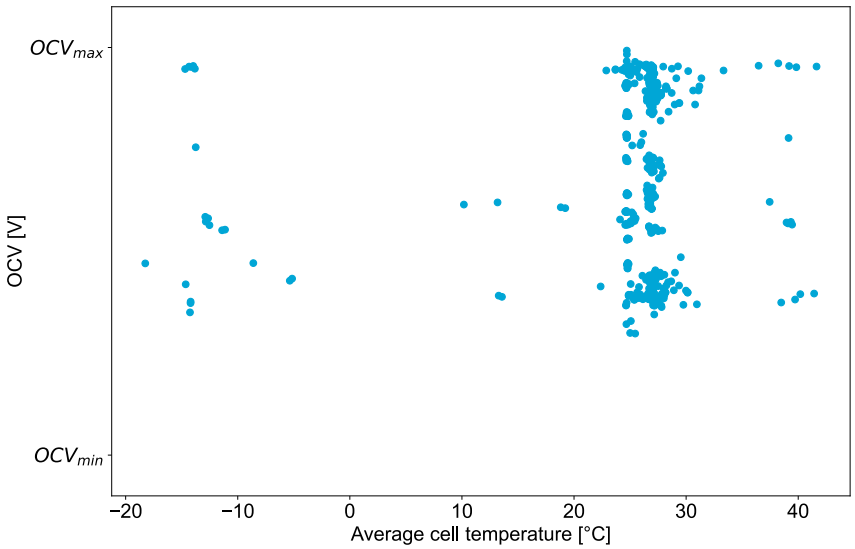


Figure A.4: Average cell temperature vs normed OCV for vehicle project B.

Figure A.5 depicts the minimum and maximum current for each measurement. The overall distribution closely resembles that of vehicle project A. Most measurements align along the vertical dotted line, corresponding to charging measurements, as the minimum current is 0, indicating no discharging occurs. There were 265 charging measurements and 127 discharging measurements, the latter aligning with the horizontal dotted line. Notably, the smallest minimum current in Project B is around 700A, which is greater than that of Project A, where the discharging current exceeds 800A. This discrepancy arises due to the differing cell types used in the vehicle projects, with Project A allowing for a higher discharging current.

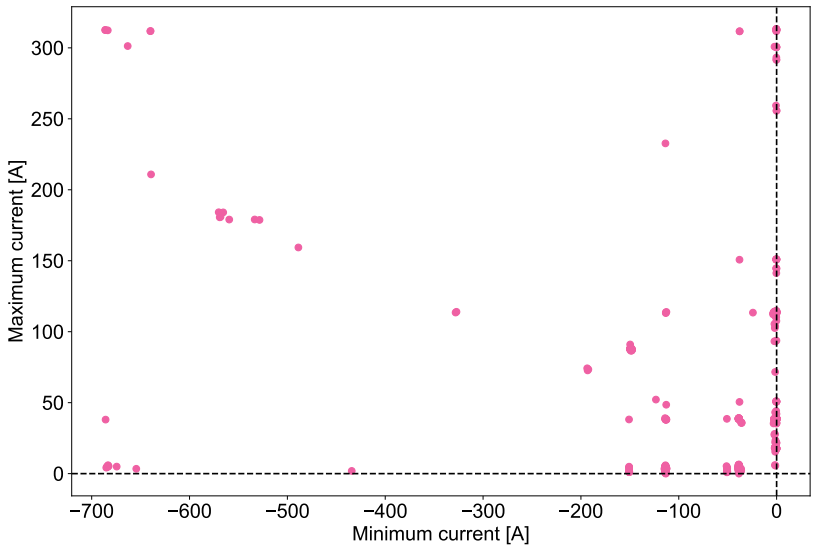


Figure A.5: Minimum and maximum current of measurements for vehicle project B.

The distribution of labels is shown in Figure A.6. This distribution is comparable to that of vehicle project A, where the majority of measurements correspond to labels -1 or 1. This trend can be traced back to the significant number of only-charging and only-discharging experiments.

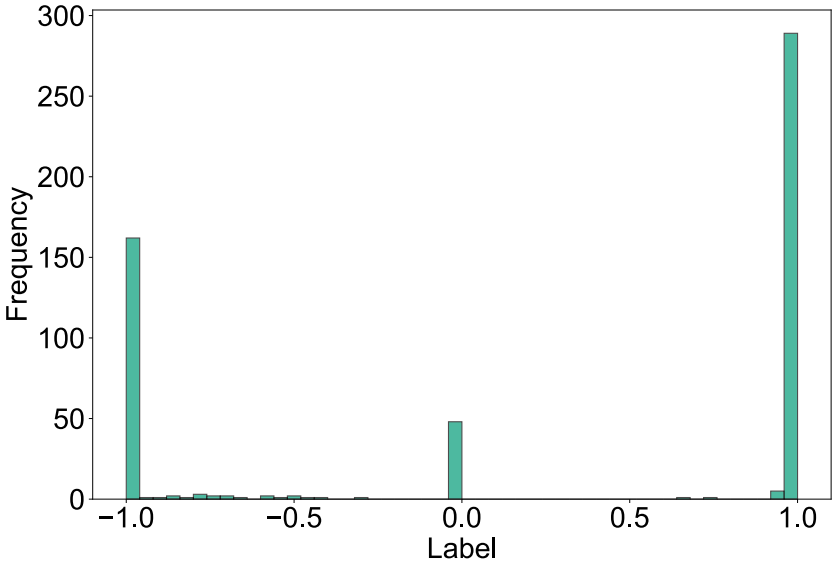


Figure A.6: Distribution of labels for different approaches for vehicle project B.

B

Details on Results

This chapter presents additional information about the results of the case studies described in chapter 5. The detailed breakdown includes the Python libraries used and details each case study. These details may include tuned hyperparameters, additional performance metrics and their description and evaluation. This additional information serves to improve the reproducibility and understanding of the presented results.

B.1. Used Python Libraries

This section lists the Python libraries and their respective versions used in the case studies.

Library	Version
numpy	1.26.4
pandas	2.2.2
tsfresh	0.20.2
beartype	0.18.5
matplotlib	3.8.4
pyarrow	16.0.0
scikit-learn	1.4.2
scipy	1.14.0
xgboost	2.0.3
torch	2.3.1+cu121
ray	2.34.0
torchsummary	1.5.1

Table B.1: Used Python libraries and their respective version.

B.2. Case Study 1

Figure B.1a and Figure B.1b present the tuned hyperparameters for various configurations of the linear regression and XGBoost models.

Hyperparameter	Learning rate		
Quantile	0.05	0.5	0.95
All (K features)	0.02	0.02	0.02
600 (K features)	0.02	0.02	0.02
300 (K features)	0.02	0.02	0.02
60 (K features)	0.02	0.02	0.02
All (PCA)	0.02	0.02	0.02
600 (PCA)	0.02	0.02	0.02
300 (PCA)	0.02	0.02	0.02
60 (PCA)	0.02	0.02	0.02

(a) Tuned parameters for the configurations for the linear regression model in Case Study 1.

Hyperparameter	Learning rate			Max depth			Subsample			Reg lambda			Reg alpha			Min child weight			Col sample by tree			Num boost rounds		
Quantile	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95
All (K features)	0.1	0.1	0.005	5	5	4	0.7	0.7	0.6	1	1	10	0.1	0.1	1	3	3	2	0.8	0.8	0.9	100	100	50
600 (K features)	0.1	0.05	0.005	5	5	4	0.7	0.7	0.6	1	0.1	10	0.1	0.1	1	3	3	2	0.8	0.8	0.9	50	200	50
300 (K features)	0.05	0.05	0.005	5	3	4	0.7	0.6	0.6	0.1	1	1	3	2	2	3	2	2	0.8	0.7	0.9	50	400	50
60 (K features)	0.05	0.01	0.005	4	4	4	0.7	0.6	0.6	10	1	10	0.1	0.1	1	1	4	2	0.8	0.7	0.9	100	1000	50
All (PCA)	0.1	0.05	0.005	5	5	4	0.7	0.7	0.6	1	0.1	10	0.1	0.11	1	3	3	2	0.8	0.8	0.9	50	250	50
600 (PCA)	0.08	0.05	0.005	4	4	4	1	0.7	0.6	0.1	0.1	10	1	1	1	4	4	2	0.8	0.8	0.9	150	150	50
300 (PCA)	0.08	0.1	0.005	4	4	4	1	0.6	0.6	0.1	10	10	1	0.1	1	4	4	2	0.8	0.8	0.9	200	150	50
60 (PCA)	0.08	0.05	0.005	4	5	4	1	0.7	0.6	0.1	0.1	10	1	0.1	1	4	3	2	0.8	0.8	0.9	50	150	50

(b) Tuned parameters for the configurations for the XGBoost model in Case Study 1.

Figure B.1: Tuned parameters for the configurations in Case Study 1 for both (a) linear regression and (b) XGBoost models.

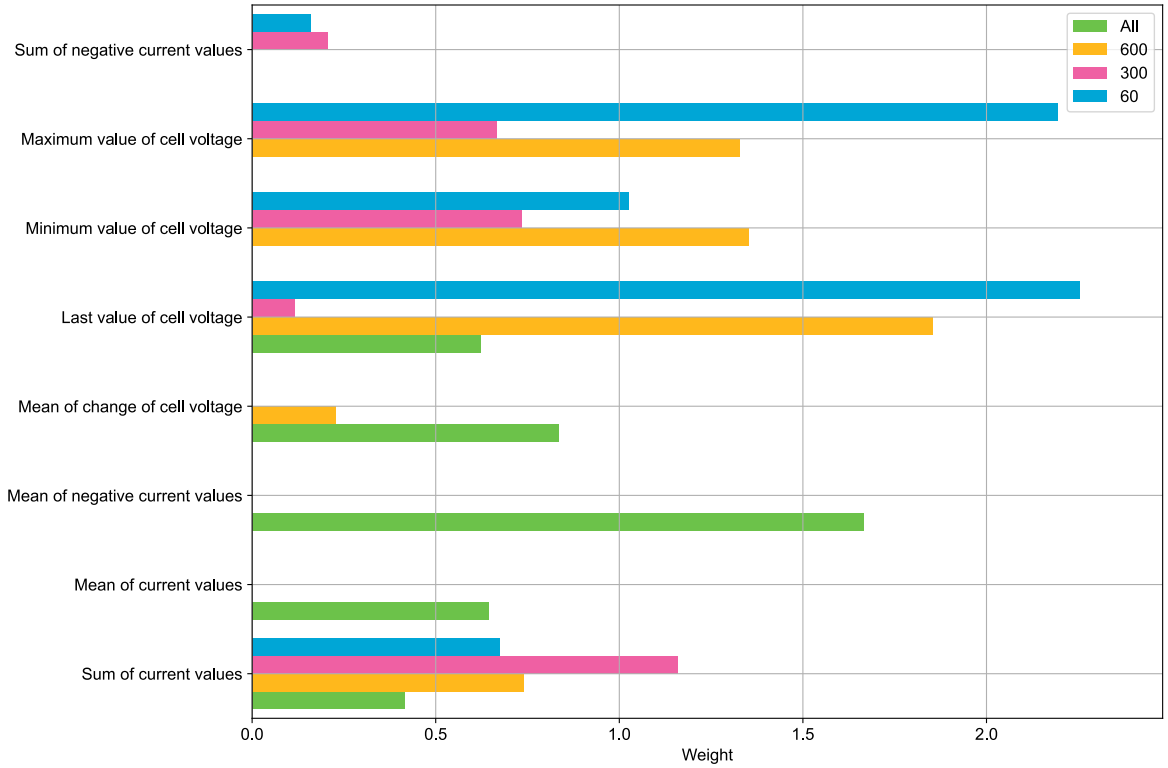
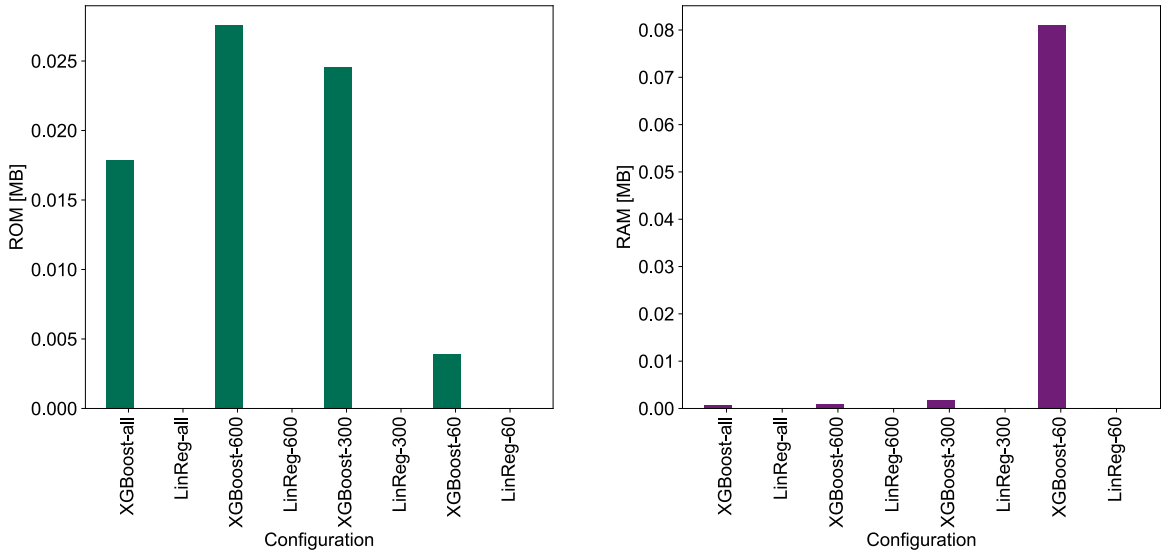


Figure B.2: The top five features and their weight for different configurations.

Figure B.2 displays the five most important features and their respective weights as determined by the trained

linear regression model. Note that in Figure B.2 different configurations are compared, each denoted by their time horizon. It is evident that across all four configurations, the sum of current values and the last value of the cell voltage consistently rank among the top five features. There may be significant variations in the weights of the features. For instance, in the "60" configuration (represented in blue), the differences in feature weights are more pronounced. Conversely, in the "all" configuration (depicted in green), the weights of the features are more closely aligned. This could either indicate that the configuration plays an important role when selecting the features of this case study or that the simple models are just too inconclusive and thus variations in the configurations should not be interpreted too much as the overall model loss is above acceptable levels regardless of the configuration.

For the K-best features selection approach, memory requirements are illustrated in Figure B.3.



(a) Required ROM.

(b) Required RAM.

Figure B.3: Memory usage for different configurations of the linear regression (LinReg) and XGBoost model in Case Study 1 (K-Features).

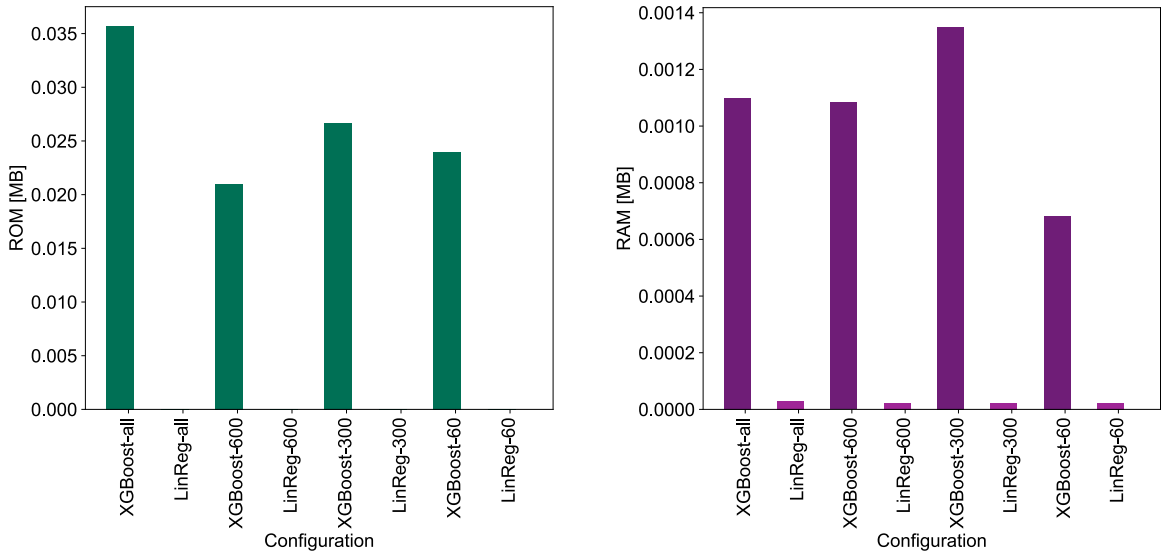
For the Principal Component Analysis (PCA) approach, memory requirements are depicted in Figure B.4.

B.3. Case Study 2

The tuned hyperparameters for Case Study 2 are presented in Table B.2.

Configuration	Learning rate	Hidden size	Num GRU layers	Batch size	Num Epochs
all-240	0.007127	32	1	8	425
all-120	0.007127	32	1	8	458
600-10	0.01088024	16	2	8	499
600-60	0.01088024	16	2	8	499
600-120	0.01088024	16	2	8	499
600-240	0.01088024	16	2	8	499
300-10	0.007127	32	1	8	250
300-60	0.007127	32	1	8	250
300-120	0.007127	32	1	8	250
300-240	0.007127	32	1	8	250
60-10	0.00874022	16	1	8	306
60-60	0.00874022	16	1	8	306
60-120	0.00874022	16	1	8	306
60-240	0.00874022	16	1	8	306

Table B.2: Tuned parameters for the configurations in Case Study 2.



(a) Required ROM.

(b) Required RAM.

Figure B.4: Memory usage for different configurations of the linear regression (LinReg) and XGBoost model in Case Study 1 (PCA).

B.4. Case Study 3

Table B.3 presents the tuned hyperparameters for the XGBoost model in Case Study 3.

Hyperparameter	Learning rate			Max depth			Subsample			Reg lambda			Reg alpha			Min child weight			Col sample by tree			Num boost rounds		
Quantile	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95
Delta Prediction	0.1	0.05	0.1	4	4	5	0.6	0.8	0.7	1	0.1	1	0.1	1	0.1	3	3	3	0.8	0.9	0.8	2000	2000	2000

Table B.3: Used XGBoost parameters for Case Study 3.

The comparison between the predicted medians and the actual labels is illustrated in Figure B.5. The significant pinball loss of 0.236421737 is evident as numerous samples substantially deviate from the ideal line (represented by the black dotted line). Additionally, many predictions exceed the theoretical hysteresis bounds of -1 and 1, with prediction values ranging from -5 to 4.5.

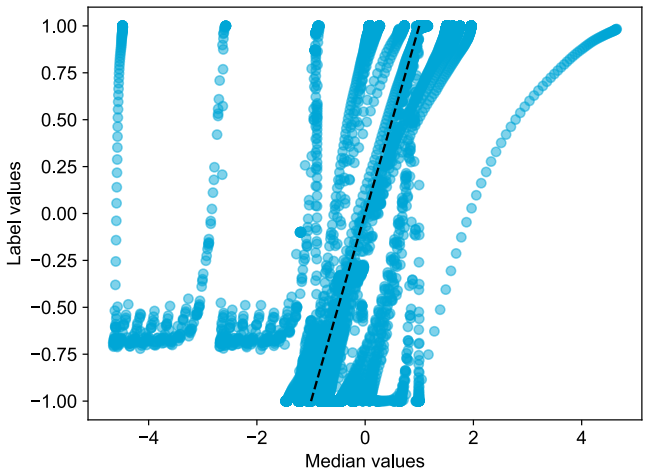


Figure B.5: Predicted median vs label for Case Study 3.

B.5. Case Study 4

The tuned hyperparameters for the linear regression and the XGBoost model of Case Study 4 are presented in Figure B.6a and Figure B.6b, respectively.

Hyperparameter	Learning rate		
Quantile	0.05	0.5	0.95
600-none	0.02	0.02	0.02
600-1	0.02	0.02	0.02
600-2	0.02	0.02	0.02
300-none	0.02	0.02	0.02
300-1	0.02	0.02	0.02
300-2	0.02	0.02	0.02
60-none	0.02	0.02	0.02
60-1	0.02	0.02	0.02
60-2	0.02	0.02	0.02

(a) Tuned parameters for the configurations for the linear regression model in Case Study 4.

Hyperparameter	Learning rate			Max depth			Subsample			Reg lambda			Reg alpha			Min child weight			Col sample by tree			Num boost rounds		
Quantile	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95	0.05	0.5	0.95
600-none	0.08	0.05	0.08	5	5	4	1	0.7	1	0.1	0.1	0.1	1	0.1	1	4	5	4	0.8	0.8	0.8	2000	2000	2000
600-1	0.1	0.1	0.1	5	5	4	0.7	0.7	0.6	1	1	10	0.1	0.1	0.1	3	3	4	0.8	0.8	0.8	2000	2000	300
600-2	0.08	0.05	0.08	5	5	4	0.9	0.7	1	1	0.1	0.1	10	0.1	1	1	3	4	0.7	0.8	0.8	2000	2000	2000
300-none	0.1	0.1	0.1	5	5	4	0.7	0.7	0.6	1	1	10	0.1	0.1	0.1	3	3	4	0.8	0.8	0.8	2000	2000	2000
300-1	0.1	0.05	0.1	5	5	5	0.7	0.7	0.7	1	0.1	1	0.1	0.1	0.1	3	3	3	0.8	0.8	0.8	2000	2000	2000
300-2	0.1	0.05	0.1	5	5	5	0.7	0.7	0.7	1	0.1	1	0.1	0.1	0.1	3	3	3	0.8	0.8	0.8	2000	2000	2000
60-none	0.1	0.05	0.1	5	4	5	0.7	0.8	0.7	1	10	1	0.1	10	0.1	3	1	3	0.8	0.9	0.8	2000	2000	2000
60-1	0.1	0.01	0.1	4	4	5	0.6	1	0.7	10	0.1	1	0.1	1	0.1	4	3	3	0.8	0.9	0.8	2000	2000	2000
60-2	0.1	0.05	0.1	4	4	5	0.6	0.8	0.7	1	0.1	1	0.1	1	0.1	3	3	3	0.8	0.9	0.8	2000	2000	2000
10-none	0.1	0.05	0.1	5	5	5	0.7	0.7	0.6	1	0.1	1	0.1	0.1	0.1	3	3	4	0.8	0.8	0.8	2000	2000	2000
10-1	0.1	0.05	0.1	5	4	4	0.7	0.8	0.7	1	1	1	0.1	0.1	0.1	4	3	3	0.8	0.8	0.9	2000	2000	2000
10-2	0.1	0.05	0.1	5	5	4	0.7	0.8	0.7	1	0.1	0.1	0.1	0.1	0.1	3	3	3	0.8	0.9	0.8	2000	2000	2000

(b) Tuned parameters for the configurations for the XGBoost model in Case Study 4.

Figure B.6: Tuned parameters for the configurations in Case Study 4 for both (a) linear regression and (b) XGBoost models.

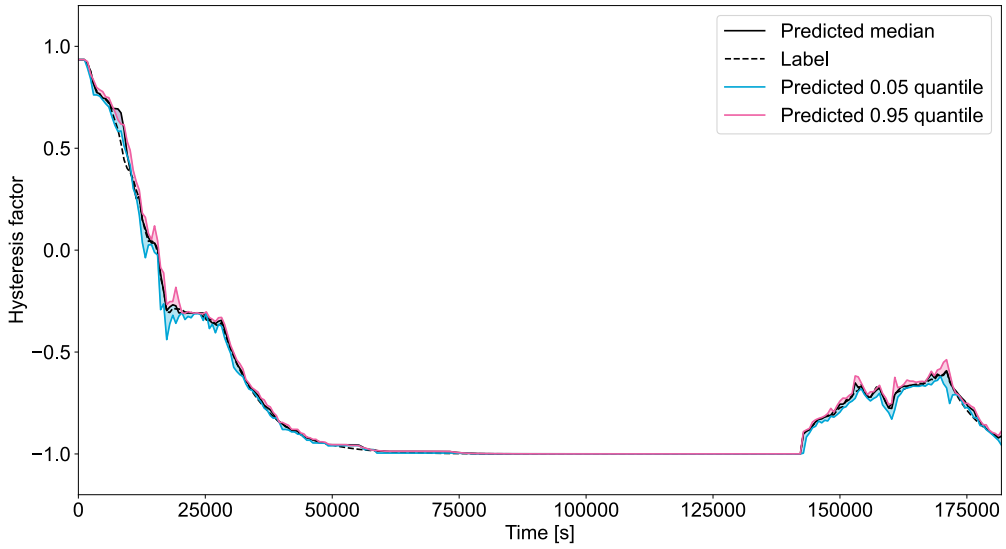


Figure B.7: Example reconstructed measurement for the "60-2" configuration of the XGBoost model in Case Study 4.

Figure B.7 represents a reconstructed measurement in Case Study 4 using the 60-2 configuration with the

XGBoost model. The quantiles closely follow the actual labels. However, minor deviations can be observed in dynamic regions, particularly around 1800 s and between 140000 s and 170000 s.

B.6. Case Study 5

Table B.4 depicts the tuned parameters for the configurations in Case Study 5. Note that these parameters are found after testing 25 (random, but with the random seed set) hyperparameter combinations and that the validation loss was only evaluated for 10 epochs as tuning with the larger data sets such as the interval size 10 configurations would otherwise be infeasible.

Configuration	Learning rate	Hidden size	Num GRU layers	Batch size
10-10	0.00019854	4	2	16
10-60	0.00019854	4	2	16
60-10	0.00094437	64	2	32
60-60	0.00094437	64	2	32
60-120	0.00094437	64	2	32
60-240	0.00094437	64	2	32
300-10	0.00127695	16	2	8
300-60	0.00127695	16	2	8
300-120	0.00127695	16	2	8
300-240	0.00127695	16	2	8
600-10	0.00127695	16	2	8
600-60	0.00127695	16	2	8
600-120	0.00127695	16	2	8
600-240	0.00127695	16	2	8

Table B.4: Tuned parameters for the configurations in Case Study 5.

Apart from the tuned parameters the number of epochs also influences the performance of the models. Therefore it was checked if the models should be stopped early if the validation loss would not improve any further. This avoids overfitting the training data set. Figure B.8 shows the loss when training for 100 epochs, without early stopping. The figure suggests that the "60-240", "10-10" and "60-60" configuration models are likely to overfit.

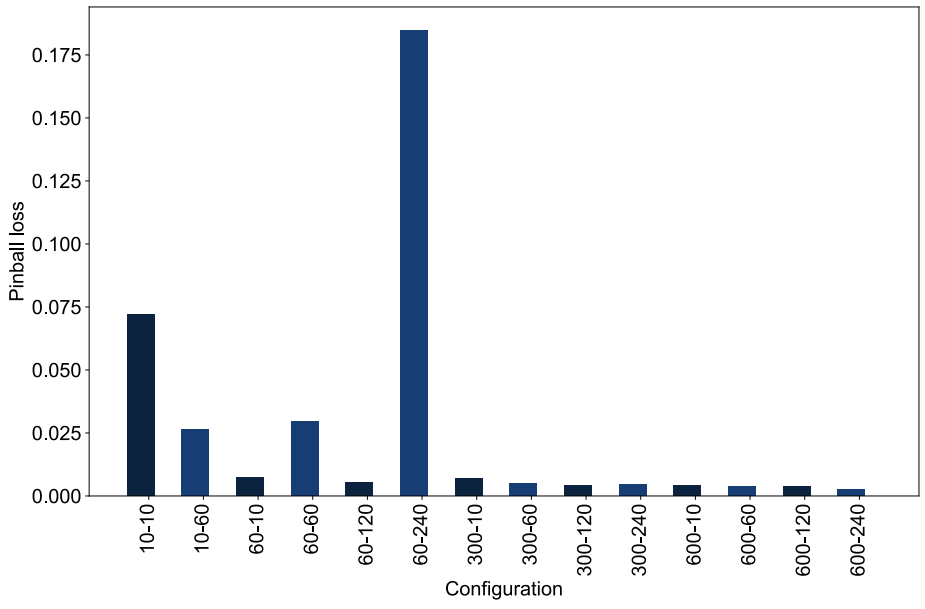


Figure B.8: Pinball losses for different configurations in Case Study 5 when training with 100 epochs.

After analysing the validation loss during training, the optimal number of epochs can be found in Table B.5. Note that the maximum number of epochs was set to 100 due to time constraints when training the models.

Configuration	Lowest validation loss	Test loss	Number of epochs
10-10	0.000201	0.059233	96
10-60	0.000219	0.024828	93
60-10	0.000542	0.004545	84
60-60	0.000542	0.004426	85
60-120	0.000489	0.003567	79
60-240	0.000530	0.004089	83
300-10	0.002018	0.005922	99
300-60	0.001589	0.004224	97
300-120	0.001485	0.003548	97
300-240	0.001581	0.004709	74
600-10	0.002817	0.004908	95
600-60	0.002464	0.003338	88
600-120	0.002282	0.004014	98
600-240	0.002278	0.003176	98

Table B.5: Optimal number of epochs for Case Study 5.

B.7. Case Study 6

The parameters for Case Study 6 are listed in Table B.6. Note that these parameters have not been tuned due to the long training time required for this model. Consequently, the optimal parameters from Case Study 5 were used, except for a different learning rate.

Configuration	Learning rate	Hidden size	Num GRU layers	Batch size	Dropout
600-240	0.01	16	2	8	0

Table B.6: Parameters for the configurations in Case Study 6.

Figure B.9 illustrates the training and validation loss over 1000 epochs. It is evident that the initial training phase with 100 epochs, which corresponds to 500 seen batches during training, was insufficient, as a significant reduction in validation loss is observed after 100 epochs. The noticeable fluctuations, particularly around 1700 seen batches, can be attributed to the random variations inherent to stochastic optimisation methods such as the Adam optimiser and the variability in the quality of batches, as some batches may be "worse" than others.

Figure B.9: Training and validation loss over number of seen batches for 1000 epochs in Case Study 6.

Figure B.10 depicts the predicted median versus the label. The majority of samples are very close to the ideal diagonal line. The most significant outliers, with the greatest distance from the ideal line, occur when the label is between -0.75 and 0.75. This could be due to the distribution of labels, specifically that there are fewer training samples in this range. It is also noteworthy that underestimations are farther from the ideal line compared to overestimations.

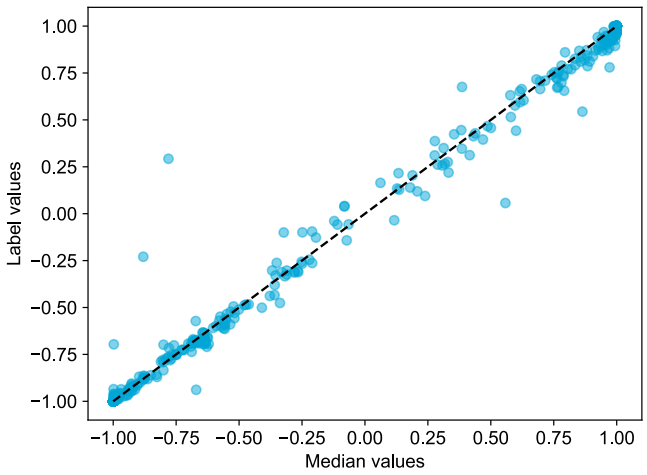


Figure B.10: Predicted median vs label for the "600-240" configuration in Case Study 6.

B.8. Evaluation

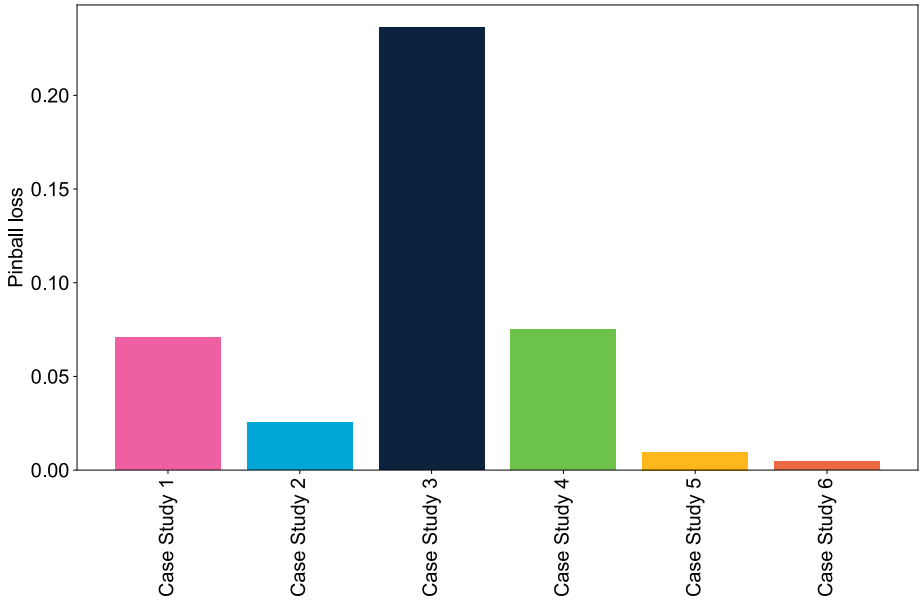
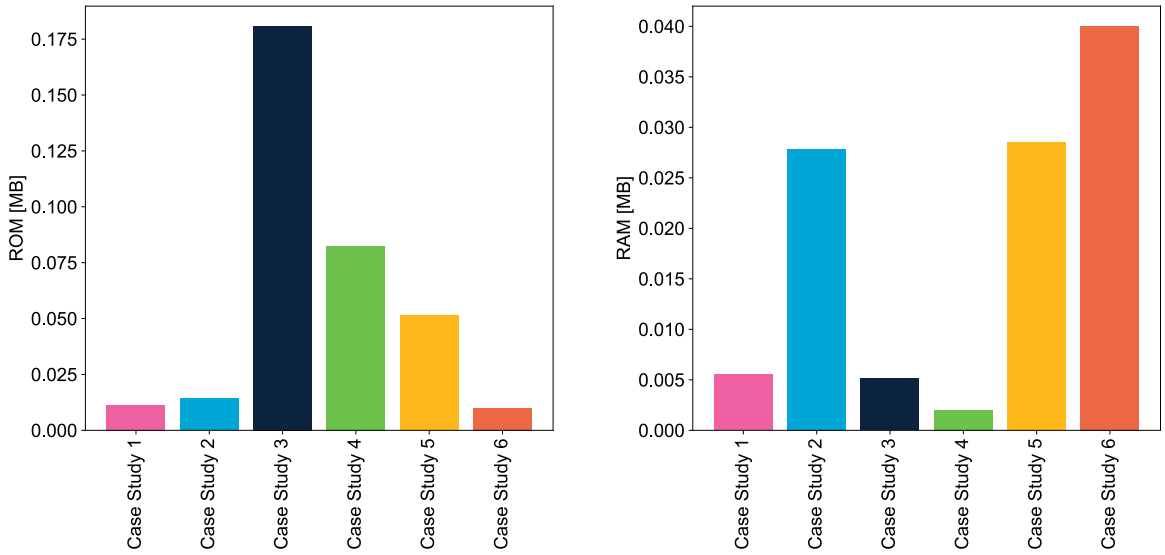


Figure B.11: Comparison of the achieved pinball loss for the average of configurations of each case study.

Figure B.11 illustrates the average pinball loss across different case studies. In comparison with Figure 5.18, which depicts the pinball loss for the best configuration, it is evident that GRU models (Case Studies 2, 5, and 6) generally outperform others. Notably, Case Study 3 exhibits the worst performance, with a pinball loss at least double that of the second-worst case study. The other simpler model case studies (Case Studies 1 and 4) show comparable loss values.

Case Study 4 demonstrates a significant sensitivity to its configurations and hyperparameters, as shown by its superior performance in the best configuration compared to Case Study 1 but the comparable performance to Case Study 1 in the average pinball loss. Conversely, Case Study 5, which also employs the interval approach but with a GRU model, appears less sensitive to configuration variations, maintaining a low average pinball loss of 0.0096, with the best configuration achieving a loss of 0.0032. On average, Case Study 6 performs the best; however, this is because it was tested with only one configuration, meaning that the average loss is identical to the best configuration.



(a) Required ROM.

(b) Required RAM.

Figure B.12: Comparison of memory usage for the average of configurations of each case study.

The memory requirements are depicted in Figure B.12. Notably, Case Study 3 exhibits the highest ROM requirements, attributed to the XGBoost model's extensive number of nodes, leaves, and trees. Specifically, the median model alone comprises 931 trees, 14876 leaves, and 13945 nodes.

Furthermore, GRU models generally demand higher RAM on average, due to the memory needed for the forward pass through the neural network. Despite these variations, all configurations remain below the set thresholds for ROM and RAM, which are 0.3MB and 0.1MB, respectively.

The achieved score, reflecting the average of configurations for each case study, is presented in Figure B.13. Similar to Figure 5.20 which presented this for the best configurations, the first three case studies yield a negative score due to their high pinball loss. As illustrated in Figure B.11, the pinball loss of Case Study 4 is highly dependent on the configuration. This dependency results in the best configuration achieving a positive score (ranking among the top three case studies), while the average of configurations yields a negative score. Consistent with previous observations on pinball loss, the GRU models generally exhibit better performance, with Case Study 6 performing the best on average.

Scenario	Weight loss	Weight memory	% CS1	% CS2	% CS3	% CS4	% CS5	% CS6
Base scenario	0.7	0.3	0	0	0	0	0	0
High scenario	0.95	0.05	74.9128459	75.3350347	37.250754	-8.41635428	-3.54062599	-10.1999293
Low scenario	0.5	0.5	-59.9302767	-60.2680278	-29.8006032	6.73308343	2.8325008	8.15994347

Table B.7: Sensitivity analysis of different case studies for three different scenarios.

Table B.7 provides detailed information on the sensitivity analysis, including the exact weight for each scenario and the percentage change compared to the base scenario for each case study. The scenarios significantly impact the first three case studies due to their high pinball loss. In contrast, Case Studies 4 to 6 exhibit only gradual changes.

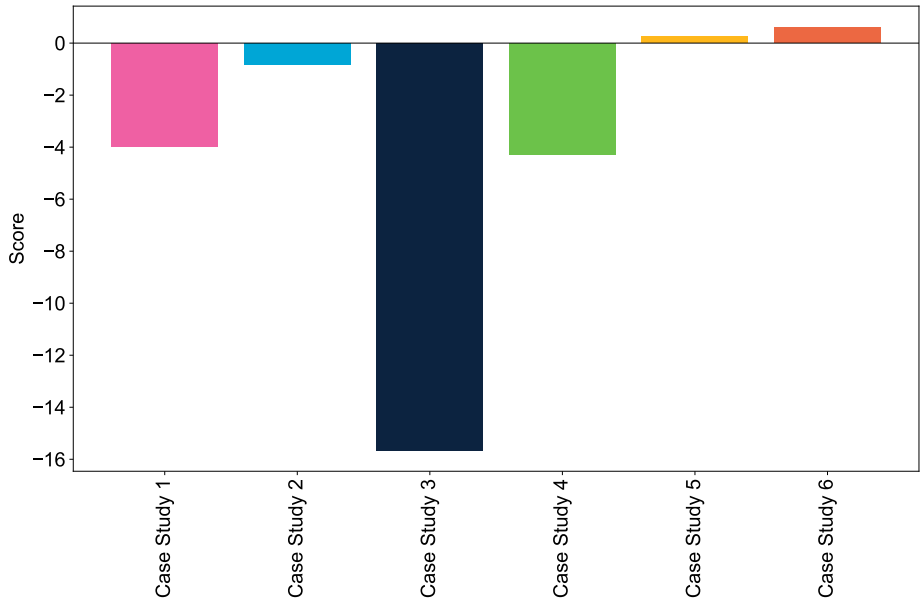


Figure B.13: Comparison of the achieved score for the average of configurations of each case study.

B.9. Case Study 7

B.9.1. Fine-Tuning

In the first step, it was checked whether increasing the complexity of the model would significantly impact its performance. For that reason and to find more optimal parameters for the model, the model of Case Study 5 was re-tuned. As it was previously seen, the minimum number of layers was set to 2. After testing the impact of increasing the number of layers, it was found that increasing the layers beyond 3 would yield only a very small decrease in loss while increasing the required memory. Furthermore, the dropout parameter was tuned (with values between 0 and 0.5). The tuning was done for 30 epochs and 25 different hyperparameter configurations were checked.

After re-tuning and checking the optimal number of epochs, the optimal parameters were found as described in Table B.8. This corresponds to the hyperparameters that were previously found in Case Study 5, thus no improvement by increasing the number of layers or varying the dropout parameter could be made.

Configuration	Learning rate	Hidden size	Num GRU layers	Batch size	Dropout
600-240	0.00127695	16	2	8	0

Table B.8: Fine-tuned parameters for the configurations in Case Study 5.

Note that these fine-tuning settings were used for all tuning done in Case Study 7.

B.9.2. Normalization According to Cell Datasheet

To prepare the data so that the model that was trained with vehicle project A could also be tested with vehicle project B, the input data was normalized in a different way to account for the differences in cells according to the cell datasheet. Table B.9 shows how each feature was normed in both datasets. Note that the current was priorly normed into a c-rate,

$$\text{c-rate} = \frac{I}{Q}, \tag{B.1}$$

using the respective nominal capacity of the battery. The integrated current was also put relative to the battery's capacity. Furthermore, the voltage was normed according to the cut-off voltages in the datasheet. As current and integrated current, once normed to the stated minimum and maximum values, reached values very close to -1 and 1 (thus the whole span of possible (integrated) current), the voltage did not reach the

Signal	Feature	Normalization	Min / Max
Current	Signal values	Cell capacity (C-rate)	According to peak (dis)charge current limits (as C-rate)
	First derivative	Cell capacity (C-rate)	According to project A
	Second derivative	Cell capacity (C-rate)	According to project A
	Third derivative	Cell capacity (C-rate)	According to project A
Integrated current	Signal values	/	According to cell data sheet (+/- capacity of the cell)
Cell voltage	Signal values	Cut-off voltages	According to project A
	First derivative	Cut-off voltages	According to project A
	Second derivative	Cut-off voltages	According to project A
	Third derivative	Cut-off voltages	According to project A
Cell temperature	Signal values	/	According to project A
	First derivative	/	According to project A
	Second derivative	/	According to project A
	Third derivative	/	According to project A
Hysteresis factor	Signal value at beginning of interval	/	-1 / 1

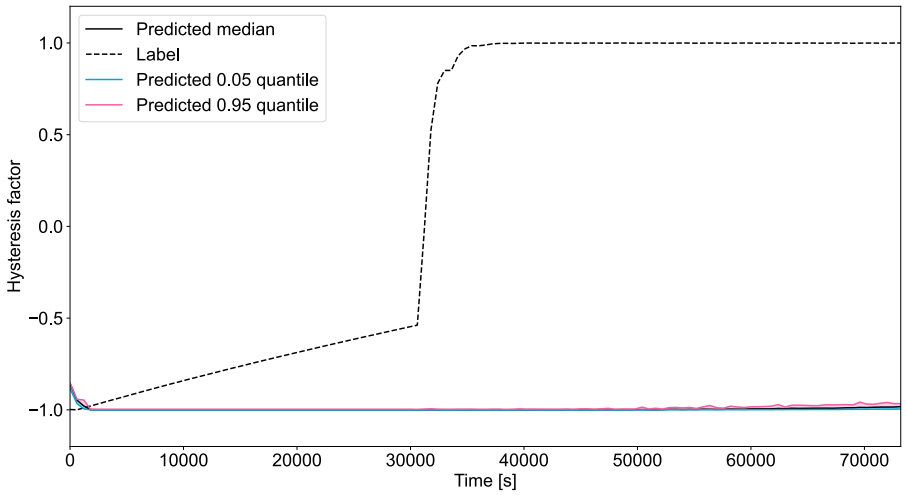
Table B.9: Normalization of the input data according to the cell datasheet.

whole span, namely, the measurements did not come very close to the lower cut-off voltage. This is due to the temperature range set. To reach the lower cut-off voltage the internal resistance of the cell has to be very high, this is achieved at low temperatures (which are excluded in this work).

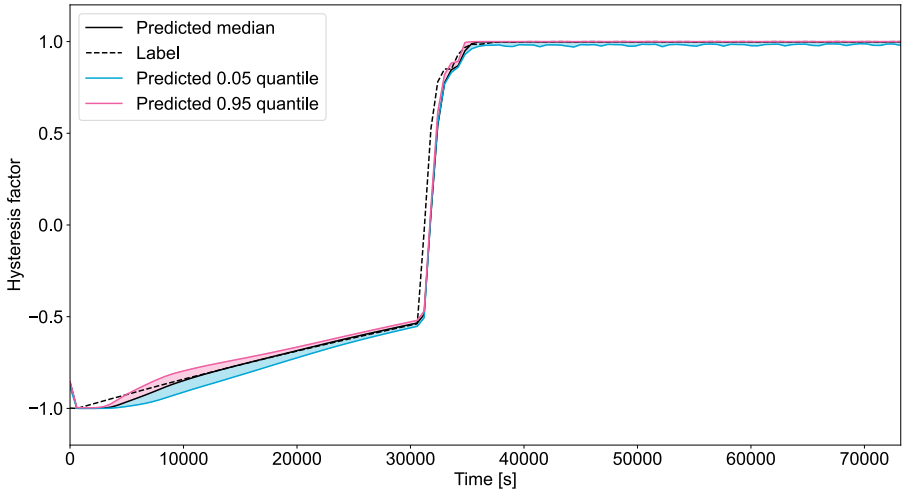
B.9.3. Vehicle Model Generalisation

Testing B With a Model Trained Solely on A

Figure B.14 shows the resulting reconstructed measurement when using the model, which was trained with vehicle project A, to test vehicle project B. The upper figure depicts autoregressive testing (using the previous prediction as input for the next) and the lower non-autoregressive testing (every interval is treated independently, thus every interval starts with the correct hysteresis factor given by the label of the previous). It can be seen that the initial wrong prediction leads to a great error. The model does not seem to be able to recover from this error. This figure also shows how dependent the model is on its previous prediction, as all other features are the same between the two training approaches.



(a) Autoregressive testing.



(b) Non-autoregressive testing.

Figure B.14: Reconstructed measurement for vehicle project B for autoregressive and non-autoregressive testing.

Training the Model With Both Vehicle Projects

The tuned hyperparameters of the two models can be found in Table B.10.

Model	Num epochs	Learning rate	Hidden size	Num GRU layers	Batch size	Dropout
New model – no DS normalisation	97	0.00127695	16	2	8	0
New model – DS normalisation	47	0.00142280	64	3	8	0

Table B.10: Tuned parameters for the new models for both projects in Case Study 7.

First, it was tested to retrain the model, not considering the different cell datasheets of the vehicle project. Thus, the data of the two projects was merged and then normed by the respective minimum and maximum of each feature. Note that in this case, the different current/voltage limits as well as the capacities were not considered.

The model was then also re-tuned with the new set of input data. This resulted in the same tuning parameters as before (see Table B.4). The resulting test loss is 0.008, worse than the one achieved for only training and testing with vehicle project A, but still below the threshold set at 0.01. The predicted median versus label plot can be seen in Figure B.15. There are more outliers, especially if the label of the hysteresis factor is in the middle of its range.

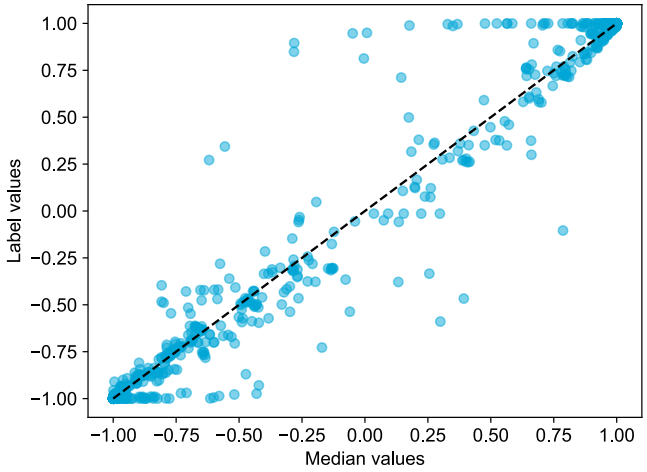


Figure B.15: Predicted median vs label for Case Study 7 after retraining the model with both data sets without considering the cell datasheet.

In Figure B.16 it can be seen that the predicted median generally follows the labels, however with deviations. Especially the slope does not seem to be estimated correctly. This could also be seen in the previous experiment where the vehicle B data was applied to the already trained model. The upper and lower quantiles are also wider.

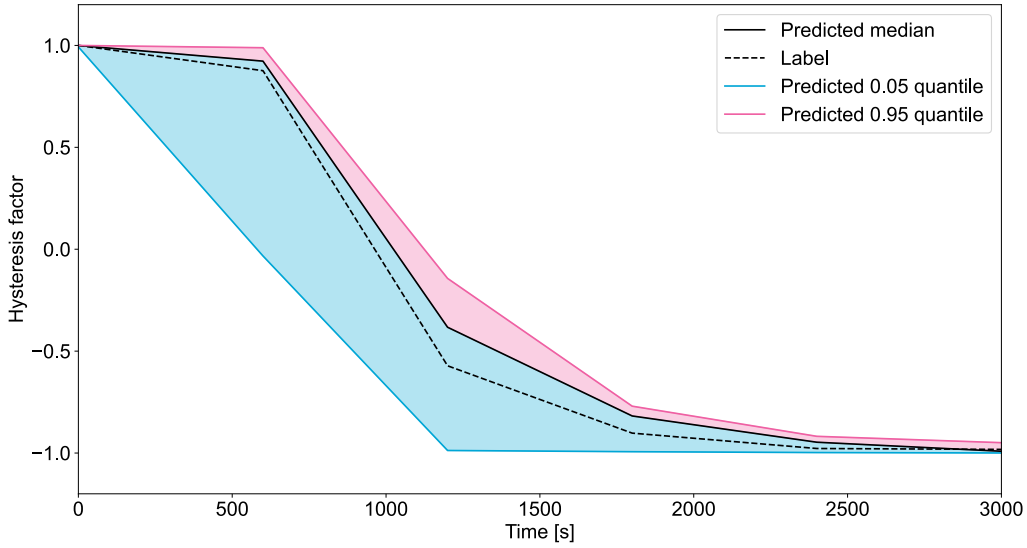


Figure B.16: Example reconstructed measurement when retraining the model with project A and B without considering the cell datasheet for Case Study 7.

Training each Vehicle Project With Its Own Model
An overview of the chosen four models and their tuned parameters can be found in Table B.11.

Model	Num epochs	Learning rate	Hidden size	Num GRU layers	Batch size	Dropout
Model A – further trained (no DS normalisation)	100	0.0001	16	2	8	0
Model A – further trained (DS normalisation)	80	0.001	16	2	8	0
Model B – no DS normalisation	93	0.00127695	16	2	8	0.2
Model B – DS normalisation	99	0.00127695	16	2	8	0

Table B.11: Tuned parameters for the models for data set B in Case Study 7.

The detailed results of the tuning process are shown in Table B.12. Note that only a limited amount of possible configurations are tested. However, the tested configurations show that the models, where every layer was retrained generally performed better than the ones where the layers of model A were frozen and an additional GRU and linear layer was added to the model and then trained on data set B. Also, it can be seen that if an extra layer is added and trained, the models without cell datasheet normalisation perform better than the ones with. Overall, however, their performance is rather poor.

B.9.4. Driving Cycle Generalisation

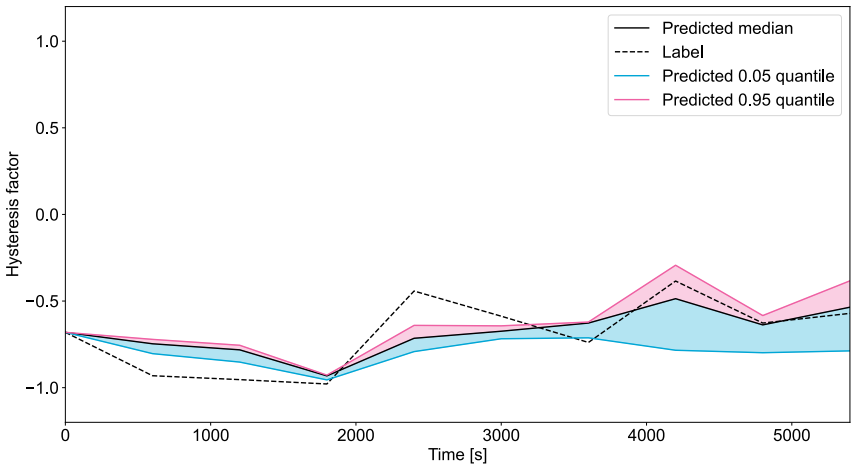
In Figure B.17 the reconstructed measurement (the same as in Figure 5.26) for three different model configurations are displayed. The configurations differ in interval size. Hence, the 60-120 configuration captures more details than the 600-240 configuration. Additionally, note that due to the absence of relaxation times at the end of the measurements, the data is "cut off" at the last complete interval. Normally, if relaxation times were included, as done in the test bench measurements, the start of the relaxation time would be used to complete the last interval. Consequently, the lengths of the measurements may vary slightly. While the overall trend of the measurements is somewhat maintained, there are significant deviations. Specifically, it seems like the predictions struggle to capture the hysteresis factor due to the steep gradient of the label. For instance, the peaks of the hysteresis factor, such as the one around 2300 s, are often not fully reached.

B.9.5. Temperature Range Generalisation

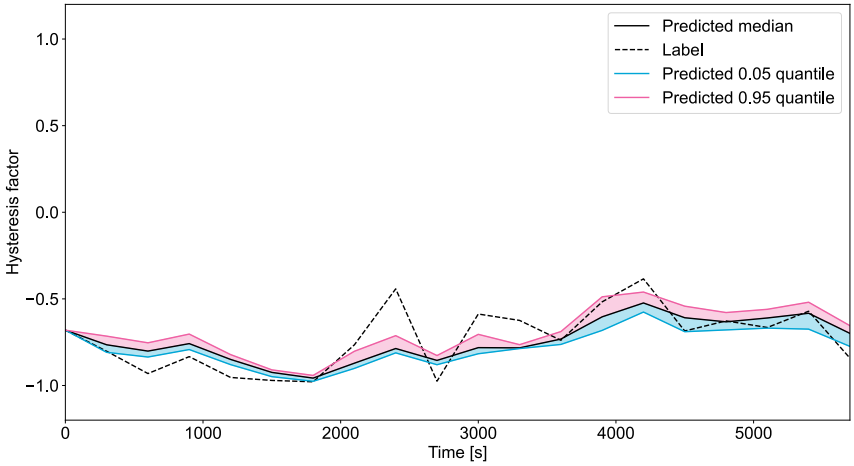
Table B.13 provides an overview of the tuned parameters used to test the temperature range generalisation of the model. This corresponds to the initial parameters of the model.

Norming	Configuration	Learning rate	Batch	Hidden size (extra layer)	Loss (100 Epochs)	Early stopping epoch	Early stopping loss
With DS	Extra GRU Layer	0.00001	8	8	0.02740925	100	0.02740925
With DS	Extra GRU Layer	0.0001	8	8	0.02649208	90	0.02694436
With DS	Extra GRU Layer	0.001	8	8	0.02186277	90	0.02266452
With DS	Extra GRU Layer	0.00127695	8	8	0.01891474	69	0.02453421
With DS	Extra GRU Layer	0.00001	8	16	0.02653545	100	0.02653545
With DS	Extra GRU Layer	0.0001	8	16	0.0244596	97	0.02459739
With DS	Extra GRU Layer	0.001	8	16	0.0218099	76	0.01975581
With DS	Extra GRU Layer	0.00127695	8	16	0.01891474	69	0.01980984
With DS	Extra GRU Layer	0.00001	8	64	0.0261107	94	0.0265608
With DS	Extra GRU Layer	0.0001	8	64	0.02374464	97	0.0237891
With DS	Extra GRU Layer	0.001	8	64	0.02085349	65	0.02065518
With DS	Extra GRU Layer	0.00127695	8	64	0.02113024	92	0.02073933
With DS	Extra linear layer	0.00001	8	8	0.02313514	100	0.02313514
With DS	Extra linear layer	0.0001	8	8	0.02354318	98	0.02382321
With DS	Extra linear layer	0.001	8	8	0.02129801	96	0.02652392
With DS	Extra linear layer	0.00127695	8	8	0.02470327	94	0.02652392
With DS	Extra linear layer	0.00001	8	16	0.02377962	100	0.02377962
With DS	Extra linear layer	0.0001	8	16	0.02617591	98	0.02693016
With DS	Extra linear layer	0.001	8	16	0.02493809	60	0.02418215
With DS	Extra linear layer	0.00127695	8	16	0.02493243	94	0.02517483
With DS	Extra linear layer	0.00001	8	64	0.02093458	100	0.02093458
With DS	Extra linear layer	0.0001	8	64	0.02608156	100	0.02608156
With DS	Extra linear layer	0.001	8	64	0.02233807	66	0.02241266
With DS	Extra linear layer	0.00127695	8	64	0.02206657	79	0.02260689
No DS	Extra linear layer	0.00001	8	8	0.0160954	100	0.0160954
No DS	Extra linear layer	0.0001	8	8	0.0119969	97	0.01184664
No DS	Extra linear layer	0.001	8	8	0.00978434	100	0.00978434
No DS	Extra linear layer	0.00127695	8	8	0.00992733	100	0.00992733
No DS	Extra linear layer	0.00001	8	16	0.0156498	100	0.0156498
No DS	Extra linear layer	0.0001	8	16	0.0113008	98	0.01119741
No DS	Extra linear layer	0.001	8	16	0.01014326	89	0.01019555
No DS	Extra linear layer	0.00127695	8	16	0.01091932	90	0.01093253
No DS	Extra linear layer	0.00001	8	64	0.0137089	100	0.0137089
No DS	Extra linear layer	0.0001	8	64	0.01049717	100	0.01049717
No DS	Extra linear layer	0.001	8	64	0.01090704	96	0.01086064
No DS	Extra linear layer	0.00127695	8	64	0.01102624	89	0.01055616
No DS	Extra GRU layer	0.00001	8	8	0.01358993	100	0.01358993
No DS	Extra GRU layer	0.0001	8	8	0.01147996	98	0.01152173
No DS	Extra GRU layer	0.001	8	8	0.00970864	93	0.0099403
No DS	Extra GRU layer	0.00127695	8	8	0.00961646	90	0.00963465
No DS	Extra GRU layer	0.00001	8	16	0.0135058	100	0.0135058
No DS	Extra GRU layer	0.0001	8	16	0.01075275	95	0.01059835
No DS	Extra GRU layer	0.001	8	16	0.01068751	80	0.01078111
No DS	Extra GRU layer	0.00127695	8	16	0.01080038	97	0.01184051
No DS	Extra GRU layer	0.00001	8	64	0.01352076	99	0.01337559
No DS	Extra GRU layer	0.0001	8	64	0.00960164	94	0.01004334
No DS	Extra GRU layer	0.001	8	64	0.00984835	99	0.01090355
No DS	Extra GRU layer	0.00127695	8	64	0.01189799	71	0.01178494
With DS	All retrained	0.00001	8	/	0.01121107	100	0.01121107
With DS	All retrained	0.0001	8	/	0.00957101	97	0.00998227
With DS	All retrained	0.001	8	/	0.00850194	80	0.00822925
With DS	All retrained	0.00127695	8	/	0.0114441	61	0.00997429
No DS	All retrained	0.00001	8	/	0.00934764	97	0.00928745
No DS	All retrained	0.0001	8	/	0.00862057	100	0.00862057
No DS	All retrained	0.001	8	/	0.01712259	89	0.00863043
No DS	All retrained	0.00127695	8	/	0.11093591	52	0.00868779

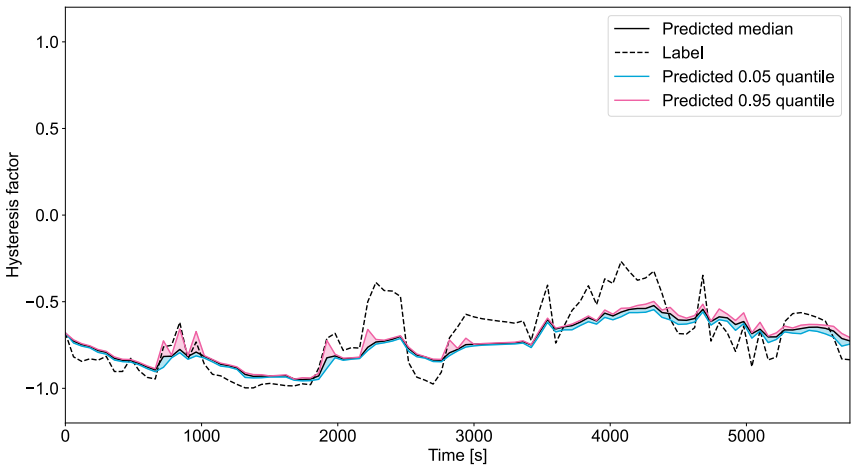
Table B.12: Results for different configurations for training a model for vehicle project B.



(a) GRU model with 600-240 configuration.



(b) GRU model with 300-120 configuration.



(c) GRU model with 60-120 configuration.

Figure B.17: Reconstructed dynamic measurement for different GRU model configurations.

Configuration	Learning rate	Hidden size	Num GRU layers	Batch size	Dropout
600-240	0.00127695	16	2	8	0

Table B.13: Fine-tuned parameters for the temperature generalisation analysis in Case Study 7.

C

Source Code and Flow Diagrams

C.1. Gated Recurrent Network Model

```
"""
GRURegressionModel is a PyTorch neural network module designed for regression tasks
    using Gated Recurrent Units (GRUs).

Attributes:
- input_size (int): The number of input features.
- hidden_size (int): The number of features in the hidden state of the GRU.
- num_layers (int): The number of recurrent layers in the GRU.
- output_size (int): The number of output features.

Methods:
- __init__(self, input_size, hidden_size, num_layers, output_size): Initializes the
    network layers.
- forward(self, x): Defines the forward pass of the model. It initializes the hidden
    state, passes the input through the GRU and a linear layer, and finally applies the
     tanh activation function to limit the output between -1 and 1.
"""

import torch.nn as nn
import torch

class GRURegressionModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size, dropout=0):
        super(GRURegressionModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True,
            dropout=dropout)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.gru(x, h0)
        out = self.fc(out[:, -1, :])
        # Apply tanh function to the output to limit it between -1 and 1
        out = torch.tanh(out)
        return out
```

C.2. Gated Recurrent Network Model With Additional GRU Layer

```
"""
GRURegressionModelExtendedGRULayer is a PyTorch neural network module designed as an extension of an existing GRU-based regression model. This model stacks an additional GRU layer and a linear output layer on the original model for enhanced predictive capability.

Attributes:
- original_model (GRURegressionModel): The original GRU-based regression model to be extended.
- additional_hidden_size (int): The number of features in the hidden state of the additional GRU layer.
- output_size (int): The number of output features.

Methods:
- __init__(self, original_model, additional_hidden_size, output_size): Initializes the extended network layers and incorporates the original model's GRU layers.
- forward(self, x): Defines the forward pass of the extended model. It involves initializing the hidden states, processing the input through the original and additional GRU layers, followed by a linear layer and applying the tanh activation function to limit the output between -1 and 1.
"""
import torch.nn as nn
import torch

class GRURegressionModelExtendedGRULayer(nn.Module):
    def __init__(self, original_model, additional_hidden_size, output_size):
        super(ExtendedGRURegressionModel, self).__init__()
        self.hidden_size = original_model.hidden_size
        self.num_layers = original_model.num_layers

        self.gru = original_model.gru
        self.additional_gru = nn.GRU(original_model.hidden_size, additional_hidden_size, batch_first=True)
        self.fc = nn.Linear(additional_hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.gru(x, h0)

        h1 = torch.zeros(1, x.size(0), self.additional_gru.hidden_size).to(x.device)
        out, _ = self.additional_gru(out, h1)

        out = self.fc(out[:, -1, :])
        out = torch.tanh(out)
        return out
```

C.3. Gated Recurrent Network Model With Additional Linear Layer

```
"""
GRURegressionModelExtendedLinearLayer is a PyTorch neural network module designed as an extension of an existing GRU-based regression model. This model adds an additional linear layer followed by the final linear output layer for enhanced predictive capability.

Attributes:
- original_model (GRURegressionModel): The original GRU-based regression model to be extended.
- additional_hidden_size (int): The number of features in the hidden state of the additional linear layer.
- output_size (int): The number of output features.

Methods:
- __init__(self, original_model, additional_hidden_size, output_size): Initializes the extended network layers and incorporates the original model's GRU layers.
- forward(self, x): Defines the forward pass of the extended model. It involves initializing the hidden states, processing the input through the original GRU layer, followed by an additional linear layer with a ReLU activation, and finally the output layer with a tanh activation function to limit the output between -1 and 1.
```

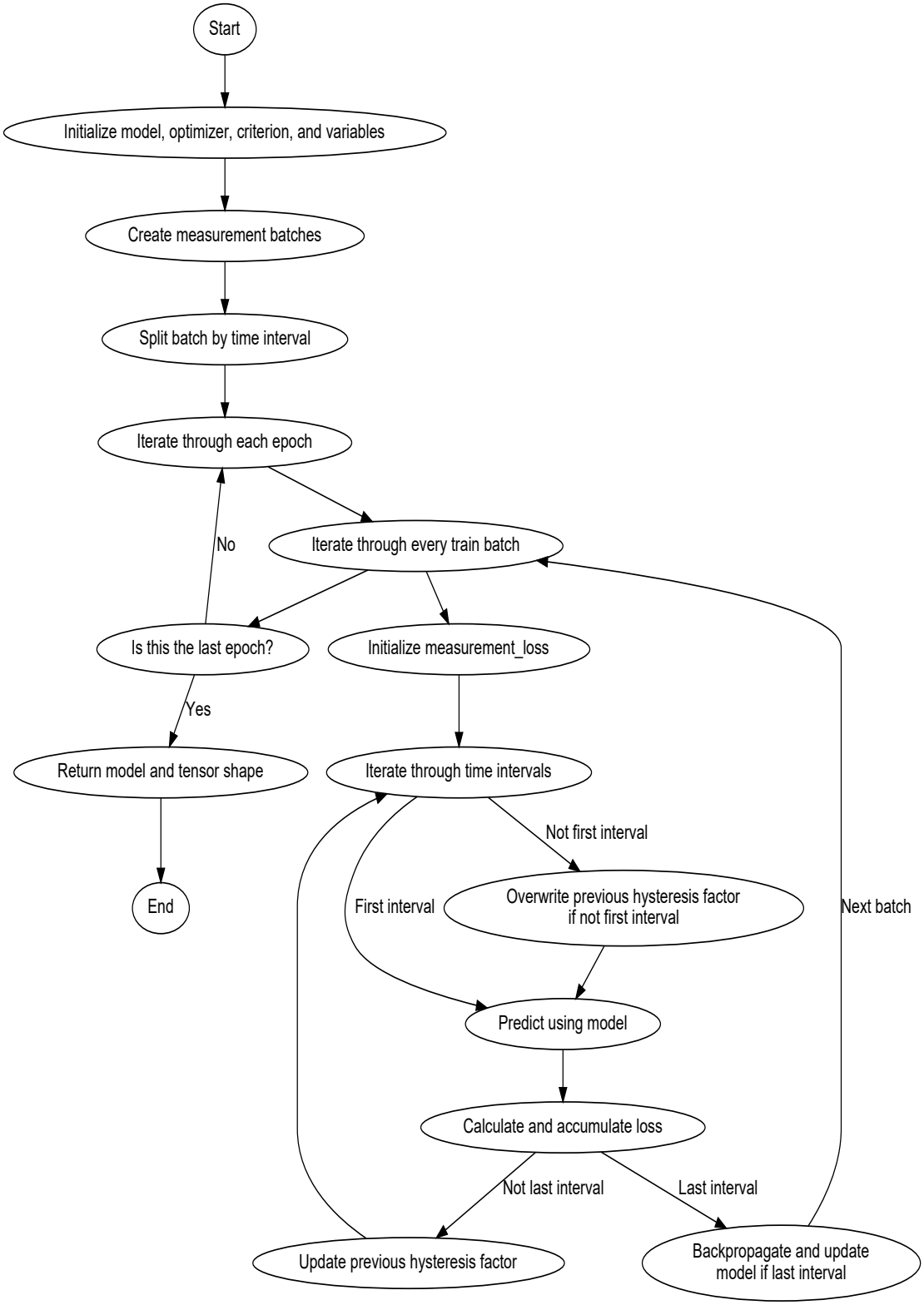
```
"""
import torch.nn as nn
import torch

class GRURegressionModelExtendedLinearLayer(nn.Module):
 def __init__(self, original_model, additional_hidden_size, output_size):
        super(ExtendedLinearRegressionModel, self).__init__()
        self.hidden_size = original_model.hidden_size
        self.num_layers = original_model.num_layers

        self.gru = original_model.gru
        self.additional_fc = nn.Linear(original_model.hidden_size,
            additional_hidden_size)
        self.final_fc = nn.Linear(additional_hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.gru(x, h0)

        out = self.additional_fc(out[:, -1, :])  # Apply the additional linear layer
        out = torch.relu(out)                    # Activation function

        out = self.final_fc(out)                 # Final output layer
        out = torch.tanh(out)                    # Activation function for output
        return out
```

C.4. Pinball Loss

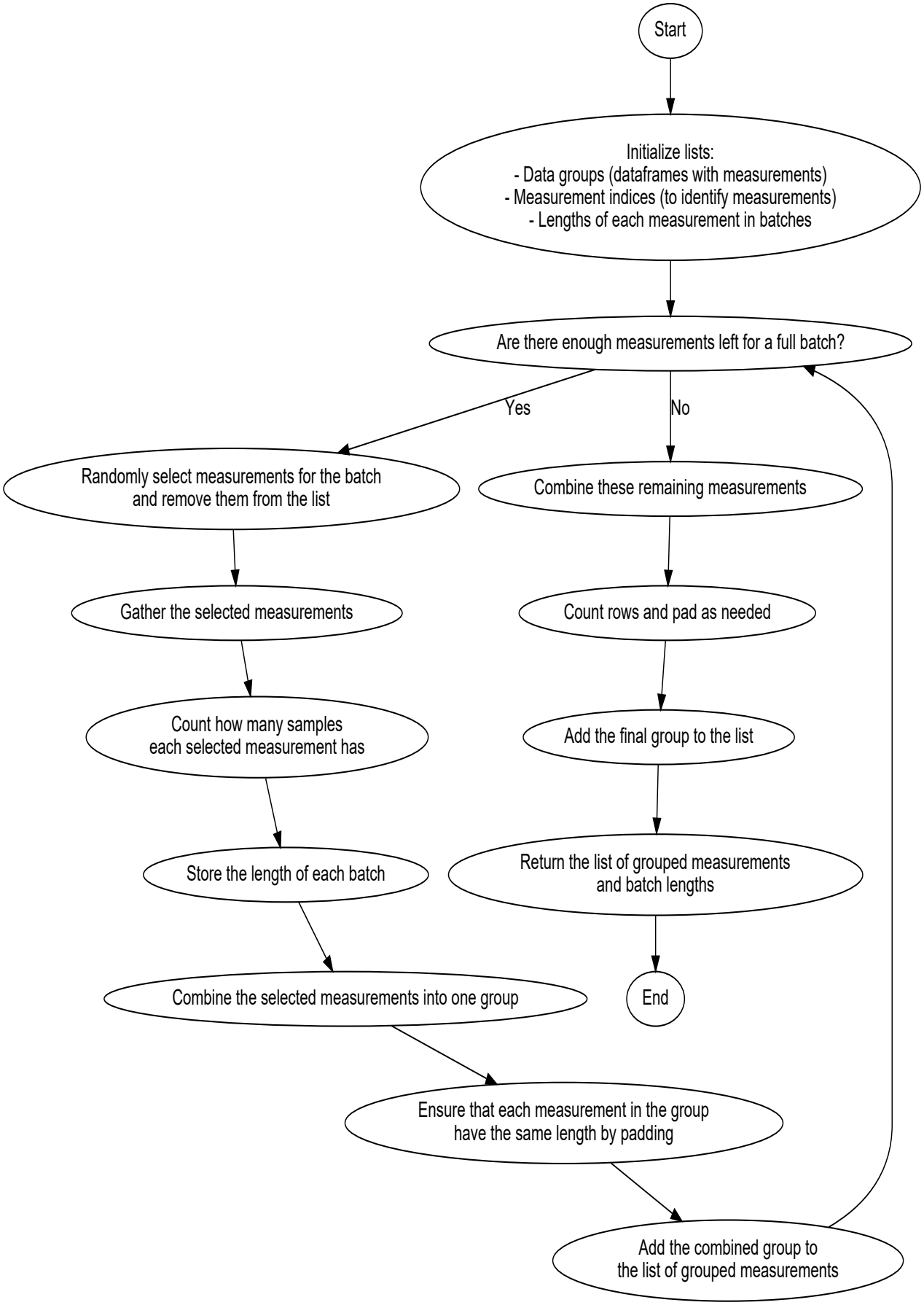
```

def pinball_loss_tensors(y_pred, y_true, tau_tensor):
    """Computes the pinball loss between `y_true` and `y_pred` in PyTorch.
    Source: https://www.tensorflow.org/addons/api_docs/python/tfa/losses/pinball_loss

    loss = maximum(tau * (y_true - y_pred), (tau - 1) * (y_true - y_pred))

    Args:
      y_true: Ground truth values with shape (batch_size, 1)
      y_pred: The predicted values with shape (batch_size, num_quantiles)
      tau_tensor: Tensor containing the to be predicted quantiles with shape (
          num_quantiles,).

    Returns:
      pinball_loss: Scalar tensor representing the pinball loss.
    """
    if tau_tensor.shape[0] != y_pred.shape[1]:
        raise ValueError(f"The␣size␣of␣the␣first␣dimension␣of␣tau_tensor␣(shape:␣{
            tau_tensor.shape})␣must␣be␣equal␣to␣the␣second␣dimension␣of␣y_pred␣(shape:␣
            {y_pred.shape}).")

    delta_y = y_true - y_pred
    loss = torch.maximum(tau_tensor * delta_y, (tau_tensor - 1) * delta_y)
    return torch.mean(loss)
```

C.5. Autoregressive Training

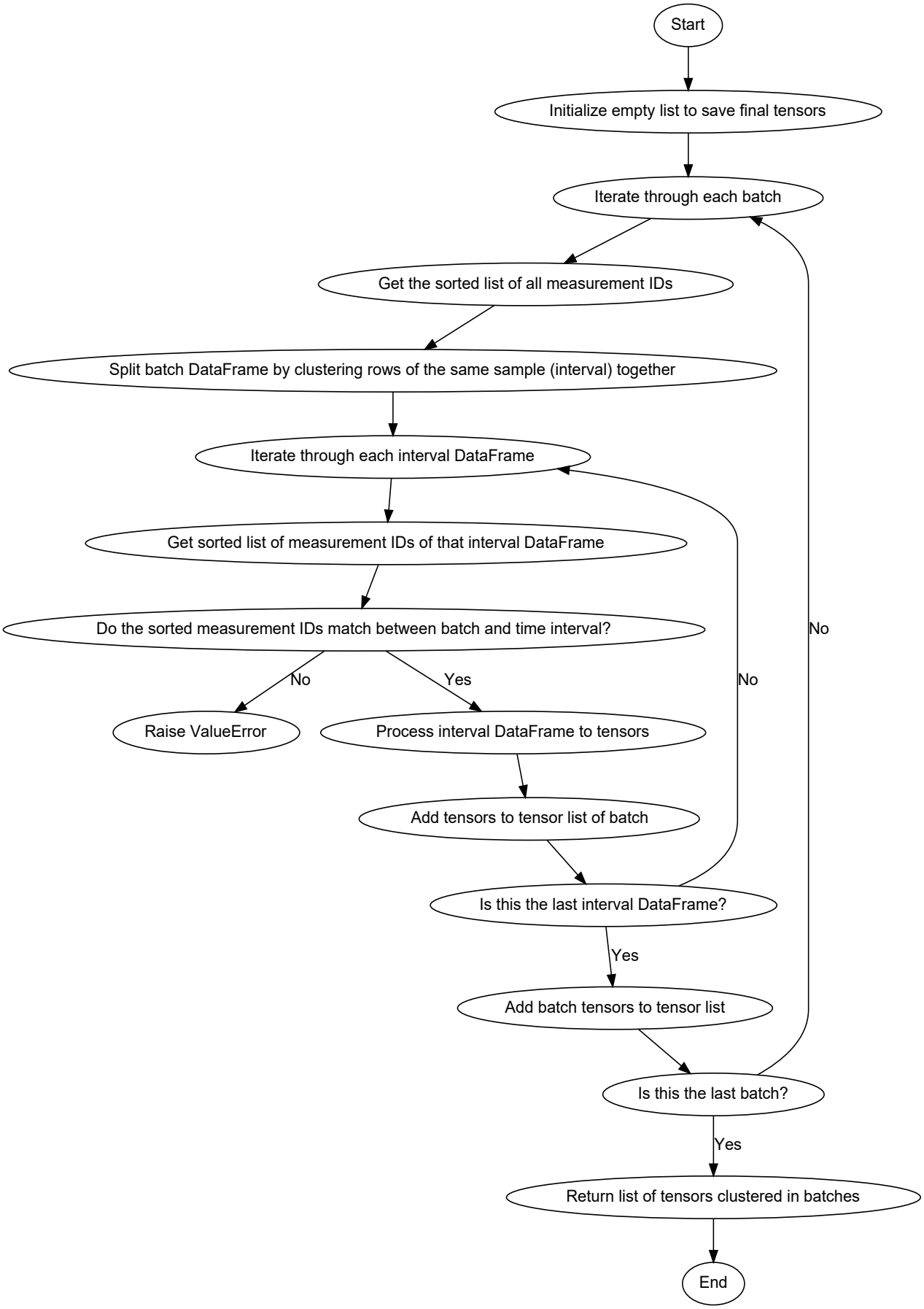
Flow Diagram for Training Autoregressively



Flow Diagram for Randomly Grouping the Measurements Into Batches



Flow Diagram for Ensuring the Batches Are Aligned in Terms of Time Intervals



C.6. Autoregressive Testing

Flow Diagram for Testing Autoregressively