Energy Consumption Prediction in Vehicle Power Nets Using High-Dimensional Network Communication Data - A Machine Learning Approach

Energieverbrauchsvorhersage in Kraftfahrzeugbordnetzen unter Verwendung hochdimensionaler Netzwerkkommunikationsdaten - ein Ansatz aus dem Maschinellen Lernen

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultät
der Universität des Saarlandes

vorgelegt von

M.Sc. Julian Müller

Saarbrücken

2025

Tag des Kolloquiums:	16.06.2025
Dekan:	Univ.-Prof. Dr.-Ing. Dirk Bähre

Mitglieder des Prüfungsausschusses

Berichterstatter:	Univ.-Prof. Dr.-Ing. Georg Frey
	Univ.-Prof. Dr.-Ing. Rainer Müller
Vorsitz:	Univ.-Prof. Dr.-Ing. Matthias Nienhaus
Akad.Mitarbeiter:	Dr.-Ing. Idriss El Azhari

To my family

"L'esprit qui invente est toujours mécontent de ses progrès, parce qu'il voit au-delà."

Jean le Rond d'Alembert

Abstract

English

The electrification of private transport and regulatory requirements to reduce emissions require the energy efficiency of electric vehicles to be maximized. In addition to increasing battery capacity, optimizing the overall energy consumption is essential. In particular, the low-voltage electrical system, which supplies numerous auxiliary components, represents a previously underestimated influencing factor. This dissertation pursues a data-driven approach to analyze and forecast the energy consumption of electronic control units and their peripherals. For this purpose, a scalable data processing pipeline is developed that imports, selects, processes and prepares measurement data for machine learning models according to the latest feature engineering and selection methods. A suitable regression algorithm is selected and validated for power consumption prediction. Explainable AI is integrated into the pipeline to increase transparency and the causes of energy consumption. This enables a detailed analysis of the variables influencing consumption and makes it easier for users to interpret the model decisions. This provides engineers with valuable insights for optimizing the vehicle electrical system architecture and component operating strategies. The methods developed contribute to the data-driven increase in efficiency of electric vehicle platforms and ultimately enable a more precise range forecast already during the development phase.

Zusammenfassung

Deutsch

Die Elektrifizierung des Individualverkehrs sowie regulatorische Vorgaben zur Emissionsreduktion erfordern eine Maximierung der Energieeffizienz von Elektrofahrzeugen. Neben der Erhöhung der Batteriekapazität ist die Optimierung des Gesamtenergieverbrauchs essenziell. Insbesondere das Niederspannungsbordnetz, das zahlreiche Nebenverbraucher versorgt, stellt einen bislang unterschätzten Einflussfaktor dar. Diese Arbeit verfolgt einen datengetriebenen Ansatz zur Analyse und Prognose des Energieverbrauchs von Steuergeräten und deren Peripherie. Hierzu wird eine skalierbare Datenverarbeitungspipeline entwickelt, die Messdaten importiert, auswählt, verarbeitet und für Machine-Learning-Modelle gemäß neuester Feature Engineering Methoden aufbereitet. Zur Verbrauchsvorhersage erfolgt die Auswahl und Validierung eines geeigneten Regressionsalgorithmus. Zur Erhöhung der Transparenz und der Ursachen des Energieverbrauchs wird Explainable AI in die Pipeline integriert. Dies ermöglicht eine detaillierte Analyse der Einflussgrößen auf den Verbrauch und erleichtert die Interpretation der Modellentscheidungen durch die Anwender. Dadurch erhalten Ingenieure fundierte Einblicke zur Optimierung der Bordnetzarchitektur und Betriebsstrategien der Komponenten. Die entwickelten Methoden leisten einen Beitrag zur datengetriebenen Effizienzsteigerung elektrischer Fahrzeugplattformen und ermöglichen schlussendlich eine präzisere Reichweitenprognose bereits in der Entwicklungsphase.

Abrégé

Français

L'électrification du transport individuel et les réglementations sur la réduction des émissions nécessitent d'optimiser l'efficacité énergétique des véhicules électriques. Au-delà de l'amélioration des batteries, l'optimisation de la consommation énergétique globale est cruciale. Le réseau de bord de basse tension, alimentant de nombreux systèmes auxiliaires, représente un facteur d'influence jusqu'ici sous-estimé. L'étude présente adopte une approche axé sur les données pour analyser et prédire la consommation énergétique des unités de contrôle et de leur périphérie. Un pipeline de traitement des données évolutif est développé pour importer, sélectionner et traiter les données de mesure, les préparant pour des modèles de l'apprentissage automatique selon les techniques d'ingénierie des caractéristiques actuelles. Un algorithme de régression sera validé pour prédire la consommation électrique. L'intégration d'Explainable AI dans le pipeline augmente la transparence et la compréhension des facteurs de consommation, facilitant l'interprétation des décisions du modèle. Les ingénieurs obtiennent ainsi des aperçus précieux pour optimiser l'architecture et les stratégies d'utilisation des composants de réseau de bord. Ces méthodes contribuent à l'amélioration data-driven de l'efficacité des plateformes de véhicules électriques et permettent des prévisions d'autonomie plus précises déjà dans la phase de développement.

Acknowledgment

I want to express my deepest gratitude to my supervisor Prof. Dr.-Ing. Georg Frey for taking care of my thesis project and for the outstanding collaboration during the entire project, starting from the first quarter of 2022.
Furthermore, I thank my colleagues Daniel Brauneis and Florian Schuchter for their continuous contribution throughout the project as well as for associated publications and all students I had the opportunity to supervise over the past years in the same context. I also would like to express gratitude to my management at Mercedes-Benz AG in particular Jochen Strenkert, Dr. Sicong von Malottki, Dr. Patrick Schmitt, Roland Lindbüchl, and Volker Seefried for their endorsement of this thesis and embarking on this endeavor giving me the opportunity to visit the necessary conferences and any other format of technical exchange and last but not least, to grow personally.
Furthermore, special thanks go to my long-time mentor and sparring partner Dr. Rainer Mäckel for his continuous advice and tips, and for setting the right deadlines for the dissertation at the right time.
Ultimately and most importantly, I want to thank my wife Dr. Constanze Braun for her everlasting support and encouragement during the entire time of this project.

Disclaimer

This thesis was conducted within the scope of a collaborative project between the ZeMA (Zentrum für Mechatronik und Automatisierungstechnik) located in Saarbrücken, Germany and the Mercedes-Benz AG, located in Sindelfingen, Germany. Hence, due to contractual non-disclosure policies, OEM-specific data such as signal names are pseudonymized and corresponding numeric values are either scaled or blacked out. This procedure shall not be a detriment for the general contribution to the state of the art this dissertation represents.

Table of Contents

List of Figures

List of Tables

List of Listings

Abbreviations and Notations

Physical Units

Table 1: Physical symbols, their units, and meanings.

Symbol	Unit	Meaning
Q	C	Electric charge
W_{el}	J	Electric work
U	V	Electric voltage
I	A	Electric current
R	Ω	Electric resistance
P	W	Electric power
t	s	Time
f	Hz	Frequency
v_s	m/s	Vehicle speed
τ	°C	Temperature

Mathematical Notation

Table 2: Mathematical notation used in this dissertation and their meanings—arranged by logical groups.

Group	Notation	Meaning
Data Representation	$X \in \mathbb{R}^{m \times n}$	Matrix of training data: m datapoints (rows), n features (columns).
	$Y \in \mathbb{R}^{m \times q}$	Matrix of target data: m datapoints (rows), q target variables (columns).
	n	Number of features in the dataset.
	m	Total number of datapoints in the dataset.
	q	Number of target variables.
	x_i	Feature vector for the i-th datapoint.
	y_i	Target value for the i-th datapoint.
Model Parameters	β	General representation of model parameters.
	β_0	Intercept in regression models.
	β_1	Curve slope in regression models.
	w_i	Weight for the i-th input feature in a model.
	b	Bias term in a model (e.g. perceptron).
Metrics and Evaluation	R^2	Coefficient of determination, measure for the fitting quality in regression.
	$\mathcal{L}$	Loss function.
	$\overline{y}$	Mean of all target values (labels).
	σ^2	Variance of a dataset.
	$k \in \mathcal{K}$	Specific metric or KPI from the set $\mathcal{K}$.
	$\mathcal{K}$	Set of metrics and KPIs.
ML Algorithms and Hyperparameters	$f(x)$, $\hat{y}$	Predicted output of a model for input x.
	$\mathcal{A}$	Set of machine learning algorithms.
	$a \in \mathcal{A}$	A specific machine learning algorithm from the set $\mathcal{A}$.
	θ	Set of possible hyperparameter combinations.
	θ^*	Optimal hyperparameter combination.
	η	Learning rate in optimization algorithms.
	$\mathcal{P}_a^k(X, y)$	Model performance for algorithm a, metric k, and dataset (X, y).

Continued on next page

Table 2 – continued from previous page

Group	Notation	Meaning
Weighted Sum Analysis	$\mathcal{D}$	Set of all decision options.
	$d \in \mathcal{D}$	A specific decision option.
	$\mathcal{C}$	Set of decision criteria.
	$c \in \mathcal{C}$	A specific decision criterion.
	$\mathcal{V}$	Set of weights for decision criteria.
	$v_c \in \mathcal{V}$	Weight assigned to criterion c.
	s_{c_d}	Normalized score of decision option d for criterion c.
	U_d	Total utility value of decision option d.
	$d_{opt}(a_{opt})$	Decision option with the highest utility value.
Domain-Specific Terms	$\mathcal{T}$	Set of test vehicles equipped with measurement equipment.
	D_t	Number of test drives for test vehicle t, where $t \in \mathcal{T}$.
	v_{s_t}	Speed of test vehicle t.
Additional Notations	$a(x)$	General activation function.
	$a(\vartheta)$	Activation function with parameter ϑ.
	$tanh$	Hyperbolic tangent activation function.
	Φ	Set of feature engineering methods with a sequence.
	Φ_{opt}	Optimal set and sequence of feature engineering methods.

Abbreviations and Acronyms

Table 3: Abbreviations and acronyms used in this dissertation.

Abbreviation	Meaning
AC	Alternating Current
A/D	Analog-to-Digital
ADAS	Advanced Driver Assistance System(s)
AI	Artificial Intelligence
AIC	Akaike Information Criterion
ALE	Accumulated Local Effects
ANN	Artificial Neural Network
API	Application Programming Interface
ASIL	Automotive Safety Integrity Level
BCF	Body Controller Front
BEV	Battery Electric Vehicle
BLF / *.blf*	Binary Logging File
BMS	Battery Management System
BO	Bayesian Optimization
CART	Classification and Regression Trees
CID	Central Infotainment Display
CNG	Compressed Natural Gas
CPU	Central Processing Unit
CV	Cross-Validation
DBC	Database CAN
DC	Direct Current
DL	Deep Learning
DIN	Deutsches Institut für Normung
ECU	Electronic Control Unit
EU	European Union
EV	Electric Vehicle
FCEV	Fuel Cell Electric Vehicle
GB	Gradient Boosting
GenAI	Generative Artificial Intelligence
GmbH	Gesellschaft mit beschränkter Haftung
GPT	Generative Pre-Trained Transformer
GPU	Graphics Processing Unit

Continued on next page

Table 3 – continued from previous page

Abbreviation	Meaning
GS	Grid Search
HVAC	Heating, Ventilation, Air Conditioning
ICE	Internal Combustion Engine
ID	Identifier
IDE	Integrated Development Environment
IFI	Impurity Feature Importance
IQR	Interquartile Range
KPI	Key Performance Indicator
LNG	Liquified Natural Gas
LOOCV	Leave-One-Out Cross-Validation
LPG	Liquified Petroleum Gas
LSTM	Long Short-Term Memory
LTS	Long-Term Support
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MCR	Median Crossing Rate
MDF	Measurement Data Format
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
MOST	Media Oriented Systems Transport
NaN	Not a Number
NN	Neural Network
OEM	Original Equipment Manufacturer
OHE	One-Hot Encoding
OLS	Ordinary Least Squares
ONNX	Open Neural Network Exchange
OS	Operating System
PAAD	Percentage Average Absolute Deviation
PAWD	Percentage Average Weighted Deviation
PCC	Pearson Correlation Coefficient
PFI	Permutation Feature Importance
PNG / *.png*	Portable Network Graphics
PRQ	Primary Research Question
PTC	Positive Temperature Coefficient Thermistor
RAM	Random Access Memory

Continued on next page

Table 3 – continued from previous page

Abbreviation	Meaning
ReLU	Rectified Linear Unit
RF	Random Forest
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
RPM	Revolutions Per Minute
RS	Random Search
SD	Standard Deviation
SGD	Stochastic Gradient Descent
SHAP	Shapley Additive Values
SME	Small and Medium Enterprise
SNA	Signal Not Available
SRQ	Secondary Research Question
SUV	Sports Utility Vehicle
TLU	Threshold Logic Unit
UML	Unified Modeling Language
US	United States
VDE	Verband der Elektrotechnik Elektronik Informationstechnik e.V.
WP	Work Package
WSA	Weighted Sum Analysis
XAI	Explainable AI
ZEV	Zero-Emission Vehicle

1 Introduction

With the increasing social awareness of climate and environmental protection, along with increasingly stringent air pollution control regulations, there are physical limits to reducing emissions from conventional combustion engines, which are still prevalent in global road traffic [1]. For example, the implementation of the EU7 emissions standard [2]—despite alleviating reworks—presents vehicle manufacturers (original equipment manufacturers, OEMs) with technical and economic challenges. This means that the development work required for this is increasingly only proving to be economical in mid and luxury class vehicles, as the contribution margins for compact cars are generally lower [3]. Additionally, the threat of fines from the European Union (EU) and other regulatory bodies like the United States Environmental Protection Agency (US EPA) for exceeding fleet CO_2 values incentivizes a shift in development efforts away from fossil fuels towards more sustainable propulsion technologies [4, 5].
Beyond environmental protection, governments also aim to reduce their dependency on politically unstable rentier states in the Middle East and Northern Africa that possess large shares of fossil fuels and thus can determine the prices to a certain extent [6, 7].

1.1 Motivation and Background

Given the context of the geopolitical tensions and the dependencies on crude oil, the following section provides an overview of the current state of the challenges causing the intended shift from fossil fuel-based towards sustainable zero-emission mobility. This transition is driven by increasing environmental concerns, regulatory pressures, and economic factors that necessitate innovative approaches in transportation technology.

1.1.1 Sustainable Mobility and Energy Efficiency

In 2024, there are several sustainable propulsion technologies for zero emission vehicles (ZEV)—which do not comprise fossil gases such as LNG, LPG and CNG—controversially discussed by the public. The most popular technology is thus the electric drivetrain combined with a traction battery as the primary energy source. Such vehicles are commonly referred to as battery electric vehicles (BEVs or EVs). Regarding ZEV popularity, EVs are followed by hydrogen-powered fuel cell electric vehicles (FCEVs) despite having a relatively small share of the total fleet [8]. Nonetheless, both technologies promise to reduce CO_2 emissions, provided that the required electric power (thus also the hydrogen) is generated from renewable energies such as solar or wind power [9]. Statistics from July 2024 indicate that EVs constitute a much larger share (approximately 4.7 million units) of the total vehicle fleet in the EU (EU27) compared to hydrogen-powered cars (approximately 4,600 units only) [10].

Key Barriers to E-Mobility Adoption

E-mobility is considered to be a key enabler for zero emission mobility. Nonetheless, with the recent discontinuations of several EV funding programs in some European countries (e.g. Germany ceased the so-called *Umweltprämie* in January 2023) [11, 12, 13] sales tend to flatline or even decrease [14]. This shows that considerable barriers for the technology adoption still persist, such as the inadequate and inconsistent charging infrastructure, high upfront costs of EVs, limited vehicle availability, consumer misconceptions about EV performance, variability in government incentives and policies, and range anxiety [15].

Efficiency as a Lever for E-Mobility Adoption

Focusing on range anxiety, there are two ways for OEMs to lower this barrier and accelerate EV adoption: either by increasing the battery size or by reducing the vehicle's energy consumption [16]. Since high-capacity traction batteries, which are mostly based on lithium-ion technology as of today, not only constitute a significant cost component of the vehicle but also contribute to its overall weight, OEMs strive to focus on developing vehicles that use less energy, hence that are more efficient. They can do so in various domains, which in addition to weight, include aerodynamics, tires, and the drivetrain. With increasing efficiency in these other domains—e.g. an EV with the electric powertrain can reach a well-to-wheel efficiency up to seven times higher than the internal combustion engine (ICE), depending on the source of the electrical energy [17]—the significance of the proportion of the low-voltage auxiliary components (e.g. displays, infotainment computers, or seat heating) increases. Thus, these components become a considerable lever to increase overall energy efficiency, too. On average, the latter accounts for around 9 % of the total energy consumption of an EV during homologation tests where only a minimum of power consumers are active [18]. Figure 1.1 depicts the five primary energy efficiency domains, including the low-voltage power net with the auxiliary components (marked in green) as the focus lever discussed in this thesis.

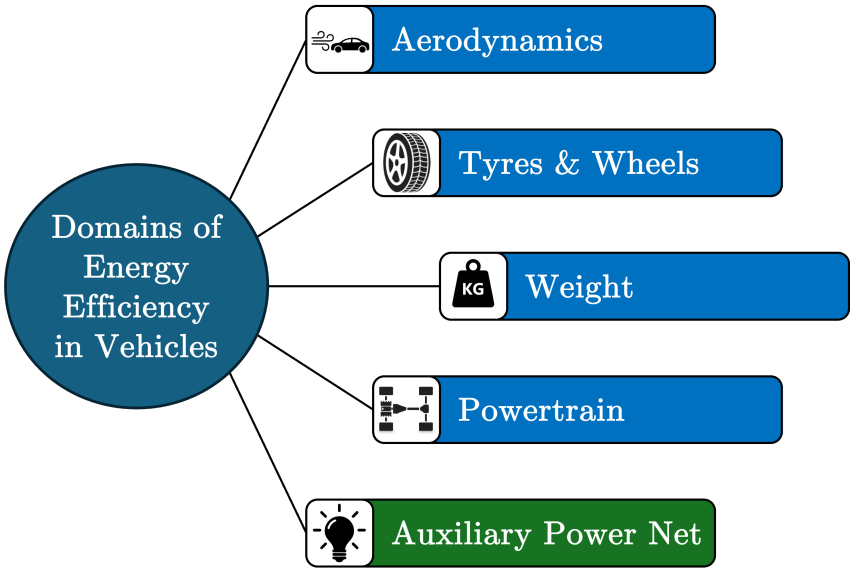

Figure 1.1: The five main levers of energy efficiency in vehicles (own representation according to [16, 18, 19, 20, 21]).

1.1.2 Economic Aspects of Energy Efficiency

The auxiliary power net is a complex system due to the high number of power consumers which include electronic control units (ECUs) and their corresponding sensors and actuators. Furthermore, these components are interconnected through various communication technologies and power supplies, forming a complex electrical and electronic (E/E) architecture (more details are outlined in Section 3.4). Adding to this complexity, the system behavior is determined by both the hardware and software of the components, as well as the underlying operational strategies of the vehicle's (customer-recognizable) functions.
Constrained by this complexity, efficiency engineers require considerable experience and knowledge to understand and optimize the system towards a reduced consumption while maintaining the level of functionality and comfort expected by the customers. This process is not only time-consuming but also costly as extensive and heavy measurement equipment is often needed in many development vehicles to measure the respective power consumption and inter-ECU communication at the same time.

1.2 Research Problem

This section addresses the main research problem which is to be assessed within the scope of this thesis. A motivation for a scientific approach is given by the formulation of a research gap.

1.2.1 General Problem Statement

Given the current challenges of the mobility transition, new methods are required for accurately measuring energy consumption and analyzing the overall E/E system and architecture. A key issue is developing approaches that generate this knowledge without necessitating deep expertise in each individual vehicle subsystem, while remaining economical and efficient. This involves minimizing the need for extensive measurement equipment and personnel involved. Furthermore, an improved process should aim to streamline internal workflows within the OEM's organization, reducing both interface dependencies and overall organizational complexity [22, 23]. To achieve this, the data generated by the inter-ECU communication of selected test vehicles is leveraged to extract valuable information about the functionality of the E/E architecture and the in-vehicle power net. At this point, it is worth mentioning that the focus of this work is on making the vehicle data usable in terms of gaining knowledge about energy consumption. The derivation of specific measures to reduce consumption, on the other hand, will be the subject of future work based on this.

1.2.2 Identification of the Research Gap

An extensive literature review has shown that no prior work has addressed the aforementioned research problem as formulated above (cf. Sections 1.2 and 3.8.2). Databases and search engines related to big data analysis, artificial intelligence (AI), and the automotive industry were thoroughly examined, including among others *ScienceDirect*, *IEEE Xplore*, *SpringerLink*, *arXiv.org*, *ACM Digital Library* and *Google Scholar*. In addition, selected analog literature was also consulted.

The review permits the conclusion that a research gap exists, which can be addressed by defining appropriate objectives and answering subsequent research questions (cf. Section 2). Specifically, the gap consists in the absence of a data-driven approach in the state of the art (cf. Section 3) which leverages ECU communication and power consumption data in the low-voltage power net of vehicles.
Current studies predominantly focus on isolated aspects such as energy efficiency, ECU communication or AI applications in automotive systems other than the low-voltage power net (cf. Table 3.2). However, none of them integrates these elements into a cohesive, data-driven approach that deals with the complexity of the E/E architecture in modern vehicles (leveraging data for system knowledge generation and energy consumption prediction on ECU level) either. This gap highlights the need for an innovative framework providing actionable insights into energy consumption patterns without requiring extensive, costly and heavy measurement equipment nor in-depth domain expertise in every vehicle subsystem.

1.3 Objectives and Limitations

Derived from the research gap, predefined objectives are formulated which need to be achieved during the course of this thesis. These objectives can be divided into general and more specific research objectives.
Generally, the data-driven energy consumption modeling and prediction are intended to support development engineers and are not to be used in the context of a customer feature inside the vehicle.

1.3.1 General Objectives

(i) **Standardization Using Data Science Best Practices**: Develop a standardized methodology or framework to derive ECU consumption models from raw data, with an emphasis on economic practicability.

(ii) **Feasibility of Power Consumption Prediction**: Demonstrate that power consumption prediction using inter-ECU communication data and previously trained ECU models is generally feasible.

(iii) **Model Explainability**: Demonstrate the explainability of selected prediction models to showcase transparency of the influential factors of the power consumption.

1.3.2 Specific Objectives

(i) **Create a Representative Database:** Establish a comprehensive and representative database for ECU model creation, ensuring that it includes a variety of vehicle types, driving conditions, and environmental factors to capture a broad range of scenarios.

(ii) **Define Performance KPIs and Metrics:** Use meaningful key performance indicators (KPIs) to assess the performance of the developed machine learning (ML) pipeline. These KPIs shall include metrics such as prediction accuracy, computational efficiency (speed), and at least one project specific metric taking into account the electric charge consumed per test drive.

(iii) **Identify Key Features:** Establish a meaningful and automated mechanism to work out relevant features for the individual power consumption models prior to the training phase.

(iv) **Conduct Objective Modeling Algorithm Selection:** Define a methodology to objectively select a "most suitable" ML modeling algorithm appropriate for the given research problem.

1.3.3 Scope Limitations

In addition to the objectives, the limits to the scope of this thesis are defined, allowing a clear focus on the aforementioned research objectives only. The limitations include:

Data Availability and Quality

The accuracy and reliability of the energy consumption predictions are dependent on the quality and comprehensiveness of the data collected from a given set of test vehicles. Since the number of available vehicles for data collection is limited so is the amount of data available for this research. Driving scenarios not represented in the datasets may not be accurately predictable. Furthermore, the collection method itself can lead to noise and incomplete data since it is subjected to the Nyquist-Shannon sampling theorem, which states that a signal must be sampled at least twice the highest frequency present in the signal to be accurately reconstructed [24] (which may not be safeguarded in this project).
Finally, the research focuses on specific types of ECUs and their communication data from a specific OEM, which may not be representative of all ECUs or vehicles in the automotive industry.

Computational Ressources and Methodology

The resources of the available computation hardware limit computation times for model generation and evaluation. Additionally, the random access memory (RAM, the working memory) sets boundaries for the amount of training data that can be processed by one machine at a time.
This study shall leverage current state-of-the-art technologies in ML and data science at the time of its accomplishment, but may not encompass future advancements or emerging technologies.

Integration and Implementation Limitations

The entire data-driven power consumption prediction pipeline including the derived ECU models is not supposed to be optimized to run on lower-performance edge hardware since it is not supposed to be applied as part of a customer function inside a vehicle, as stated earlier, where computational power is usually sparse.
Any code implementation made is limited to be utilized with the development tools, including the integrated development environment (IDE), the operating system (OS) of the host machine and the software packages only. Generalizability of the latter is not tested.

1.4 Organization of the Thesis

This thesis is divided into five main work packages (WPs), each addressing a specific aspect of the research. They are depicted in Fig. 1.2 and described subsequently.

1.4.1 Research Questions

The first WP is dedicated to the definition and formulation of the research questions which guide this study. They are derived from the identified research gap (cf. Section 1.2.2) and serve to focus the investigation on specific, measurable, and achievable goals. This WP is split into three parts:

- **Identification of the Questions:** Derive primary and subordinate dedicated research questions.
- **Justification of the Questions:** Provide a reasoning for each question, explaining how it addresses the identified research gap.
- **Link to Objectives:** Show the connection between the research questions and the specific objectives outlined in Section 1.3.

1.4.2 State of the Art

The second WP addresses the state of the art (cf. Section 3) and outlines the current technical development and scientific research in the relevant domains for this thesis. It covers both automotive and computer science, with a particular focus on the field of ML as well as fundamentals of electrics. Another subsection of the state of the art serves to underline the research gap by reviewing existing literature.

1.4.3 Preliminaries, Methodology and Modeling

The third WP involves the development of the required preliminaries and of a methodology necessary for training ECU power consumption prediction models as well as the execution of the modeling process itself (cf. Section 4).

- **Data Collection and Transformation:** Describes the process of gathering relevant data from selected test vehicles, including details on the types of data collected and the methods used for acquisition.
- **Evaluation Metrics:** Defines appropriate metrics for evaluating the performance of the trained ML models, ensuring that the chosen metrics align with the research objectives and industry standards.
- **Pipeline Composition and Implementation:** Details the creation and implementation of a holistic processing pipeline, including data preprocessing steps, feature selection, hyperparameter tuning, the application of ML algorithms as well as KPI generation and documentation.

- **Modeling Algorithm Selection:** Includes the actual modeling and evaluation part, the data processing pipeline is applied to the collected data to generate ECU consumption models, predictions, and explanations. A selection of ML models provides trial results on a given set of power consumers allowing the conclusion on an optimal algorithm choice among them.

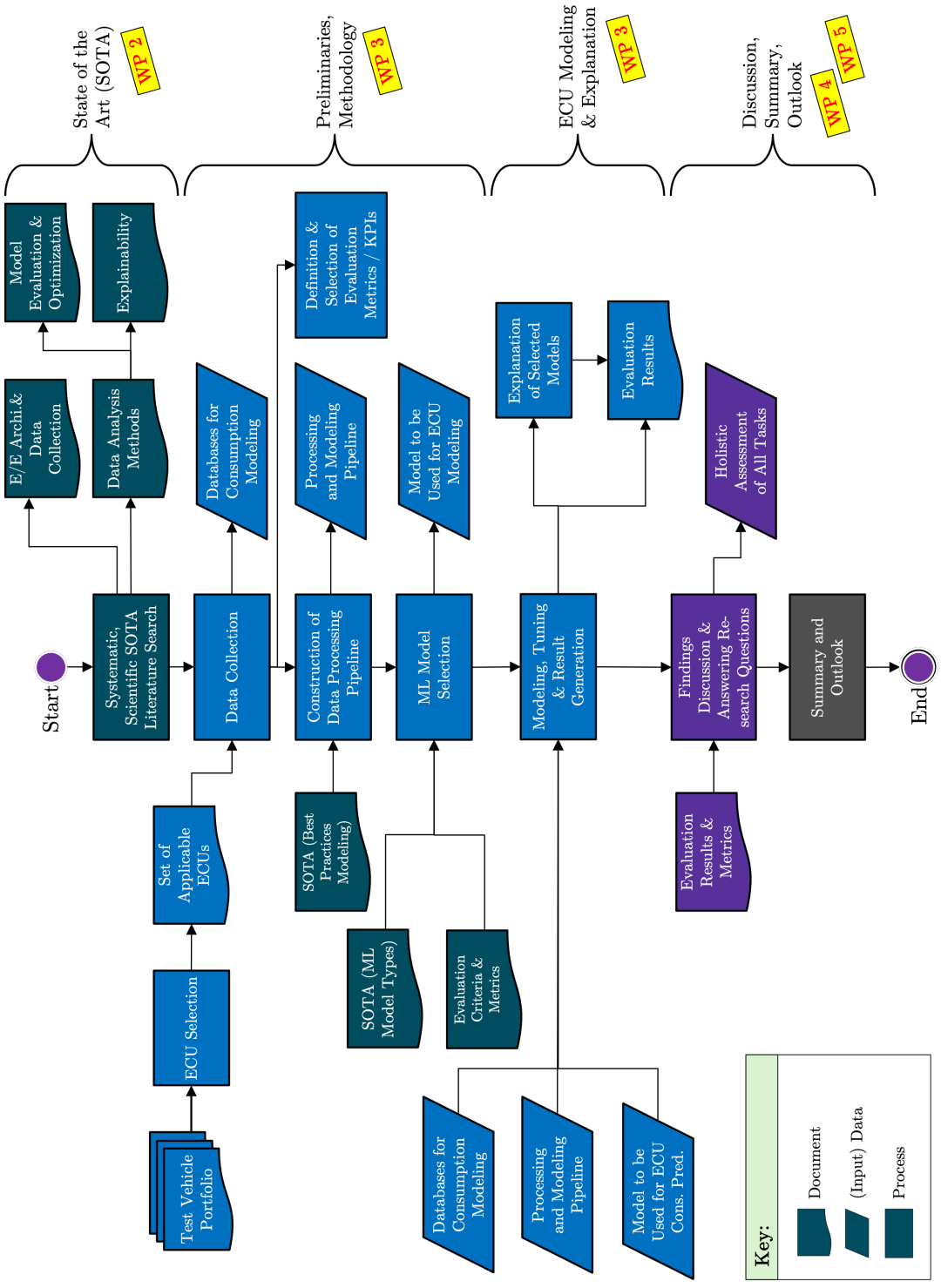

Figure 1.2: The thesis process as a graphical representation. A distinction is made between "processes", "data" and "document" indicating either directed input or output information.

1.4.4 Discussion

The fourth WP focuses on the discussion of the results presented in Section 4. It comprises a final assessment of the performance of the generated ECU consumption prediction models given the application of the data processing pipeline together with the selected ML modeling algorithm. Additionally, it provides answers to the research questions, based on the findings from the data analysis and model evaluations. The model explainability and the limitations are also discussed. Finally, the implications of these findings for the automotive industry and for the way-of-working for development engineers are assessed.

1.4.5 Conclusion and Future Work

The fifth and final WP provides a summary of the thesis, highlighting the key developments and underlining the contributions in Section 6. It also outlines potential areas for future research and development in the industrial fields considered. This includes possible extensions of the approach presented.

1.5 Main Contributions and Prior Publications

The main contribution of this thesis is the development of an automated and integrated pipeline that is capable of processing high-dimensional in-vehicle communication data. Additionally, there is the application of state-of-the-art ML modeling algorithms to predict energy consumption of ECUs and other power consumers in the vehicle with the possibility to examine the model with the help of explainable AI (XAI) techniques. This novel approach thus reduces the need for costly (time and money) measurement equipment in certain use cases during the development phase of a passenger car. The data-driven approach provides development engineers with insights of the data, the model and the energy consumption in a highly automated way so that as little data science knowledge as possible is needed. To this end, the pipeline comprises the following features which themselves are again a contribution investigated on and realized within the scope of this thesis project:

- **Handling High-Dimensional Data**: The data which is communicated in vehicles is vast and complex due to its high dimensionality and frequency. Hence, it is a necessity to develop a way to automatically handle the data upon importing and before working with it e.g. by training an ML model. This comprises the identification of meaningful data which contributes to the ML model's predictive quality, the handling of missing values, the correct resolution of the various data types and adequate scaling where the need arises. Since the in-vehicle data is so unique and different for each power consumer (ECU) a novel and highly automatable approach is developed.

- **Model Comparison and Selection**: Together with AI and domain experts a set of criteria is defined and weighted according to which the suitability of an already existing ML modeling algorithm is selected for the main problem of this research (cf. Section 1.2). The respective metrics provide the degree of fulfillment in a weighted sum analysis (WSA). Given this fact, a selection of possibly suitable models is evaluated.

- **Explainability**: To explain individual predictions or understand overall feature importance and feature interactions leading to a certain ECU power consumption state-of-the-art XAI techniques are applied to provide intuitive insights into the data and the ML model's behavior. The application of these techniques to the in-vehicle data helps to understand how different factors such as the environment, the ECU configuration or the driver's behavior influence energy consumption.
- **Showcasing Real-World Application**: The pipeline and evaluations are applied on real-world datasets collected in a meaningful way from specially equipped test vehicles proving its validity and practical utility for productive application in vehicle development activities. Furthermore, the scalability of the approach demonstrates application perspectives for a broad variety of data and power consumers without the need to adapt the code basis.
- **Interdisciplinary Approach**: Bridging the gap between automotive engineering, data science, and ML demonstrates the interdisciplinary nature of modern engineering challenges which are no longer limited to one specific field but which are intertwined with other adjacent disciplines.
- **Contribution to the Mobility Transition**: Contributing to the broader goal of sustainable mobility by facilitating the optimization of energy consumption in vehicles especially of those with an electric drive train, this dissertation ultimately contributes to reduced emissions and better overall resource utilization.

During the course of this research project, several related publications have been achieved by the author together with several co-authors which are summarized in Table 1.1.

Table 1.1: Publications related to this dissertation and underlying research project.

No.	Author(s)	Year	Title and DOI
1	Müller, Julian; Sprave, Joachim; Frey, Georg	2023	The Effect of Target Variable Rescaling on Energy Consumption Prediction in a Vehicle Powernet using Multi-Target Regression Trees: A Study From the Automotive Industry [25], ICMLT 2023 Conference, DOI: 10.1145/3589883.3589905
2	Müller, Julian; Czekalla, Lukas; Schuchter, Florian; Frey, Georg	2023	Illuminating the Black Box: A Comparative Study of Explainable AI for Interpreting Time Series Data in Vehicle Power Net Consumption Models [26], ICMLA 2023 Conference, DOI: 10.1109/ICMLA58977.2023.00031
3	Müller, Julian; Schuchter, Florian; Brauneis, Daniel; Frey, Georg	2024	Enhancing Power Net Efficiency with Data-Driven Consumption Prediction - A Machine Learning Approach [27], IVS 2024 Conference, DOI: 10.1109/IV55156.2024.10588508
4	Müller, Julian; Schuchter, Florian; Frey, Georg	2024	Licht in der Blackbox – Eine Vergleichsstudie unterschiedlicher Explainable AI Methoden zur Erklärung von Stromverbräuchen im Niedervolt-Bordnetz von Kraftfahrzeugen [28], VDI Automation 2024, DOI: 10.51202/9783181024379
5	Müller, Julian; Schuchter, Florian; Frey, Georg	2024	Verbrauchsprognose im Kfz-Bordnetz mit erklärbarer KI: Vorschlag einer datenbasierten Methodik [29], atp magazin, DOI: 10.17560/atp.v66i8.2735
6	Müller, Julian; Liu, Xuanheng; Maier, Jonas; Frey, Georg	2024	Transferability of Machine Learning Models for Energy Consumption Prediction of In-Vehicle Low-Voltage Equipment: A Case Study [30], IEEE CASE 2024, DOI: 10.1109/CASE59546.2024.10711740
7	Müller, Julian; Wagenknecht, Jonas; Schuchter, Florian; Frey, Georg	2024	Proposal of an Automated Feature Engineering Pipeline for High-Dimensional In-Vehicle Bus Data Using Reinforcement Learning [31], RAAI 2024 Conference, DOI: 10.1109/RAAI64504.2024.10949538

1.6 Note on Code Examples and UML Diagrams

In selected subsections, this thesis contains simple program code excerpts in the Python programming language, which serve in particular to provide a better understanding of the methodology developed. The reader is therefore advised to possess a basic knowledge of this programming language. Similarly, the Unified Modeling Language (UML) is used to represent code object relations.

2 Research Questions

In this chapter the primary and secondary research questions guiding and defining the workflow of this thesis are defined. The questions are derived from the research gap outlined in Section 1.2.2 and align with the overall objectives from Section 1.3. Addressing these questions will contribute to the state of the art in the prediction of energy consumption in vehicle power nets using high-dimensional network communication data, an integrated ML pipeline design and in the productive application of XAI.

2.1 Primary Research Questions

The primary research questions (PRQs) comprise the main focus of this thesis and address the primary aspects of the research gap:

(i) **PRQ1: How can high-dimensional ECU network communication data be utilized to predict energy consumption in vehicle power nets?**

(ii) **PRQ2: What are the most suitable ML modeling algorithms and preparatory techniques for predicting ECU energy consumption and how can they be integrated into a holistic pipeline?**

(iii) **PRQ3: How can the explainability of the trained ECU models be ensured whilst maintaining high prediction accuracy at the same time?**

2.2 Secondary Research Questions

The secondary research questions (SRQs) support the primary ones by addressing specific subproblems:

(i) **SRQ1: What network communication signals are relevant for predicting energy consumption in vehicle power nets, and what preprocessing and feature selection techniques are needed to handle high-dimensional data effectively?**

(ii) **SRQ2: Which evaluation metrics are appropriate for assessing the performance of the ML models?**

(iii) **SRQ3: How can the data processing pipeline be defined to ensure efficient and automated model generation?**

(iv) **SRQ4: What methods can be employed to interpret and explain the predictions made by the ML models?**

(v) **SRQ5: Is there one "most suitable" modeling algorithm to be selected in order to reduce the complexity of the pipeline?**

Both PRQs and SRQs are eventually answered in Sections 5.1.4 and 5.1.5.

2.3 Justification and Link to Objectives

The primary and secondary research questions are justified as follows: PRQ1 addresses the core challenge of utilizing high-dimensional network communication data for energy consumption prediction, which is central to this thesis. This question aims to fill the primary research gap identified by exploring how such data can be leveraged to provide predictions without direct measurement of the currents. PRQ2 focuses on identifying and integrating adequate ML modeling algorithms, a critical step in developing the training and prediction pipeline. This question aligns with the objective of using data science best practices to derive ECU consumption models from raw data while considering economic practicability. PRQ3 ensures that the models used are not only accurate but also interpretable (or explainable), addressing the need for XAI techniques whose application is a key objective of the thesis. Ensuring model transparency and understanding is crucial for gaining the trust of development engineers and fostering AI adoption in their daily work.
SRQ1 supports PRQ1 by identifying relevant data types and necessary preprocessing steps. These questions are essential for finding the most important features relevant to power consumption prediction, thereby enabling effective model training. SRQ2 supports PRQ2 by establishing a way how to measure model performance effectively, ensuring that meaningful KPIs are defined and evaluated. This alignment is crucial for assessing the developed ML pipeline's effectiveness and robustness. SRQ3 ensures the automation of the data processing pipeline, supporting the practical application of PRQ2. By optimizing the data processing steps, the research aims to handle large volumes of inter-ECU communication data efficiently and in an automated way to lower the AI knowledge needed by future users targeting at domain instead of AI experts. SRQ4 directly supports PRQ3 by focusing on methods for interpreting and validating model predictions, ensuring the explainability of the AI models. The justification for SRQ5 is the practical need to balance predictive performance with simplicity of the pipeline. While diverse ML algorithms can be employed to model energy consumption, each one introduces unique requirements for feature engineering and selection, hyperparameter tuning, and computational resources needed. Identifying a "most suitable" algorithm simplifies the pipeline, reduces development and operational complexity and enhances the interpretability and maintainability of the model. Additionally, such a choice can minimize redundancy and improve scalability, making the solution more economically and technically applicable for real-world applications.

3 State of the Art

In the following chapter the state of the art is worked out for the various sub domains relevant for this thesis. For the corresponding research a structured way is developed to find adequate sources and resources.

3.1 State of the Art Assessment Strategy

Due to the multidisciplinarity of the research problem there is the need to develop a structured way to assess the state of the art for this thesis. Thus, it can be divided into different topical sections. Furthermore, a wide range of available document sources and types must be covered which is why source systems with different focal points are considered. They are depicted in Fig. 3.1 and a research process is defined which is described in greater detail in Section 3.2. The main focus shall be on the basics of electrics (cf. Section 3.3) in order to build a technological foundation for the reader.
Subsequently, the current developments in automotive E/E architectures are worked out to underline the cutting-edge technologies under investigation. This includes the most recent networking and on-board power supply (power net) technologies as well as ways OEMs collect data in their test vehicles (cf. Section 3.4.5).
This is followed by the second focal research area which is about data science and ML as a subdomain of AI (cf. Section 3.5). Best industry and research practices as well as recent developments in setting up AI (or ML) projects and algorithms are shown which the next chapters then build upon. Due to the numeric problem dealt with in this project which is to predict the ECU current consumption (cf. Chapter 2) the research on ML modeling algorithms focuses on regression (instead of classification) algorithms since regression is capable of solving this type of problem. After the modeling algorithms, research on possibly suitable XAI methods is carried out as well (cf. Section 3.6) before defining a mechanism for ML algorithm assessment (cf. Section 3.7).
To round off the state of the art, an overview of related existing approaches similar to the one carried out in the course of this thesis is presented in Section 3.8.

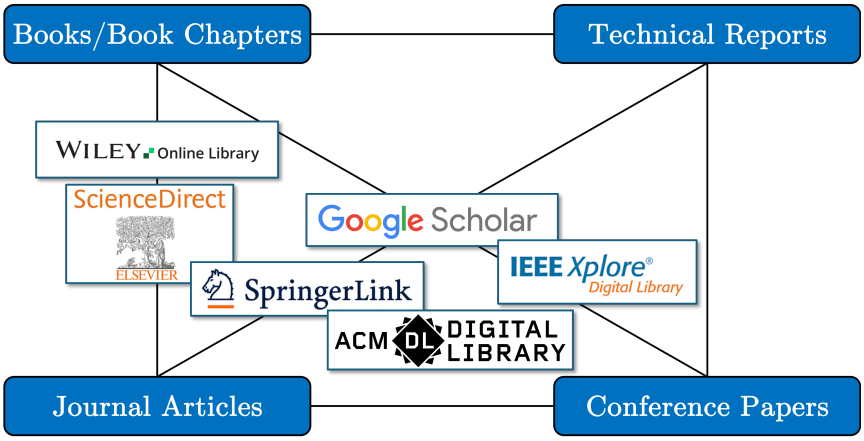

Figure 3.1: To obtain a broad overview of the state of the art for this thesis digital libraries are crawled which provide diverse types of references such as books, chapters of books, (journal or conference) papers and technical reports.

3.2 The State of the Art Process

Fig. 3.2 depicts the different steps of the state of the art investigation process. The research for relevant references is carried out in a structured and reproducible way so that an objective and broad overview over the latest advancements can be given.
The process starts with a brainstorming for keywords to find promising references related to the general topics of this thesis which are in the fields of electricity, automotive engineering, data processing and ML technologies. Afterwards, search engines are selected followed by the execution of the search itself. At the first decision point the general relevance of a promising document for this thesis is being assessed regarding the abstract and the title of the respective reference.
If this is not the case (if results are too unspecific) key words and clusters either need to be coupled or modified. The next step is checking if the reference is peer reviewed (does not apply for text books since they are reviewed by the editor), if not the reference is rejected. Then, the number of citations is assessed meaning when a reference is cited more than 20 times it is directly considered suitable. If fewer citations are listed for a particular reference the authority of the first author is checked with regard to academic qualifications and affiliations.

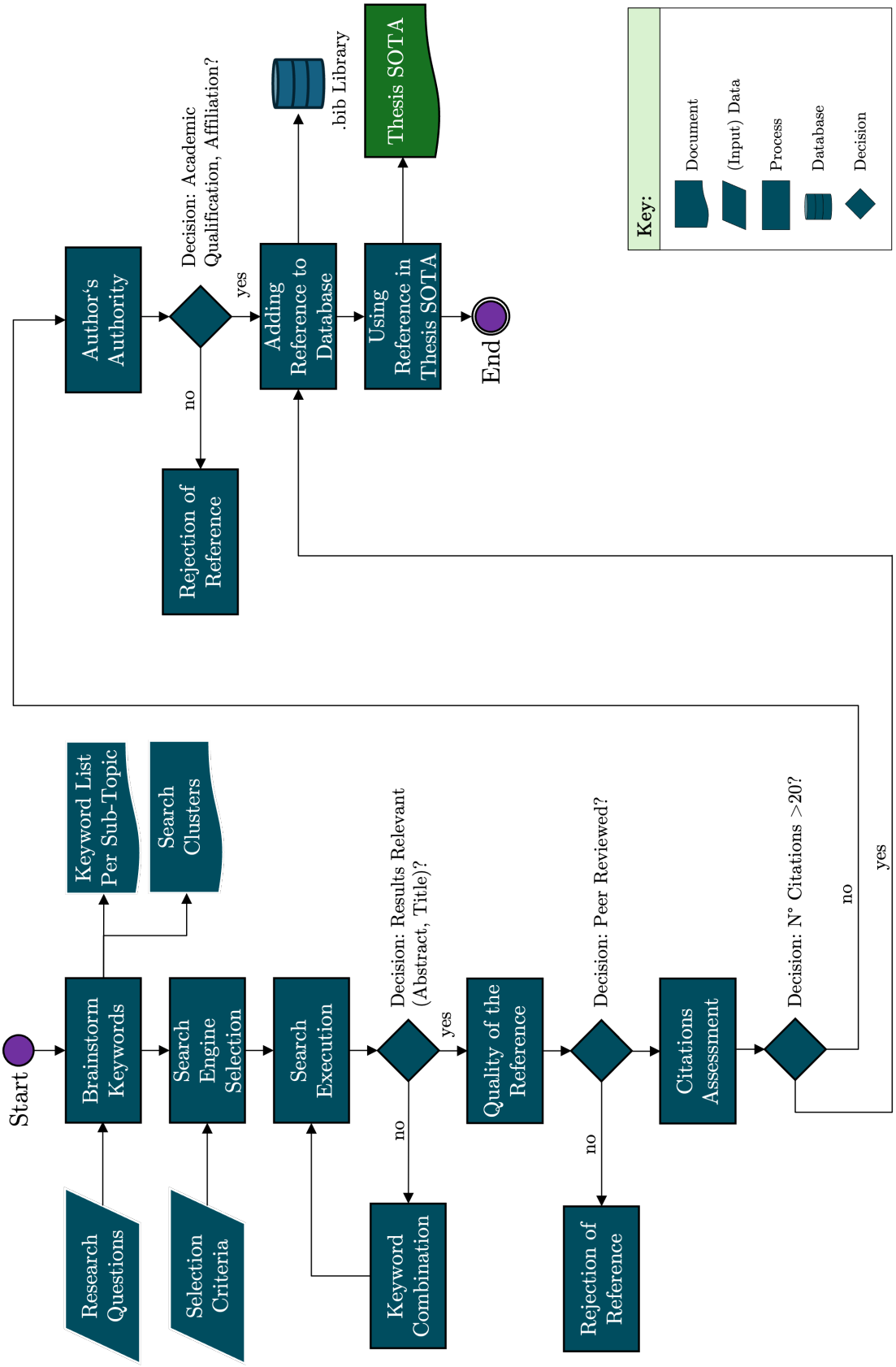

Figure 3.2: Structured process to obtain an elaborate state of the art.

3.3 Fundamentals of Electrics

This section explains the basic concepts and norms of electrics with a focus on the automotive industry. Hence, the most important relationships between physical variables needed for this thesis are established.

3.3.1 Electric Voltage, Current and Resistance

Whenever positive and negative electric charges Q are separated from each other (within an electrically neutral body), an amount of work W_{el} is needed to do so. The relation between this work and the amount of electric charge being separated is called *electric voltage* U, measured in *volts* [32]. The relation is defined in Equation 3.1 below:

$$U = \frac{W_{el}}{Q} \quad [V] \tag{3.1}$$

Where:

U: is the voltage measured in volts [V].
W_{el}: is the work to separate the electric charges measured in joules [J].
Q: is the amount of electric charge measured in coulombs [C].

In electric circuits such as in the power net of vehicles the separation of the electric charges is carried out by voltage sources such as batteries and alternators. The flow of electric charges within an electric circuit from the negative (excess of negative charges) to the positive pole (excess of positive charges) via one or more power consumers striving to equalize the charges is called *electric current* I which is measured in *ampere* [33, 32]. During this flow the power consumers, wiring and other conductors provide a certain resistance R. The relation between the voltage U, the current I and the resistance R is defined by Ohm's law as follows [33, 34]:

$$R = \frac{U}{I} \quad [\Omega] \tag{3.2}$$

Where:

R: is the electric resistance measured in ohm [Ω].
U: is the electric voltage measured in volts [V].
I: is the electric current measured in ampere [A].

3.3.2 Adding the Time Dimension: Electric Power and Charge

Since the current I can also be defined as electric charges that flow per time t by the formula $I = \frac{Q}{t}$ a time component is added implicitly. Using Equation 3.1 and the common definition of *power* which is "work per time" the definition of **electric** power P measured in *watts* is given by the following formula [32]:

$$P = U \cdot I \quad [W] \tag{3.3}$$

Where:

P: is the electric power measured in watts $[W]$.
U: is the voltage measured in volts [V].
I: is the current measured in ampere [A].

On the contrary the integral of the electric current I over the time component t is the electric charge Q, measured in the unit coulombs or also in ampere-seconds [As] [35]. It is thus defined as follows [32]:

$$Q = \int I \, dt \quad [C] \tag{3.4}$$

Where:

Q: is the electric charge measured in coulombs $[C]$.
I: is the current measured in amperes $[A]$.
t: is the time measured in seconds $[s]$.

3.3.3 Automotive Voltage Standards

This thesis works with the so-called *low-voltage* power net of passenger vehicles which needs to be defined since voltage levels in vehicles differ from the definitions in general electrical engineering. In accordance with the VDE e.V. (Verband der Elektrotechnik und Informationstechnik) and the DIN VDE 0105-100 standard, the term *low-voltage* is defined as the range of $(120\,\mathrm{V}, 1500\,\mathrm{V}]$ for direct current (DC) and $(50\,\mathrm{V}, 1000\,\mathrm{V}]$ for alternating current (AC). Voltages exceeding these thresholds are classified as *high-voltage*, whereas voltages less than or equal to 120 V (DC) and 50 V (AC) are designated as *extra-low-voltage* [36].
However, in the automotive industry, there are only two voltage levels which are defined in Table 3.1 [37, 38]. The levels relevant for this dissertation are located in the low-voltage section since they are smaller than 60 V DC. The main power source for the low-voltage power net in the vehicle are the alternator (or generator) for ICE vehicles or the DC/DC converter for EVs [39].

Table 3.1: Automotive voltage level classification for both AC and DC.

Voltage		Classification
AC	DC	low-voltage
≤ 30 V	≤ 60 V	
AC	DC	high-voltage
≤ 1000 V	≤ 1500 V	

3.4 E/E Architectures, ECU Communication Data and Data Collection

In this section the basic concepts of E/E architectures are defined as well as the state of the art of both automotive power nets and data collection in this context.

3.4.1 Common Automotive Networking Technologies

Automotive E/E architectures are complex cyber-physical systems consisting of ECUs and their peripherals (such as actuators and sensors) which are interconnected by data buses so that messages can be exchanged [40]. The vehicle's E/E architecture is central to defining how ECUs are interconnected, thereby governing the efficacy and efficiency of inter-ECU communication. Modern premium passenger cars may contain more than one hundred ECUs, leading to a significant data flow that can reach several gigabytes per hour [40, 41]. This complex network requires advanced networking technologies to handle this extensive data traffic and ensure reliable exchange of messages among the various ECUs on different automotive safety integrity (ASIL) levels[1].
The general behavior of ECUs in modern (passenger) vehicles is typically influenced by various external mechanisms, including self-regulation by the devices themselves, responses to environmental conditions, and inputs from the vehicle's passengers—especially by the driver. These control mechanisms enforce a robust communication structure within the vehicle, enabling the reliable exchange of information between the different components. This is realized and facilitated by several dedicated key networking technologies, such as the Controller Area Network (CAN), Local Interconnect Network (LIN), and FlexRay, which transmit data via messages [43, 44, 45, 46]. Since the three aforementioned bus systems are designed to be used for different control tasks within the vehicle they provide access to important state information of the most of the power consumers expected to carry the information needed to derive their energy consumption.

Controller Area Network (CAN)

CAN is one of the main digital communication technologies between ECUs in vehicles today, applicable for a large amount of use cases including time-critical ones. Initially developed by Bosch and standardized in ISO11898 it is now widely used across OEMs [44]. To implement CAN on the physical layer, ECUs must be equipped with a CAN controller and a transceiver. A message formulated by the CAN controller within the ECU is put onto the CAN bus by the transceiver via a *CAN_high* and a *CAN_low* wire and is broadcast to all participants of the bus, so no dedicated master controller is needed. The *CAN_high* and *CAN_low* lines form a differential pair, where *CAN_high* rises and *CAN_low* falls during data transmission (and vice versa), providing robust, noise-resistant communication by maintaining a stable voltage difference between them. Wiring is realized by a twisted pair cable. The ISO standard also specifies mandatory termination resistances of 120 Ω

[1] Automotive Safety Integrity Level (ASIL)—according to ISO 26262—classifies automotive hazards based on Severity (S), Exposure (E), and Controllability (C). Levels range from QM (lowest) to ASIL D (highest safety requirements) [42].

(cf. CAN topology in Fig. 3.3). A lean wiring using only two cables as well as the high transmission speed are advantages of the CAN bus system [47, 48, 49]. CAN supports data rates up to 1 Mbit/s[2] which makes it a promising data source [50].

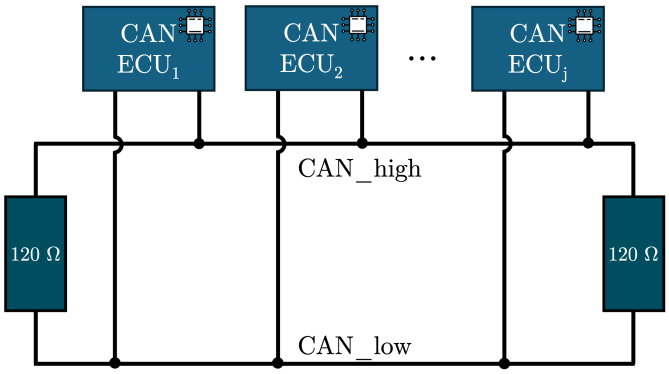

Figure 3.3: CAN bus topology with j bus devices (ECUs) as well as two (mandatory) and two 120 Ω termination resistors (own representation based on [51]).

.

Local Interconnect Network (LIN)

Compared to CAN, the LIN bus system is mostly applied to less time- and data rate-critical applications and where a CAN bus system would be too costly. LIN is designed for simpler, low-cost applications and operates with a single master node that controls communication with multiple slave nodes. The master initiates all data transfers, ensuring synchronized communication among the slaves. LIN uses a non-differential signaling method with a maximum data rate of 20 kbit/s[3], making it ideal for applications like climate control (mostly stepper motors) or window regulators. In contrast to CAN, LIN only uses a single wire connecting all its bus devices [52, 46, 53]. As a result, LIN is also a promising data source for the scope of this thesis since it controls mostly volatile power consumers and so the number of possible ECUs to consider in this project can be enlarged.

FlexRay

A third data bus technology leveraged for this research is FlexRay. Initially developed in the early 2000s by the FlexRay Consortium (cf. the Consortium's logo in Fig. 3.4), an association of OEMs and suppliers such as the Volkswagen AG, the DaimlerChrysler AG (now Mercedes-Benz AG), NXP Semiconductors and Bosch. The FlexRay bus was specifically developed to complement CAN which is limited in speed and not suitable for reliable safety-critical automotive applications requiring timely synchronized communication, such as advanced driver-assistance systems (ADAS), brake-by-wire, and steer-by-wire technologies [54]. Data rates up to 10 Mbit/s and dual channel communication (with then 4 twisted-pair wires instead of two as with CAN) are supported which means that also a higher variety of signals and messages can be transmitted via the data bus compared to CAN or LIN [55].

[2]Megabit: 1 Mbit = 10^6 bits.
[3]Kilobit: 1 kbit = 10^3 bits.

Figure 3.4: Logo of the FlexRay Consortium [56].

Other Networking Technologies

For the sake of completeness automotive ethernet and Media Oriented Systems Transport (MOST) are briefly introduced even though they are not considered in this dissertation. This is due to limitations of computational resources (ethernet) and the absence of logging data (MOST).

Automotive Ethernet: With the rise of digital functionalities such as audio and video broadcasting which has to be distributed throughout the vehicle as well as with the direction to a more service-based design of the in-vehicle communication common technologies like the ones mentioned above reach their limits. To handle this high-volume data, ethernet as the subsequent technology comes into play providing a higher bandwidth [40]. It provides communication speeds of up to 10 Gbit/s[4] and highly precise time synchronization (time-sensitive networking, TSN[5]) for critical applications such as automated driving [58].

Media Oriented Systems Transport: The MOST data transmission system, defined in ISO 21806 and maintained by the MOST Cooperation[6] is specifically designed for high speed multimedia applications in vehicles [59]. The communication focuses on relations between infotainment ECUs and can reach bandwidths of up to 150 Mbit/s for MOST150 divided into different channels [59, 60].

3.4.2 Automotive Communication Data

On the aforementioned data buses, information of various types is exchanged between the ECUs. Thus, it is crucial to be aware of what kinds of data types are prevalent to properly conduct data (pre-)processing in order to obtain adequate prediction results. The bus data can be classified into four main categories based on their nature and method of acquisition (categorization inspired by [27]).

[4]Gigabit: 1 Gbit = 10^9 bits.

[5]TSN is part of the IEEE Standard 802.1 family [57], last accessed: January 15, 2025.

[6]See also: `https://www.mostcooperation.com/specifications`, last accessed: January 17, 2025.

Quantitative Data:
Quantitative data in the automotive communication context can be divided into two subcategories:

- *Discrete Data:* This includes, for example, engine revolutions per minute (RPM), the number of passengers in the vehicle, the vehicle's speed at a given moment, and other numeric data with a fixed (limited) resolution.
- *Continuous Data:* This category comprises analog signals recorded during vehicle operation such as temperature or pressure sensor values. Every numeric value within the respective range can be adopted.

Categorical Data:
Categorical data is classified based on the attribute type. There are the following two attribute types present in the in-vehicle communication:

- *Nominal Data:* An example is the state of ambient light color, which is measured in discrete categories (e.g. "red", "green", "blue", etc.) neither an order nor a ranking between them can be established.
- *Ordinal Data:* This includes variables such as on/off states or the stages of the seat heating, which have a ranking order (a higher seat heating stage means a higher intensity).

Event Data:
Event data is collected to monitor specific actions or reactions within the system at specific points in time, such as key presses and threshold crossings that may indicate potential deviations or anomalies.

Diagnostic Data:
In addition to the aforementioned data types, bus-specific messages are collected for system diagnosis and troubleshooting (fault and status information). This includes checksums, clock signals, and sequence counters that ensure the integrity and proper functioning of the bus system itself.

3.4.3 Automotive Power Nets and On-Board Power Supply

Every ECU and all other power consumers in the vehicle need to be supplied with electrical power in order to function properly. Therefore, there is—next to the communication architecture consisting of the data buses (cf. Section 3.4.1)—also a complex power distribution network present in the vehicle which is also called the *power net*. In modern vehicles there can exist several voltage levels at the same time to feed different kinds of power consumers. However, this dissertation focuses solely on the low-voltage power net with the nominal voltage levels of 12 V respectively 13,5 V (cf. Table 3.1, top row).
Fig. 3.5 depicts a schematic representation of a simple structure of the vehicle electrical system: essential to a power net is the power source which is an alternator in ICE cars and a DC/DC converter in EVs which converts from high to low-voltage. In most of the cases the power source feeds a 12 V battery that serves as a buffer and for smoothing out voltage peaks and ripple [61]. Subsequently, the power is distributed via several fuse boxes (here: depicted as a single fuse F_1) which secure

the power net from overload and—in safety critical-cases—ensure the freedom of interference [42, 49]. In modern vehicles, melting fuses are used almost exclusively. In melting fuses a small metal wire disintegrates in the event of a persisting overload, so that the affected electric circuit is opened mechanically. The design of the flat plug melting fuse developed in 1976, which is used for currents of up to 30 amperes, is standardized in accordance with DIN 72581/3C [62] which also comprises standardized dimensions according to the nominal current. Figure 3.6 depicts the essential components of the flat melting fuse as found in most motor vehicles.
As the degree of branching in the wiring system increases, the nominal current that has to be protected by the fuses decreases, and so does the amount of power consumers behind the fuse. As a result, almost every individual consumer in the vehicle electrical system can be assigned its own fuse, which simplifies fault assignment during troubleshooting (cf. Fig. 3.5) [49, 62].
The power (distribution) networks in combination with the communication architecture eventually define the automotive E/E architecture.

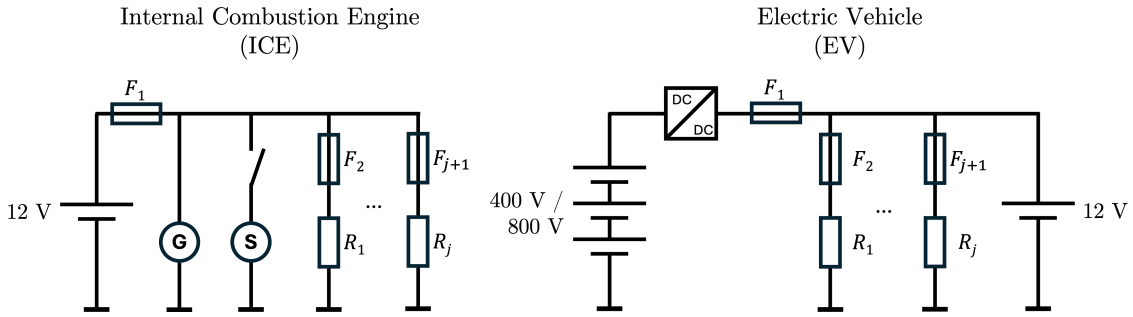

Figure 3.5: Simplified power net architectures for ICE vehicles with a generator (**G**) and starter (**S**) (left) and EVs with a high-voltage system (right), based on [62, 61, 63]. Resistors R_p ($p \in 1, \ldots, j$) represent power consumers, secured by fuses F_p ($p \in 1, \ldots, j+1$). Wiring harness losses are omitted. Voltage differences are indicated by different battery circuit symbols joint in series.

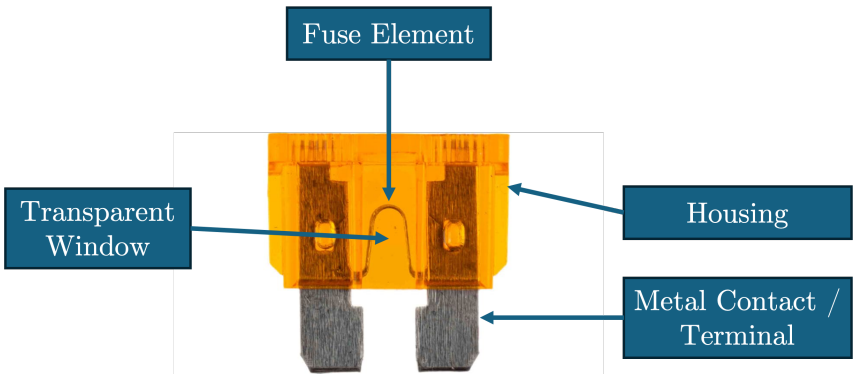

Figure 3.6: Side way illustration of a flat plug melting fuse. In the center, the metallic fuse element is visible which melts when the current is too high.

3.4.4 Current Measurement Techniques

When power net or homologation engineers need to get a closer look on the components and how power consumption evolves during the vehicle's operation they need to measure the current over time during various operational procedures. This can either be a driving cycle e.g. the Worldwide Harmonized Light Vehicles Test Procedure (WLTP) [64] or other research conducted e.g. driving under real road conditions.

Introduction to Shunts

Since accurately capturing current consumption near the power consumer(s) is crucial during training data collection. Shunt resistors, also known as current sense resistors, are commonly employed and are available as a replacement for the melting fuses [65, 66]. A shunt is a precision resistor used to measure electric current by detecting the voltage drop across it [67]. According to Ohm's law, as defined in Equation 3.2, the voltage drop U across the shunt is directly proportional to the current I flowing through it, given the resistance R. By rearranging Equation 3.2, the current can be determined as:

$$I = \frac{U}{R} \tag{3.5}$$

Shunt resistors are widely used in various applications, including battery management systems, automotive power networks, and industrial energy monitoring due to their simplicity and reliability [68]. They are typically designed with low resistance values to minimize power dissipation (own power consumption) while ensuring accurate current sensing [67]. To ensure accurate current measurements proper thermal management is necessary to prevent drift in resistance values due to temperature variations, which can impact measurement accuracy. Therefore, selecting shunt resistors with low temperature coefficients is essential for maintaining measurement precision [66, 68]. In summary, shunt-based current sensing is preferred for its simplicity, reliability, and cost-effectiveness compared to alternative methods such as Hall-effect sensors or current transformers especially in the case of the automotive low-voltage power net it is favorable to replace the melting fuses with equal shaped shunts [69]. However, for this study, careful consideration of shunt placement and thermal effects is necessary to ensure accurate and stable measurements and thus a precise data collection.
Since one flat melting fuse approximately corresponds to one power consumer, it makes sense to replace them with shunts and record the measured data per ECU. Companies like IPETronik have specialized in such measurement equipment offering high precision shunts which meet even certification (homologation) requirements.

Current Measurement Hardware Used for This Research

The shunts used for this research are supplied by the aforementioned company IPETronik. Their *IPEshunt 1* is a high-precision current sensor designed for automotive applications, capable of monitoring both operational and quiescent currents directly at the vehicle's fuse holder. Available in current ranges of ±5 mA, ±1 A, ±5 A, ±10 A, and ±30 A, it converts current into a proportional voltage signal for data acquisition according to Ohm's law.

The output sensitivities vary depending on the current range:

- Quiescent current (±5 mA): 20 mV/mA with an output range of 0 to 0.1 V.
- Operational current (±1 A): 1 V/A with an output range of 0 to 1.0 V.
- Operational current (±5 A): 0.2 V/A with an output range of 0 to 1.0 V.
- Operational current (±10 A): 0.1V/A with an output range of 0 to 1.0 V.
- Operational current (±30 A): 0.033 V/A with an output range of 0 to 1.0 V.

The sensors used in this research cover the entire product range, except for the quiescent current sensors, which are not employed here, as ECUs with different consumption characteristics—and thus magnitudes—are investigated. All of the shunts offer an overload protection up to 1.5 times their nominal current for short durations (maximum of 1 second) as the safety for the vehicle must be guaranteed since the original melting fuse is replaced. The integrated overload protection of the shunts therefore ensures the safety of the vehicle during test drive operations.
The *IPEshunt 1* maintains high accuracy, with a ±1 % deviation at 25 °C, and is designed to operate within a temperature range of -40 °C to 85 °C. This makes the component suitable for data collection in this research (cf. Section 4.1.1) [66]. Figure 3.7 shows a photograph of the shunt which has the exact shape of the original melting fuse.

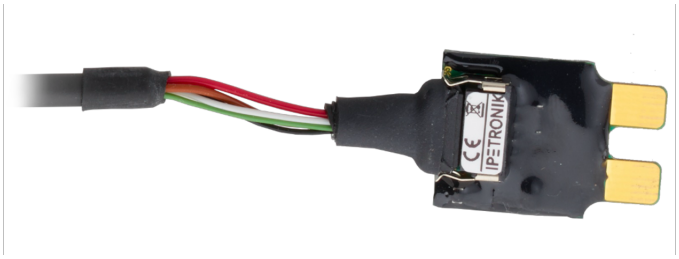

Figure 3.7: Photograph of the *IPEshunt 1*—eventually replacing the vehicle's melting fuses (image source: [70]).

3.4.5 Measurement System and Logging Technology

Measurement System Architecture

As relevant components for current measurement have been introduced, the measurement and logging system as depicted in Figure 3.8 can be put together and adapted to this project's needs [27]. Shunts are in use to collect the current data from the power consumers. Their voltage signal can then be passed to an analog-to-digital (A/D) converter and then be put on a dedicated measurement CAN bus which is fed to the general data logger together with all other data buses from the vehicle which include LIN, CAN and FlexRay buses. The A/D converter used is also supplied by the company IPETronik to match with the shunt hardware. More precisely the *MSENS-2* A/D converter module is utilized in this research. It is a 4-channel analog measurement module and supports both voltage and current inputs, making it suitable for various automotive applications. It is capable of providing sensor power supplies up to 15 V with currents up to ±60 mA. The *MSENS-2* module operates within a temperature range of -40 °C to 125 °C and transmits measurement data via its CAN interface with configurable data rates up to 1 Mbit/s. Its accuracy at 25 °C is: ±0.05 % for bipolar and ±0.13 % for unipolar voltage ranges [71].
For logging, the incoming the data a BLUEPIRAT Rapid logger is used which is an advanced data logger developed by MAGNA Telemotive GmbH, designed primarily for high-performance logging in automotive environments. Within the data logger the signals are synchronized in time and saved in the attached memory to be retrieved after a test drive. The sampling rate can be set manually between frequencies of 1 Hz and 4 kHz [72]. The logger provides dedicated measurement files in a binary logging file, also called *.blf* file which can be downloaded, extracted and converted with a dedicated software [72].

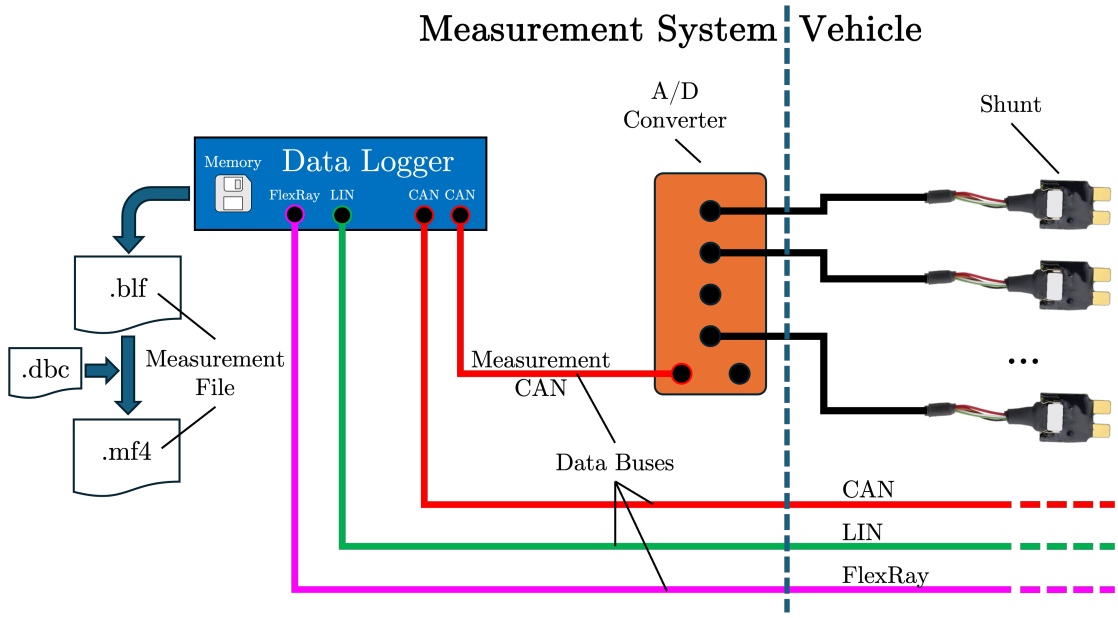

Figure 3.8: Measurement system for electric current and network traffic logging. The data logger ensures time synchronization of the recorded measurements, a fixed sampling rate and the creation of measurement files in a binary logging format (*.blf*).

Data Export and Measurement File Generation

The *.blf* file provides the hexadecimal values of the measurement channels configured in the data logger. However, it cannot be interpreted directly since the information on the physical values and the signal names are missing. These are added by the conversion of BLF to the so-called measurement data file (*.mdf*) or—in version 4—*.mf4* using a library providing the necessary metadata information [73]. The library—also called database (*.dbc* file)—provides bus specific information on how to translate the raw values into meaningful and human interpretable information. This includes information about bus messages, signals, and their properties, such as data type, range, and physical units. The conversion process is also depicted in Figure 3.8, however it requires external software such as vSignalyzer from Vector Informatik GmbH [74, 75, 73].
In general, BLF is optimized for storing bus data, offering high-speed logging and compact storage. MF4, a version of the MDF standard, is more versatile, supporting complex datasets as well as compression. In contrast, ASCII for example is a plain-text format that, despite being human-readable, is less efficient and scalable, making it less suitable for handling large-scale, high-speed data logging in real-time systems as it is required in this research [73, 76].
Hence, the conversion from the raw data to the more meaningful enriched MDF format becomes necessary for further processing the information as input for ML model training since more information is added to the data through the conversion process. This ensures the interpretability of the investigated models.

3.5 Machine Learning and Data Science Fundamentals

This section introduces the basics of ML and data science with the most important algorithms and methods needed to tackle the underlying research project.

3.5.1 Artificial Intelligence and Machine Learning

The Non-Trivial Definition of AI

Until today there is no universally valid and recognized definition of AI and even, Alan Turing only defines the so-called *Turing Test* (instead of AI directly) which is a measure of a machine's ability to exhibit intelligent behavior indistinguishable from that of a human in his seminal works on that subject [77]. In the scientific literature, a variety of definitions of AI can be found. Kreutzer and Sirrenberg define AI in a way that is particularly relevant to this research project: *"Artificial Intelligence refers to the ability of a machine to perform cognitive tasks that we associate with the human mind"* [78, p. 123]. According to this definition, AI serves as an umbrella term encompassing model building (i.e. the abstraction of reality), machine learning (i.e. the generation of knowledge through experience), and deep learning (DL). (cf. Fig. 3.9). Using this methodology, the cognitive task of extracting information from the recordings of in-vehicle bus communication with the help of a "machine" is accomplished in this project, as humans are unable to do so due to the high frequency and number of signals (data points) available.

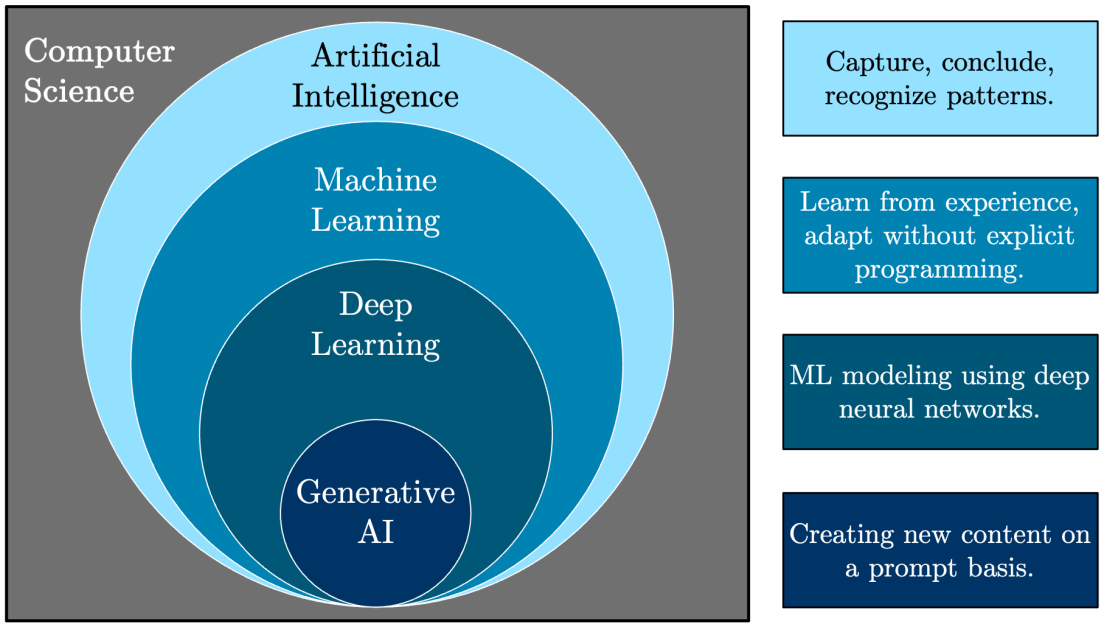

Figure 3.9: The different and nested scopes of ML, DL and GenAI as subdisciplines of computer science.

Stepping down from AI as the umbrella term, DL is considered a subfield of ML and involves artificial neural networks (ANNs, cf. Section 3.5.3) that have more than one logical connection between the input and output of the model (multilayered model). They are constructed after the functioning of the human brain where neurons are interconnected with each other through synapses. For this reason, the term "machine learning" will subsequently be used as an umbrella term for both subfields of AI.
Additionally, with the rise of powerful chatbots such as OpenAI's *ChatGPT*[7] and other applications that are capable of generating prompt-based new (previously unseen) output such as texts or images, generative AI (also called GenAI) plays a vital role as a sub-field of DL [79]. The term *GPT* stands for *generative pre-trained transformer* which is an application of a deep neural networks for generative AI and shall be mentioned here for the sake of completeness and will not be further used for this research project [80]. Hence, the focus of the present work lies on "classic" ML tasks such as predicting an output based on a given input with the help of a pre-trained (regression) model.

3.5.2 Introduction to Supervised Machine Learning

For ML model training one can distinguish between supervised and unsupervised learning. Supervised learning involves training models on labeled data where the correct output is provided together with the corresponding input data during the training process. On the contrary, unsupervised learning involves discovering patterns or structures in unlabeled data without explicit guidance on what the correct output should be [81]. Hereafter, only supervised learning is discussed in greater detail since it is the suitable procedure used for the underlying research project of this thesis since the bus trace data is labeled in this project.

[7] `https://chatgpt.com`, last accessed: December 17, 2024.

Basic Principles of Supervised Learning

Expressed mathematically, in ML tasks "learning" refers to fitting a curve to a given set of data points in an multi-dimensional space. The dimensionality refers to the number of input (independent) variables as well as datapoints available to make a prediction (cf. Equation 3.6). The resulting curve is subjected to minimize a so-called *loss function* which defines the quality of a fit to the given set of data points [82].
When the combination of input—the so-called *features*—and output data—the so-called *target*—is already available during the training phase the algorithm constructs (fits) a function that maps input data to output data based on these labeled examples best. ML algorithms then have the goal to minimize the error between predicted and the corresponding actual outputs across the labeled dataset [83].

Introduction of a Mathematical Notation

A variety of sources describe ML tasks with a similar mathematical notation i.e. [84, 85] and [86]. According to them the construction of the aforementioned fitting function can be described as in the following mapping notation:

$$f : X \to Y \tag{3.6}$$

Where:

X	represents the input data (features) in $\mathbb{R}^{m \times n}$.
Y	represents the output labels (target values) in $\mathbb{R}^{m \times q}$.
n	represents the number of features.
m	represents the number of data points.
q	represents the number of targets considered simultaneously.

Given a dataset of labeled samples $\{(x_i, y_i)\}_{i=1}^{m}$, where $x_i \in X$ and $y_i \in Y$, the goal of supervised learning is to minimize the error between the predicted and actual outputs by optimizing a model. The training dataset consists of labeled pairs (x_i, y_i). Here, x_i is the feature vector for the i-th data point in the dataset, represented as $x_i \in \mathbb{R}^n$, containing n feature values corresponding to the n columns of the feature matrix X. The corresponding label or target for the i-th data point is $y_i \in \mathbb{R}^q$, where q is the number of target variables. The dataset as a whole is structured as follows: $X \in \mathbb{R}^{m \times n}$ is the feature matrix with m rows and n columns, where each row represents a data point, and each column represents a specific feature. Similarly, $Y \in \mathbb{R}^{m \times q}$ is the target matrix with m rows and q columns, where each row corresponds to the target variables for a data point. In this context, m denotes the number of data points (rows in X and Y), n denotes the number of features (columns in X), and q denotes the number of target variables (columns in Y). Hence, the formal objective of supervised learning is to estimate a function $f : \mathbb{R}^n \to \mathbb{R}^q$ that maps each feature vector x_i to its corresponding label y_i. This involves learning and adapting the model parameters such that the error between the predicted outputs $\hat{y}_i = f(x_i)$ and the true outputs y_i is minimized, using the loss function $\mathcal{L}$.

The loss function can be defined as $\mathcal{L}(y, f(x))$ and it quantifies the error between the predicted output $f(x) = \hat{y}$ and the actual label y. The model's internal parameter set β is optimized using techniques like gradient descent (especially weights in the case of Neural Networks (NNs), cf. Section 3.5.3) or by optimizing a dedicated loss

function [87]. There are multiple loss functions that can be used such as the mean squared error (MSE, cf. Section 3.5.4). A loss function can be chosen according to the individual ML tasks and needs of a project [87, 85].

$$\mathcal{L}(y, f(x)) = \frac{1}{m} \sum_{i=1}^{m} (y_i - f(x_i))^2 \tag{3.7}$$

Where:

m	represents the number of data points.
y_i	represents the label or target for the i-th data point.
$f(x_i)$	represents the predicted output for the i-th data point.

In the following, the prediction is referred to as $\hat{y}$ and the actual label as y for the sake of simplicity and to be aligned with common notations in literature [88, 80, 83].

Regression vs. Classification Tasks

As mentioned in the state of the art assessment strategy (cf. Section 3.1), this dissertation focuses on regression tasks. The main differentiation between regression and classification tasks is the output type for a given set of input features X. For regression the output of the ML model is a continuous numerical value ($y \in \mathbb{R}$) as for example the prediction of power consumption for ECUs. For classification however, the main task is to assign a class label y to the given input X as for example the categorization of images into the two classes "cat" and "dog" [85, 80].

3.5.3 Data Analysis Methods and Algorithms

Due to the high complexity of the data in ML projects and especially with the large amounts of bus communication data in vehicles it is particularly important to follow a standardized, structured and validated process to ensure high output quality and consistency during an ML project's life cycle [89]. A common feature of many process models is the application of ML algorithms to some extent in solving the given problem. Those relevant for the further course of this thesis are outlined in this section.

Machine Learning Methodology and Process Models

Data science projects are often complex in terms of both the amount of data involved and the skillset required of the personnel responsible for applying the methodologies and interpreting the output. Hence, especially on an enterprise level it is important to have a standardized framework in place that guides the creation, implementation, and management of productive AI or ML systems. Therefore, several dedicated process models have been developed and continuously improved over the recent years to handle differences between AI and conventional software development projects. In their comparative study involving seven process models, Kutzias et al. conclude that none of the current ones is complete or entirely suitable for the complex needs of data science projects, especially not for small and medium enterprises (SMEs) [90].

CRISP-DM: One of the early process models is CRISP-DM tackling this gap (Cross-Industry Standard Process for Data Mining) which is a widely adopted framework for data mining and AI (ML) projects. It follows a structured, iterative process comprising six phases from *Business Understanding* until eventually the *Deployment* of the project takes place.

CRISP-ML(Q): Studer et al. identify the lack of adaption of CRISP-DM to contemporary data science and AI problem statements and extend it by developing CRISP-ML(Q) where *ML* stands for machine learning and *Q* for quality. They take into account the dynamics and impact of decisions derived from an AI project's outcome by including model maintenance and monitoring in the process as lifecycle elements, too. CRISP-ML(Q) is designed to be universally applicable across various industries and applications [91].
Given the extensive and diverse automotive datasets utilized in this research, CRISP-ML(Q) approach is used to shape this research reflected in Chapter 4's structure. The initial phase involves understanding the context and the data simultaneously. This is followed by data preparation, model development, and evaluation. Once the models are deployed, their performance is continuously monitored and validated. The single steps of CRISP-ML(Q) together with an initial mapping to the underlying project's tasks (where evident) is given below:

(i) **Data Collection and Understanding**:
Gather and understand the bus network data and the types of information available.

(ii) **Algorithm Selection**:
Choose an appropriate regression algorithm.

(iii) **Model Generation**:
Train (fit) the ML model. Adapted to the present research:

  (a) Systematically select relevant Electronic Control Units (ECUs) from a specific model series.

  (b) Prepare the data by removing bus signals that have minimal or no impact on the prediction.

  (c) Train the ECU models, validate with cross-validation (CV) and KPIs.

(iv) **Model Quality and (Computational) Efficiency Assessment**:
Evaluate the quality and computational efficiency of the model.

Machine Learning Tooling and Frameworks

The ML algorithms proposed for this research project capable of solving the given ML task (cf. below) are complex. A large number of mathematical operations need to be computed to obtain the final results, making manual implementation tedious and impractical [83]. As a result, a wide range of ML tools and frameworks have been developed to efficiently encapsulate these implementations, providing standardized application programming interfaces (APIs) for streamlined model development. Among these, TensorFlow (with the Keras API) and scikit-learn are two of the most

widely used frameworks, offering optimized implementations of ML operations that enable efficient training, evaluation, and deployment.
TensorFlow, with its high-level Keras API, simplifies DL model construction through an intuitive interface and support for graphics processing unit (GPU) acceleration, making it particularly efficient for training large-scale NNs. On the other hand, scikit-learn provides a comprehensive collection of classical ML algorithms, including regression, classification, clustering, as well as hyperparameter tuning, making it a powerful toolkit for supervised and unsupervised learning tasks [92]. Therefore, scikit-learn's popularity originates from several key advantages: a seamless integration within the Python ecosystem, a comprehensive API enabling switching between algorithms with little effort. Additionally, its extensive community support comes along with a complete documentation, and community-based quality assurance, making it a promising choice. [93, 92].
Both frameworks support model deployment through APIs such as ONNX for interoperability [94]. While TensorFlow excels in DL applications, scikit-learn remains the preferred choice for traditional ML tasks due to its efficiency and broad applicability across various domains—including regression. These tools reduce the need for implementing the aforementioned mathematical operations manually, allowing to focus on hyperparameter tuning, model interpretability, and real-world application development which are crucial tasks around the actual execution of the respective algorithms [95].
In the following selected ML techniques and modeling algorithms are presented together with the chosen implementation and parametrization in either scikit-learn or TensorFlow Keras.

Regression Analysis Methods

As already discussed in Sections 3.1 and 3.5.2 regression is a technique used in statistics, ML and data analysis to model and understand the relationship between one or more independent variables (features) and a dependent variable (target). The goal of regression is to predict the numerical value of the dependent variable based on the values of the independent variables [96]. The simplest case is the linear regression with one feature variable $X \in R^{m \times n}$ and one target variable $y \in R^{m \times 1}$ assumed to have a linear relationship. This relationship can then be approximated by the regression curve as shown in Fig. 3.10 (own representation using the Python scikit-learn library version 1.5.2[8]).

This trivial regression function can be defined as in Equation 3.8. The goal is to find an optimal intercept and slope of the linear curve which in combination minimize an error measure.

$$y = \beta_0 + \beta_1 \cdot x \tag{3.8}$$

Where:

- y is the target (dependent) variable.
- x is the feature (independent) variable.
- β_0 is the intercept, determined during model training (fitting).
- β_1 is the slope of the line, also determined during model training (fitting).

[8] `https://scikit-learn.org/dev/whats_new/v1.2.html`, last accessed: October 22, 2024.

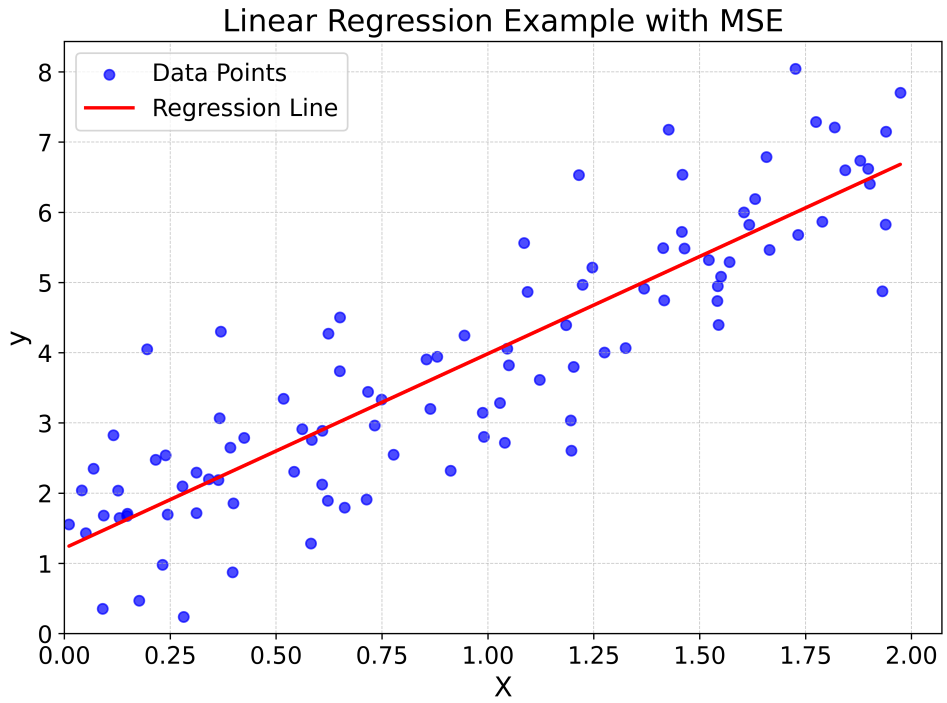

Figure 3.10: Simple linear regression using 100 random data points having a linear relationship with noise added to it. The loss function used is MSE resulting in a MSE value of 0.8066 in this example.

However, for more complex data such as the one dealt with in this project linear regression is not capable enough. Therefore, multiple regression with more than one feature variable is necessary [97]. When dealing with the prediction of multiple target variables, multi-target regression is required [25].

Tree-based Algorithms

Another way to conduct predictions for either regression or classification tasks is the use of decision trees. Tree-based methods have gained significant popularity in supervised machine learning due to their ability to efficiently classify and predict both continuous and categorical variables, often with comparably fast evaluation times [98]. Most decision tree algorithms can handle multivariate input data and can predict both single and multitarget outputs, providing benefits, such as the ease of interpretation and the ability to handle incomplete data (missing values) [99, 25]. Fig 3.11 shows a simple tree-like visualization of possible decisions, whether to buy a classic or a family car. Based on the criteria *budget*, *purpose* and *maintenance costs* decisions can be made to eventually classify which type of car to buy. The decisions are made sequentially starting at the root node passing through the tree's inner nodes until a leaf node is reached which means that no further decision can be made at that point [100]. In this example the tree has a *depth* of 2 which means there are two edges between the root and the leaf node [101]. In the case of single-target regression each leaf node represents a numeric value that represents a single possible prediction $\hat{y}$ instead of a class [102].

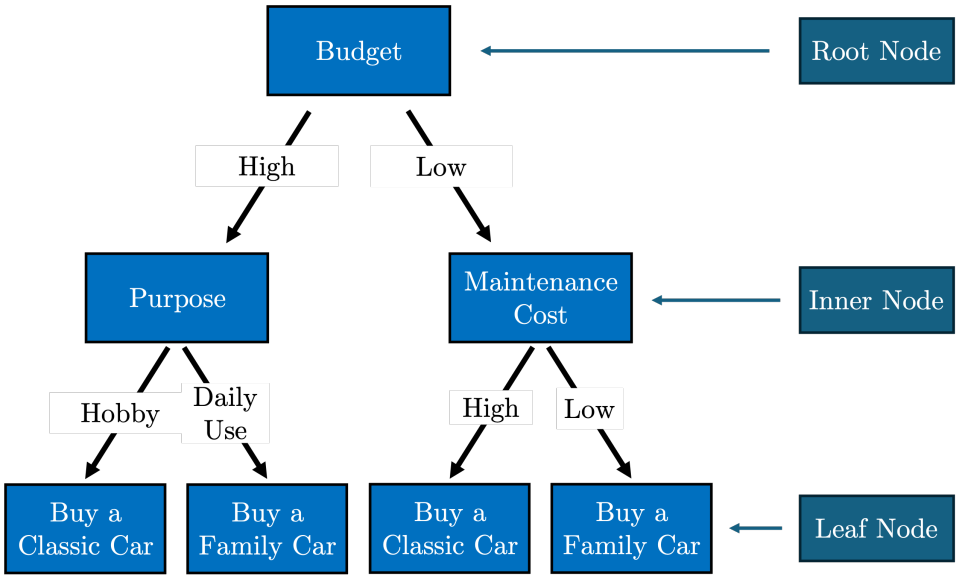

Figure 3.11: Simple decision tree on what car to buy. This example tree has a depth of two and four leaf nodes representing the final decisions.

Simple Decision Trees: One highly cited algorithm commonly used to induce (train) single decision trees is CART (Classification and Regression Trees) developed by Leo Breiman in 1984, which is a decision tree algorithm that can be employed for both classification and regression tasks. The main concept of CART is to iteratively split the data into smaller subsets that show greater homogeneity concerning the target variable, ultimately forming a tree-like structure. CART splits the (sub-)dataset into two child nodes based on minimizing an impurity measure for classification or a predefined error (or loss) for regression. In the case of regression, CART uses the MSE as the loss function (cf. Equation 3.7) which is also the default metric of the *scikit-learn* framework and library used for this research [103, 92].

Ensemble Methods 1 - Random Forest: Since a single decision tree tends to overfit the training data and is prone to noise, ensemble methods have been developed to make this ML technique more robust, improve generalization, and—for regression trees—reduce variance [103].
Random Forests (RF) are an ensemble learning method that enhances the performance and robustness of decision trees by combining multiple trees. This involves drawing multiple bootstrap samples from the training data and the training of a separate decision tree on each sample[9] [102, 104]. Additionally, during the construction of each tree, only a random subset of features is considered for splitting nodes, which introduces extra randomness and reduces the correlation between trees which—again—increases robustness. The final prediction is made by aggregating the predictions of all trees, by using majority voting for classification and averaging for regression tasks [105]. The *RandomForestRegressor* class from scikit-learn is an implementation of the random forest regression algorithm used in the further course

[9]Bootstrapping is a statistical technique of resampling with replacement from a dataset to create multiple new samples which are themselves representative for the actual population.

of this research[10].
This algorithm induces multiple decision trees during the training phase and delivers the average prediction of the individual trees minimizing overfitting and variance [103]. Key hyperparameters play a vital role in tuning the performance of the `RandomForestRegressor` (cf. Listing 3.1). The `n_estimators` parameter determines the number of trees in the ensemble, where increasing this value typically enhances performance but also raises computational performance demands. The `max_depth` parameter sets an upper limit for the tree depth, helping to prevent overfitting by restricting tree growth (cf. Fig. 3.11 above). The `min_samples_split` and `min_samples_leaf` parameters specify the minimum number of samples needed to split an internal node and to constitute a leaf node, respectively. The `max_features` parameter dictates the proportion of features considered for each split, introducing randomness to reduce tree correlation. Furthermore, the `random_state` parameter controls the randomness in bootstrapping and feature selection, ensuring reproducibility across different runs. The `max_leaf_nodes` parameter limits the number of leaf nodes in each tree whereas `bootstrap` decides whether bootstrap samples are used when building trees [92].

```
from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(
    n_estimators=100,
    criterion='squared_error',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features=1.0,
    max_leaf_nodes=None,
    bootstrap=True,
    random_state=None,
    [...]
)
```

Listing 3.1: Constructor of the `RandomForestRegressor` instance in Python using key parameters (some have been omitted for brevity).

Ensemble Methods 2 - Gradient Boosting: Gradient boosting (GB) can be seen as a type of gradient-based optimization, a concept that was explored by Mason et al. [106] prior to Friedman's seminal framework published in 2001 [107]. Unlike random forests, which build (induce) trees independently and which average their predictions, gradient boosting sequentially adds new trees to the overall model that specifically target and correct the residual errors made by the previously combined ensemble of learners [108]. Scikit-learn's `GradientBoostingRegressor` [92] is one such implementation for regression tasks which is also used in this research project. Generally, GB operates by iteratively fitting new models to minimize a differentiable loss function $\mathcal{L}$. Each new tree is trained on the negative gradient of the loss function with respect to the model's predictions, effectively performing gradient

[10] https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.RandomForestRegressor.html, last accessed: October 22, 2024.

descent. This approach allows the algorithm to identify and correct patterns of errors in the predictions of the existing ensemble [107]. By tuning parameters such as the number of boosting stages (`n_estimators`), the learning rate (`learning_rate`), and the subsampling of the training set (`subsample`), overfitting can be controlled and thus be limited. While parameters such as `max_depth`, `min_samples_split`, `min_samples_leaf`, `n_estimators` and `max_features` have already been described in the context of the `RandomForestRegressor` (cf. above). In GB they also determine the same model behavior [92]. In fact, the incremental, gradient-based optimization distinguishes gradient boosting from random forests as well [108]. As depicted in Listing 3.2, the `GradientBoostingRegressor` can be instantiated with a chosen `loss` (scikit-learn default is `'squared_error'` for regression) and a `learning_rate` defining the individual contribution of each additional tree. The parameter `subsample`, when set to a value less than 1.0, introduces stochasticity into the boosting process, acting similarly to a bootstrap sample and helping further reduce overfitting [108, 109].

```
from sklearn.ensemble import GradientBoostingRegressor

regr = GradientBoostingRegressor(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features=None,
    subsample=1.0,
    loss='squared_error',
    [...]
)
```

Listing 3.2: Constructor of the `GradientBoostingRegressor` instance in Python using key parameters (some have been omitted for brevity).

Artificial Neural Networks (ANNs)

Yet another way to induce ML regression models is the use of ANNs. This kind of model intends to imitate the way a (e.g. human) brain is built and learns. The initial concept and fundamental basis of ANNs was first introduced by McCulloch and Pitts in 1943 [110].
To represent knowledge a NN consists of neurons (nerve cells) which are interconnected with synapses—electrochemical interconnections of neurons. In general, once there is an input to a neuron it can be activated and subsequently activate following neurons according to both the connected synapses and the respective activation thresholds [111]. In the case of ANNs, basic neurons are represented by a binary Threshold Logic Unit (TLU) which computes the weighted sum of its inputs $w_1x_1 + w_2x_2 + \cdots + w_Nx_N$ (with N as the number of inputs) and outputs 1 if a threshold is reached and 0 if not.
A more advanced approach is the concept of the perceptron, first introduced by Frank Rosenblatt in 1958 [112, 113]. A perceptron enables training through supervised learning, explained further in Section 3.5.2, which involves adjusting the

weights using a learning algorithm (perceptron learning rule) that compares the predicted output $\hat{y}$ with the actual target output y (cf. Equation 3.9) [83, 114]. Additionally, a bias term b is added to the weighted sum, allowing the model to shift the activation function independently of the input data [114, 111].
Therefore, a single perceptron using this weight update rule can only classify linearly separable data points [114].

$$\Delta w_i = \eta \cdot (y - \hat{y}) \cdot x_i, \quad i \in \{1, \ldots, N\} \tag{3.9}$$

Where:

- w_i is the weight associated with an input feature.
- η is the learning rate.
- y is the actual output label (ground truth).
- $\hat{y}$ is the predicted output.
- x_i is the respective input feature.

Figure 3.12 illustrates a simple perceptron with N input features and a predicted output $\hat{y}$. The output is generated by an activation function $a(\vartheta)$, where $\vartheta = \sum_{i=1}^{n} x_i w_i + b$ is the weighted sum of the inputs plus the bias term. The choice of activation function is critical for the predictive quality of the neural network (or perceptron) [115]. Common activation functions (cf. Fig. 3.13) include the binary step function (used in the TLU), ReLU (Rectified Linear Unit), sigmoid, and hyperbolic tangent (tanh).
The **binary step** function is defined as:

$$a(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{3.10}$$

This function outputs discrete values based on a threshold which is computationally cheap [110]. One of the main disadvantages of this activation function is that it cannot be differentiated for $x = 0$, which is why it cannot be used for NNs [116].

The Rectified Linear Unit (**ReLU**) is defined as:

$$a(x) = \max(0, x) \tag{3.11}$$

It outputs zero for negative inputs and the input itself for positive inputs. Despite being fast, simple and the avoidance of vanishing gradients (they are either 1 or 0) it can suffer from the so-called *dying ReLU* when a neuron stops updating due to the fact of the gradient being 0 [117, 118, 119].

The **sigmoid** function is shown below:

$$a(x) = \frac{1}{1 + e^{-x}} \tag{3.12}$$

Sigmoid maps inputs smoothly to outputs between 0 and 1 using an S-shaped curve. However, it suffers from the vanishing gradient problem (gradient becomes very small for large positive or negative inputs) which can slow down the learning process significantly [120]. Furthermore, it is computationally expensive with regard to the derivative calculation compared to ReLU [121].

The hyperbolic tangent function (**tanh**) is:

$$a(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.13}$$

Tanh also provides an S-shaped curve, but maps inputs to outputs between -1 and 1 [121, 122, 111]. This means it is zero-centered which helps to achieve better gradient updates than sigmoid [123], however it also suffers from the vanishing gradient problem [124].

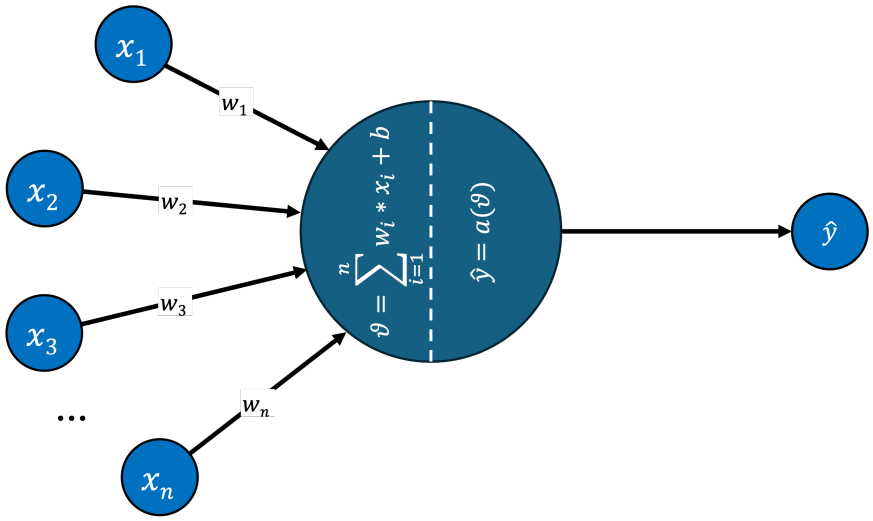

Figure 3.12: Perceptron architecture with n inputs, an activation function $a(\vartheta)$, and a predicted output $\hat{y}$.

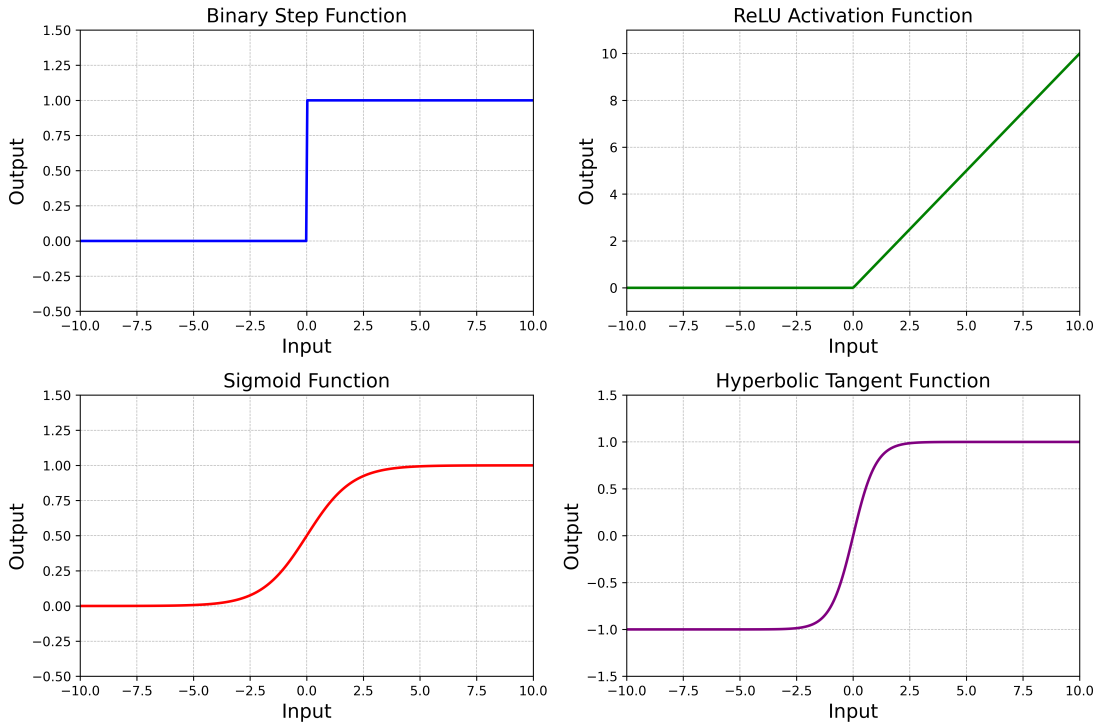

Figure 3.13: Common functions used for the activation of neurons in neural networks.

Multi-Layer Perceptron (MLP): To handle more complex and non-linearly separable data MLP has been developed which combines several perceptrons into a feed-forward neural network (cf. Fig. 3.14) where the input is passed in one direction through the network. This means there is at least one input and one output layer. Optionally, so-called *hidden layers* can be added between the input and output if the complexity of the ML problem is high [125]. The perceptron update rule (cf. Equation 3.9) is no longer applicable due to the concatenation of perceptrons and due to the multitude of weight inter-connections between them.
To update the weights within the MLP the so-called *gradient descent* algorithm must be used since it provides a systematic and recursive way to minimize the loss function, thus improving the network's performance on the given ML task during training. This algorithm involves initializing weights and biases, then iteratively performing forward propagation (passing input data through the network) to compute predictions $\hat{y}$, and calculating the loss by comparing them to the corresponding target labels y using a loss function $\mathcal{L}(y, f(x))$. The gradients of the loss with respect to each weight w and bias b are computed using *backpropagation* [126]. They are then used to update the weights and biases respectively as follows:

$$w_{new} \leftarrow w - \eta \frac{\partial \mathcal{L}(y, f(x))}{\partial w}$$

$$b_{new} \leftarrow b - \eta \frac{\partial \mathcal{L}(y, f(x))}{\partial b}$$

where η is the learning rate, a decisive parameter in finding the optimal weights and biases [122, 83]. This process is recursive, as it involves repeatedly performing forward propagation, loss calculation, backpropagation, and weight updates until the loss converges to a satisfactory level defined by the user.
One thing that must be considered when updating and initializing weights is the phenomenon of the *vanishing gradients* which occurs when gradients in deep neural networks become exceedingly small, hindering effective weight updates during backpropagation and slowing down or even preventing learning [127]. Variants like stochastic gradient descent (SGD) and Adam (also a method for stochastic optimization) can enhance this process by optimizing the update steps [111, 128].
There are several Python libraries that support the construction of MLPs. One of them is also provided by scikit-learn[11] and its constructor is shown in Listing 3.3 below, depicting selected hyperparameters and their default values of scikit-learn version 1.2.1 [92]. The `hidden_layer_sizes` parameter specifies the number of neurons in each hidden layer, defaulting to `(100,)` for one layer with 100 neurons. Multiple layers can be created by providing a tuple, such as `(100, 50)`. The activation function is set via `activation`, with `'relu'` being the default [117]. Other options include `'logistic'`, `'tanh'`, and `'identity'`. Regularization is controlled by `alpha`, applying L2[12] regularization to penalize large weights, with a default value of `0.0001` [92]. The `batch_size` parameter defines the number of samples processed before weight updates, defaulting to `min(200, n_samples)` when set to `'auto'` [92].

[11] https://scikit-learn.org/dev/modules/generated/sklearn.neural_network.MLPRegressor.html, last accessed: October 22, 2024.

[12] The L2 regularization adds a squared penalty term (squared magnitude of all weights w_i) to the model's loss function [108].

Smaller batches may speed up the algorithm's convergence [129]. Learning behavior is adjusted via `learning_rate`, with the default `'constant'` keeping the rate fixed. Alternatives, like `'invscaling'` or `'adaptive'`, allow dynamic adjustments. The initial learning rate is set by `learning_rate_init` (default `0.001`), balancing convergence speed and stability of the training process [92, 130]. Reproducibility is ensured using `random_state`, which controls the random seed for weight initialization and data shuffling [92]. Finally, `early_stopping` can be enabled to interrupt the training process if validation performance stops improving after a set number of iterations, specified by `n_iter_no_change`, with a default of 10 [92].

```
from sklearn.neural_network import MLPRegressor

regr = MLPRegressor(
    hidden_layer_sizes=(100,),
    activation='relu',
    alpha=0.0001,
    batch_size='auto',
    learning_rate='constant',
    learning_rate_init=0.001,
    random_state=None,
    early_stopping=False,
    n_iter_no_change=10,
    [...]
)
```

Listing 3.3: Constructor of the `MLPRegressor` instance in Python using key parameters (some have been omitted for brevity).

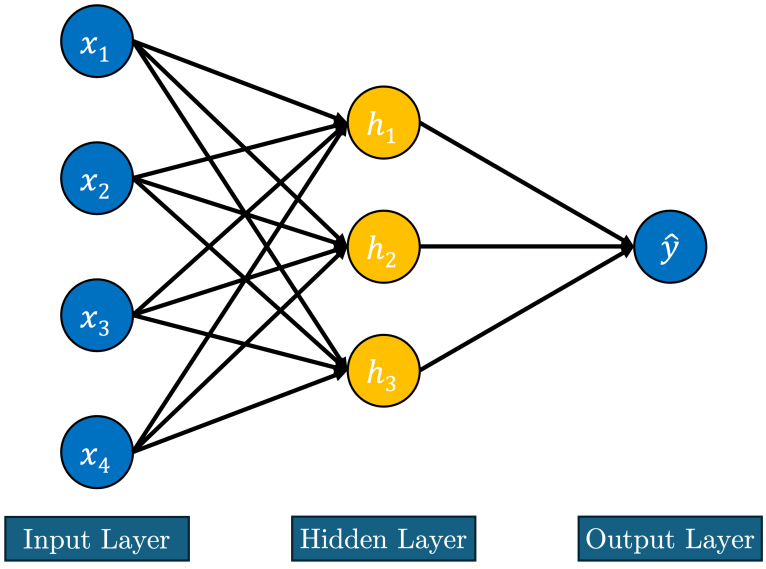

Figure 3.14: Representation of a feed-forward MLP with one input layer capturing four distinct features, one hidden layer with size three (h_1, h_2, h_3) and an output layer for one target variable.

Long Short-Term Memory (LSTM): A major challenge in training recurrent neural networks (RNNs) over long sequences is the vanishing or exploding gradient problem, which makes it difficult for standard RNNs to retain information across extended time spans. This impediment initiated the development of long short-term memory (LSTM) networks, which address these challenges by incorporating mechanisms like special gates to control the information flow (e.g. by controlling input, output and the forgetting of information) making it easier to retain important long-term dependencies while discarding irrelevant data [131, 132]. To capture these long-term temporal dependencies in the data properly, Hochreiter et al. [133] developed LSTMs as a sub-type of RNNs. The difference between RNNs and feed-forward NNs such as MLP is that the output of one cell can be re-inserted into a cell of a previous layer of the network. This means, information does not necessarily flow unidirectionally from input to output but it can also flow backwards through the network allowing it to persist across steps (cf. Fig. 3.15) [131].

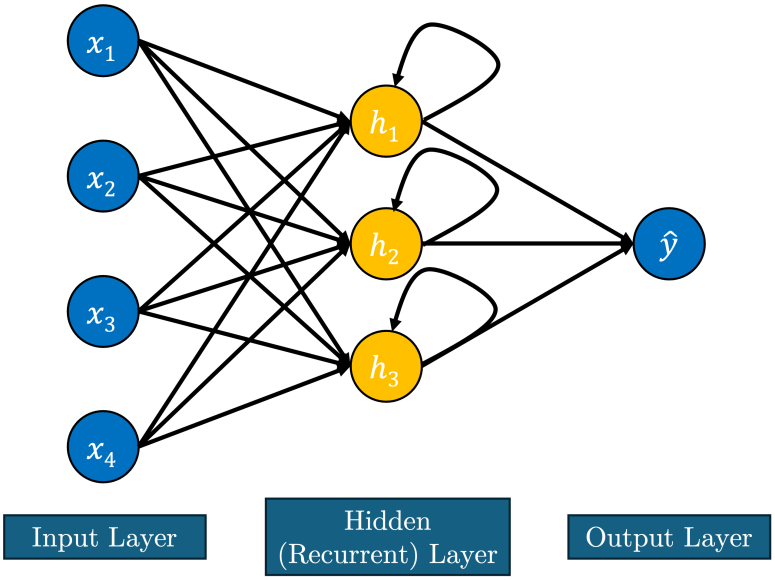

Figure 3.15: Simple RNN with one hidden recurrent layer (h_1, h_2, h_3). In this representation hidden-layer information flows back into its origin cell, also capable of crossing layers (own representation inspired by [94]).

The Tensorflow Keras 3 API[13] provides a convenient way to create LSTM ANN architectures and is used in this research [134]. The constructor and select variables are defined as follows:

```
from tensorflow.keras.layers import LSTM

regr = LSTM(
    units=50,
    activation='tanh',
    recurrent_activation='sigmoid',
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False
    # [...]
)
```

Listing 3.4: Constructor of the `LSTM` layer in Keras with key parameters (some omitted for reasons of clarity).

In Listing 3.4 above, the `LSTM` constructor from TensorFlow's Keras API shows the key parameters. The `units` parameter specifies the number of neurons in the LSTM layer, whereas the `activation` parameter, defaulting to *tanh* (cf. Equation 3.13), defines the activation function within the LSTM cells, while `recurrent_activation` determines the activation function to use for the recurrent step. Its default is sigmoid (cf. Fig. 3.13). The `dropout` parameter, with a default value of 0.0 helps avoid overfitting by randomly setting a given fraction of the input to zero during the training process, similarly to the `recurrent_dropout` parameter, which applies to the recurrent step. This way, not all input units have an effect in each forward (and recurrent) pass [135]. The `return_sequences` parameter, when set to `True`, returns the full sequence of outputs for each input sequence, rather than just the last output. Eventually, the `return_state` parameter, when set to `True`, returns the last state of the LSTM cell in addition to the output itself [134].

3.5.4 Quality Assessment for AI Model Predictions

To enable the objective selection of an adequate ML modeling algorithm among those presented in Section 3.5.3, metrics need to be defined that can compare training and prediction performance across them [136, 137]. Unlike loss functions which are used to train and optimize an ML model, KPIs are used to monitor performance post-training hence they do not need to be differentiable. Given that the prediction of power consumption is composed of continuous numerical values, the following base (unspecific to the respective regression task) KPIs from the state of the art are proposed. However, these require a project-specific adaptation and complementation which will be developed in Section 4.1.6.

[13] `https://keras.io/api/layers/recurrent_layers/lstm`, last accessed: December 15, 2024.

Prediction Quality

Mean Squared Error - MSE: The mean squared error (MSE) is the average of the squared differences between the predicted and actual (here: measured) target labels. For n labelled observations of a validation dataset, the squared difference between the actual value y_i and the predicted value $\hat{y}_i$ MSE is defined as in Equation 3.14 after [80]:

$$MSE = \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{m} \tag{3.14}$$

Coefficient of Determination - R^2: The Coefficient of Determination (R^2) is a measure indicating the proportion of the variance in the dependent variable that is predictable from the independent variables. When the R^2 has a value of 1, this indicates that the regression model fits the data perfectly while an R^2 value of 0 shows that the regression does not explain the variability in the response variable but its average only. Negative values hence indicate that the predictions are even worse than only predicting the average. Mathematically, R^2 is defined in Equation 3.15 [80, 88]:

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{m}(y_i - \overline{y})^2} \tag{3.15}$$

In addition to the already introduced notation: $\overline{y}$ is the mean of all target labels.

Model Speed

Especially when it comes to economically productive applications of ML, time plays a vital role for the success of a project. This includes the speed of training and model inference which is the time to receive predictions from it once an input is ingested [138, 139]. These speeds shall thus be considered for this research especially for research questions *SRQ2* and *SRQ3* (cf. Section 2.2).

3.5.5 Preprocessing of Input Data and Feature Engineering

Another key challenge of large-scale high-dimensional ML projects is data preparation. In addition to preparing the dataset for the ML task a check for completeness must be conducted and missing values, or duplicates be handled and—if needed—features can then be scaled before further processing [140, 141].
Subsequently, the process of *feature engineering* and *selection*—transforming raw data into meaningful features for the ML model—can be both complex and time-consuming especially when the number of available data points m is high [130]. Consequently, selecting the optimal feature engineering techniques to cope ideally with the given data requires not only domain-specific knowledge of the dataset and task but also an understanding of the model's architecture [139, 142]. In general, feature engineering is essential to ensure the interpretability and the accuracy of an ML model which is also why it can be employed not only for numeric regression problems but also in other domains such as e.g for text and multimedia data [143]. The methods can be classified into *feature selection*, *feature creation* and *feature extraction* [143, 130]. In the following, selected methods of all three aforementioned categories as well as for data preprocessing are presented according to [31].

Data Preprocessing

The following data preprocessing methods are considered relevant for this thesis due to the measurement system used to record the considered data.

Handling Missing Values: Whenever data is being recorded by a data logger (cf. Section 3.4.5), it can occur that not every data point is properly saved. Therefore, missing data points can exist in the raw dataset which can mainly be handled by imputing them through filling in missing data points using estimated or calculated values [144].

(i) **Mean Imputation:**
Replaces the missing values with the mean of the remaining observed values for that feature. This approach is useful for numeric data that is symmetrically distributed and has no extreme outliers which could potentially influence the mean. However, it can reduce overall data variance, potentially leading to biased estimates [145].

(ii) **Median Imputation:**
Fills in missing values with the median of the remaining observed values, making it more robust to outliers than mean imputation. Median imputation is suitable for biased data distributions since it stabilizes the general tendency without a significant influence of extreme outliers [146].

(iii) **Zero Imputation:**
Substitutes missing values with 0 (zero), commonly used when zero already signifies a meaningful baseline or absence in the data. This method can introduce bias if missing values are not truly representative of zero but have a different meaning, potentially distorting feature relationships [146].

Feature Scaling: Especially when working with NNs, the input data needs to be scaled between 0 and 1 so that the network's neuron's calculations are not biased by the differences in nominal input values [114, 123]. Therefore, three common scaling methods are presented (in accordance with [25]):

(i) **Min-Max Scaler**:
This scaler transforms features by scaling them to a fixed range, usually $[0, 1]$, which preserves the relationships between data points. It is sensitive to outliers since they can affect the minimum and maximum values significantly.

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{3.16}$$

Where:

X_{min} is the minimum value in X.
X_{max} is the maximum value in X.

(ii) **Robust Scaler**:
This scaler centers and scales data according to percentiles, e.g. the interquartile range (IQR), as per default in the scikit-learn Python implementation[14]), making it less sensitive to outliers compared to the min-max scaler. It is thus useful for data with highly different values.

$$X_{\text{scaled}} = \frac{X - \text{median}(X)}{\text{IQR}(X)} \tag{3.17}$$

Where IQR is the interquartile range, or $Q_3 - Q_1$, representing the spread of the middle 50 % of the data (data points between first quartile (Q_1) and third quartile (Q_3)).

(iii) **Standard Scaler**:
This scaler standardizes feature values by removing their mean and scaling them to unit variance, hence transforming the data to follow a standard normal distribution (mean 0, variance 1). It is effective for algorithms that assume normally distributed data. The standard scaler can be influenced by outliers, too.

$$X_{\text{scaled}} = \frac{X - \mu_X}{\sigma_X} \tag{3.18}$$

Where:

μ_X is the mean of X.
σ_X is the standard deviation of X.

All scalers can be influenced by outliers to a certain degree. If a dataset is susceptible to this, it is advisable to remove it first before applying the scaler.

Feature Selection

Feature selection methods target the reduction of the number of features in the dataset so that model size is reduced and accuracy rises to eventually utilize less memory (RAM) and return more precise predictions [108]. Additionally, a well selected feature set helps to reduce overfitting, thus fostering generalizability of the ML model [147]. Three common effective methods are presented below.

Pearson Correlation: In a dataset, redundancy can occur when features share the same root origin. This means that one feature can be derived from one or more other features. Consequently, there is a correlation between them and thus one can be removed as it does not provide additional information to the model. This can be done by applying the Pearson correlation coefficient (PCC) to each possible pair of features in the dataset to check for correlations [144]. The value range of the PCC reaches from -1 to 1 where -1 means complete negative correlation and +1 complete positive correlation. The closer the correlation coefficient is to 0 the more independent two variables are, the closer to 1 (respectively -1) the higher their dependency (correlation) is [84]. It is then possible to define a correlation threshold

[14] https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.RobustScaler.html, last accessed: October 30, 2024.

above (and below) which one of the two correlated features is removed from the dataset. PCC between features a and b is computed as follows:

$$PCC_{x_a,x_b} = \frac{\sum_{i=1}^{m} (x_{a_i} - \mu_{x_a})(x_{b_i} - \mu_{x_b})}{\sqrt{\sum_{i=1}^{m} (x_{a_i} - \mu_{x_a})^2} \sqrt{\sum_{i=1}^{m} (x_{b_i} - \mu_{x_b})^2}}, \quad a, b \in X;\ a \neq b \tag{3.19}$$

Where:

m	represents the number of data points.
x_{a_i}	represents the value of feature x_a at the i-th data point.
x_{b_i}	represents the value of feature x_b at the i-th data point.
μ_{x_a}	represents the mean of feature x_a.
μ_{x_b}	represents the mean of feature x_b.

Low-Variance Feature Removal: Low-variance features should be removed from the dataset because they provide minimal or no useful information for predictive modeling while unnecessarily increasing model complexity and size. When a feature has nearly the same value across all instances m, it does not support distinguishing between different classes (in classification tasks) or predicting a numeric target variable (in regression tasks), since its contribution to the model is minimal or zero [147]. The variance of a feature a is computed as follows:

$$\sigma_{x_a}^2 = \frac{1}{m} \sum_{i=1}^{m} (x_{a_i} - \mu_{x_a})^2, \quad a \in X \tag{3.20}$$

Low-Importance Feature Removal: Yet another method to reduce the dataset to a minimum number of features required to obtain a maximum accuracy is the removal of features with low importance (cf. also Section 3.6). By setting an importance threshold adequately unimportant features can be filtered and removed from the dataset automatically [148, 147].

Feature Creation

One-Hot Encoding (OHE): One-hot encoding (OHE) is a feature engineering technique used to convert categorical variables into a format that can be provided to ML algorithms to improve predictions. Categorical variables are those that have a fixed number of possible values (categories), such as "red", "green", and "blue" for the ambient light color feature of a vehicle. ML algorithms, especially NNs, require numerical input, however. Hence, categorical data, which consists of labels or categories, cannot be used directly. OHE thus transforms them into a numerical format that can be used effectively by the ML model [149]. The presence of one category state at a given timestamp is then encoded with 0 and 1 respectively, depending on the state. For the ambient light color example the encoding could look like this:

- *Color_Red*
- *Color_Green*
- *Color_Blue*

If at a given point in the dataset the actual feature value is "red," the transformed and encoded features would be:

- *Color_Red*: 1
- *Color_Green*: 0
- *Color_Blue*: 0

The encoder used for this research is the scikit-learn *OneHotEncoder*[15] which automatically detects and encodes categorical features as a one-hot numeric array which can then be treated further in the subsequent steps of the ML pipeline [92]. It has to be considered that OHE counteracts the feature reduction principle carried out in the feature selection and feature extraction steps since it adds new features (in the example above one feature becomes three through the encoding process). It therefore makes sense to conduct (another) feature reduction after the application of OHE.

Feature Extraction

Feature extraction finds meaningful representations of data through algorithms that transform raw data into more informative features, e.g. with dimensionality reduction.

Principal Component Analysis - PCA: Principal Component Analysis (PCA) is a statistical method which is used for the reduction of the input feature dimensionality of ML problems and thus a constitutes a part of feature extraction. It transforms a dataset with potentially correlated variables into a (smaller) set of linearly uncorrelated variables known as principal components. The main objective of PCA is to capture the maximum variance in the data with the fewest number of principal components [144]. These principal components are abstract and do not correspond directly to the original features, making it difficult to interpret the results in the context of the original data [150].

[15] https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.OneHotEncoder.html, last accessed: November 1, 2024.

3.6 Knowledge Extraction with Explainable AI

Despite their advantages, complex ML models often miss the necessary transparency which makes it difficult to reconstruct how they make their predictions or decisions derived from the ingested input data. This opacity necessitates the development of methods to explain the inner functioning of ML models, particularly in critical applications (medicine, military, etc.). Such methods are essential to foster trust, ensure transparency, and comply with legal and ethical requirements [151, 152]. Political initiatives also emphasize the importance of trustworthy AI, as demonstrated by the EU's "Artificial Intelligence Act" (AI Act) [153] which entered into force in August 2024 and the similar "AI Bill of Rights" in the United States [154].
The XAI techniques presented below address both the overarching and detailed perspectives on ML models to draw conclusions about the entire dataset (globally) as well as specific instances or events (locally).

3.6.1 Explainable AI Techniques

During the course of the present research, a survey among engineers of a large German OEM about the usefulness of certain XAI techniques for a similar problem was conducted and published in [29]. The results demonstrate the general suitability of this preselected subset of XAI methods by applying them to a virtual ECU with especially engineered training data and a previously known influence of the latter on the virtual ECU's power consumption [26]. They also allow the selection and application of the following three explainers for this thesis.

Permutation Feature Importance (PFI)

Feature importance—a **global** explainer—assigns a numerical value to each feature based on its impact on a model's output. A higher value indicates a greater influence on the model's predictive capability. Specifically, the model-agnostic permutation feature importance (PFI) method involves randomly shuffling the values of a feature and measuring the subsequent decrease in the model's performance score (here: R^2), while keeping other features unchanged. A larger decrease in the score means a more important feature, indicating a greater impact on the prediction [155]. Additionally, impurity feature importance (IFI) is used for ensembles of decision trees, such as RF or GB. However, it is avoided in this research (it is only used to determine unimportant features during data preprocessing, cf. Section 4.2.2) due to its potential bias when the model overfits [92].
PFI can produce negative values when shuffling the feature values improves the prediction quality by chance, whereas IFI only outputs non-negative values [103].

Accumulated Local Effects (ALE)

The Accumulated Local Effects (ALE) algorithm is an explainer that quantifies the impact of a specific feature on a model's output **globally** while keeping other features constant. ALE plots depict how the model's predictions change as the feature of interest varies, where ALE values indicate the average change in the model's output across the feature's range. The algorithm approximates the partial

derivative of the model output with respect to the feature using finite differences, and then integrates this derivative over the feature's range to obtain the ALE value [156]. ALE can thus be compared to and interpreted as a sensitivity analysis for each feature.

Shapley Additive Explanations (SHAP)

The SHAP (**SH**apley **A**dditive Ex**P**lanations) algorithm generates **local** explanations for individual predictions even made by complex models. This method employs game-theoretic principles to allocate fairly each feature's contribution to the prediction, ensuring properties such as local accuracy, missingness, and consistency, as outlined in [157]. For ensemble tree-based models, an optimized variant known as *TreeSHAP* is utilized [158]. By aggregating explanations across multiple instances, SHAP can also offer a sort of global insight into the model's behavior, although this process is computationally expensive and not considered here. Additionally, *KernelSHAP* provides a model-agnostic approach that approximates SHAP values through weighted linear regression on perturbed samples, while *DeepSHAP* leverages the structure of deep neural networks.

3.6.2 XAI Suitability to the Research Problem

Another research by Mueller et al. shows formally that the aforementioned XAI methods generally work on the regression data used in their research which is similar to the one treated in this dissertation (cf. Section 4.1.1) by both theoretical and practical validation. Additionally, they show that the computation times needed to execute the explainer computation do not contradict their economic application [26].

3.7 Model Evaluation and Optimization

As already mentioned in Section 3.5.3, the hyperparameters influence the predictive performance of an ML model significantly. Moreover, the selection of a suitable ML model is another degree of freedom and a significant step in the CRISP-ML(Q) process model [91]. Therefore, a choice must be made for an appropriate ML algorithm and for an "optimal" set of hyperparameters θ^* (cf. Equation 3.23). Due to the novelty of the research problem, the experimental design carried out in Chapter 4 automated black-box model selection and optimization libraries (AutoML) such as *Auto-WEKA* [159] or *Auto-Sklearn* [160] are not considered any further.

3.7.1 Weighted Sum Analysis for Algorithm Selection

A common method to tackle multi-criteria decision problems in an objective way is the weighted sum analysis (WSA). It is mainly used when choices and decisions must be made based on a range of both qualitative and quantitative objectives with varying degrees of importance, typically in business or engineering [161].

To conduct a WSA, the available decision options $d \in \mathcal{D}$ and the individual decision criteria $c \in \mathcal{C}$ must be outlined first. Subsequently, the decision objectives are broken down into measurable criteria, each assigned a weight $v_c \in \mathcal{V}$ to reflect

its importance [161]. Stakeholders or experts determine these weights, which can for example be done by a pairwise comparison (cf. below).
Each decision option d is then scored based on how well it fulfills each criterion $c \in \mathcal{C}$, with the help of scores s_{c_d}—normalized for consistency. The total utility value U_d of a decision option is calculated as:

$$U_d = \sum_{c \in \mathcal{C}} v_c \cdot s_{c_d}, \quad \forall d \in \mathcal{D} \tag{3.21}$$

This utility value provides a quantitative measure of each alternative's fulfillment of the respective objectives since it relies on the additivity of singular utilities per criterion [162]. The alternative with the highest utility value d_{opt} is selected, since it fulfills the entire weighted criteria best, given the other options. Consequently, d_{opt} can be defined as follows:

$$d_{\text{opt}} = \arg\max_{d \in \mathcal{D}} U_d = \arg\max_{d \in \mathcal{D}} \sum_{c \in \mathcal{C}} v_c \cdot s_{c_d} \tag{3.22}$$

Pairwise Comparison for Criteria Ranking

The weights of the respective decision criteria can be determined by conducting a so-called *pairwise comparison*. In this method each criterion is compared in a pairwise manner to each and every other criterion and it has to be decided if it is more (encoded by 1), less (encoded by -1) or equally important (encoded by 0) than the other one. As a result, a triangular matrix is generated from which the rank and weight of each decision criterion are derived [163]. This process is ideally conducted by stakeholders, subject matter experts, and other decision makers involved in the process and project [161].

3.7.2 Hyperparameter Tuning Strategies

Once the data has been cleaned and the features are engineered and selected, the model training process takes place [91]. Given the high number of hyperparameters available per ML algorithm as well as a specific loss function (cf. Section 3.5.2) a systematic approach on how to find the optimal ones given an individual ML task seems more favorable compared to a tedious manual procedure. Therefore, different hyperparameter optimization strategies are available which can be applied to find a parameter set θ^* [129]. According to [164], the hyperparameter optimization process involves four key components: an estimator (ML algorithm) with its loss function $\mathcal{L}(y, f(x))$, a search space that defines possible configurations, an optimization method for exploring hyperparameter combinations, and an evaluation function to assess the performance of each configuration (e.g. MSE or R^2). Formally, the problem of finding a hyperparameter set that minimizes the loss of an ML model can be defined as follows:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(y, f_\theta(x)) \tag{3.23}$$

In the following, the three optimization methods *grid search*, *random search*, and *bayesian optimization* are outlined in greater detail as they offer a balance of simplicity, effectiveness, and versatility, making them suitable for different types of ML modeling algorithms and datasets—including regression problems [165, 164, 166]. They are depicted in Fig. 3.16.

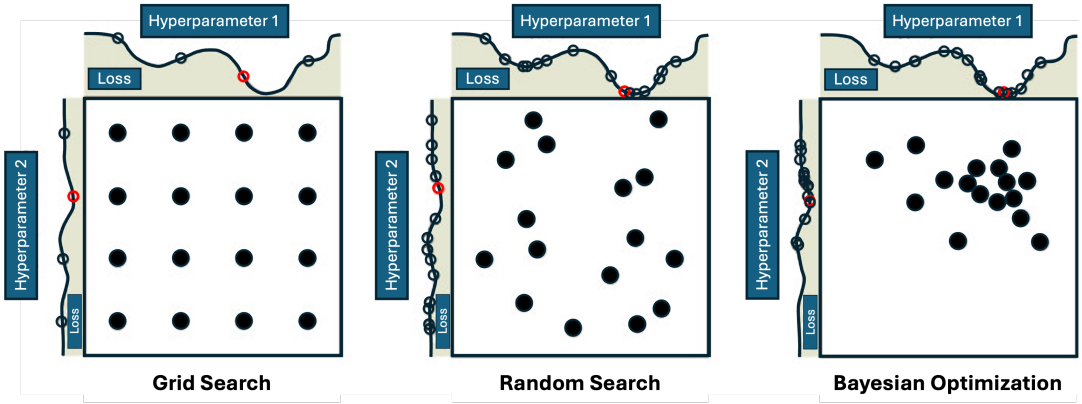

Figure 3.16: Visual representation of the three considered hyperparameter search strategies *grid search*, *random search*, and *bayesian optimization* applied to two hyperparameters with different magnitudes of influence on the loss function. (Figure adapted from [167] and [168]; generously granted by Prof. Yoshua Bengio.)

Grid Search (GS)

Grid search (GS) is a simple baseline hyperparameter optimization strategy to evaluate all possible parameter combinations θ. This means their Cartesian product is calculated. The higher the number of parameters or resolution within the parameters the more function evaluations are needed which makes grid search computationally expensive [165]. Another inefficiency of the GS is that also parameters with a low impact on the predictive performance of the ML model are evaluated (cf. Fig. 3.16, left plot) which wastes computational resources [169], too.

Random Search (RS)

The middle plot of Fig. 3.16 shows parameter combinations generated by the RS. This strategy sets a fixed number of evaluations and randomly selects parameter values from predefined ranges [165]. Bergstra et al. state that for high-dimensional parameter spaces *theta*, RS can outperform GS, as it allows finer searches for influential parameters [168, 166]. With RS, sampling strategies vary by parameter type. For example, the learning rate *eta* is typically drawn from a log-uniform distribution to cover multiple orders of magnitude, while the number of trees in a random forest is sampled from a uniform distribution [168, 129].

Bayesian Optimization (BO)

Bayesian optimization (BO) constructs a probabilistic model to approximate the target function. It then uses this model to select the subsequent set of hyperparameters (θ') to test, balancing exploration of unknown regions with exploitation of promising areas. Acquisition functions, such as *expected improvement* or *entropy*

search—both not expensive to compute—assess the potential benefit of evaluating each point [170]. This approach is especially effective for expensive, non-convex loss functions (with local and global minima), aiming to find the global optimum with fewer trials [171]. In Fig. 3.16, this strategy is depicted by hyperparameter combinations clustering around the minima of the respective losses.

3.8 Related Work

3.8.1 Approaches to Modeling

Complementing the CRISP-ML(Q) approach (cf. Section 3.5.3), Isermann [172] distinguishes between different modeling strategies. According to Isermann, these strategies can be categorized into two main types: theoretical (white box) modeling and experimental (black box) modeling, with hybrid approaches—such as grey box modeling—lying between these two extremes.

Theoretical Modeling

Theoretical modeling is based on mathematically defined natural laws. This involves the application of fundamental constitutive and phenomenological equations, as well as balance equations, entropy balances, and circuit equations. The modeling process typically starts with simplifying assumptions to make the problem tractable. The model parameters are assumed to be known in advance. Given that these models are often sophisticated and detailed, they are usually simplified for practical use, such as through linearization or by reducing the model's order [172]. As an example, the current consumption I of a component can be modeled as a function of various input features $x \in X$ such as the component's operational state, and general model parameters β.

$$I = f(x_{i,j}, \beta) \tag{3.24}$$

The advantage of this approach lies in the deterministic nature of the model, often referred to as a white-box model. However, a notable disadvantage is the complexity of deriving the mathematical equations and the challenge of estimating model parameters, such as through measurement. Additionally, the operational strategies of the component (ECU) must also be considered.

Experimental Modeling

Experimental modeling is based on the creation of models from empirical data obtained through experiments (e.g. measured), observations, or simulations. Common types for experimental modeling include the aforementioned regression models (cf. Section 3.5.2), NNs [111], and more. This approach proves especially beneficial for highly complex systems where theoretical modeling becomes less effective. The recent advancements in computational power and the availability of large datasets have significantly expanded the potential of data-driven experimental modeling. However, one of the challenges of this approach is the tendency for models to overfit (memorize) the training data, as well as the non-trivial issue of ensuring that relevant data for specific model areas is available [172].

Experimental Models for Energy Consumption Prediction

For passenger vehicles, experimental models are predominantly used to predict overall energy consumption (cf. review Table 3.2 below), while in other fields, such as smart buildings, energy consumption forecasting is more common [173, 174].
Data-driven models are currently the most advanced tools for predicting energy consumption accurately. However, their application within the complex power distribution systems of vehicles, particularly based on bus data, has not yet been extensively explored.

3.8.2 Literature Review

In this section, a review of the existing literature in the relevant research field is conducted to enable an alignment of this thesis with current tendencies and research focuses. A comprehensive overview of the reviewed sources is depicted in Table 3.2.
After a thorough analysis of the existing (scientific) literature it can be stated that only little work has been achieved with regard to the detailed prediction of the auxiliary power net components and ECUs of passenger vehicles. However, there is a considerable amount of relevant work that focuses on data-based prediction of the entire energy consumption of vehicles such as [175], [176], [177], [178] or [179] which analyze possible influential factors on the prediction before generating their predictive models which are regressions due to the numeric nature of the general problem. They state that the influential factors are versatile reaching from environmental (e.g. weather) conditions to individual driving behavior (e.g. speed and cruise time) as well as traffic situations.
Schäfers et al. [180] confirm that currently there is little research on a detailed representation of the auxiliaries in (EV) vehicle power consumption prediction and if there is, the power net components are often summarized by either a scalar or simple linear functions depending on time and temperature [181]. Therefore, Schäfers et al. focus on modeling the thermal components such as the PTC[16] auxiliary heater with only a limited amount of information available on a data bus and from external sources using LSTM NNs. [180].
Nonetheless, except for [176] which predict the energy consumption to enhance the function of the battery management system (BMS) most of the sources aim at counteracting range anxiety by more accurately predicting the total power consumption of the vehicle. In addition to their work in the field of propulsion energy prediction Zhou et al. also present a high-level framework to train and obtain AI models from their raw data. It includes feature engineering and data preprocessing methods [182].
Derived from the current state-of-the-art research conducted in the field of energy consumption prediction and modeling at the ECU level, it can be concluded that there is a gap this research work fills as—to the best of the author's knowledge—there is a lack of a holistic approach that covers the entire auxiliary power net and its components as well as adequate explainability of the predictions made in that context.

[16] PTC: Positive temperature coefficient thermistor.

Table 3.2: Comparison of ML methods for predicting EV energy consumption.

Source	ML Model	Quality Metrics	Data Source	Feature Count and Types
[175]	XGBoost	RMSE, MAPE	Real-world driving data from 55 electric taxis in Beijing.	12 features from the vehicle, environment, and driver-related factors.
[176]	Heuristic methods, no traditional ML model	Average error of predicted power requirements.	Real-time vehicle data (speed, acceleration, road information) with historical data.	5 key features: vehicle and road parameters.
[177]	Random Forest, Ridge Regression, k-Nearest Neighbors, MLP	MAE, MSE, R^2, MAPE	Aggregated sensor data from 62 electric trucks.	10 features including mileage, speed, temperature, auxiliaries.
[178]	LSTM-Transformer	MAPE, MAE, RMSE	Real-world vehicle data (one year) and weather data.	20 features from the vehicle and the environment.
[179]	Ordinary Least Squares (OLS) Regression, Multilevel Mixed Effects Regression	R^2, AIC	GPS data from 68 EVs in the Aichi Prefecture, Japan.	14 features including speed, distance, gradient, HVAC usage, temperature.
[180]	Bidirectional LSTM	RMSE, Prediction Error Percentage	Simulation data from thermal management models and real measurements from EVs.	9 features including temperature, humidity, solar irradiance, and cabin preconditioning.
[181]	Multiple Linear Regression (Macro, Hybrid, Micro models)	R^2, Relative Error	Real-world driving data from a Nissan Leaf (2013-2014).	7 features including speed, elevation, temperature, acceleration, auxiliaries.
[182]	Quantile Regression Neural Network	RMSE, and more	Real-world driving data from 55 battery electric taxis in Beijing.	17 features including driving distance, vehicle velocity, acceleration and elevation.

3.9 Conclusion on the State of the Art

Finally, it can be concluded that this state of the art analysis provides a comprehensive overview of the foundational concepts, architectures, data collection methods, analysis techniques, and evaluation strategies relevant for predicting energy consumption in vehicle power nets. These insights can now guide the development of an integrated, explainable ML pipeline tailored to the unique challenges of automotive E/E architectures in the following sections and is used as a foundation to eventually answer the research questions.

4 Empirical Methodology: Design and Results

In the following chapter the state of the art as well as novel approaches are brought together to answer the research questions forming WP 3 of this thesis. Starting with the necessary preparatory work outlined in the preliminaries (cf. Section 4.1) a methodology to obtain data-based energy consumption predictions for selected power consumers is developed in Section 4.2. Subsequently, the methodology is applied to real-world data in order to obtain experimentation results in Section 4.3.

4.1 Preliminaries

Prior to the experiment design—according to CRISP-ML(Q) [91]—data must be collected to train the ML model and it must be analyzed beforehand to identify its influence on the ML model's predictions as well as any prior transformations that may arise which the data must undergo.

4.1.1 Data Collection and Database Generation

It is crucial for data collection that a large variety of the diverse ECU communication network traffic is generated as well as ensuring that as many operation states of the considered power consumers as possible are reflected in the training data (features and target labels). This enables the resulting ML models to react with more precise predictions for a larger variety of driving situations.
Hence, the data collection does not take place in laboratory conditions or in simulations but on actual roads. Therefore, different road types, ambient conditions, speeds and other possibly influential factors are varied during dedicated real-world data collection drives. In addition to that, power consumers which depend on the influence of the driver or passengers must be manipulated manually such as the seat heating, the seat ventilation, the power steering and the ambient light.

Data Collection Mechanism

In preparation of the test drives, two vehicles are equipped with the measurement equipment described in Section 3.4.5.
The first vehicle (thereafter referred to as Vehicle A) is a luxury sedan equipped with a diesel engine. The second vehicle (thereafter referred to as Vehicle B) is an SUV EV with electric propulsion.
The set of vehicles is denoted as $\mathcal{T} = \{VehicleA, VehicleB\}$. Both vehicles are equipped with nearly all available options. The respective individual options lists are shown in Appendix A. Both vehicles are based on the same E/E architecture platform. This means, they share certain E/E components, though it is not investigated which ones exactly. Each available ECU is equipped with a measurement

shunt that collects power consumption data, which is subsequently transmitted via the measurement CAN bus. The logger is configured to record data at a rate of 100 Hz. Table 4.1 provides an overview of the bus types available for each vehicle and lists the number of power measurement points—118 for Vehicle A and 110 for Vehicle B.

Table 4.1: Number of available (recorded) data communication buses on both Vehicle A and Vehicle B.

Bus Type	**Vehicle A**	**Vehicle B**
CAN	14	11
LIN	11	8
FlexRay	1	1
Measurement CAN	1	1
N° of meas. points	118	110

Eventually, for each recorded data bus one *.blf* data file is produced per test drive which is then converted into an *.mf4* file enriched with additional knowledge using a *.dbc* file. This includes information on data type, value range and possible conversion rules (cf. Section 3.4.5) as well as with the signal names themselves.

Database Description

The entirety of the data collected forms two distinct databases of *.mf4* files. The number of test drives is D_t with $t \in \mathcal{T}$. It is 15 for Vehicle A and 25 for Vehicle B. A detailed statistical summary (cf. Table 4.2) provides insights into the (numerical) environmental conditions and operational ranges of the two vehicles during the test drives, highlighting variations in temperature, speed, and bus signal availability. These value ranges allow for an initial assessment of the limits of the predictive capabilities of the ML models trained later on. The real-world data collection ensures that a diverse range of driving situations is captured, thereby enhancing the precision of the resulting ML models.

Table 4.2: Statistical summary for **numerical** variables of Vehicles A and B.

Vehicle	**Numerical Variables**	**Min**	**Max**	**Average**	**Median**
Vehicle A	Outside temperature τ_A [°C]	-0.45	23.50	9.45	9.50
	Vehicle speed v_{s_A} [km/h]	0.00	232.66	74.83	70.71
	Bus Signals [count]	23	7796	1058	263
Vehicle B	Outside temperature τ_B [°C]	3.00	21.50	10.47	10.00
	Vehicle speed v_{s_B} [km/h]	0.00	186.49	87.19	92.61
	Bus Signals [count]	45	7861	548	1283

Vehicle A is exposed to an outside temperature τ_A ranging from -0.45 °C to 23.50 °C, with an average temperature of 9.45 °C and a median of 9.50 °C. The vehicle speed v_{s_A} for Vehicle A ranges from 0.00 km/h to 232.66 km/h ($\approx$ 64.64 m/s), with an average speed of 74.83 km/h ($\approx$ 20.79 m/s) and a median of 70.71 km/h ($\approx$ 19.64 m/s). Additionally, the number of bus signals per data bus recorded for Vehicle A ranges from 23 to 7796, with an average of 1058 and a median of 263 signals captured. Overall, a total of 263370 data points are collected with a downsampling from 100 Hz to 5 Hz[17] which results in 52674 seconds (14.63 hours) of driving time. Conversely, Vehicle B shows an outside temperature τ_B ranging from 3.00 °C to 21.50 °C, with an average temperature of 10.47 °C and a median of 10.00 °C. The speed v_{s_B} of Vehicle B varies between 0.00 km/h and 186.49 km/h ($\approx$ 51.80 m/s), with an average speed of 87.19 km/h ($\approx$ 24.22 m/s) and a median of 92.61 km/h ($\approx$ 25.73 m/s). The number of signals per data bus for Vehicle B ranges from 45 to 7861, with an average of 548 and a median of 1283. Here, 384290 data points are sampled with the same raster as Vehicle A which is 0.2 or 5 Hz, resulting in 76858 seconds (21.35 hours) of driving time.
Disclaimer: All bus signals mentioned explicitly hereafter are referred to with a human readable signal name which does not reflect the actual signal name in the dataset and in the vehicle. This also serves privacy concerns by the OEM. Not all data buses recorded might be used in the further process steps since the actual buses needed depend on the ECUs selected for the final experimentation.

Table 4.3 summarizes categorical metadata of the compiled database for both vehicles. Omitted categories have occurrences of less than 5 data points and are thus considered irrelevant or incorrectly recorded. For Vehicle A, the day/night distribution shows that 56.66 % of the data was collected during the day and 43.34 % at night, with a negligible 0.00 % (rounded value) falling into omitted categories. When considering whether the vehicle was driven with a passenger, 54.70 % of the instances include a passenger, while 45.29 % do not, and 0.01 % are omitted. The data confirms that the vehicle is always driven with a driver—which is obvious—accounting for 100.00 % of the cases. Regarding the driver's seatbelt usage, it is fastened 99.45 % of the time and unfastened 0.55 % of the time, with 0.00 % omitted. For the passenger seatbelt, it is fastened in 54.88 % of the instances and unfastened in 45.12 %, with 0.00 % omitted.
In contrast, Vehicle B's day/night distribution indicates that 63.31 % of the data was collected during the day and 36.69 % at night, with 0.00 % omitted. When examining whether the vehicle was driven with a passenger, 57.38 % of the instances included a passenger, 42.62 % did not, and a negligible 0.00 % (rounded value) involve a child seat or were omitted. Similar to Vehicle A, Vehicle B is always driven with a driver, accounting for 100.00 % of the cases. The driver's seatbelt was fastened 99.64 % of the time and unfastened 0.36 % of the time, with 0.00 % (rounded value) omitted. The passenger seatbelt however, is fastened in 57.50 % of the instances and unfastened in 42.50 %.

[17] 5 Hz returned best results compared to the model size in preparatory experiments.

Table 4.3: Statistical summary for **categorical** variables of Vehicles A and B (including omitted categories).

Vehicle	Categorical Variables	Categories and Distribution
Vehicle A	Day/night distribution	Day: 149219 (56.66 %), Night: 114146 (43.34 %), omitted categories: 5 (0.00 %)
	Driving with a passenger	Yes: 144060 (54.70 %), No: 119288 (45.29 %), omitted categories: 15 (0.01 %)
	Driving with a driver	Yes: 263370 (100.00 %)
	Driver seatbelt	Fastened: 261923 (99.45 %), unfastened: 1443 (0.55 %), omitted categories: 4 (0.00 %)
	Passenger seatbelt	Fastened: 144530 (54.88 %), unfastened: 118834 (45.12 %), omitted categories: 6 (0.00 %)
Vehicle B	Day/night distribution	Day: 243277 (63.31 %), night: 140998 (36.69 %), omitted categories: 15 (0.00 %)
	Driving with a passenger	Yes: 220488 (57.38 %), No: 163780 (42.62 %), child seat: 17 (0.00 %), omitted categories: 5 (0.00 %)
	Driving with a driver	Yes: 384290 (100.00 %)
	Driver seat belt	Fastened: 382901 (99.64 %), unfastened: 1385 (0.36 %), omitted categories: 4 (0.00 %)
	Passenger seat belt	Fastened: 220976 (57.50 %), unfastened: 163314 (42.50 %)

Figures 4.1 and 4.2 show the outside air temperature τ_A and speed profiles v_{s_A}. The graphics underline the desired randomness the vehicles are exposed to during their test drives. A certain standardized driving cycle or profile (e.g. WLTP) is not included in the database. In addition to that, Figures 4.3 and 4.4 demonstrate the same for Vehicle B.

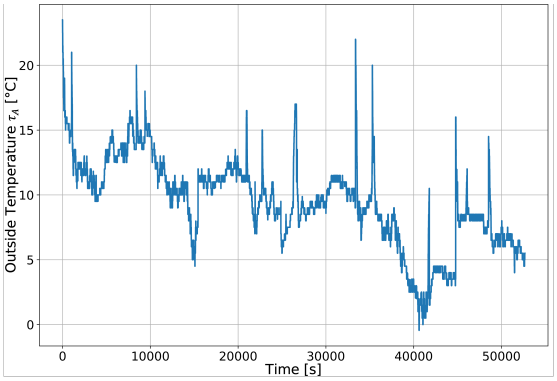

Figure 4.1: Outside air temperatures τ_A captured by Vehicle A.

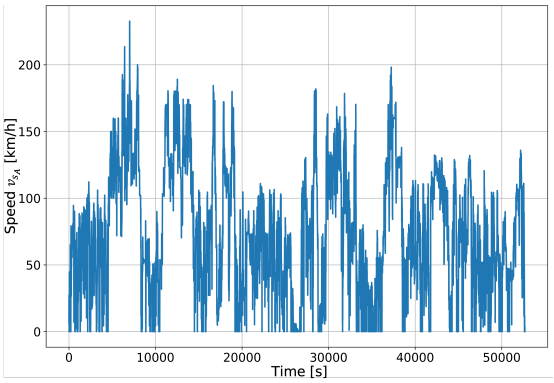

Figure 4.2: Speeds v_{s_A} captured by Vehicle A.

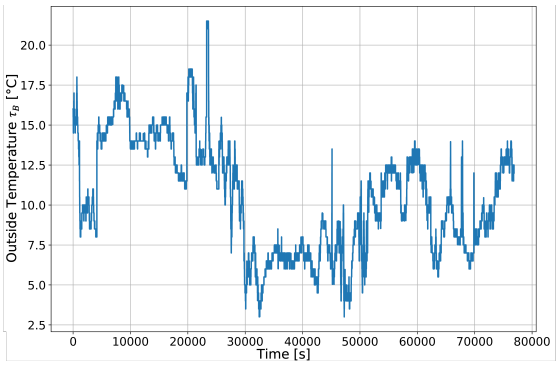

Figure 4.3: Outside air temperatures τ_B captured by Vehicle B.

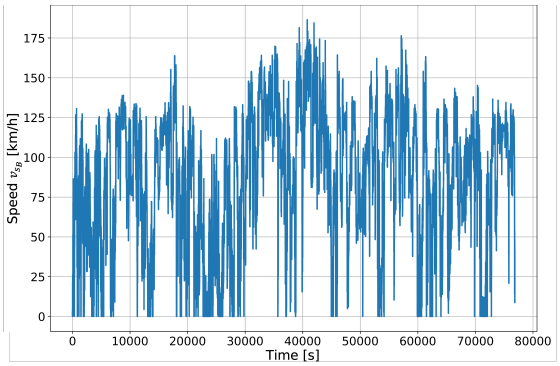

Figure 4.4: Speeds v_{s_B} captured by Vehicle B.

4.1.2 Data Type Analysis and Blacklisting

Vehicle bus data can be of all kinds of data types as described in Section 3.4.2 due to the variety of information that needs to be transmitted between ECUs. However, special attention is to be paid to bus control data, event-based data as well as any type of diagnostics data since it is known ex ante that some signals do not provide any information on the prediction of the energy consumption nor on its explanation. For example, bus control data such as checksums and tickcounts are not related to the energy consumption (if not by spurious correlation). This is why it is useful to already filter these signals out beforehand, via a static blacklist. The same applies to diagnostics data and other information such as the current local time or date as well as measurement signals which are not part of customer-ready vehicle ECU software. The exact blacklist is displayed in Table 4.4.
Event data is non-cyclic information which can be triggered by users, systems, devices, or external conditions spontaneously. This means if it is not triggered it is not available in the dataset. Missing data (Not-a-Number, NaN) which e.g. NNs cannot handle properly [129] must be avoided by the data import strategy and at the latest during data preprocessing.

Table 4.4: Signals flagged for blacklisting signals during the data import.

Category	Signal Names
Bus Control Markers	`Sequence Counter`, `Tickcounts`, `Authentication Info`, `Checksums`
Irrelevant Information	`Odometer Reading`, `Measurement Signals`
Metadata Fields	`Current Date and Time`, `Timestamps`

4.1.3 Developing a Robust Data Import Strategy and Algorithm

Algorithm 1 is designed to import and process vehicle measurement data from *.mf4* files using the open source Python library *asammdf* [183]. It starts with initializing a dedicated `Data.py` class with several input parameters (some optional), inlcuding the directory containing the *.mf4* files, a fuse-to-bus allocation file, a prefix used in the *.mf4* files, and optionally, a blacklist file with signals to exclude (additional to the static standard blacklist shown in Table 4.4), a target signal name which can manually be specified if not already done in fuse-to-bus mapping, and a

boolean flag (`inner_join`) indicating the DataFrame concatenation method.
The fuse-to-bus allocation file is a text file indicating the fuse's (ECU's) name in the measurement CAN as well as the data buses relevant for each fuse (connected to the corresponding ECU) in the following format $< fuse\_name >:< bus_1 >, < bus_2 >, \ldots, < bus_n >$. More than one fuse can be specified. Additional fuses require similar entries in new lines of the text file. Creating the mapping file is a manual process and requires knowledge about the respective E/E architecture since the exact relations between ECU (fuse) and the data it comes in touch with must be reflected. The next step of the algorithm involves organizing the files. The fuse-to-bus mapping is read and a dictionary created from it (`fuse_bus_dict`). It then lists all *.mf4* files and groups them by time frames into another dictionary, (`dict_files`). This ensures that data points from different test drives are not mixed up.
Subsequently, the algorithm identifies the available fuses by searching the measurement files using the specified `fuse_prefix` and stores the results in a set called `set_fuses_names`.
For each fuse in `set_fuses_names`, the algorithm proceeds to process the data. If an additional blacklist is specified, it creates a list of signals which are then not imported. Then, for each group of files in `dict_files`, it initializes a collection, `mf4_collection`. For each file in the group, it is checked whether the file contains measurement data (`Mess_CAN`). If so, it reads the measurements for the fuse and adds them to `mf4_collection`. If a file corresponds to the data bus relevant for the fuse, it reads the bus signals, applies the blacklist, and adds the data to `mf4_collection`, too.
If `mf4_collection` eventually contains the desired data, the algorithm stacks the data into `mf4_stacked` and converts it into a Pandas DataFrame[18], named `df_mf4_stacked`. It then fills any NaN values with zeros in `df_mf4_stacked`, and appends it to a list of DataFrames, `dataframes`.
Once all fuses (specified in the fuse-bus-allocation file) are processed, the algorithm concatenates all DataFrames in `dataframes` into a single one which is `df_result`. If the `inner_join` flag is set to `True`, it concatenates the DataFrames using an inner join, keeping only the common columns which handles missing data caused by events that were not triggered during all the test drives effectively. Otherwise, an outer join is used, keeping the superset of all columns.
The final output of the algorithm is `df_result`, a Pandas DataFrame containing the processed data ready to be processed further within the holistic ML pipeline.

[18] A Pandas DataFrame is a two-dimensional, mutable data structure in Python that allows for the storage and manipulation of heterogeneous data types, organized into labeled rows and columns. It is comparable to a table in a relational database or an Excel spreadsheet, providing a flexible and efficient means of handling structured data [184].

Algorithm 1: Algorithm for importing and processing vehicle measurement data in preparation for ML tasks.

Input: `filedir`: directory with `.mf4` files
`fuse_bus_file`: fuse-to-bus mapping file
`fuse_prefix`: prefix used in `.mf4` files
`blacklist`: (optional) file with signals to exclude
`target_signal`: (optional) target signal name
`inner_join`: (optional) boolean flag for DataFrame concatenation method
Output: `df_result`: DataFrame containing processed data

```
1  1. Initialize Data class with input parameters
2  2. Organize files: Read fuse-to-bus mapping; create fuse_bus_dict
3  List .mf4 files; group by time frames, write into dict_files
4  3. Identify available fuses: Search measurement files using fuse_prefix;
     store in set_fuses_names
5  4. For each fuse in set_fuses_names: foreach fuse do
6  |  if blacklist is specified then
7  |  |  Create blacklist of signals
8  |  end
9  |  foreach group in dict_files do
10 |  |  Initialize mf4_collection
11 |  |  foreach file in group do
12 |  |  |  if file is measurement data (Mess_CAN) then
13 |  |  |  |  Read measurements for fuse; add to mf4_collection
14 |  |  |  end
15 |  |  |  else if file corresponds to bus for fuse then
16 |  |  |  |  Read bus signals; apply blacklist; add to mf4_collection
17 |  |  |  end
18 |  |  end
19 |  |  if mf4_collection has data then
20 |  |  |  Stack data into mf4_stacked; convert to DataFrame
   |  |  |   df_mf4_stacked
21 |  |  |  Add file column with file name to the mf4_stacked; Append
   |  |  |   df_mf4_stacked to dataframes
22 |  |  end
23 |  end
24 end
25 5. Concatenate all DataFrames into df_result
26 if inner_join is True then
27 |  Concatenate using inner join (keep common columns)
28 end
29 else
30 |  Concatenate using outer join (keep all columns)
31 end
32 6. Output df_result
```

4.1.4 Methodology to Select Simulation-worthy ECUs

Another research problem-specific element that needs to be developed before constructing the ML pipeline is a mechanism to reduce the number of ECUs (fuses) to be examined, since not every component in the vehicle is "worth" predicting by an ML model. This is especially applicable to ECUs without any variation in their consumption (constant power drawing). The procedure of selecting the right ECUs for modeling can be considered part of the model training process since only simulation-worthy ECUs are then processed further by the ML modeling pipeline.
To realize this, an algorithm is developed whose functions include processing the current measurement data, computing relevant metrics, and scoring each fuse based on them. As a result, a ranking for "simulation-worthiness" can be derived for each vehicle in $\mathcal{T}$ from which the ECUs considered for further examination are chosen.
The ranking algorithm is structured to first use the function computing the median crossing rate (MCR) of an ECU current signal, which serves as a key metric in the analysis as it is an indicator of volatility and ECUs with a volatile consumption are considered more "simulation-worthy" than those with a constant one. After reading the measurement files, a function processes the data to extract various additional statistics (cf. enumeration below), and saves the results in an Excel file. A score function then evaluates each fuse (and each metric) against certain thresholds, scoring them based on their performance and saving the scoring results.
The algorithm is designed to be run as a standalone program, hence the corresponding script—not discussed further at this point—accepts command-line arguments for the car model series, source data path, whether to plot histograms, whether to use automatic or manual thresholds, and the save path. When executed, it performs the entire analysis and scoring process, providing valuable insights into the performance and reliability of the fuses. The considered metrics for ECU ranking are the following:

- **Median Crossing Rate (`MCR`)**
  - **Global MCR (`MCR`):** This metric measures the frequency at which the signal crosses its median value over the entire duration of the measurement. It provides insight into the overall variability and stability of the signal.
  - **Local MCR (`MCR_loc`):** A metric similar to the global MCR but it is computed over smaller, localized windows over the progression of the signal. It helps to identify local variations and instabilities within the signal.
- **Minimum Value (`Min`)**
  - The smallest signal value observed, indicating the lower bound of the signal's behavior.
- **Maximum Value (`Max`)**
  - The largest signal value observed, indicating the signal's upper bound.

- **Mean Value (`Mean`)**
  - The average signal value, providing a measure of the central tendency of its behavior.
- **Median Value (`Median`)**
  - The median value of the signal, representing the central point of the data. Compared to the mean, it is less influenced by outliers.
- **Standard Deviation (`SD`)**
  - This metric measures the amount of variation or dispersion in the signal. A higher SD indicates that the signal values are spread over a wider range.
- **Normalized Standard Deviation (`SD_Norm`)**
  - The standard deviation of the signal after normalization. Normalization rescales the data to a standard range, facilitating easier comparison between signals with different scales.

Once the metrics are computed, the script and the corresponding algorithm evaluate each fuse against predefined thresholds (explained below). This evaluation process is encapsulated in the `score` function. The scoring methodology involves the following steps:

(i) **Threshold Determination**

- The script either calculates thresholds for each metric automatically based on the data ("auto threshold") or loads them from a predefined JSON file. These thresholds serve as benchmarks to evaluate the performance of each fuse (here: "auto threshold" is used).

(ii) **Metric Evaluation**

- Each ECU is evaluated to determine whether its metrics meet the respective thresholds.
- If a metric meets the threshold, it is marked as "Y" (yes); otherwise, it is marked as "N" (no).

(iii) **Score Assignment**

- For each fuse (ECU), an overall score is calculated by counting the number of 'Y' marks across all metrics.
- The score eventually represents the number of metrics for which the fuse meets the thresholds, providing a quantitative measure of its performance. Since five metrics are used for the evaluation, the highest possible score is 5.

Thresholds for each metric are useful to distinguish between acceptable and potentially problematic fuse (ECU) power consumption performances. They are determined as follows when set to "auto threshold": The maximum value threshold (`Threshold_MAX`) is the 30th highest maximum value among all ECUs, which identifies fuses with unusually high maximum values. The mean value threshold (`Threshold_MEAN`) is the 30th highest mean value, flagging fuses with significant deviations in average performance.
The global median crossing rate (MCR) threshold (`Threshold_MCR`) is the 30th lowest global MCR value, indicating fuses with stable signals. Similarly, the local median crossing rate threshold (`Threshold_MCR_LOC`) is the 30th lowest local MCR value, identifying fuses with fewer local signal variations.
Finally, the normalized standard deviation threshold (`Threshold_SD_NORM`) is the 30th highest normalized standard deviation, highlighting fuses with higher signal variability.
These thresholds effectively differentiate between fuses performing within simulation-worthy parameters and those which need to be rejected for this research. Given the total number of ECUs available in contemporary luxury vehicles (cf. measurement points in Table 4.1) the threshold of 30 is chosen to reflect around one quarter of the best performing components per selection criterion. So if a score is among the top 30, then "yes" is awarded, otherwise "no".
Table 4.5 provides an extract of the top-ranked ECUs from a larger pool of 118 for Vehicle A and 110 for Vehicle B forming the ECU set $\mathcal{E}$. Only the most promising (high-ranked) ECUs, as determined by the previously introduced metrics, are to be considered. Lower-ranked components, which do not meet the criteria for high simulation relevance, are omitted from the table and from any further consideration in this research.

Table 4.5: Result of the ECU selection algorithm. Ranks in descending order. For Vehicle A and B 9 ECUs are considered respectively for the further experimentation.

ECUs Vehicle A	**Max**	**Mean**	**MCR**	**MCR_loc**	**SD_Norm**	**Score**
Body Controller Front (BCF)	Y	Y	Y	Y	Y	**5**
Extractor Fan	Y	Y	Y	Y	Y	**4**
Central Infotainment Display (CID)	N	N	Y	Y	Y	**4**
Fuel Supply ECU	Y	Y	Y	Y	N	**4**
Coolant Pump	Y	Y	Y	Y	N	**4**
Right Pixel Headlamp	Y	Y	Y	Y	N	**4**
Left Pixel Headlamp	Y	Y	Y	Y	N	**4**
Adaptive Suspension	Y	Y	Y	Y	N	**4**
Driver Display	N	N	Y	Y	Y	**3**
ECUs Vehicle B	**Max**	**Mean**	**MCR**	**MCR_loc**	**SD_Norm**	**Score**
Right Headlamp	Y	Y	Y	Y	Y	**5**
Left Headlamp	Y	Y	Y	Y	Y	**5**
Steering Column ECU	Y	Y	Y	Y	Y	**5**
Coolant Pump	Y	Y	Y	Y	Y	**5**
Seat ECU Driver	Y	Y	Y	Y	N	**4**
Door ECU Front Left	Y	Y	Y	N	Y	**4**
Body Controller Front (BCF)	Y	Y	Y	N	Y	**4**
Central Infotainment Display (CID)	Y	Y	Y	N	Y	**4**
Active Air Suspension	Y	Y	Y	N	Y	**4**

4.1.5 Features and Data Buses for Selected ECUs

Table 4.6 provides an overview of the ECUs selected for both Vehicle A and Vehicle B. Each of them is associated with a set of data buses (to which they are physically connected in the vehicle). Additionally, the corresponding number of raw features (bus signals) available per ECU is shown. The table also highlights the diversity of data sources and features captured for each ECU, showing the complexity of the datasets analyzed. For example, the adaptive suspension in Vehicle A and the steering column ECU in Vehicle B are connected to the FlexRay bus and thus have the highest number of raw features (5,399 and 5,608 respectively). In contrast, simpler components, such as the coolant pump, use a LIN bus with significantly fewer features.
The buses connected to the 18 selected ECUs are nonetheless only a subset of all the recorded buses during database compilation. For privacy reasons, the actual bus names used by the OEM are omitted. Instead, the buses are identified by their protocol types.

Table 4.6: Overview of the power consumers under investigation with their corresponding data buses and the number of distinct bus signals in the raw data.

ECUs Vehicle A	**Data Buses**	**N° of Features (inner join, raw)**
BCF	3x CAN, 5x LIN	3480
Extractor Fan	1x LIN	138
CID	1x CAN	972
Fuel Supply ECU	1x CAN	216
Coolant Pump	1x LIN	138
Right Pixel Headlamp	1x CAN	1334
Left Pixel Headlamp	1x CAN	1334
Adaptive Suspension	1x FlexRay	5399
Driver Display	1x CAN	972
ECUs Vehicle B	**Data Buses**	**N° of Features (inner join, raw)**
Right Headlamp	1x CAN	1386
Left Headlamp	1x CAN	1386
Steering Column ECU	1x FlexRay	5608
Coolant Pump	1x LIN	134
Seat ECU Driver	1x CAN	1064
Door ECU Front Left	1x CAN	1599
BCF	3x CAN, 5x LIN	2748
CID	1x CAN	987
Active Air Suspension	1x FlexRay	5608

4.1.6 Metrics and KPI Selection for Model Evaluation

This subsection addresses the selection and subsequent weighting of KPIs to assess an ML model's quality. Therefore, a workshop was conducted in February 2023 with two AI research experts from an OEM as well as with five domain experts from the OEM's E/E architecture department who take the customer perspective. The goal of the workshop was to discuss a predefined set of metrics and a set of proposed KPIs $\mathcal{K}$ which range from project-specific ones to common ML metrics as introduced in Section 3.5.4. The workshop results thus permit the answering of *SRQ2*—which is about appropriate metrics for performance assessment—later on.

Need for a Project-specific Metric

It is the aspiration of this research project to not solely rely on state-of-the-art metrics for the quality assessment. Hence, a specific KPI is developed to address the project's main prediction challenge which is the correct prognosis of the electric current over time, thus the electric charge (cf. Section 3.3, Equation 3.4). It shall measure the absolute percentage difference of electric charge consumed by a given ECU averaged over a number of test drives D_t. Since the power consumption might be different for every drive a weighting mechanism is to be included to give more importance to a drive where the respective ECU consumes more power than in others. This metric is especially important when k-fold or leave-one-out cross-validation (LOOCV, cf. Section 4.2.2) is used for model training, where the folds are not of equal length and thus consumption might not be comparable over the different drives [27]. The KPI is therefore named *Percentage Average Weighted Deviation* (PAWD) and is defined as follows:

$$PAWD = \frac{\sum_{i=1}^{D_t} |((\frac{Q_i}{\hat{Q}_i} - 1) * 100) * Q_i|}{\sum_{i=1}^{D_t} Q_i} \quad [\%] \tag{4.1}$$

Where:
D_t: is the number of drives (CV folds) of a vehicle $t \in \mathcal{T}$.
Q_i: is the measured electrical charge of drive i.
$\hat{Q}_i$: is the predicted electrical charge of drive i.

State-of-the-Art Metrics for Regression Models

Additional common metrics suggested to use in the context of ML model quality assessment for this research are the R^2 (cf. Equation 3.15), the MSE (cf. Equation 3.14) and the inference speed measured in seconds. More precisely, the average and cross-validation-weighted R^2 (in analogy to PAWD) are proposed, as well as its standard deviation (SD) over all test drives (CV folds). The same is proposed for the MSE. For the inference and training times—the latter also measured in seconds—the calculation by sample (inference time) and by number of features and samples (for the training time) are suggested to the experts. Complementing PAWD, the standard deviation (SD) as a measure of the volatility of the electric charge prediction per test drive is proposed. The model size measured in kilobytes when exporting the ML regressor via the standardized ONNX[19] format, is another quantitative metric.

[19]Open Neural Network Exchange (ONNX) is an open-source format designed to represent ML models in a standardized, thus exchangeable way [185].

A Note on Explainability as a Metric

One discrete, stepwise metric developed in this thesis is the **explainability** indication, to what extent both the local and global state-of-the-art XAI methods from Section 3.6 can be applied to the respective regressor. A value of 1.0 thus refers to the ML algorithm that is most and 0.0 least explainable amongst the ones considered in this research. The KPI is defined below.
Random forest shall be given the highest explainability score of 1.0 due to its interpretable tree-like structure. Both local and global XAI methods, such as PFI, ALE and SHAP are applicable. Gradient boosting also works with these methods, although its sequential training introduces a slightly increased complexity compared to RF. It shall therefore be given the explainability score of 0.8. In contrast, MLP shall rank lower (with a score of 0.5) in explainability since it has a non-linear and complex network structure, hence a black-box nature. Although SHAP and ALE can still be applied, the insights derived from these methods are less intuitive compared to tree-based models. Eventually, the LSTM model is the least explainable (score: 0.0), which is due to its black-box nature and the additional challenge of interpreting temporal dependencies derived from the given set of explainers.

4.1.7 KPI Assessment Workshop, Pairwise Comparison and Results

During the aforementioned workshop, AI and domain experts carry out a so-called *pairwise comparison* on the KPIs and their previously mentioned variations. The result is not only a ranking but also a set of weights $\mathcal{V}$ specific to the present problem. Each KPI weight $v_c \in \mathcal{V}$ can later be applied to different ML model results (decision criteria) to carry out a weighted sum analysis (WSA, cf. Section 4.3.3) to ultimately determine a "most suitable" modeling algorithm. The results of the weight determination process are presented in Table 4.7.

Table 4.7: Metrics and their respective rankings and weights, according to a pairwise comparison conducted during an expert workshop.

KPI / Metric k	**Rank**	**Weight** v_c
X-val weighted PAWD [%]	1	10
X-val weighted R^2	2	7.8
Inference time per sample [s]	3	7.4
SD(PAAD) over all X-vals [%]	4	6.5
Explainability (local & global methods)	5	6.1
X-val avg. R^2	6	5.7
Model size after export [kB]	6	5.7
SD(R^2) over all X-vals	8	5.2
X-val weighted MSE [A^2]	9	3.9
MSE [A^2]	10	2.6
SD(MSE) [A^2]	11	1.3
Training time per feature & sample [s]	12	0.4

As shown in Table 4.7, metrics like the **X-val weighted PAWD** (rank 1, weight 10) and the **inference time per sample** (rank 3, weight 7.4) are ranked high because of their direct influence on model performance and their training, hence deployment efficiency. These factors are critical for the underlying real-world application, for which workshop participants (which include end users of the resulting models, as well) prioritize prediction accuracy and computational efficiency.
In contrast, metrics like **SD(MSE)** (rank 11, weight 1.3) and **MSE** (rank 10, weight 2.6) are ranked lower, due to their secondary importance for ML problems or due to redundancy with other KPIs already ranked higher (e.g. $\mathbf{R^2}$ is generally ranked higher even though **MSE** is a standard error metric in the ML context). Criteria like **explainability** (rank 5, weight 6.1) reflect growing concerns by the domain experts about transparency in their resulting ML models, but they do not overweight core performance metrics in terms of priority, however. Similarly, the **ONNX model size** (rank 6, weight 5.7, sharing the same rank with **X-val avg.** $\mathbf{R^2}$) is important for productive model deployment, but resource efficiency is secondary to model accuracy and runtime, according to them.
These rankings suggest that task-specific needs dominate the prioritization of metrics, with a strong emphasis on performance, efficiency, and scalability. Metrics that measure raw prediction error or error variability are ranked lower due to their reduced relevance in this context compared to performance measures providing insights into model reliability, speed, and ease of deployment.
The detailed results of the pairwise comparison can be found in Appendix C.

4.1.8 Computational Resources

All subsequent experiments are conducted on the same local Linux workstation with the following computational capabilities:
The machine is powered by an Intel Xeon Gold 6226R Central Processing Unit (CPU), which features a total of 32 logical processors. The processor architecture is based on the x86_64 instruction set, supporting both 32-bit and 64-bit operation modes. It has a single socket containing 16 physical cores, each supporting two threads, providing a total of 32 threads through hyper-threading [186]. The CPU operates at a base frequency of 2.90 GHz [186]. The machine includes 502 GiB[20] of system memory (RAM), with an additional 4.0 GiB of swap space available and it runs on Ubuntu 22.04.5 Long-Term Support (LTS, Jammy Jellyfish).
The software environment is configured with Python 3.10.12, a programming language commonly used for ML and data analysis tasks. The libraries and dependencies required for the experiments are listed in the accompanying `requirements.txt` file (cf. Appendix B), ensuring the reproducibility of the results.

[20]Gibibyte: 1 GiB = 2^{30} byte $\approx 1,074$ gigabytes. 1 byte = 8 bit.

4.2 Methodology

At this point of the present study, the preliminaries are fully worked out. More precisely, the experimental data is collected from the test vehicles under real-world conditions, and code is developed to import the data in a way and format the ML algorithms can deal with it. Furthermore, the framework has been set to which enables objective and project-specific ML model comparison including respective metrics and KPIs.
In this section, the methodology is developed to train and assess the ML modeling algorithms used. This consists of an ML pipeline processing the raw data from the import to the final regressor model as well as the detailing of its contents such as the selection of a (cross-)validation and documentation mechanism together with an appropriate hyperparameter tuning strategy suitable for the given ML task.
One overarching goal is also to ensure practicability which means that the pipeline must be as automatable as possible so its usability in a real-world engineering environment can be guaranteed without the need of an extensive ML training and qualification for the development engineers using it.

4.2.1 Construction of a Pipeline Architecture

According to the CRISP-ML(Q) process model, its step 3 can now be realized at this point which includes the model training (cf. Section 3.5.3) and the necessary steps to attain acceptable prediction results.
Therefore, a stepwise algorithm (pipeline) is subsequently presented that comprises the approach to achieve that goal.

(i) **Feature Engineering and Selection**
After the data import there is a considerable number of features present in the training data (cf. Table 4.6). Hence, an algorithm has to be applied to reduce the size of the feature set further to speed-up subsequent processes such as hyperparameter search and model training, as well as to reduce the model size. Therefore, the methods by the authors of [31] are applied with a static feature engineering pipeline (fixed sequence) to ensure the reproducibility of the results.

(ii) **Model Hyperparameter Tuning Strategies**
Following the feature engineering and selection each ML model's hyperparameter set θ_a with $a \in \mathcal{A} = \{RF, GB, MLP, LSTM\}$ can be tuned to further optimize its predictive capabilities. Therefore, the tunable parameters as well as an adequate search range needs to be determined before selecting a suitable search and sampling algorithm among the ones presented in Section 3.7.2.

(iii) **Model Validation Strategy**
Once the hyperparameters are tuned and the features are selected as well as engineered the resultant data can be used to apply CV strategies, which resample the training data and execute several fits for the same regression task with slightly altered training and validation datasets. The KPIs and metrics for each run can then be averaged over all CV folds and a more concise image of the algorithm's generalization performance on variations of the input data can be derived.

(iv) **Model Training**
With the feature set prepared and the parameters tuned the actual productive regression can take place as a result of which the regressor can be exported as an ONNX model file and the respective KPIs can be calculated to be able to assess the algorithm's performance for the given ML task. Ideally, the KPIs are not the result of a single run of the regression algorithm but of a systematic validation strategy including several fits with variations of training and validation data.

(v) **Regressor and KPI Export**
After the regression task, the regressor (the actual ML model) must be able to be exported so that it can be transferred freely to any other computation environment where it can then be fed with unseen ECU communication data to make predictions, eventually replacing a complex and costly physical measurement system (cf. Section 3.4.5). For a thorough assessment of the regressor's performance on the training and validation data, the KPIs (cf. Section 4.1.6) must be saved to a file in a tabular format where they represent the columns and each run adds a new line of KPI values to the table. This way, different runs can be compared to each other with greater ease.

(vi) **Explainable AI Method Implementation**
To be able to explain root causes and important influential factors of the ECU energy consumption it shall optionally be possible to turn on the execution of XAI methods for a training run. Those helpful for the given ML task are described in Section 3.6 and are selected based on a survey with automotive development engineers from the same domain conducted by the authors of [29].

The base structure of the ML pipeline developed can be found in Fig. 4.5 below. It starts on the left side with the relevant *.mf4* measurement files as well as with the fuse/bus allocation (cf. data import in Section 4.1.3). The data is then passed through until the final training and results export stage. The points above, with the exception of point (iv), are explained in greater detail in so-called *detailed views* in Section 4.2.2. The resulting full architecture is schematically shown in Appendix D.

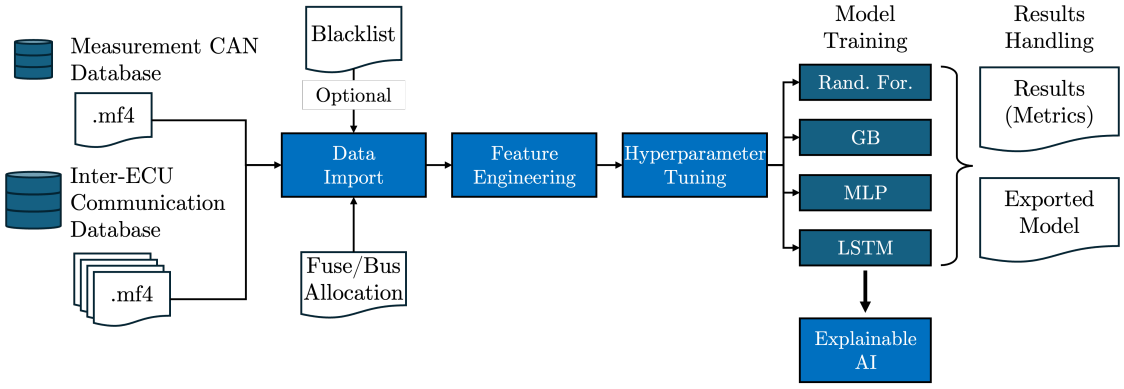

Figure 4.5: Simplified visualization of the automated pipeline to obtain ECU consumption prediction models including data import, feature engineering, model training, results export as well as XAI.

4.2.2 Pipeline Components and Order

Subsequently, detailed perspectives of the order of the ML and data processing pipeline's components are laid out as well as specifics on selected Python code implementations are presented. This pipeline is then eventually utilized to produce the modeling and XAI results in Section 4.3.

Detailed View 1: Feature Engineering and Selection

Once the data is imported, a **training dataset** and a so-called **test dataset** are created from the raw data. The test dataset is set aside and excluded from the model generation process. However, it remains available for later (optional) validation of the model using unseen data, should the need arise. Importantly, the transformations resulting from feature engineering must also be applied consistently to the test dataset. This ensures the feature columns remain identical across datasets, which is essential for most ML modeling algorithms to function correctly—in particular when features are constructed or removed.
In order not to tamper with the original training data, a copy of it is produced and the target column is removed so that only the input features remain on which the feature engineering methods are then executed (virtual training data). This way, helper columns which are not features—such as the identifier of a test drive, the target column and the timestamps—can remain in the original training and test datasets. The results of the processing methods are then applied to that original training dataset only. Therefore, a fit-and-apply approach is developed which first fits a feature engineering method on the virtual training data before applying it to the actual one which is then used in the further steps of the ML processing pipeline. For this research, a static pipeline is used, applying methods proposed by the authors of [31][21]. Additionally, the static pipeline explicitly excludes the creation of polynomial features (e.g. multiplying two features, where their product constitutes a new one) even though they might capture non-linear relationships between the features and the target variable that linear models might not otherwise [187]. The reason for that is the lack of explainability of polynomial features as squared or cubic features might lose their original meaning. This would then contradict the goal of enabling explainability.
The process is graphically depicted in Fig. 4.6 where it is shown that at the end of the steps the feature engineered dataset is available for further processing.
Formally written, the static pipeline always consists of a learning algorithm $a \in \mathcal{A}$ with $\mathcal{A} = \{RF, GB, MLP, LSTM\}$ and a performance metric $k \in \mathcal{K}$. Therefore, the model's performance shall be denoted as $\mathcal{P}_a^k(X, y)$. Feature engineering methods and sequence, represented by Φ, transform the original feature set $X = \{x_1, \ldots, x_n\}$ into a modified set X', expressed as $\Phi(X)$. Ideally, Φ_{opt} shall thus designate the maximization of the model's performance by determining an optimal set of features X' for the given regression problem.

$$\Phi_{opt} = \arg \max_{X' \in \Phi(X)} \mathcal{P}_a^k(X', y) \tag{4.2}$$

[21] In this publication—conducted during the course of this research—a more sophisticated reinforcement learning (RL) based approach is introduced which finds an optimal sequence of the methods based on a reward function. Nonetheless, it not used in this thesis due to the approach being non-deterministic which prohibits the reproducibility of the results generated.

The feature engineering applied for this use case starts with a first feature selection part eliminating columns (features) with no variance since the absence of variance does not provide any information for an ML model to learn from. Then, missing values (NaN) and "signal not available" values are handled with an implementation of median imputation. After that, OHE is applied to the categorical features before another low-variance removal is executed since OHE might produce low-variance columns. Finally, features with a low importance to the model are removed from the training data. There is a threshold applied which cuts off features with an importance lower than that. In the final step, highly correlated features among the remaining ones are removed using PCC (cf. Equation 3.19).

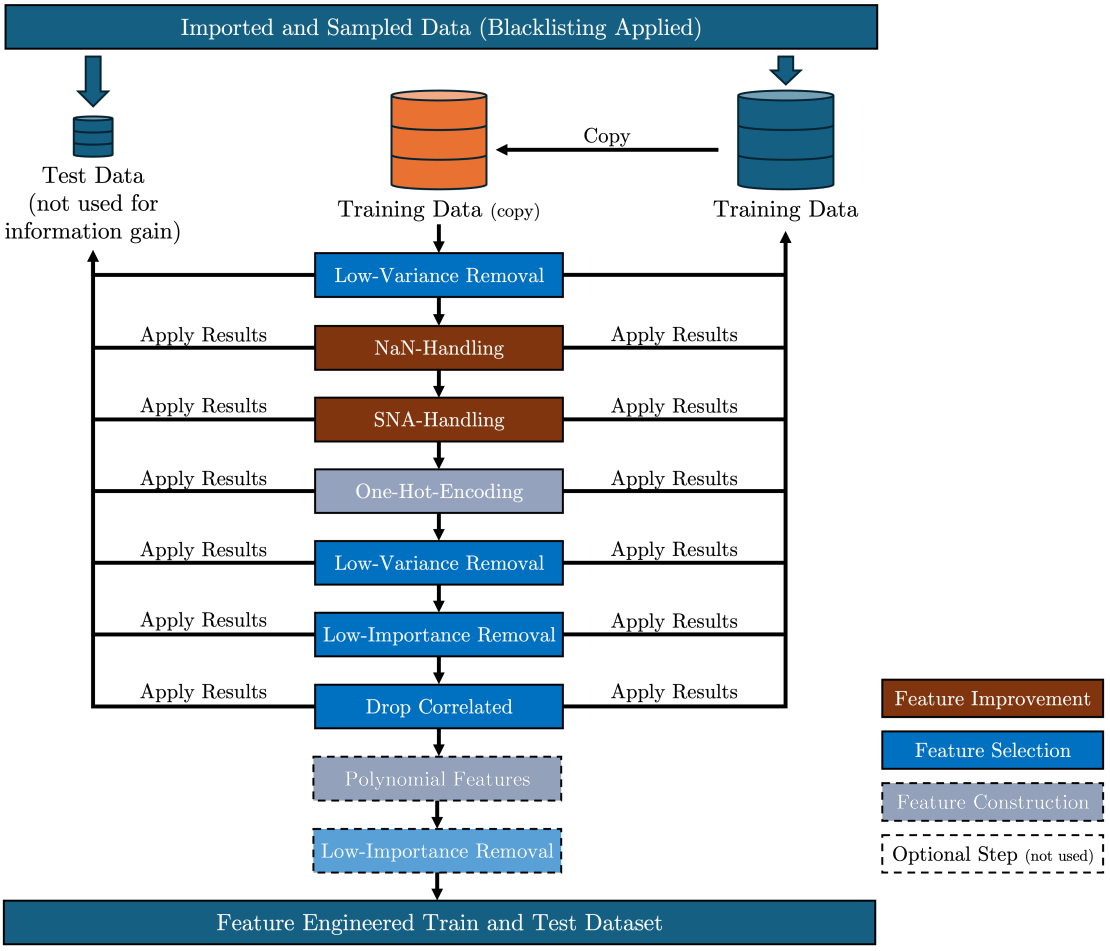

Figure 4.6: Stepwise feature engineering and selection process with a fixed (static) sequence of methods omitting polynomial feature generation.

Feature Selection Methods:

(i) **Zero Variance Elimination**
In this step of the feature engineering and selection process, all features with zero variance are eliminated. This means, if the feature values are constant over the entire time recorded, they are removed from the dataset since without alteration they do not add value to the ML model. The Python implementation to find and eliminate zero variance features Python can be referenced from Listing E.1 shown in Appendix E, Section E.1.

(ii) **Feature Importance Removal with Threshold**
To remove features with a low importance and thus low benefit to the ML model the same fit-and-apply logic as for the low-variance removal is used. First, the importances are determined using a `RandomForestRegressor` or `GradientBoostingRegressor` in the fit function, depending on the ML algorithm the data is being prepared for. Then, the identified feature columns whose importance is lower than the threshold are removed in the transform (application) part of the function pair.
In this case *impurity feature importance* is used which measures how much a feature reduces the MSE [92] when it is used for splitting a node in a decision tree [188]. More precisely, first the importance of the corresponding feature data is assessed with either an RF-based feature importance algorithm using scikit-learn version 1.2.1[22] default values (since the method is not applicable per se for MLP and LSTM). The exception is GB where a GB-based feature importance can also be calculated using scikit-learn version 1.2.1[23] with its default values. The detailed implementation of the corresponding functions are shown in Listing E.2, in Appendix E, Section E.2.
It is pointed out that the application of feature removal with importance on the dataset eventually results in different reduced feature sets for GB compared to the other considered algorithms (cf. Table 4.12) since the latter use RF to determine the importances.

(iii) **Removal of Correlated Features with Threshold**
To further reduce the complexity and size of the dataset features which are correlated to each other shall be inspected and one of them eventually be removed. To do so, a fit function is developed that takes a threshold for the minimum correlation required among two features (here the threshold is set to 0.9) and the DataFrame `df` containing the training data as another argument. Then, at first, the helper columns (cf. above) are removed before pairs of features whose correlation exceeds the given threshold are identified. The feature (column) with a lower importance is dropped. Therefore, the aforementioned importance implementation from scikit-learn is used in the way that the importances calculated at this preceding stage need to be handed over to the private `_drop_features(df, correlated_pairs, importances)` function (cf. Listing E.3, line 27) where the column with the lower (respectively higher) importance is identified from a pair of columns and eventually dropped. The detailed listing of the respective functions to realize this preprocessing functionality is shown and described in Appendix E, Section E.3.

[22] https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor.feature_importances_, last accessed: October 30, 2024.

[23] https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor.feature_importances_, last accessed: December 31, 2024.

Feature Improvement Method: For feature improvement a **median imputation** is applied to the training dataset. It is applied only to columns in the dataset that have at least the threshold value of 32 unique values, which has proven to be a generally suitable parameter in preliminary experiments with similar data [31]. For the affected columns, the median is calculated and used to replace NaN values. The strategy to find respective feature columns with NaN values also follows the fit-and-apply approach. In the fit function a dictionary (`sna_dict`) is created mapping each column in the input DataFrame to a "filler" value. The filler is either the column's median (in case of numeric values) or the string "sna" if the threshold number is not met. In a method called `fill_sna_median_fit([...])` the `sna_dict` is then applied to the real training DataFrame.
The corresponding Python implementation with a more in-depth explanation of the respective functions can be found in Listing E.4 in Appendix E, Section E.4.

Feature Construction Method: The third module of the feature engineering and selection process is the application of OHE to the categorical features in the dataset (cf. Section 3.5.5). Therefore, the dedicated `FeatureEncoding` class is implemented providing two main functions: `one_hot_encoding_fit([...])` for fitting and `one_hot_encoding_transform([...])` for applying OHE. They perform OHE on categorical columns in the training DataFrame using scikit-learn's `OneHotEncoder`[24] implementation. The latter is part of the aforementioned fit function and is set to automatically detect categorical features, extract them into distinct categories and store them in a special decoder object.
In the corresponding transform function the fitted encoder is used to transform the categorical columns of the training DataFrame into one-hot encoded representations while maintaining the structure of the original data. Since the number of categorical features is unknown prior to the feature engineering process, there is a risk that the number of columns increases significantly, potentially leading to a memory (RAM) overflow or at least to substantially larger training data. This has to be handled accordingly by the user (e.g. by adding sufficient RAM memory to the computation resource). The detailed implementation of the OHE fit and transform functions together with a more detailed description are outlined in Appendix F, Section E.5.

Detailed View 2: Hyperparameter Tuning Strategies

Now as the training as well as the test dataset are both preprocessed according to the methods described, the next step in CRISP-ML(Q) is the tuning of the modeling algorithms' hyperparameter sets θ_a, $\forall a \in \mathcal{A}$.
Therefore, an adequate tuning mechanism for the given ML modeling task must be defined globally—this means for all four ML algorithms under investigation so their results remain comparable. Subsequently, the tunable parameters for each algorithm must be identified and meaningful search spaces for each hyperparameter must be defined. The concept of hyperparameter tuning is introduced in Section 3.7.2 and specifically Equation 3.23.

[24] https://scikit-learn.org/1.5/modules/generated/sklearn.preprocessing.OneHotEncoder.html, last accessed: January 4, 2024.

In the Python implementation for this thesis, the hyperparameters are set through a configuration object (which has the format of a Python dictionary) upon the instantiation of the respective regressor objects in the codebase as shown in the Listing 4.1 below:

```
# Load configuration dictionary
config_dict = self.config.get_dict(
    config_path=self.config.configFile,
    default_path=(
        pathlib.Path(__file__).parent.parent.resolve().as_posix() +
        "/default_configs/randomforest_defaults.json"
    )
)

# Instantiate the Random Forest Regressor
regr = RandomForestRegressor(**config_dict)
```

Listing 4.1: Instantiation of a `RandomForestRegressor` with hyperparameters from a configuration dictionary.

In this listing the configuration which includes the hyperparameters is stored in a human and machine readable `randomforest_default.json` configuration file. This is the case for any of the ML algorithms in $\mathcal{A}$. This way, the parameter set θ_a can be exchanged in a modular way (e.g. with a set of tuned hyperparameters). A selection of the considerable hyperparameters together with their functionalities is presented in Section 3.5.3.

Tunable Parameters and Search Space Definitions: In the following, tunable parameters are defined for each of the ML algorithms to enhance their individual performance. This is preceded by a systematic sensitivity analysis which is carried out for each parameter with a subset of the real-world ECU communication data to obtain meaningful search spaces specific to the present ML task. In total, six power consumers from Vehicle A are chosen randomly but with the goal to cover a large variety of vehicle domains. Since this analysis took place before the KPI assessment workshop (cf. Section 4.1.6) only $\mathbf{R^2}$ and **PAWD** are considered, both averaged over all test drives (CVs, cf. Section 4.2.2). No data preprocessing had been developed yet at that point of the research project, so the sensitivity analysis is carried out on the raw data. This assessment is also useful to determine the impact of individual hyperparameters. The overall results are complemented by findings from literature where possible.
Due to the high number of possible combinations of hyperparameters for each algorithm, the sensitivity analysis is carried out by varying one parameter at once only whilst keeping others constant. Even though this procedure may not capture all interactions and dependencies between hyperparameters it is considered a practical way as the combinatorial complexity decreases.
Hereafter, the condensed results of the sensitivity analysis are presented, combined with findings from the literature, in the form of respective search spaces for hyperparameter tuning.

(i) **Sensitivity Analysis for Random Forest (RF)**
The search space ranges as well as the choice for tunable parameters for RF are supported by literature [189, 190]. Even though it is stated there that the actual search range depends strongly on the size and complexity of the training dataset [191].
For visualization, Figure 4.7 illustrates the development of the two performance KPIs, $\mathbf{R^2}$ and **PAWD**, across different values of the `n_estimators` hyperparameter for the `RandomForestRegressor`, using two of the six ECUs from the sensitivity analysis as examples. From this excerpt it can already be derived that the R^2 value improves significantly as `n_estimators` increases, while the **PAWD** decreases, indicating better model performance with more trees (estimators) in the forest.
A good value for any considered parameter can then be derived visually. The overall range for a parameter is then determined by the superset of "best parameters" over all six ECUs considered in the analysis. This procedure is repeated for all hyperparameters considered "tunable" which ultimately results in the values shown in Table 4.8.

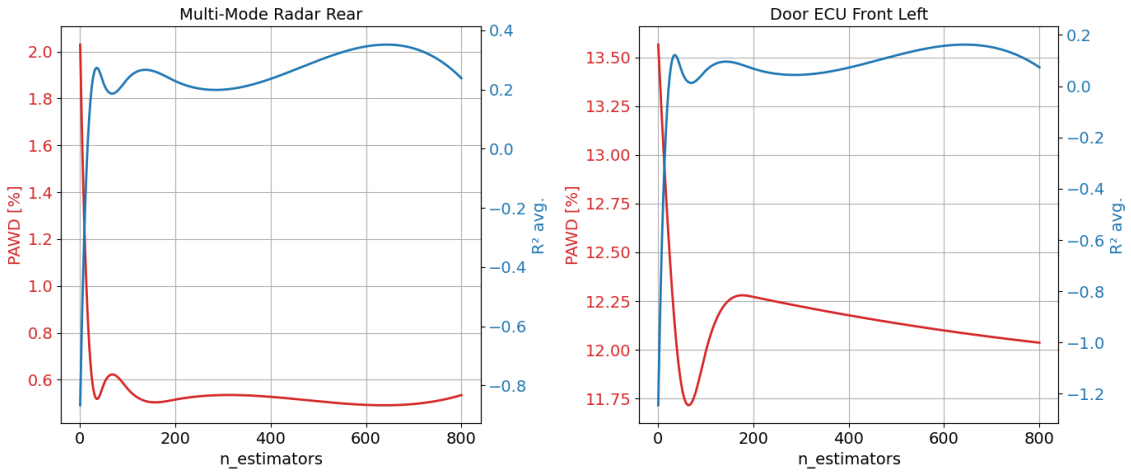

Figure 4.7: Sensitivity analysis example for the rear multi-mode radar and for the front left door ECU, and slope of the corresponding two KPI curves over different values of the `n_estimators` hyperparameter.

Table 4.8: Hyperparameter, search space, spacing, and default values for `RandomForestRegressor` (scikit-learn version 1.2.1).

Hyperparameter	Search Space	Value Distribution	Package Defaults
n_estimators	[50, 300] (20 values)	Equidistantly spaced	100
max_depth	[2, 20] (7 values) + [30, 100] (6 values) + None	Equally spaced	None
min_samples_split	[2, 200] (10 values)	Equidistantly spaced	2
min_samples_leaf	[1, 100] (10 values)	Equidistantly spaced	1
max_features	[0.1, 1.0] (10 values)	Equidistantly spaced	1.0
max_leaf_nodes	[50, 1000] (10 values) + None	Equidistantly spaced	None
bootstrap	[True, False]	Binary	True

(ii) **Sensitivity Analysis for Gradient Boosting (GB)**
The part of the tunable parameters for GB wich are similar to the ones for RF are taken over since they signify the same model behavior and influence. Whereas the GB-specific hyperparameters (`learning_rate`, `subsample` and `loss`) are taken from [109]. They are summarized as a whole in Table 4.9.

Table 4.9: Hyperparameter, search space, spacing, and default values for `GradientBoostingRegressor` (scikit-learn version 1.2.1).

Hyperparameter	Search Space	Value Distribution	Package Defaults
n_estimators	[50, 300] (20 values)	Equidistantly spaced	100
learning_rate	[0.01, 0.05, 0.1, 0.2]	Categorical	0.1
max_depth	[2, 20] (7 values)	Equidistantly spaced	3
min_samples_split	[2, 200] (10 values)	Equidistantly spaced	2
min_samples_leaf	[1, 100] (10 values)	Equidistantly spaced	1
max_features	['sqrt', 'log2', None, 0.8, 1.0]	Categorical	None
subsample	[0.5, 0.7, 1.0]	Categorical	1.0
loss	['squared_error', 'huber']	Categorical	'squared_error'

(iii) **Sensitivity Analysis for Multilayer Perceptron (MLP)**
With regard to the `MLPRegressor`, the choice of tunable hyperparameters as well as their ranges is also supported by literature (by both [164] and [192]). Ancillary, the procedure of determining ranges for this specific research is also carried out through a sensitivity analysis and with the same six power consumers and data as explained above. Figure 4.8 is an example for the `alpha` hyperparameter which influences the KPIs R^2 and PAWD over different values (here displayed—again—for two out of six power consumers). From these graphs (and from those not depicted) it can be derived that smaller alpha values are more favorable. The same procedure is applied to the other MLP-specific KPIs as well.
However, one of the more decisive factors of NNs is their `architecture` consisting of input, output and hidden layers (cf. Section 3.5.3). Despite the existence of several approaches to automate the architectural design, called *neural architecture search*—short NAS [193]—in this case, a systematic and dedicated sensitivity analysis is carried out for this parameter as well. This approach is chosen due to the novelty and complexity of the data, which makes it challenging to apply general-purpose NAS methods effectively, because of the aspiration to provide the most specific and tailored results for the given problem. Here, for each of the six ECUs a vast number of designed trials—a total of 161—are carried out to approximate a suitable range for the number and sizes of the hidden layers (parameter name: `hidden_layer_sizes`). The trial range for the sensitivity analysis reaches from no hidden layers (`Arch_[0]`) to a single one (`Arch_[5]`) with five nodes to up to four hidden layers with 150 nodes each (`Arch_[150,150,150,150]`).
Single-layer architectures are designed to vary in size from 5 to 50 neurons, while multi-layer models include configurations with uniform or mixed layer sizes. The deeper networks with three or four layers allow for learning complex, hierarchical patterns, enabling the analysis of non-linear relationships in the data—with a remaining risk of overfitting. However, larger networks might be a good fit for the presently researched ML problem as well, since the data (feature) complexity and number both vary as shown in Table 4.6.
The risk of overfitting is underlined by the results of the analysis which show a discouragement of very large hidden layer architectures with a tendency to fewer hidden layers and especially to smaller node numbers depending on the power consumer and the complexity (number of features) of the input data involved. This result is plausible since NNs with large hidden layers tend to overfit the data which leads to worse results in model validation, hence poorer generalization. Consequently, a set of architectures can be selected to be incorporated in the hyperparameter optimization strategy (cf. Table 4.10). The largest hidden layer configurations from the sensitivity analysis could eventually be discarded to keep the search space smaller. This way, a reduction from initially 161 to eventually 127 considered hidden layer configurations can be achieved.

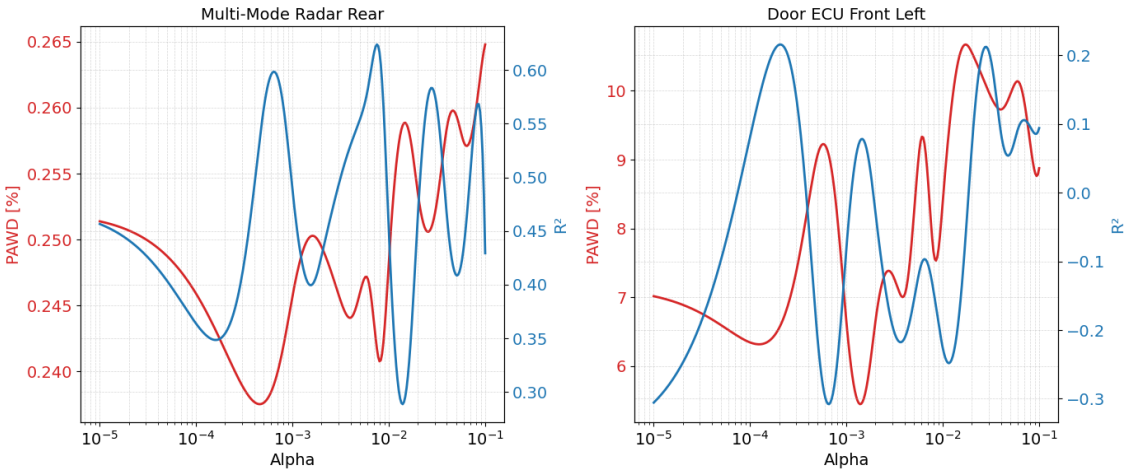

Figure 4.8: Sensitivity analysis example for the rear multi-mode radar and for the front left door ECU, and slope of the corresponding two KPI curves over different values of the `alpha` hyperparameter. Due to the sampling focus towards smaller values the x-axis is log scaled.

Table 4.10: Hyperparameter, search space, spacing, and default values for `MLPRegressor` (scikit-learn version 1.2.1).

Hyperparameter	**Search Space**	**Spacing**	**Package Defaults**
activation	[”tanh”, ”logistic”, ”relu”]	Categorical	”relu”
alpha	[0.001, 0.1] (10 values)	Equidistantly spaced	0.0001
batch_size	[16, 32, 64, 128, 256]	Power of two	”auto”
hidden_layer_sizes	Various architectures (see below)	Categorical	(100,)
learning_rate_init	[0.1, 0.01, 0.001, 0.0001, 0.00001]	Log spaced	0.001

The following scheme is used to build a search space for the MLP hidden layer architectures which is then converted to and stored in a JSON file of architectures for the hyperparameter `hidden_layer_sizes`. It results directly from the sensitivity analysis:

- No hidden layers
- Single-layer architectures: [5], [10], [15], [...], [40], [50].
- Two-layer architectures:
  - Same values: [5, 5], [10, 10], [15, 15], [...], [40, 40], [50, 50].
  - Different values: [5, 10], [5, 15], [5, 20], [5, 25], [...], [15, 30], [15, 35], [15, 40], [15, 50], [...], [50, 60], [50, 100], [50, 150].
- Three-layer architectures:
  - Same values: [5, 5, 5], [10, 10, 10], [...], [40, 40, 40], [50, 50, 50].
  - Different values: [5, 10, 5], [5, 15, 5], [...], [25, 40, 25], [25, 50, 25], [...], [50, 60, 50], [50, 100, 50], [50, 150, 50].
- Four-layer architectures:
  - Same values: [5, 5, 5, 5], [10, 10, 10, 10], [...], [50, 50, 50, 50].
  - Different values: [5, 10, 10, 5], [5, 15, 15, 5], [...], [20, 25, 25, 20], [20, 30, 30, 20], [20, 35, 35, 20], [20, 40, 40, 20], [...], [50, 60, 60, 50], [50, 100, 100, 50], [50, 150, 150, 50].

(iv) **Sensitivity Analysis for Long Short-Term Memory (LSTM)**:
For the LSTM algorithm the tunable parameters as well as their ranges are determined with the same experimental sensitivity analysis setup as for MLP and RF. They are outlined in Table 4.11 and the choice for the parameters is also supported by literature [194, 195].

Table 4.11: Hyperparameter search space, spacing, and package defaults for `LSTM` (Tensorflow Keras version 2.12.0).

Hyperparameter	**Search Space**	**Spacing**	**Package Defaults**
layer_sizes	Various architectures (see MLP above)	Categorical	User-defined
batch_size	[64, 128, 256, 512, 1024]	Power of two	32
activation	[”tanh”, ”sigmoid”]	Categorical	tanh
dropout	[0, 0.1] (continuous)	Uniform	0
learning_rate	[0.01, 0.001, 0.0001]	Log spaced	0.001

The `layer_sizes` parameter contains the same NN architecture definition options for hidden layers as determined above for MLP. However, the package Ray Tune[25] and specific functions from it are used for the hyperparameter definition, spacing and eventually tuning since there is no dedicated scikit-learn implementation for LSTM. This way, the function `tune.choice([...])` is used for drawing discrete samples for hyperparameters such as `layer_sizes`, `batch_size`, `activation`, and `learning_rate` [196]. The package function `tune.uniform([...])` is utilized to sample continuous values for the `dropout` hyperparameter given its upper and lower bounds. More details on the Ray Tune package are outlined below.

Selection of the Tuning Mechanism: Once the hyperparameters and their ranges are defined, a search strategy must be developed that meets the criteria of efficiency in terms of tuning time and results improvement. A manual tuning is considered impossible given the high number of manual trials that would need to be executed. In addition to that, a manual process would contradict the aim of establishing an automated data processing and model training pipeline for this ML task. Therefore, an automated tuning strategy must be implemented. In the supporting literature about hyperparameter tuning three strategies prevail: Grid Search (GS), Random Search (RS), and Bayesian Optimization (BO) [80, 165, 189, 194] (cf. also Section 3.7.2).
In fact, GS is the most extensive search strategy since it covers all possible parameter combinations with a brute-force approach. Nonetheless, it is impractical given the large datasets and possible parameter combinations θ_a as there are too many trials to be executed which takes too much time. Given the defined parameters for RF for example, this would result in 6.16×10^6 different combinations to be assessed. In previous preparatory experiments with the data of both Vehicles A and B, experiments have shown that BO does not provide significant improvement despite being faster than GS. These findings are supported and underlined by the fact that BO struggles with scaling and parallelization which is an impediment for large datasets. Furthermore, the influence of each hyperparameter within the set θ_a $a \in \mathcal{A}$ is not known ex ante which can also inhibit BO to be effective should none of them have a significantly greater impact on the model performance (cf. also Fig. 3.16 on the right).
Apart from this fact, Bergstra et al. state that RS outperforms GS which they found out by empirical experiments and they also underline the statements made about BO stated above and in Section 3.7.2. This is why—in this case—RS is chosen as the most promising way to find good hyperparameter configurations for the ML algorithms investigated.

- **Search Strategy Definition** As previously discussed, RS samples randomly composed sets of hyperparameters θ_{a_i} for each ML modeling algorithm $a \in \mathcal{A}$ with i being in a predefined range. This range depends on the project's constraints regarding the total processing time since for each drawing of hyperparameters a complete training and validation run on the given training dataset is conducted. In addition, the large number of available parameters needs to

[25] Python package version 2.3.0, `https://pypi.org/project/ray/2.3.0`, last accessed: December 5, 2024.

be considered, as the more parameters there are, the more trials need to be conducted to achieve significant coverage.
Therefore, it is defined that the counter index i shall range from 1 to 100: $i \in \{1, \ldots, 100\}$ wich equals 100 trials with different randomly sampled parameter combinations.

- **Search Strategy Implementation**

  (i) **Scikit-learn `RandomizedSearchCV` Class**
  For RF, GB and MLP there is an implementation available from scikit-learn which offers a constructor for RS parametrization. For this research it is used as a part of a dedicated Python wrapper class `RandomSearch.py`. The predefined parameters and their distributions are received via a private `_get_search_parameters(model_type)` function, according to the model type in use, which randomly draws parameter combinations with put back. The number of drawings is defined by the predefined `n_iter` parameter and a verbosity level for terminal (console) output is set.
  Especially worth mentioning is that `RandomizedSearchCV` performs an internal CV by splitting the data into five parts of equal length (`cv` = 5 in this case, default value). For each combination of hyperparameters, it trains the model on `cv-1` folds and evaluates it on the remaining fold. This procedure is repeated `cv` times, with each fold being used as the validation set once. Multiplied by 100 parameter drawings this results in 500 individual fittings for each ECU. The performance metrics are then averaged over the `cv` folds per fit to provide a robust estimate of the model's performance for a given θ_{a_i}.
  Finally, the regressor is fitted with the parameters found returning the best score for further processing. The detailed implementation is shown in Listing F.1 in the Appendix F, Section F.1.1.

  (ii) **Ray Tune `Tuner` Class**
  Since there is no out-of-the-box scikit-learn implementation for LSTM hyperparameter optimization—as stated previously—a special wrapper function `_get_best_param_ray([...])` is defined as part of the superordinate class `RandomSearch.py` using the parameter tuning package Ray Tune. The search strategy and parametrization is similar to the other three ML modeling algorithms described above and the detailed implementation is shown in Appendix F, Section F.1.2, Listing F.2.

Detailed View 3: Model Validation Strategy

Selection of a Cross-Validation Design: Another part of the CRISP-ML(Q) process model step 3 "model generation" is the validation of the generalization capability of the ML model [91]. Therefore, CV is performed, where the training data, consisting of a given number of test drives $D_t - 1$ (one drive is already put aside as test data), is partitioned into the same number of subsets. For each model training of a power consumer during the CV stage, another drive is excluded from the training process, and the model is then fitted with the $D_t - 2$ remaining drives (cf. Section 4.1.1). This process is repeated $D_t - 1$ times.

Since one test drive (a distinct subset of the training data) is iteratively excluded from the fitting process, this procedure is also referred to as *leave-one-out-cross-validation* (LOOCV, cf. also Section 4.1.6). Standard train-test split ratios (e.g., 70:30 or 80:20 [197]) are not used here to avoid mixing data from different drives, which could lead to unrealistic driving scenarios at timestamps where one drive transitions into another.
All metrics in the results part (cf. Section 4.3) are then averaged over all $D_t - 1$ CV runs in order to obtain a robust statement on a model's generalization performance and correctness [198]. Figure 4.9 illustrates the generalized LOOCV process with $D_t - 1$ test drives and resulting model fits. For simplicity reasons all drives are depicted as of equal length which is however not the case in this project. In addition to that, it is visualized that each LOOCV fit produces a regressor model and the respective KPIs which are then averaged as explained before.
All results presented in Section 4.3 use this LOOCV strategy.

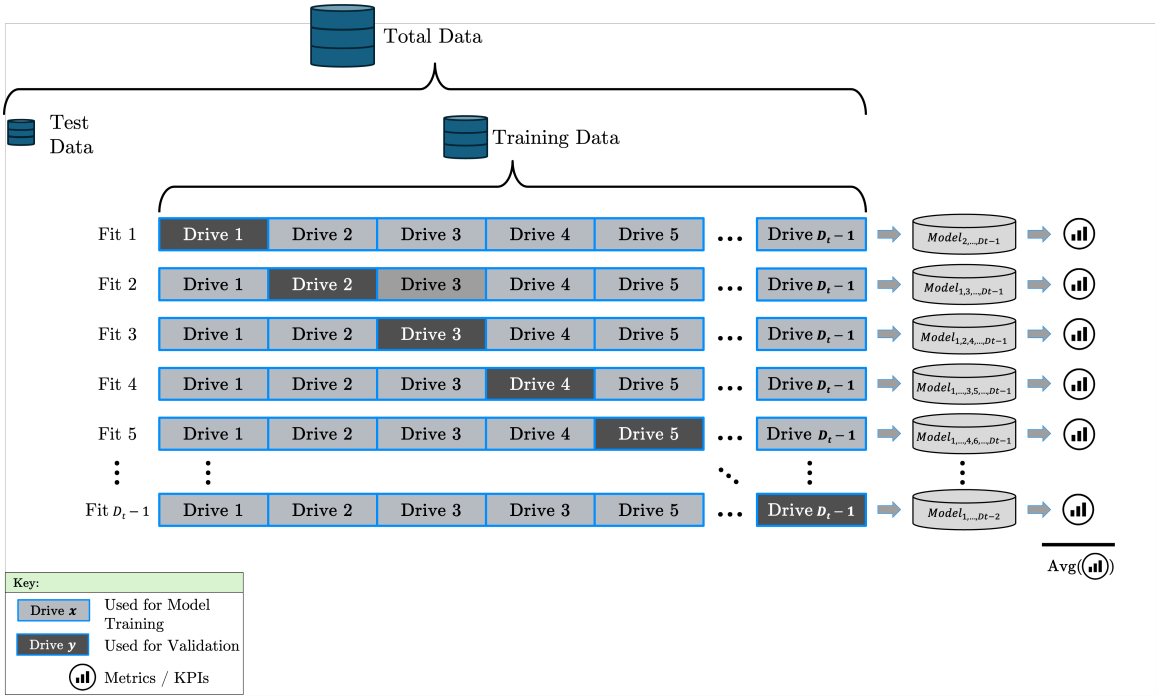

Figure 4.9: Schematic depiction of the LOOCV strategy applied in this research aiming at preserving the integrity of the test drive data. Additionally, the test dataset remains untouched in this stage, too. Only the actual training data is used.

LOOCV Implementation Strategy: The implementation to realize the LOOCV strategy for this project follows a modular implementation strategy as well, so the functionalities around creating CV splits are encapsulated in a dedicated class `CrossValidation.py`. The class is particularly designed for segmented data—e.g. distinct test drives, identified by a unique file name. Each test drive has one dedicated and unique filename which is also part of the imported data so these identifiers can be used to distinguish between the respective drives and to subsequently build the CV folds accordingly. To be more precise, this means that drives (splits) are separated by that identifier.
Ultimately, a compiled list of CV splits which can then be further processed by the

four investigated ML modeling algorithms in the set $\mathcal{A}$ is returned. By this standardized and reproducible procedure it is ensured that for a detailed comparison, every algorithm is fed with the same (split) data. Avoiding such sources of randomness is a contribution to consistency in the assessments conducted and to the assumptions made in the following results and discussion sections.
The detailed Python implementation of the `CrossValidation.py` class together with its associated class methods is shown in Appendix F, Section F.2, Listing F.3.

Detailed View 4: Regressor and KPI Export

Another work item of this research is the assessment of the model's predictive capabilities on a selection of example ECUs. To be able to make this assessment, certain KPIs and metrics as well as model sizes must be calculated and made accessible. As already shown in Fig. 4.9, this is realized by the calculation of the average values of all CVs conducted during a training run.

Regressor Export: As previously introduced in Section 4.1.6, the trained models (regressors) are exported in the standardized ONNX format, which ensures that they are both comparable and interchangeable across platforms and tools. To enable this functionality in the code, a so-called *export* flag `-e` can be set as a command-line parameter, which in turn sets the corresponding boolean flag in the configuration to the value `True`. When this is the case, the export function in the orchestrating `Model.py` class is triggered. The actual functionality of the export process is encapsulated in the dedicated `ModelExport.py` class, also adhering to the general object-oriented programming paradigm for better modularity. This means, an object is then instantiated, which facilitates the export process through the public methods it makes available which can then—again—be handled centrally from the overarching `Model.py` class described later in this section. The actual Python implementation of the `ModelExport.py` class is shown in Appendix F, Section F.3, Listing F.4
It is also possible to export metadata and model information from a given training run. An example of a metadata export from a pipeline run for the driver display of Vehicle A, in a human- and machine-readable JSON format, is provided in Appendix G.

KPI and Metrics Export: The KPI export is organized similarly as the previously explained regressor and metadata export. Some KPIs are already part of the metadata JSON file (cf. Appendix G). Nonetheless, a holistic collection of necessary KPIs is achieved through the dedicated `main_collector` and `model_collector` objects which are then used to prepare an Excel (*.xlsx*) file listing the respective KPIs in a structured way. One line per complete run of the ML pipeline. This means, these KPIs are thus averaged values over all CV runs. Figure 4.10 shows the results of 20 training runs for an example ECU. More KPIs are not displayed for brevity reasons, however the most important ones identified in Section 4.1.6 as well as selected meta data such as the number of CVs, the longest respectively shortest CV, the number of features (bus signals) involved or the ML modeling algorithm used are recorded.

C	D	E	F	H	J	AA	AB	AE	AF	AG	AJ	AK	AL
N° CVs	Min. CV length [s]	Max. CV length [s]	Frequency [Hz]	ML Algorithm	N° Features	SD(PAAD) over all X-vals [%]	X-val weighted PAWD [%]	X-val avg. R^2	SD(R^2) over all X-vals	X-val weighted R^2	MSE [A_2]	SD(MSE) [A_2]	X-val weighted MSE [A_2]
14	976	6634	5	randomforest	972	13,399	9,067	-1,225	2,579	-0,510	0,005	0,005	0,006
14	976	6634	5	mlp	972	11,669	7,235	-0,686	2,196	-0,050	0,004	0,005	0,004
14	976	6634	5	lstm	972	21,666	9,471	-68,536	245,508	-34,220	0,007	0,011	0,006
14	976	6634	5	randomforest	80	4,530	3,080	0,258	0,518	0,431	0,002	0,002	0,003
14	976	6634	5	mlp	80	7,696	6,792	-0,620	3,395	-0,650	0,003	0,004	0,005
14	976	6634	5	lstm	80	19,196	8,652	-2,007	4,367	-0,818	0,007	0,012	0,006
14	976	6634	5	randomforest	80	2,650	2,576	0,588	0,287	0,652	0,002	0,002	0,002
14	976	6634	5	mlp	80	52,575	49,105	-79,440	145,360	-51,312	0,070	0,086	0,102
14	976	6634	5	lstm	80	13,458	8,717	-0,734	1,644	-0,405	0,007	0,009	0,009
14	976	6634	5	lstm	80	9,484	8,390	-0,105	0,789	0,084	0,004	0,004	0,005
14	976	6634	5	gradientboosting	972	3,204	2,555	0,411	0,772	0,626	0,001	0,001	0,002
14	976	6634	5	gradientboosting	6	1,389	1,286	0,627	0,309	0,709	0,001	0,001	0,001
14	976	6634	5	gradientboosting	6	1,368	1,335	0,629	0,320	0,707	0,001	0,001	0,001
14	976	6634	5	randomforest	972	13,983	8,916	-1,185	2,203	-0,566	0,006	0,007	0,006
14	976	6634	5	randomforest	80	5,147	3,514	-0,031	1,013	0,268	0,003	0,003	0,003
14	976	6634	5	randomforest	80	3,483	2,595	0,177	0,746	0,425	0,002	0,002	0,002
14	976	6634	5	gradientboosting	6	1,368	1,335	0,629	0,320	0,707	0,001	0,001	0,001
14	976	6634	5	gradientboosting	6	1,389	1,286	0,627	0,309	0,709	0,001	0,001	0,001
14	976	6634	5	gradientboosting	6	1,368	1,335	0,629	0,320	0,707	0,001	0,001	0,001
14	976	6634	5	gradientboosting	6	1,368	1,335	0,629	0,320	0,707	0,001	0,001	0,001

Figure 4.10: Excerpt of a result Excel file, showcasing the most relevant KPIs and metrics of one of the ECUs for 20 individual training runs with different configurations of the ML pipeline arranged one below the other.

Detailed View 5: XAI Method Implementation

Once the model training is completed, all the necessary data, information, and other precursors necessary to conduct XAI are available. These necessary elements notably consist of a trained regressor model and the associated engineered training and validation data. The corresponding XAI Python implementation follows the already presented strategy where thematically related functions are summarized in one class. In the final code, any XAI method from Section 3.6 can thus be retrieved by creating an `Explanation.py` object and calling the desired encapsulated function. This modular logic is depicted in the UML diagram in Fig. 4.11. In that case, the class name is `Explanation` and—as shown in the UML diagram—the components `X_trn`, `X_val`, `y_trn` and `y_val` for the training and validation data, the trained model (regressor) `regr` and the `fuse` to identify the target name are attributes of such an object. Especially for the SHAP explainer the feature names `feature_columns` and the absolute number of targets `num_fuses` is also needed.
All XAI methods are primarily calculated on the training data (even though for some the validation data is needed, too) which ensures that the explanations are done with the patterns the model was trained with, providing insights into the reasoning behind the predictions based on that original data.
The actual instantiation of the class is depicted in Listing F.6 (cf. Appendix F, Section F.5).

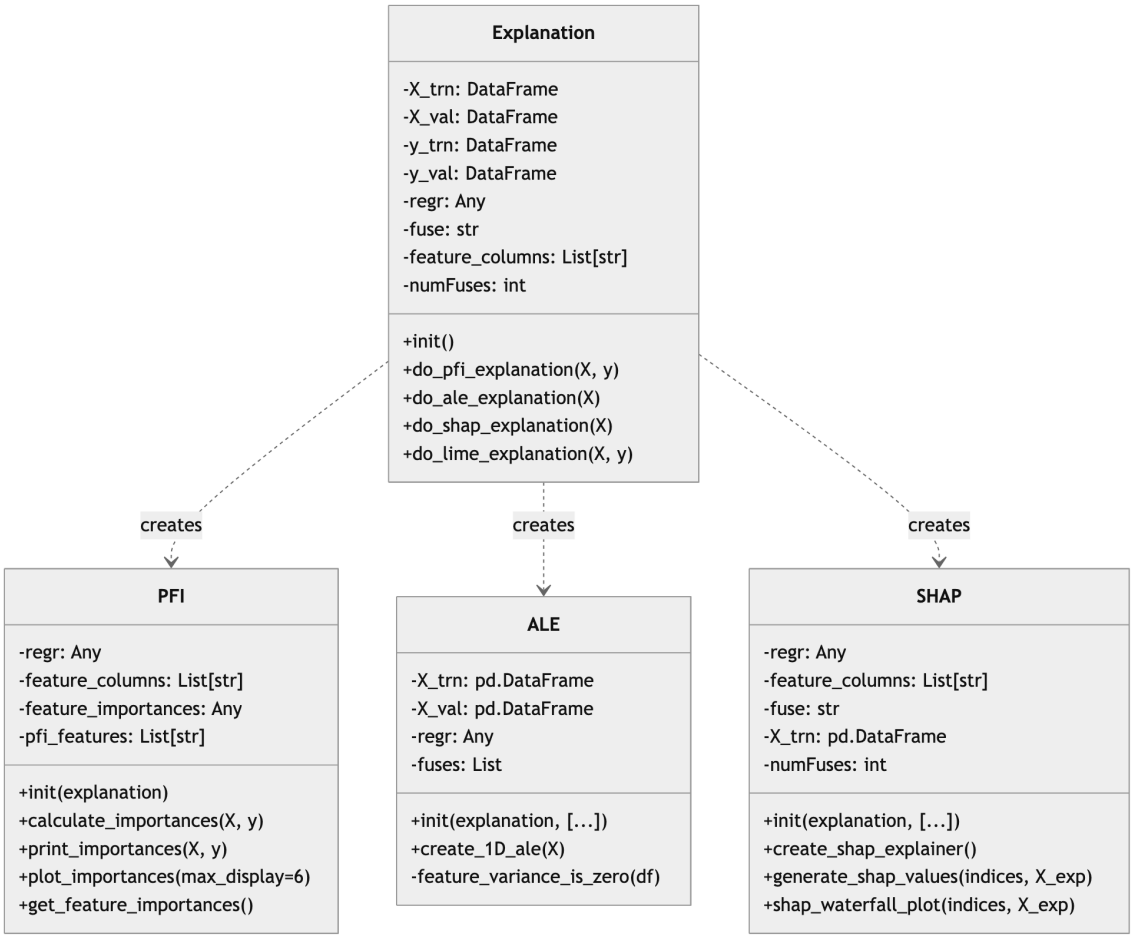

Figure 4.11: Representation of the XAI implementation strategy in a UML format. The actual XAI logic is encapsulated in the corresponding classes, callable through functions of a more generic `Explanation` object.

Permutation Feature Importance (PFI): The function responsible for PFI, which wraps the project-specific PFI Python implementation, is assigned the name `do_pfi_explanation(self, X, y)` and is shown in Listing 4.2. It supports all examined ML modeling algorithms defined in $\mathcal{A}$. However, since it relies on randomly shuffling feature values, special caution must be taken when interpreting results, particularly for LSTM models. The shuffling may disrupt potential (assumed) temporal dependencies, thereby impacting the reliability of the explanations for sequential data. The PFI implementation used in this research leverages the scikit-learn `sklearn.inspection` API[26] [92].

[26] `https://scikit-learn.org/1.5/api/sklearn.inspection.html`, last accessed: December 8, 2024.

```
1 def do_pfi_explanation(self, X_trn, y_trn) -> List[Tuple[float, str
    ]]:
2     ''' Here all class variables of PFI.py are made available to
          the Explanation.py class '''
3     pfi_explainer = PFI(self)
4     pfi_explainer.print_importances(X_trn, y_trn)
5     pfi_explainer.plot_importances()
6     self.pfi_features = pfi_explainer.pfi_features
7 
8     return pfi_explainer.get_feature_importances()
```

Listing 4.2: Object-oriented Python implementation of the `do_pfi_explanation([...])` function. The actual implementation is encapsulated in the `get_feature_importances()` function.

The wrapper function creates an instance of the `PFI.py` class (cf. Appendix H) to compute permutation feature importances, log them to a file (cf. Listing 4.2, line 4), and visualize the top features in a bar plot (cf. Listing 4.2, line 5). The calculated importances are stored in `self.pfi_features`—a class variable of `Explanation`—for further use. They are returned as a sorted list of tuples containing feature names and their importance scores with `get_feature_importances()`.

Accumulated Local Effects (ALE): In the case of ALE, the same logic as for PFI is applied. The `do_ale_explanation(self, X_trn)` function is also part of the `Explanation.py` class. However, the underlying calculation logic is drawn from the package `PyALE`[27] encapsulated inside an `ALE` object (cf. Listing 4.3, line 3).
For this research, the focus shall be on the analysis of one feature at a time with the target variable (1-dimensional ALE) only. To do so, the respective regressor needs to be handed over to the `ALE` class upon generation of the `ale_exp` object in order to generate the relevant predictions (cf. Appendix I).

```
1 do_ale_explanation(self, X_trn) -> None:
2     ''' Creation of an ALE object. An instance of the dedicated ALE
          class. '''
3         ale_exp = ALE(
4             self,
5             X_trn,
6             self.regr,
7             self.fuses,
8         )
9         ''' Trigger execution of ALE value calculation and plot
             generation. '''
10        ale_exp.create_1D_ale(X_trn, "X_trn")
11 
12        return None
```

Listing 4.3: Object-oriented Python implementation of the `do_ale_explanation([...])` wrapper function for the creation of 1-dimensional plots.

[27] Package version 1.1.3, available at `https://pypi.org/project/PyALE/1.1.3`, last accessed: December 8, 2024.

The function `create_1D_ale(self, X, path)` (Listing 4.3, line 10) first creates directories for storing plots according to the `path` variable, then it iterates over all features of `X`, checking for sufficient variance to ensure an impactful analysis. For "valid" features, ALE values are computed. Features with significant effects are visualized in plots, saved as portable network graphics (PNG or *.png*) files. A feature is considered "valid" when its value variations cause a current prediction difference (range) of at least 0.1 A, otherwise no plot is created due to lacking significance. This functionality is needed to avoid the creation of numerous plots which have no further use as even after feature selection there can still be several hundreds of features left for ML model training (cf. Table 4.12).

Shapley Additive Explanations (SHAP): As a third XAI element, SHAP value and SHAP explanation generation is integrated into the `Explanation.py` class. The logic is similar to the two aforementioned explainers where within `Explanation.py` inside the dedicated function `do_shap_explanation(self, X)` an object `shap_exp` of the underlying `SHAP` class is created and all the variables needed for this constructor are supplied through the class variables of `Explanation.py`. Hence, the created object can be used to call `shap_waterfall_plot()` to create a waterfall plot, visually demonstrating either the positive or negative impact of the most important features on the local prediction. To be more precise, the plot then presents the feature-wise justification for the deviation of a given prediction $\hat{y} = f(x_{i,j})$ at one specific timestamp (SHAP is a local XAI method) from the expected value $E[f(x_{i,j})]$, $\forall(i,j) \in \{1,\ldots,m\} \times \{1,\ldots,n\}$ of all predictions (cf. Mueller et al. [26]). The implementation logic of SHAP is referenced in Listing 4.4. The similarity with the aforementioned two implementations is intended and does not only serve a better structure and readability of the code base but also contributes to the object-oriented, modular code design.

```
1  def do_shap_explanation(self, X) -> None:
2      ''' Creation of a SHAP object. An instance of the dedicated
           class. '''
3          shap_exp = SHAP(
4              self,
5              self.regr,
6              self.feature_columns,
7              self.X_trn,
8              self.fuse,
9              self.numFuses
10         )
11
12     ''' Trigger execution of SHAP value calc. depending on the
           regressor type and plot generation. '''
13         shap_exp.shap_waterfall_plot()
14
15         ''' Keeping SHAP-relevant features as Explanation class
               variables '''
16         self.shap_features = shap_exp.imp_features
17
18         return None
```

Listing 4.4: Object-oriented Python implementation of the `do_shap_explanation([...])` wrapper function within the `Explanation` class.

The SHAP value calculations are conducted using the Python package `shap`, version 0.41.0[28] [157]. For MLP, SHAP is applied in its standard configuration. However, *TreeSHAP* is employed for random forests (RF) and gradient boosting (GB) models, as it is specifically optimized for tree-based ML algorithms, as described in Section 3.6 [158]. Consequently, differentiation between regressor types must be implemented within the `SHAP` class (cf. Appendix J). The SHAP values are then computed accordingly. For LSTM, explainability is more challenging. While `shap` provides *DeepExplainer* and *KernelExplainer* for deep learning models, these explainers are less optimized for sequential data and may result in higher computational costs or less interpretable outputs. This poses limitations in the explainability of LSTMs compared to tree-based models.

On a Modular Python Implementation - Using the Example of XAI

The explainer object needed to call the functions introduced in the sections above is optionally created in the superordinate class `Model.py` which has been mentioned several times already. The UML diagram depicted in Fig. 4.12 shows the modeling class with the most important parameters and functions discussed. The `Explanation` class in Fig. 4.12 is the same as shown in Fig. 4.11. It can be seen that for each ML modeling algorithm a different object is created based on the configuration. The same logic is applied to the XAI class (`Explanation.py`). Public class methods in the related classes make their respective functionalities accessible to the model class thus allowing the centralized flow control of the model-related part of an entire pipeline run through this instance. The actual implementation of the related functions (e.g. `fit(X, y)` or `predict(X)`) is encapsulated in the corresponding classes allowing for individual implementations should the need arise. This structure is also used for other functionalities implemented in `Model.py` to control the different elements of the overall code: It is used for the actual model creation depending on the desired modeling algorithm $a \in \mathcal{A}$ (cf. Appendix F, Listing F.5, line 23 ff) as well as for the fitting of the regressor, for the collection of metrics and KPIs, and for the model export (the latter two not shown for brevity). This way, the flow of a part of the pipeline, namely everything downstream of data preprocessing, can be controlled centrally in `Model.py`.
A centralized yet modular code structure as such enables the addition or removal of modules—such as new XAI methods, metrics, or ML modeling algorithms—with minimal effort. The program can then be parameterized via the command line, whose parameters are read in and then stored in a central configuration class (`Config.py`) so that they can then be made accessible at any location by means of an import `from config import Config` (cf. Listing F.5, line 12).

[28] `https://pypi.org/project/shap/0.41.0`, last accessed: December 8, 2024.

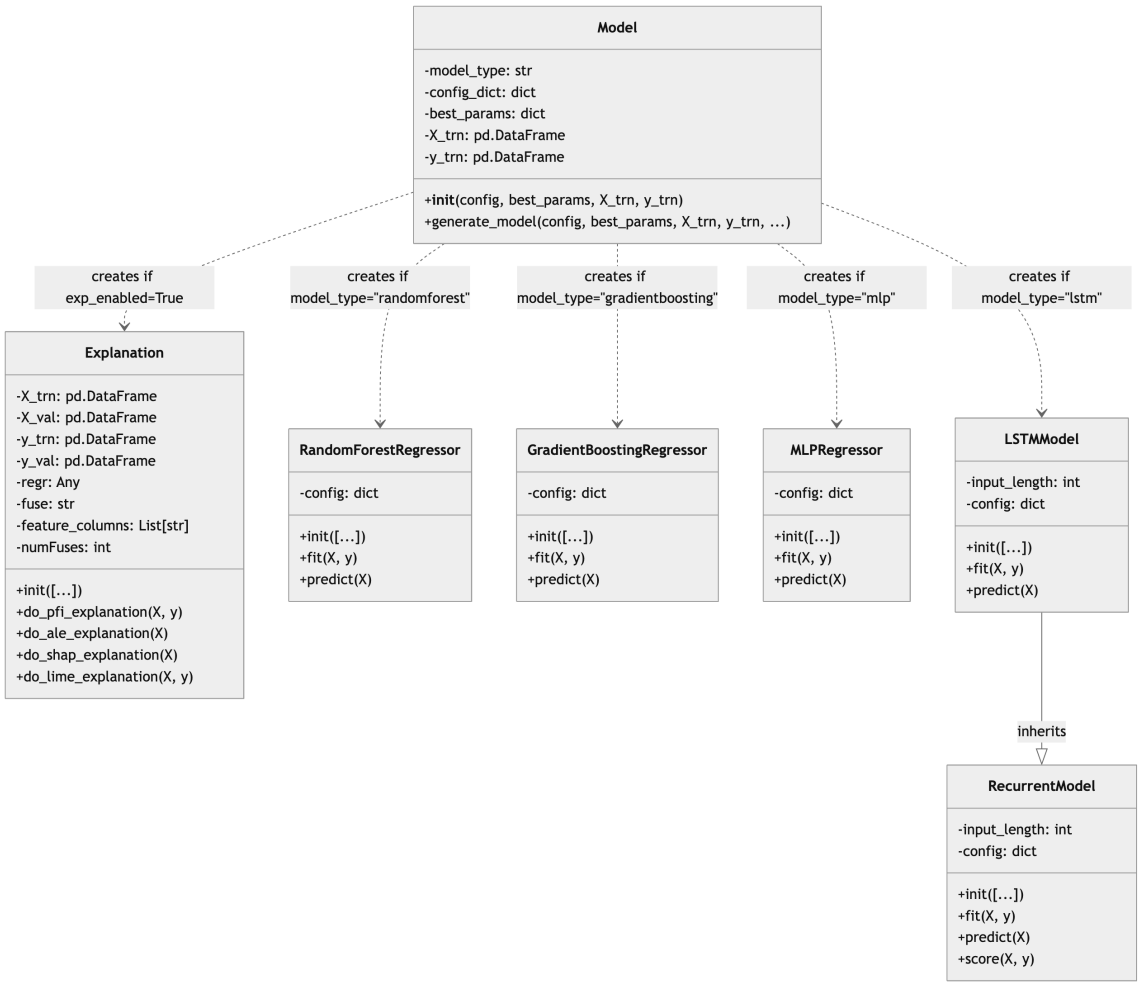

Figure 4.12: UML diagram of an excerpt of the `Model.py` class focusing on the model and explainer creation (other functionalities omitted for brevity).

4.3 Modeling and Evaluation Results

Given the implementation of the ML data processing and training pipeline just introduced, this section deals with its application to real-world data. This includes the full use of the implemented functionalities as described in the previous sections as well as the presentation of the results of each important process step. Hence, this complies with the CRISP-ML(Q) step 4 which involves the quality assessment. All results subsequently presented involve the 18 ECUs present in the set of power consumers $\mathcal{E}$ compiled through the selection process in Section 4.1.5. Eventually, a selection for one ML modeling algorithm is made, hence all subsequent considerations (notably XAI) are conducted utilizing that selected regressor.
However, due to a large number of resultant evaluations and runs some results are presented in an aggregated way, while selected methods are still discussed at a detailed ECU-level. Additional details can be found in the related referenced appendices.

4.3.1 Feature Engineering and Selection

Starting with the data preprocessing pipeline, the results for the feature engineering and selection are assessed by the reduction of the features and the capabilities of the pipeline to improve the respective ML model's prediction quality KPIs.

Feature Reduction

After applying the feature engineering and selection pipeline, the first observation is that the number of remaining signals per ECU is reduced significantly for both the RF- and GB-based data preprocessing runs. As shown in Table 4.12, the reduction ratios range from -73.1 % for the fuel supply ECU (Vehicle A, RF-based) until up to -99.9 % for the steering column ECU (Vehicle B, GB-based). The mean of all feature reductions is -90.2 % for RF-based (also used for MLP and LSTM regression later on) and -97.7 % for GB-based reduction respectively. These high reduction ratios show the effectiveness of the entire preprocessing pipeline in terms of one of its goals which is to reduce the number of features thus model size significantly.

Table 4.12: Results of the signal reduction per ECU from $\mathcal{E}$ after applying the feature engineering and selection pipeline to the raw input data. In brackets: results for GB-based feature reduction. In boldface: highest and lowest reduction ratios.

ECUs Vehicle A	**N° of Raw Features (GB)**	**N° of Engineered Features (GB)**	**Ratio (GB) [%]**
BCF	3410	223 (63)	-93.6 (-98.2)
Extractor Fan	138	29 (9)	-79.0 (-93.5)
CID	972	95 (21)	-90.2 (-97.8)
Fuel Supply ECU	**216**	**58** (6)	**-73.1** (-97.2)
Coolant Pump	138	26 (4)	-81.2 (-97.1)
Right Pixel Headlamp	1334	128 (29)	-90.4 (-97.8)
Left Pixel Headlamp	1334	108 (25)	-91.9 (-98.1)
Adaptive Suspension	5399	150 (40)	-97.2 (-99.3)
Driver Display	972	80 (6)	-91.8 (-99.4)
ECUs Vehicle B	**N° of Features (raw)**	**N° of Features (engineered)**	**Ratio (GB) [%]**
Right Headlamp	1386	91 (15)	-93.4 (-98.9)
Left Headlamp	1386	99 (3)	-92.9 (-99.8)
Steering Column ECU	**5608**	143 **(5)**	-97.5 **(-99.9)**
Coolant Pump	134	11 (3)	-91.8 (-97.8)
Seat ECU Driver	1064	61 (14)	-94.3 (-98.7)
Door ECU Front Left	1599	147 (55)	-90.8 (-96.6)
BCF	2748	190 (29)	-93.1 (-98.9)
CID	987	150 (36)	-84.8 (-96.4)
Active Air Suspension	5608	152 (22)	-97.3 (-99.6)

Table 4.13 presents a detailed comparison of the feature engineering pipeline's impact across the different ML algorithms in $\mathcal{A}$ regarding the quality performance metrics and KPIs only (excluding efficiency KPIs), which are a subset of $\mathcal{K}$ (cf. Section 4.1.6). Each metric is evaluated for its optimization target (either increase

or decrease), and the effect of the feature engineering pipeline is quantified in terms of the average change in performance over all considered ECUs and the number of enhanced trial runs compared to a run without data preprocessing.

Quality KPIs: Random Forest and Gradient Boosting

Therefore, Table 4.13 shows that the feature engineering pipeline offers a clear benefit for RF, with an average of 13.5 enhanced trials[29] out of 18. For example, the **weighted R^2 over all CVs** shows a significant improvement of 58.01 % on average, with 14 out of 18 trials showing an enhancement according to the definition.
In contrast, MLP has fewer amended experimental runs (8.875 on average) and generally shows less improved KPIs than RF. Nonetheless, the pipeline still helps to achieve a 7.12 % increase in **X-val weighted R^2** over all GB runs, though with a lower number of enhanced trials (9 out of 18). The key takeaway here is that RF benefits more consistently from the feature engineering pipeline, whereas GB shows mixed results, with certain metrics such as **X-val weighted PAWD** indicating a slight reduction in the average performance and **X-val avg. R^2** comprising significant negative outliers with -260.95 % on average (even though it still shows 9 out of 18 amended trials).

Quality KPIs: Multi-Layer Perceptron and Long Short-Term Memory

The evaluations for MLP however, show only a slightly reduced average of amended trials compared to GB whereas there is only one KPI (**X-val weighted R^2**) which improves—on average—by 11.92 %. Another evident result is that the dispersions of **MSE** and **R^2** rise significantly on average by 27.07 % and 65.26 % respectively. LSTM reveals moderate improvements, with an average of 9 enhanced trials out of 18. For instance, **PAWD** increases by 15.72 %, although the number of individual improvements is lower compared to RF, GB or MLP.
The analysis of the feature engineering pipeline across different ML algorithms shows varying degrees of improvement depending on the modeling algorithm and the actual ECU under investigation. RF shows robust performance across a variety of metrics as all optimization targets are met on average. GB achieves three KPIs which are improved on average with 8.875 trials showing more than half of the metrics ameliorated. MLP, however, shows only moderate advancements, with only one optimization target actually met (**X-val weighted R^2**). A view on the detailed data shows significant scattering in the effects of the feature reduction on the prediction itself. LSTM achieves just moderate improvements as well (here also just one optimization target is met), although with fewer enhanced trials compared to RF and slightly more compared to GB and MLP.
The actual performance of the ML models, together with the actual values of the performance KPIs also taking into account the efficiency KPIs (the entire set $\mathcal{K}$, is then evaluated in the weighted sum analysis in Section 4.3.3. There, it is combined with the results of the hyperparameter tuning, too.

[29] A trial is considered "enhanced" when more than half (>50 %) of the quality KPIs have a better value compared to the respective other experimental run.

Table 4.13: Performance comparison of the feature engineering and selection pipeline with no data preprocessing across the model quality KPIs and the ML modeling algorithms. The table is split into two parts: one for RF and GB, and one for MLP and LSTM. For a facilitated reading, the optimization directions for each KPI are indicated in the second column.

Performance Metric	Optimization Direction	RF		GB	
		Avg. Change [%]	Enhanced Trials	Avg. Change [%]	Enhanced Trials
X-val weighted PAWD [%]	Reduce	-17.23	14	0.16	10
X-val weighted R^2	Increase	58.01	14	7.12	9
SD(PAAD) [%]	Reduce	-5.14	11	10.72	9
X-val avg. R^2	Increase	51.60	13	-260.95	9
SD(R^2)	Reduce	-11.92	12	15.27	9
X-val weighted MSE [A^2]	Reduce	-15.97	15	-1.85	8
MSE [A^2]	Reduce	-15.79	15	-1.0	8
SD(MSE) [A^2]	Reduce	-17.46	14	1.56	9
Average Enhanced Trials			**13.5**		8.875

Performance Metric	Optimization Direction	MLP		LSTM	
		Avg. Change [%]	Enhanced Trials	Avg. Change [%]	Enhanced Trials
X-val weighted PAWD [%]	Reduce	5.49	10	15.72	9
X-val weighted R^2	Increase	11.92	10	-16.09	9
SD(PAAD) [%]	Reduce	12.53	7	8.53	8
X-val avg. R^2	Increase	-20.94	8	19.65	10
SD(R^2)	Reduce	65.26	8	7.92	11
X-val weighted MSE [A^2]	Reduce	27.63	9	33.42	9
MSE [A^2]	Reduce	30.72	8	15.89	9
SD(MSE) [A^2]	Reduce	27.07	9	55.13	7
Average Enhanced Trials			8.625		9

Results on Trial Level

At the level of the individual experiments (cf. Appendix K, Tables K.1 to K.4), it becomes evident that feature engineering and selection generally perform better on Vehicle A than on Vehicle B especially for the MLP and LSTM regressors. In addition to that, the overall performance varies significantly with the algorithm used as for example with the active air suspension of Vehicle B. As a result of that, depending on the baseline behavior without any optimization made beforehand, the actual performance of the feature engineering pipeline depends on the general capabilities of an algorithm. In other words, the better the baseline performance without preprocessing, the smaller the potential for additional improvements through feature engineering and selection.

4.3.2 Hyperparameter Tuning Results

One additional step to enhance ML model performance even further, is the execution of the hyperparameter tuning, notably the random search (RS), as described in Section 4.2.2 using the feature engineered and selected data.
The high-level results of the hyperparameter tuning process step are presented in Table 4.14 and provide insights into the performance impact of RS on the investigated

ML modeling algorithms in $\mathcal{A}$. They are evaluated relative to and as a subsequent process of the observations made during the feature engineering phase. Hence, insights into the joint effect of feature engineering and hyperparameter tuning can be gained from these results.

Additional Hyperparameter Tuning Impact - RF and GB

Random forest maintains its status as the most consistently performing algorithm, also after hyperparameter tuning. With an average of 13.375 enhanced trials[30] out of 18, RF shows stable improvements across all key metrics. For instance, the **X-val weighted R^2** increases by 46.7 % on average compared to just feature engineering and selection and the number of enhanced trials remains high at 16. This aligns with the results observed for RF in the feature engineering phase. Similarly, RF achieves a significant reduction in metrics like **PAWD** (-19.3 %) and **X-val weighted MSE** (-7.9 %) as intended.
Gradient boosting demonstrates fewer consistent enhancements than RF, still showcases meaningful performance improvements across some key metrics. It achieves an average of 11.125 enhanced trials out of 18. Notable improvements include a reduction in **X-val weighted MSE** by -4.1 % and increases in metrics such as **X-val avg. R^2** in 13 trials. However, **PAWD** for example shows an unintended average increase of 6.5 %, reflecting variability in the optimization impact of the hyperparameter tuning.

Additional Hyperparameter Tuning Impact - MLP and LSTM

The performance of MLP and LSTM after the hyperparameter tuning demonstrates differences in their optimization consistency and impact. Compared to RF and GB, MLP and LSTM—both NN-based algorithms—show more variable results, with especially MLP struggling to achieve performant improvements regarding the evaluated quality metrics.
MLP achieves an average of just 5 enhanced trials out of 18, with some metrics showing extreme deviations caused by outliers (cf. detailed results in Appendix K, Tables K.5 to K.8). For example for **X-val weighted PAWD**, MLP achieves an average increase of 384.3 % instead of a reduction. Similarly, **X-val weighted R^2** suffers from a significant average decrease of -62,536.1 %, mainly caused by the seat ECU driver (Vehicle B) and by the driver display (Vehicle A), highlighting the challenge in tuning MLP effectively within the chosen RS strategy. Although a few trials show slight improvements, the overall performance of MLP is more unreliable compared to RF, GB, and even LSTM.
In contrast, LSTM demonstrates much more stable and significant improvements, achieving an average of 10.25 enhanced trials out of 18. This includes for example a reduction of the MSE dispersion **SD(MSE)** by -6.5 % and an increase in **X-val avg. R^2** by 25.1 % over all 18 trials. Nonetheless, LSTM does not outperform RF or GB in overall performance. However, it is more competitive and shows promising results as an alternative approach for individual power consumers. In addition to that, the reduction in **SD(PAAD)** (-0.8 %) and the improvement in **X-val weighted R^2** (+1.0 %) further support LSTM's utility in this context.

[30] Enhancement based on the resultant data after the feature engineering and selection pipeline.

Table 4.14: Performance comparison of the RS search strategy across the model quality KPIs and the investigated ML modeling algorithms. The table is split into two parts: one for the RF and GB regressors and one for MLP and LSTM. For a facilitated reading, the optimization directions for each KPI are indicated in the second column.

Performance Metric	**Optimization**	**RF**		**GB**	
	Direction	**Avg. Change [%]**	**Enhanced Trials**	**Avg. Change [%]**	**Enhanced Trials**
X-val weighted PAWD [%]	Reduce	-19.3	15	6.5	6
X-val weighted R^2	Increase	46.7	16	-23.8	12
SD(PAAD) [%]	Reduce	-16.7	12	9.9	12
X-val avg. R^2	Increase	42.3	14	-22.7	13
SD(R^2)	Reduce	-16.8	13	43.2	11
X-val weighted MSE [A^2]	Reduce	-7.9	14	-4.1	13
MSE [A^2]	Reduce	-7.5	12	-1.2	12
SD(MSE) [A^2]	Reduce	-2.6	11	3.2	10
Average Enhanced Trials			**13.375**		11.125

Performance Metric	**Optimization**	**MLP**		**LSTM**	
	Direction	**Avg. Change [%]**	**Enhanced Trials**	**Avg. Change [%]**	**Enhanced Trials**
X-val weighted PAWD [%]	Reduce	384.3	5	78.5	10
X-val weighted R^2	Increase	-62536.1	6	1.0	10
SD(PAAD) [%]	Reduce	420.0	4	-0.8	10
X-val avg. R^2	Increase	-9956.0	3	25.1	10
SD(R^2)	Reduce	5669.9	7	18.5	11
X-val weighted MSE [A^2]	Reduce	-2016.2	6	2.5	11
MSE [A^2]	Reduce	2242.9	4	1.6	11
SD(MSE) [A^2]	Reduce	5288.6	5	-6.5	9
Average Enhanced Trials			5		10.25

Remarks on Trial Level

On an individual trial level (cf. Appendix K, Tables K.5 to K.8) the MLP evaluations show significant negative outliers especially for the $\mathbf{R^2}$ metric for both Vehicle A (e.g. driver display) and Vehicle B (e.g. seat ECU driver and steering column ECU) after the hyperparameter tuning. They are especially responsible for causing the overall MLP results to be negative. There are also outliers present regarding the **X-val weighted MSE** with the fuel supply ECU of Vehicle A contributing negatively to the overall MLP results, too.

Visual Evaluation Using the Driver Display as an Example

The effect of the ML pipeline on the quality of the predictions can also be assessed visually. Figures 4.13, 4.14, and 4.15 depict the modeling results of the first CV iteration of the driver display ECU of Vehicle A. They are obtained with the raw and the feature-engineered data as well as with additional RS-based hyperparameter tuning (in that order). They show plots which are divided into three subplots each. For privacy reasons the entire data has been min-max-scaled before being plotted, hence retaining its qualitative statements.
In each subplot the blue line represents the **training data** and the orange line the

associated **prediction** made by the corresponding ML model.
The top subplot depicts the measurement and prediction of the training data. There, the blue and orange curves are almost congruent in Figures 4.13 and 4.14 since the training data is already memorized by the model (as it has been trained with it). However, in Fig. 4.15, where an additional RS is performed, there is more of the actual measurement visible (top subplot, blue line), which demonstrates less overfitting of the training data.
The middle plot is the test drive used for that very CV fold. This means it is not part of the training data in that particular run, hence the corresponding model has not "seen" that data beforehand during the training and it thus serves as **validation data**. It therefore represents the actual prediction for that test drive using the ML model generated from the data of the top subplot. Comparing the three figures, it becomes visible that for this ECU each additional step of the ML pipeline makes the prediction more accurate as the blue and orange lines gradually become more congruent (cf. red circle in the middle subplot in all three Figures).
The effect is underlined as well by the lower subplot which shows the measurement and prediction of the electric charges which are the integral over the current consumption over time (cf. Equation 3.4). The comparison of the blue and orange lines of the lower subplot indicates a cumulative error. Comparing the figures subsequently, the curves also become more congruent (cf. red circle in the lower plot), hence the overall error diminishes the more sophisticated the ML pipeline is.

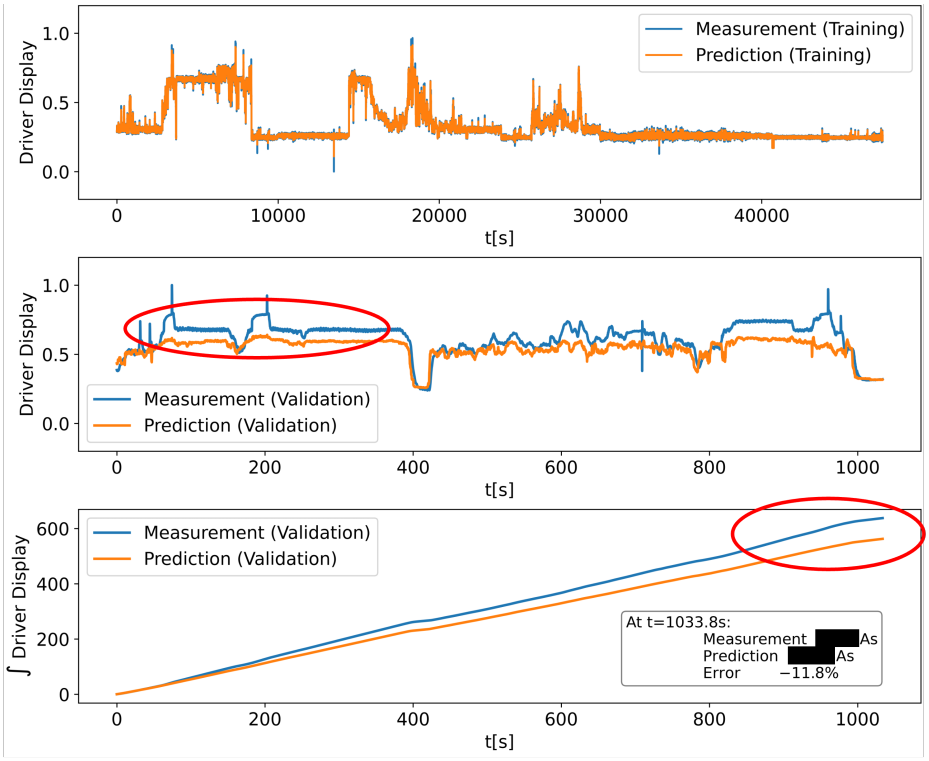

Figure 4.13: Measurement (blue curve) and predictions (orange curve) of current and electric charge consumption for the driver display in Vehicle A. Here, RF with raw data and default hyperparameters is used.

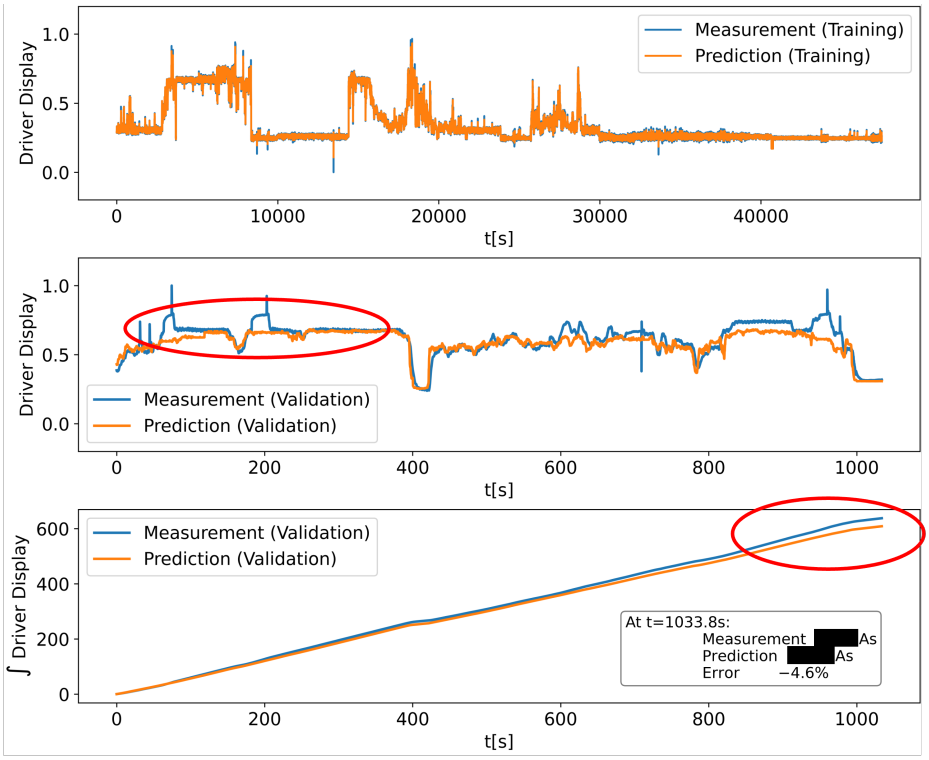

Figure 4.14: Measurement (blue curve) and predictions (orange curve) of current and electric charge consumption for the driver display in Vehicle A. Here, RF with feature-engineered data and default hyperparameters is used.

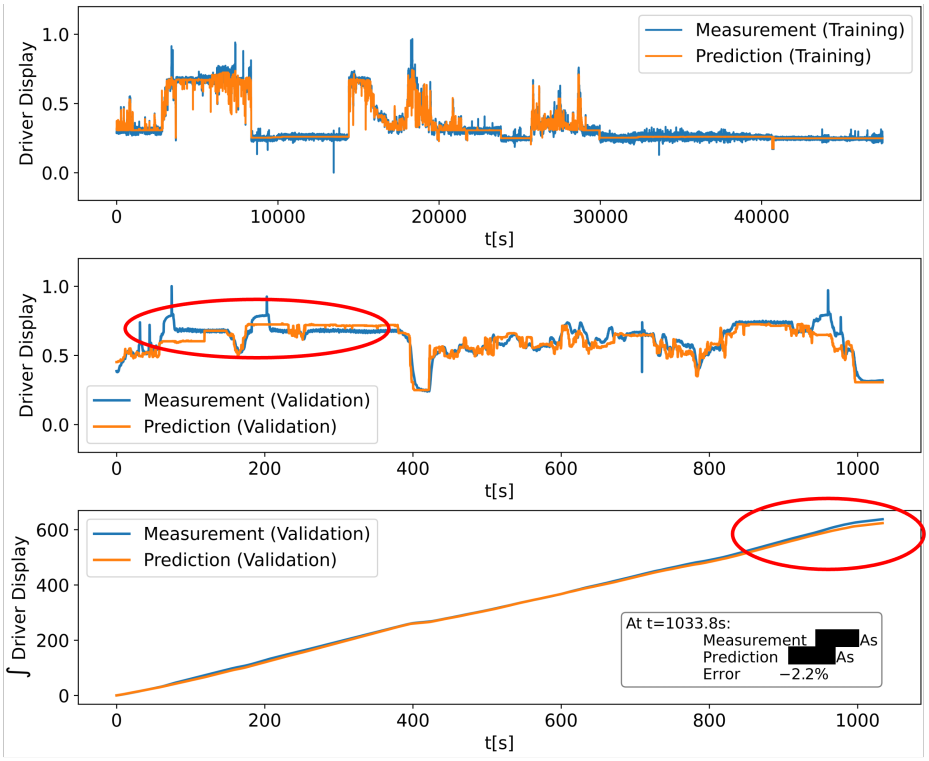

Figure 4.15: Measurement (blue curve) and predictions (orange curve) of current and electric charge consumption for the driver display in Vehicle A. Here, RF with feature-engineered data and RS-tuned hyperparameters is used.

4.3.3 Model Selection Process Using Weighted Sum Analysis

In the following, the entire ML pipeline is taken into account to make predictions for the 18 selected ECUs. Furthermore, for the subsequent evaluation, the entire KPI set $\mathcal{K}$ is considered. This includes also the time measurements to assess computational efficiency and the model size after export.
Due to the number and diverse nature of the evaluated metrics, a WSA as described in Section 3.7.1 is chosen to obtain an objective result reflecting the importance of the different evaluation criteria set by the domain experts, through weighted KPIs. Its goal is to reduce the complexity of the entire ML pipeline and code base enabling an objective selection of one ML modeling algorithm out of $\mathcal{A}$ for the given task. This selection also fulfills step 2 of the CRISP-ML(Q) process model which comprises the algorithm selection [91].

KPI Normalization and Cut-Off Strategy

Before conducting the WSA, the metrics of all 18 examined ECUs must be normalized to have the same value range in order not to bias the final result, which is why a normalization (scaling) strategy is needed. The selection of that strategy depends on the actual value ranges of the KPIs themselves.
Additionally, a strategy to cope with outliers must be defined as well which ensures a meaningful application of the scaling strategy as well as the correct interpretability of the resulting utility values.

Normalization Strategy Selection: An analysis of $\mathcal{K}$ (also refer to Table 4.7 and Section 4.1.6) reveals that the metric measuring the explainability of the respective models produced by the algorithms in $\mathcal{A}$ ranges from 1 (most explainable) to 0 (least explainable) including selected increments in between. Therefore, it is considered favorable to normalize all remaining metrics into that range which can be achieved by the min-max-scaler introduced in Equation 3.16. Some metrics are considered "enhanced" when their value decreases which means that the scaled result then needs to be inverted by changing the algebraic sign.

KPI Range Capping and Adjustment Strategy: As already demonstrated and as it can also be seen in Appendix K, some of the results suffer from outliers compared to their general mean, depending on the metric in consideration. Metrics, like **explainability** and **model size**, retain their full ranges for scaling because they lack significant outliers or do not otherwise require specific constraints. However, outliers influence the normalization negatively since they lead to a compression of the total range in which acceptable values are located significantly, making meaningful differentiation in the range between 0 and 1 impossible.
Therefore, a subset of $\mathcal{K}$ undergoes an individual value range pre-treatment before the min-max-normalization is applied. The individual strategies are described in Table 4.15.

Table 4.15: Pre-treatment of KPI and metrics value ranges before min-max-normalizing their values prior to utility value calculation.

KPI	Native Range	Acceptable Range	Reasoning (when capping applied)
X-val weighted PAWD [%]	$[0, \infty)$	$(0, 10]$	No special requirements for the project. Discretionary decision by the author.
X-val weighted R^2	$(-\infty, 1]$	$[0.4, 1]$	At least 40% of the variance to be explainable. (Not higher due to partially noisy data.)
Inference time / sample [s]	$(0, \infty)$	$(0, Q_{90})$	The 90% quantile is used to cut off a moderate number of outliers.
SD(PAAD) [%]	$[0, \infty)$	$[0, 10)$	10 shall be the max. allowed value in harmony with PAWD.
Explainability	$[0, 1]$	-	-
X-val avg. R^2	$(-\infty, 1]$	$[0.4, 1]$	See *X-val weighted R^2*.
Model size [kB]	$(0, \infty)$	$(0, \infty)$	Large value range, no significant outliers, hence no cut-off.
SD(R^2)	$[0, \infty)$	$[0, 0.2)$	High variable data and performance across ECUs and training data.
X-val weighted MSE [A^2]	$[0, \infty)$	$[0, 0.25)$	This cut-off was chosen by the author in consultation with the domain experts (cf. 4.1.6).
MSE [A^2]	$[0, \infty)$	$[0, 0.25)$	See *X-val weighted MSE*.
SD(MSE) [A^2]	$[0, \infty)$	$[0, \overline{\text{MSE}}]$	The SD should be a maximum of 30% of the avg. MSE (author's discretion).
Training time per feature & sample	$(0, \infty)$	$(0, Q_{90})$	See *Inference time / sample*.

For the sake of completeness it shall be mentioned that there is no commonly valid solution for capping the KPI value ranges before the application of a min-max-normalization. However, the author advises to conduct it before the calculation of the utility values to avoid retrospective bias that might be introduced through shifted distributions within the interval [0,1].
To be in harmony with the WSA nomenclature introduced in Section 3.7 the term "score" is used to denominate the normalized experimental results.

Condensed WSA Results and Final Algorithm Selection

With the evaluation results and the KPI weighting conducted in Section 4.1.6 and the normalizing eventually set-up, the utility values $U_{a,e}$ with $a \in \mathcal{A}$ and $e \in \mathcal{E}$ for each ECU and ML algorithm can be calculated according to Equation 4.3. In a second step, the sum of the utility values can be aggregated for each ML algorithm—they are additive—thus permitting to determine the one with the highest utility over all ECUs in $\mathcal{E}$ as shown in Equation 4.4. Both equations are adaptions of the general WSA descriptions previously outlined in Section 3.7.

$$U_{a,e} = \sum_{k \in \mathcal{K}} v_k \cdot s_{k_{a,e}}, \quad \forall a \in \mathcal{A}; \forall e \in \mathcal{E} \tag{4.3}$$

As a result, the total utility U_a of one modeling algorithm $a \in \mathcal{A}$ is calculated as follows:

$$U_a = \sum_{e \in \mathcal{E}} U_{a,e} = \sum_{e \in \mathcal{E}} \sum_{k \in \mathcal{K}} v_k \cdot s_{k_{a,e}}, \quad \forall a \in \mathcal{A} \tag{4.4}$$

Equation 4.4 thus exploits the general additivity of utilities. As a final step the ML modeling algorithm with the highest utility can be selected according to Equation 4.5 which is an adaption of the general notation formulated in Equation 3.22:

$$a_{opt} = \arg\max_{a \in \mathcal{A}} U_a = \arg\max_{a \in \mathcal{A}} \sum_{e \in \mathcal{E}} \sum_{k \in \mathcal{K}} v_k \cdot s_{k_{a,e}} \tag{4.5}$$

Table 4.16: Resultant utility values of the WSA: cumulative, as well as min, max and their dispersions over the examined ECUs per algorithm are shown (values rounded to the third decimal place).

Algorithm	U_a	$U_{a,e_{min}}$	$U_{a,e_{max}}$	$SD(U_{a,e})$	$\overline{U_{a,e}}$
RF	548.665	8.195	**54.910**	13.894	30.481
GB	**614.115**	**12.418**	48.500	11.583	**34.118**
MLP	349.110	10.653	45.111	**10.034**	19.395
LSTM	362.423	4.918	40.112	11.745	20.135

Table 4.16 demonstrates that GB has the highest accumulated utility value with **614.115** since $U_{GB} > U_{RF} > U_{LSTM} > U_{MLP}$, followed by RF with around 65 value points of distance. Falling behind are LSTM and MLP with 362.423 and 349.110 points respectively.
Additionally, GB also has the highest average utility value $\overline{U_a}$ over all considered ECUs with 34.118 and the highest minimum $U_{a,e_{min}}$ of 12.418 (door ECU front left, Vehicle B).
Given the importances of the metrics and the modeling results MLP achieves the lowest utility value despite having lowest dispersion (standard deviation $SD(U_{a,e})$ of all utility values with 10.034. On the contrary, RF has the highest maximum utility with 54.910 $U_{a,e_{max}}$ (coolant pump, Vehicle A) whilst only having a lower cumulative utility value than GB with 548.665.
The individual utility values for every ECU and ML modeling algorithm are shown in Appendix L. Hence, in all subsequent (XAI) considerations—this includes the final discussion in Chapter 5, only GB is further considered thus reducing the overall dimensionality of this ML task. The aforementioned results then serve to answer *PRQ2* and *SRQ3*.

Remarks: The detailed results of the WSA (cf. Appendix L, Table L.1) indicate that GB does not perform well universally across the considered ECUs.
Considering for example the steering column ECU or the (front left) door ECU (both Vehicle B), RF achieves a higher utility value than GB. Even though especially for the door ECU all of the algorithms in $\mathcal{A}$ do not perform as well as for others. This adds to the fact that the performance on Vehicle B (cumulative utility: 269.355; average: 29.928) is generally not as good as in Vehicle A (cumulative utility: 344.760; average: 38.310), given the training data and the selected ECUs.

4.3.4 Explainable AI Using the Example of the Driver Display

To conclude the results part of this thesis—following the KPI assessments and model selection—the XAI results are shown in the following and their plausibility discussed in Section 5.1.3. The XAI explorations are made on the fully feature engineered dataset with RS hyperparameter tuning activated. Due to the satisfactory results with $U_{GB,e} = 40.117$, the driver display of Vehicle A shall be considered. Table 4.17 shows the KPI values for this ECU as raw values and normalized after the capping strategy is applied. The XAI evaluations commence with a simple PCC calculation among the features themselves as well as with the target. The correlation analysis is then followed by the more advanced methods introduced in Section 3.6.

Table 4.17: KPIs and metrics $k \in \mathcal{K}$ as well as corresponding normalized scores $s_{k_{GB,e}}$ for the driver display using the GB algorithm, feature engineering, selection and RS for hyperparameter optimization (values rounded to the third decimal place).

KPI / Metric	Value	Score (Normalized)
X-val weighted PAWD [%]	1.335	0.880
X-val weighted R^2	0.707	0.523
Inference time per sample [s]	2.095×10^{-6}	0.774
SD(PAAD) over all X-vals [%]	1.368	0.882
Explainability (local & global methods)	0.8	0.8
X-val avg. R^2	0.629	0.382
Model size after export [kB]	413.95	0.993
SD(R^2) over all X-vals	0.320	0
X-val weighted MSE [A^2]	0.001	1.0
MSE [A^2]	0.001	1.0
SD(MSE) [A^2]	0.001	0.994
Training time per feature & sample [s]	3.463×10^{-5}	0.0

Pearson Correlation Coefficient (PCC)

A basic explanation of the correlation of the features with the target and among each other is delivered by PCC calculated according to Equation 3.19. Figure 4.16 shows a PCC matrix over the six most important features relevant for the prediction of the energy consumption of the aforementioned driver display. Here, the effectiveness of the feature engineering and selection pipeline becomes visible since at this point no features are correlated more by a factor greater than 0.9 which further reduces the size of the training data and the model complexity.
In this case the feature *Illumination Level* followed by *Head-Up Display Sensor Value* are most correlated to the target with correlation coefficients of 0.85 and 0.8 respectively. The least correlated are *Supply Battery Current* and *Supply Battery Voltage* with PCCs of -0.0026 and 0.16 respectively. Since correlation has no direction the resultant correlation matrix is symmetric.

Figure 4.16: Symmetric PCC matrix of the six most important features for the driver display's energy consumption prediction (Vehicle A).

Permutation Feature Importance (PFI)

The PFI for the driver display is calculated as described in Section 3.6.1 and in the present example on the validation data from the first LOOCV iteration. The validation data comprises a total driving time of 1033.4 seconds (cf. Fig. 4.17) with the results shown in Fig. 4.15 and Table 4.18—ranked by importance in descending order. From them it can be derived that for that validation test drive the display's illumination level has the highest impact on the model's prediction with a PFI of 0.624 and a variability of ± 0.014. Much less drop in model performance are caused by the voltage of the supply battery (0.229 ± 0.009) and the value of the display's brightness sensor (0.184 ± 0.006) even less important are the supply battery current and the calculated elevation angle of the sun. This is in accordance with the findings in the PCC analysis. An even negative importance (unintended positive impact on performance) is caused by a signal related to the head-up display (*Head-Up Display Sensor Value*).

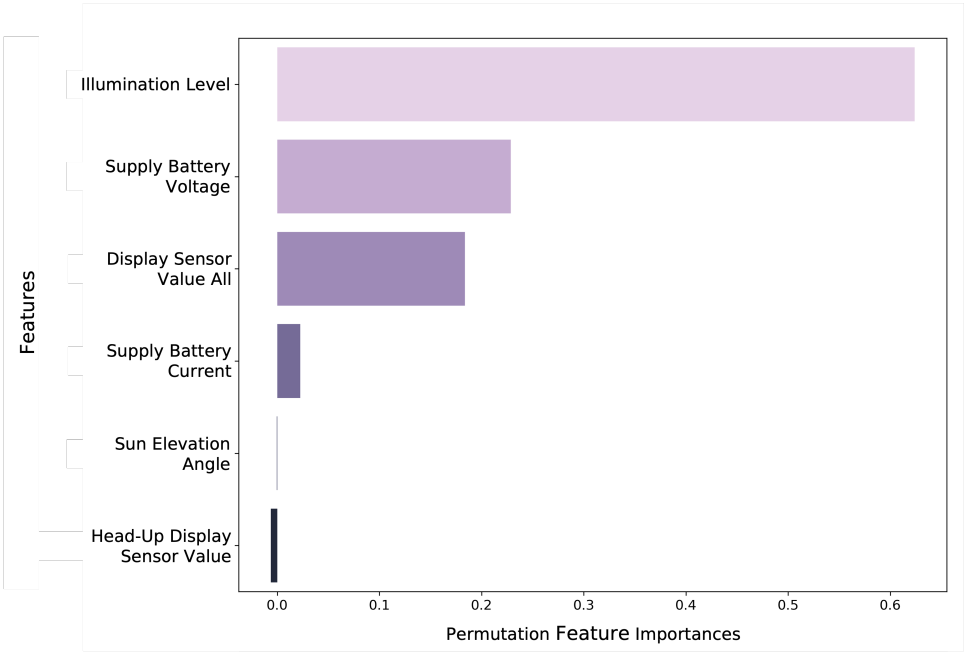

Figure 4.17: The six most important features for the first out of 15 CV folds (validation drive) of the drivers display of Vehicle A.

Table 4.18: Numeric PFI values with variability (six most important) for the driver display.

Feature	Importance (Variability)
Illumination Level	0.624 ± 0.014
Supply Battery Voltage	0.229 ± 0.009
Display Sensor Value	0.184 ± 0.006
Supply Battery Current	0.023 ± 0.003
Sun Elevation Angle	-0.000 ± 0.00
Head-Up Display Sensor Segment Value	-0.006 ± 0.004

Accumulated Local Effects (ALE)

In contrast to PFI, ALE focuses on the individual features whilst exploring the entire dataset for a specific one. This implies that there is one plot per feature per se but—as previously discussed—it is limited to a minimum value range of 0.1 A in this XAI implementation (cf. Section 4.2.2). This results in two ALE plots for (*Illumination Level* and *Display Sensor Value*) from which the first one—as an example—is analyzed in greater detail.
As shown in Listing 4.3, the `do_ale_explanation(self, X_trn)` function takes the training dataset as input but also the validation data could possibly be passed. However, to show the effect of one feature a set of more explanatory data points is preferred in this case which is fulfilled by the training data (the model has been trained with it and more data points are available). Figure 4.18 shows the ALE plot for the feature *Illumination Level*. Here, it becomes apparent that this feature is capable of influencing the power consumption significantly depending on the value it assumes, which leads to an approximate total influential range of 0.3 A. The highest gradient of the curve is reached in the feature value range between 40 and 80 flattening out for values above 80.
Additionally, the density indication above the x-axis shows an almost completely consistent coverage of values in the training data. The exception is the range between 20 and 30, and short before 100 where values thin out.

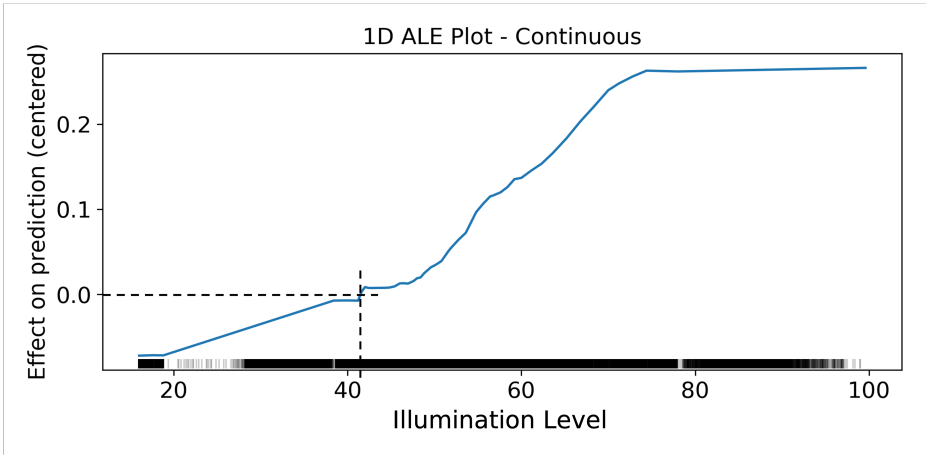

Figure 4.18: Graph of the 1-dimensional ALE plot for the bus signal *Illumination Level*, a continuous numeric feature for the GB-based regression model of the driver display of Vehicle A—on the **training** data.

As it can be seen in Fig. 4.19, the ALE plot for the same ECU calculated on the much smaller evaluation dataset, the value range of the x-axis is more limited compared to the ALE value calculations based on the significantly larger training data, visible in Fig. 4.18. Furthermore, the density indication above the x-axis shows areas where values are more sparse e.g. between illumination levels 20 and 30 as well as between 40 and 55. In these areas the ALE curve is less detailed and more interpolated than in the area between 60 and 70 where the datapoint availability is significantly higher.

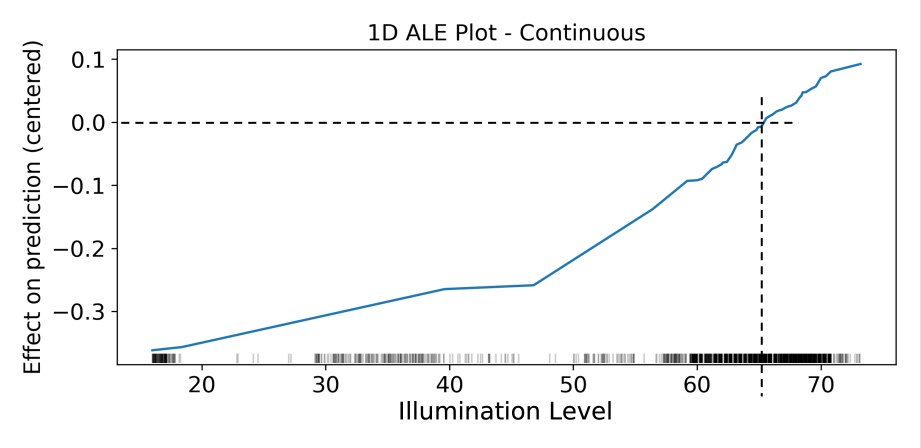

Figure 4.19: Graph of the 1-dimensional ALE plot for the bus signal *Illumination Level*, a continuous numeric feature for the GB-based regression model of the driver display of Vehicle A—on the **validation** data.

Shapley Additive Explanations (SHAP)

The SHAP explanations are calculated on the training data only, since more data points are then available to explain the model's (and thus the power consuming component's) behavior. Figure 4.20 shows the influence of a total of 8 features on the prediction. In this case, the 90 % quantile (Q_{90}) of the available power consumption measurement values y is chosen by the author as the local index to be examined by SHAP in order to provoke sufficient distance between the expected value $E[f(x)]$ and the considered value which can then be explained[31]. This is realized automatically within a dedicated function of the program code as part of the `Explanation.py` class introduced in Listing F.6, which returns the desired indices as a list which then serves as input for the SHAP method (cf. Listing 4.5, line 8). This modularity can be utilized to automatically examine any other data point of interest as long as it can be retrieved by a function (e.g. the maximum or minimum, turning points, etc.) before it is passed to the actual `generate_shap_values([...])` function as the `indices` parameter (cf. Listing J.1, line 80). Given the example index chosen for this research, Fig. 4.20 demonstrates that at Q_{90} the prediction deviates from the average of all predictions (exact values blacked for privacy reasons). According to the SHAP algorithm the *Display Sensor Value Segment 2* contributes most (+0.22 A) to the elevated energy consumption of the driver display at that moment. It is followed by the *Illumination Level*—already discussed in the ALE section—as well as by the *Supply Battery Voltage* with +0.18 A and +0.07 A respectively. The subsequent features only have little impact on the energy consumption.
However, SHAP is also able to indicate negative influence on the deviation from the average prediction as it can be derived from Fig 4.21. In this example, which is the 9$^{\text{th}}$ CV iteration, of the right pixel light unit (Vehicle A), the *Supply Battery Voltage* and *Temperature* at the chosen index reflecting the value of (Q_{90}) influence the current consumption negatively as well as the sum of 23 other features. Nonetheless, the features (bus signals) causing an overall positive deviation from the global average outweigh the negative ones.

[31]Note: Depending on the aim of the analysis, any index (data point) from the dataset can be passed to the SHAP method.

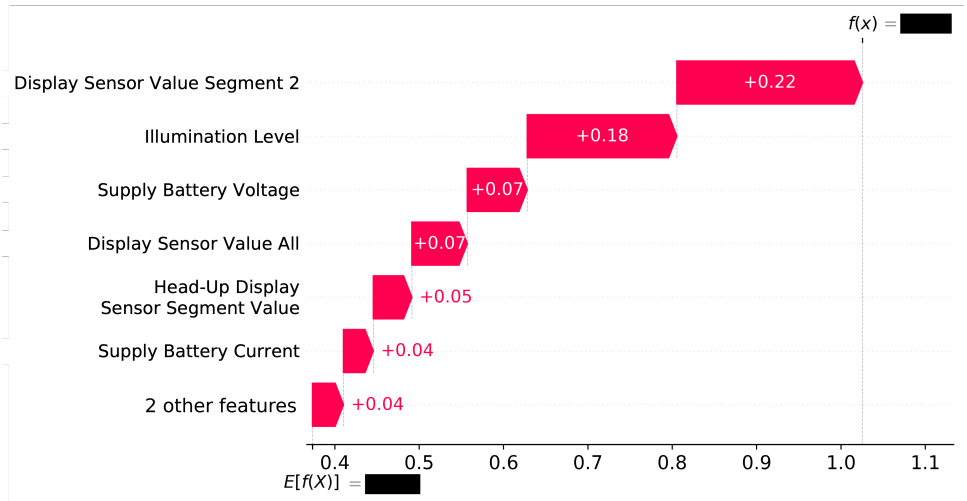

Figure 4.20: SHAP plot for the driver display of Vehicle A, CV iteration 1 (out of 15), calculated using the GB regression algorithm. Actual values are blacked due to a non-disclosure policy.

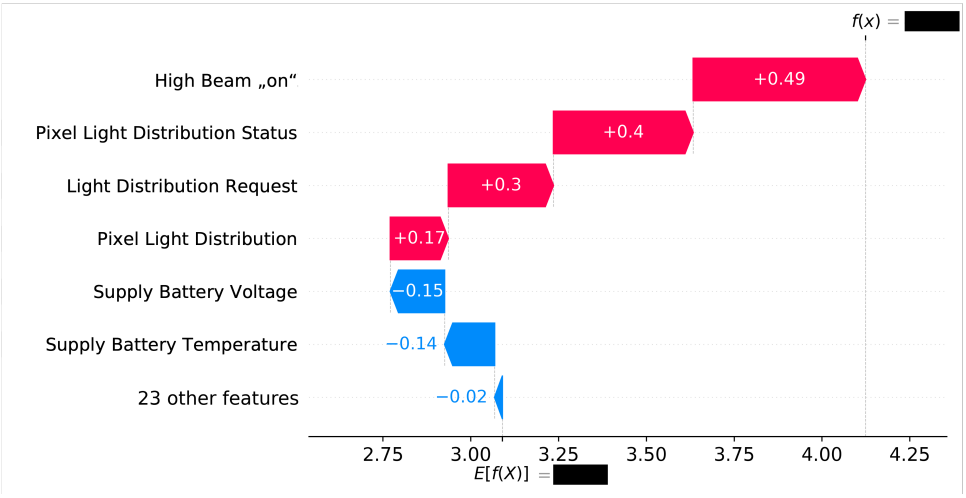

Figure 4.21: SHAP plot for the right pixel light unit (headlamp) of Vehicle A, CV iteration 9, calculated using the GB regression algorithm. Actual values are blacked due to a non-disclosure policy.

```python
import numpy as np

def set_local_index(self, y):
        max = np.argmax(y)
        percentile_90 = np.percentile(y, 90)
        closest_val_90 = y.flat[np.abs(y - percentile_90).argmin()]
        min = np.where(y == closest_val_90)[0][0]
        self.index_list = [min, max]
        return None
```

Listing 4.5: Python function to automatically determine distinctive points in the measurement data. Here: the maximum of all measured values y and the 90 % quantile are identified using built-in numpy functions and stored in `index_list`.

5 Discussion of Findings and Answering of the Research Questions

With the research results from the previous chapter now available, they can be categorized and discussed as part of WP 4 of this dissertation. The results also give the opportunity to answer both the primary and secondary research questions (*PRQs* and *SRQs*).

5.1 Summary of the Key Results

5.1.1 Energy Consumption Predictability with an Integrated Pipeline

During the course of this research it could be demonstrated that the inter-ECU communication data of contemporary passenger vehicles carries valuable information for the prediction of the low-voltage energy consumption of selected power consumers. To systematically select relevant ECUs, an adequate methodology was developed in Section 4.1.4 which provides a ranking of promising power consumers in an automated way.
From two luxury passenger vehicles (Vehicle A and Vehicle B), 14.63 respectively 21.35 hours of training data for the selected ECUs were able to be collected. Therefore, an integrated special shunt-based measurement system with a data logger as described in Section 3.4.5 was used, focusing on the coverage of diverse real-world driving conditions. This data served as an input for all the subsequent evaluations and for an integrated data processing and ML training pipeline.

Data Import, Feature Selection and Engineering

To preprocess the data, state-of-the-art feature engineering and selection methods were employed as the first pipeline step after importing the raw data. The feature engineering and selection part is composed of a set of methods for feature selection, improvement and construction in a fixed order. The two main goals of that process step were to achieve a significant reduction of the number of features in order to make the resultant ML models smaller as well as—ideally—an improvement of the model's predictive capabilities. These capabilities were assessed with measured KPIs which were defined, selected and weighted according to their importance by AI and automotive domain (low-voltage energy efficiency) experts during a workshop.
The feature preprocessing pipeline however, was able to reduce the number of bus signals (features) used for prediction of at least -73.1 % until up to -99.9 %. With regard to the actual performance of the 18 selected ECUs it became apparent that the four ML modeling algorithms from the set $\mathcal{A}$ showed different results reaching from as much as 13.5 (RF) to as few as 8.625 (MLP) enhanced trials (cf. Table 4.13).

5.1.2 Hyperparameter Tuning and Systematic Algorithm Selection

After a detailed analysis of various hyperparameter tuning strategies, a random search was chosen for this ML task. This conclusion was preceded by the systematic narrowing down of the search spaces of the respective parameters using a sensitivity analysis with selected power consumers.
The results of the RS show that further improvements in model performance are possible even after feature engineering, depending on the regression algorithm. However, this does not apply to every power consumer and so the overall effect of the improvements per regression algorithm varies significantly. Of the 18 ECUs examined, RF showed the greatest additional improvement in the KPIs for model quality with an average of 13.375 improved trials. MLP, again, performed the worst with only 5 further improvements. At the same time, the average values for this regressor were strongly influenced by individual outliers, in which the model performance was considerably weakened by the random search (cf. Table 4.14). The positive effects evolving from an analysis using the raw, feature engineered and hyperparameter tuned RF algorithm could also be shown graphically using the example of the driver display (cf. Figures 4.13 to 4.15)
Based on the results obtained for 18 ECUs, each of them applied to the four ML modeling algorithms, as well as the weighted KPIs and metrics, it was then possible to carry out a WSA with the aim of selecting a regression algorithm from the four that best meets the requirements across the ECUs. The aim was to remove complexity from the overall ML pipeline and to find a universally well-performing way to model the energy consumption behavior of diverse ECUs.
Eventually, this was achieved by calculating the utility values for each ML modeling algorithm according to Equation 4.3 and Equation 4.4 for each of the 18 ECUs and then accumulating them per algorithm. In the final result, GB achieved the highest cumulative utility value of 614.115, followed by RF with 548.665. LSTM (362.423) and MLP (349.110) were at the bottom by some distance.

5.1.3 Explainability

With regard to the XAI methods investigated, this research could add to the findings made in previous publications [26, 28]. Using the example of the driver display, it could be demonstrated that the methods proposed are also applicable to additional power consumer ECUs. Additionally, the appropriateness of the methods could be shown on the GB algorithm.
The results presented in the previous chapter show the plausibility of the physical coherences between signals as well as their values and energy consumption:

- **Permutation Feature Importance (PFI)** The PFI plot from Fig. 4.17 shows that the *Illumination Level* of the driver display has the highest impact on the prediction ($\hat{y}$, output) which is plausible since the background LED lighting of the central display is also associated with the evaluated fuse. This fact is also related to the *Display Sensor Value* signal—measuring the ambient brightness—that regulates the display's illumination accordingly. The brighter the outside the more intense the illumination of the display needs to be the more power is drawn from it. Less important but also mentioned amongst the top six features is the sun elevation angle which itself relates to the ambient brightness. The occurrence of the *Head-Up Display Sensor Value* in the PFI

(and SHAP; cf. below) plot might as well result from spurious correlation with regard to brightness as there is no coherence known to the author.

- **Accumulated Local Effects (ALE)** These coherences are also underlined by the detailed observations of the *Illumination Level* signal in the ALE plots of both the training and validation data (cf. Fig. 4.18 and Fig. 4.19). Higher levels (a brighter screen) are related to higher consumptions.
- **Shapley Additive Explanations (SHAP)** Given the specific local index selected for the SHAP explainer, it can also be derived that the signals already mentioned above are responsible for causing the positive deviation from the global average of all predictions which is also plausible.

These results may suggest that the actual content displayed on the screen might not play a crucial role for the energy consumption since the focus is mainly on the screen brightness. However, the data streamed to the screen is not part of the database as it is not captured by the measurement system, which is why no statement on its influence can be derived from the model.

5.1.4 Answers to the Primary Research Questions

All findings made above, now permit the answering of the primary research questions (PRQs).

PRQ1: How can high-dimensional ECU network communication data be utilized to predict energy consumption in vehicle power nets?

This question can be answered through the entire conception of the pipeline as it graphically shown in Appendix C. Each of the modules shown there contributes to the achievement of an accurate prediction made by an adequate regression algorithm. More generally, it can be stated that the network communication data can be exploited by an adequate data collection mechanism (cf. Sections 4.1.1 and 4.1.3), suitable data preprocessing (feature engineering and selection, cf. Section 4.2.2) as well as by the usage of GB as the appropriate regression algorithm (selection made in Section 4.3.3). The combination of these methods returned different KPIs for a set of 18 ECUs examined. The KPIs were then combined in an aggregated utility value per algorithm to eventually assess the overall project-specific performance of the respective ML model.

PRQ2: What are the most suitable ML modeling algorithms and preparatory techniques for predicting ECU energy consumption and how can they be integrated into a holistic pipeline?

To answer these research questions, the nature of the problem was first identified as a regression algorithm due to the numerical in- and output. Subsequently, a set of candidate algorithms $\mathcal{A}$ was composed which was then evaluated using a WSA. Input to the WSA was delivered by a weighted set of KPIs $\mathcal{K}$, compiled and assessed by a multidisciplinary project team. Prior to that, the data preprocessing and hyperparameter tuning needed to be executed to receive the most satisfactory results for all ML algorithms. Data preprocessing involved blacklisting previously

known non-value-adding features (bus signals) and handling erroneous and missing data points. This was then followed by OHE, low-variance and importance feature removal as well as by the dropping of highly correlated features. This eventually led to significantly smaller and manageable datasets as well as faster evaluation trial runs within the scope of the actual WSA, where all the aforementioned methods were employed in a fixed sequence.
To receive an additional boost in model performance the algorithm's hyperparameters were also tuned using random search before the final evaluation run.
As a result of the WSA, the GB regressor could be identified as the most suitable, given the weighted KPIs and metrics (cf. Section 5.1.2).

PRQ3: How can the explainability of the trained ECU models be ensured whilst maintaining high prediction accuracy at the same time?

First, high-precision and accuracy are ensured by the objectively selected ML modeling algorithm which is GB. Subsequently XAI methods were executed and assessed using the models trained and the engineered input data with exactly that algorithm. The respective assessment of the explanatory local and global methods considered (PCC, PFI, ALE and SHAP) does not only show the effectivity but also the functioning of the methods, despite the application of an algorithm (GB) which had not yet been employed in the referenced literature (e.g. [26] and Section 4.3.4).

5.1.5 Answers to the Secondary Research Questions

SRQ1: What network communication signals are relevant for predicting energy consumption in vehicle power nets, and what preprocessing and feature selection techniques are needed to handle high-dimensional data effectively?

The fact that for the most power consumers different data buses are available due to the topology of the E/E architecture, different features are available for each ECU for prediction as well. Therefore, this research question cannot be answered generally but requires the construction of a data preprocessing pipeline that automatically compiles the relevant methods to retrieve the best-fitting set of features specific to each power consumer. The necessary feature engineering and selection techniques are a result of a concise assessment depending on the characteristics of the data available. Therefore, the static pipeline as described in Section 4.2.2 was developed and put to use. It has the main goals to not only improve the predictive performance of the respective ML model but also to reduce the complexity of the high-dimensional input data significantly.

SRQ2: Which evaluation metrics are appropriate for assessing the performance of the machine learning models?

Evaluating the aforementioned predictive performance of the ML models is a crucial step in assessing the practicability of the entire approach. Therefore, meaningful KPIs could be defined and one of them could be developed specific to the present ML task (PAWD). To ensure a high degree of explanatory power as well as informative value of these metrics, a joint workshop of both AI/ML and domain experts was conducted to build a foundation with regard to the selection of such

KPIs as a precursor to their subsequent weighting. The exact procedure to obtain these metrics as well as the outcome of the KPI workshop (pairwise comparison) is outlined in Section 4.1.6.

SRQ3: How can the data processing pipeline be defined to ensure efficient and automated model generation?

One of the main goals of this research is to develop a pipeline which can be executed by energy efficiency specialist engineers without detailed knowledge of ML. Therefore, an automated approach has been developed over the entire course of Chapter 4 of this dissertation. In this chapter, methods from importing, converting and preprocessing respective input data from the domain experts which eventually retrieves trained and hyperparameter-tuned ML models are sequentially compiled. They can then be used again by the domain experts to make on- or offline predictions or run XAI on them.
However, the essence of the approach and of the corresponding outcome is the seamless handling of the data between the pipeline steps without necessary intervention by its users as graphically depicted in the figures in Appendix C.

SRQ4: What methods can be employed to interpret and explain the predictions made by the ML models?

Following previously published work by the author ([26]), the general concept of XAI was selected and eventually employed to make the predictions which result from both the input data as well as from the trained ML models explainable in a way that is convenient and meaningful to the target group (cf. above). According to a survey among domain experts—also conducted by the author ([28])— the three most preferred XAI methods were applied to a different power consumer ECU. Additionally, GB was used (instead of RF) as ML modeling algorithm. The results show that these methods applied on a new power consumer as well as a new algorithm produce similar plausible results as in the preliminary work. This manifests the correct selection of the three principal XAI methods PFI, ALE and SHAP, optionally combined with a preceding PCC analysis between features and between features with the target.

SRQ5: Is there one "most suitable" modeling algorithm to be selected in order to reduce the complexity of the pipeline?

This research question is substantially connected with *PRQ2*. In order to select a "most suitable" ML modeling algorithm in the most objective way, first the criteria for the selection were agreed upon by fixing the KPIs in the KPI assessment workshop together with their weight attribution (cf. Section 4.1.7 and Table 4.7). Subsequently, a set of affected ECUs was systematically selected (cf. Section 4.1.6) and then the ML pipeline was applied to their respective bus data. The resulting models and KPIs then served as an input to a WSA to determine the cumulative utilities for all investigated algorithms. As a result, the gradient boosting algorithm accumulated the highest utility value (cf. *PRQ2* above) which is why it shall be considered the "most suitable" one, given the algorithms available in $\mathcal{A}$.

5.2 Interpretation of the Key Results

Taking into account the results and the answered research questions the overall outcome of the present research can be interpreted and put into a context to the state-of-the-art literature.

5.2.1 Power Consumption Prediction

Given the related literature—as stated in the state of the art research in Table 3.2—it can be stated that this research as well as its results pose a novelty in the domain of high-dimensional regression analysis for energy prediction to the best of the author's knowledge. This is mainly due to the fact that the primary goal of this research was to find a way to predict energy consumptions of in-vehicle ECUs based on their associated communication data with other bus participants within the E/E architecture of passenger cars.
For many of the considered ECUs, the GB algorithm—selected through the WSA—produced results measured by appropriate KPIs and metrics that fall within an acceptable range. This indicates that the inter-ECU communication data carries information correlated with energy consumption. Furthermore, the XAI algorithms applied to the example ECU revealed that the most correlated features are not the result of spurious correlations. This is supported by the observation that the features identified for the driver display ECU mainly relate to the display's brightness settings (influenced by external conditions) and variations in the power net voltage level. These voltage variations can impact the operation of internal components, such as DC-DC converters, leading to changes in current consumption required to maintain the necessary power (cf. Fig. 4.17).
Nonetheless, it can be stated that the KPIs are volatile even when considering the "most suitable" regression algorithm which is GB as stated in the Appendix Table K.6. This may have several causes one of which can be a missing relevant variety in the training data. Moreover, the differences in the results can be caused by insufficient data preprocessing or hyperparameter tuning. Runs with different sets of hyperparameters θ can lead to improved results.

5.2.2 Methodological Reflections

The reference literature in Table 3.2 demonstrates that ML tasks are generally complex regarding data collection, the number of input features, and the selection of the appropriate modeling algorithm. As a result a key feature to be implemented was the automation of the entire pipeline. Except for the data collection mechanism, which still relies on a manual file format conversion process, all subsequent steps could be automated until the retrieval of the final results. This way the ML and data science complexity knowledge is incorporated into the pipeline itself and does not need to be handled by the final customers who are notably the domain and energy efficiency experts respectively development engineers. This user group usually lacks the required data science and XAI knowledge according to a survey conducted by the author in [29], shown in Fig. 5.1.

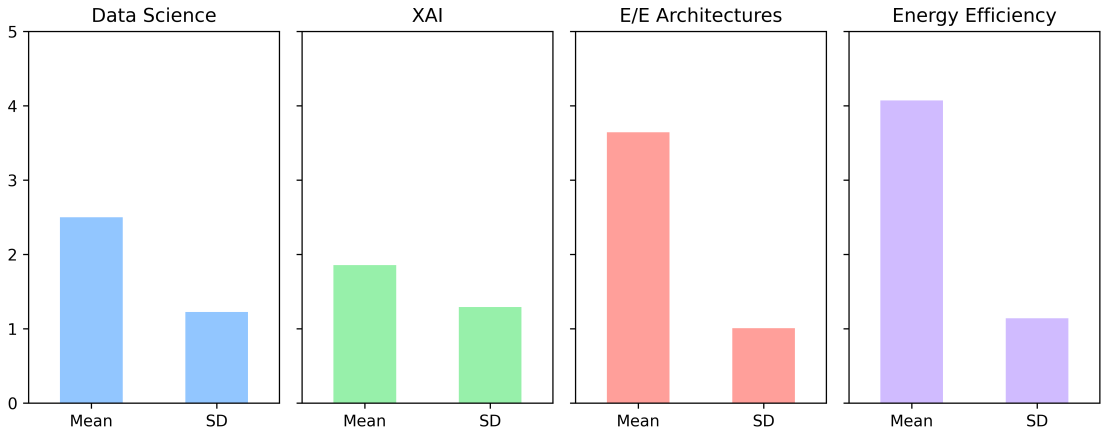

Figure 5.1: Experience of the surveyed development engineers regarding their prior experience in the relevant fields (adapted from [29, p. 61]).

Consequently, the present complexity imposed by the high-dimensional data could be reduced by a pipeline step that lowers the number of remaining bus signals to an acceptable minimum (data preprocessing). Furthermore, the WSA as a precursor to the final pipeline composition served to handle the manifold requirements for the resulting ML models—previously formulated by weighted KPIs—since it reduced the number of available regression algorithms. Nevertheless, it must be mentioned that the apparent objectivity of the WSA is based on subjective assessments of the importance of the KPIs, which are developed as part of workshops and the pairwise comparison method. Due to the different influences of expert opinions during such KPI assessment workshops on the weightings of the metrics, complete objectivity cannot be guaranteed. However, given these weightings, the subsequent WSA process is intrinsically objective again and allows a relative comparison of the individual decision options, which can ultimately be quantified in a cumulative overall benefit.

5.3 Statement on Implications

By filling a gap in the state of the art, this dissertation has significant practical implications within the automotive industry, too, which can be divided into implications on the way engineers work (efficiency) as well as into implications for the vehicles themselves.

5.3.1 A New Way-of-Working for Efficiency Engineers

The method developed here, going from raw training data to fully trained regression models for predicting energy consumption in low-voltage vehicle electrical systems is a new tool for the (energy efficiency) engineers involved. In addition to the complex measurement technology, which is associated with high costs, weight, and set-up times, they now have another option for obtaining current values offline—within certain accuracy tolerances. This means that the number of measurement systems that need to be installed in dedicated test vehicles and the number of actual measurement runs (road or dynamometer driving) can be reduced, as existing bus traces can be utilized to make a statement about the power consumption behavior

of an ECU using the trained models. Data from vehicles that are equipped with a data logger but not with measurement technology can also be used for this purpose.

5.3.2 Contributions to More Efficient Vehicles

The outcome of this research not only provides a new tool for engineers to work in a more efficient data-driven way, but also contributes to more efficient vehicles. The tool offers the possibility of examining the complex vehicle electrical system and thus the entire E/E architecture like an X-ray machine, with a focus on energy consumption. XAI plays a crucial role by revealing and effectively displaying the influencing variables as bus signals. This allows for a meaningful and efficient interpretation of the data. Additionally, applying the presented methodology to real-world data eliminates the need for tedious manual work. This way, the automated data pipeline developed during this research makes a contribution to data-driven development in the automotive industry, hence to the overall technical progress.

5.4 Limitations of this Research

5.4.1 Model Predictive Quality

The detailed results of the evaluations conducted (shown in Appendix K) demonstrate that, given the same amount of training data the individual results per vehicle, differ significantly. This applies even for the selected "most suitable" regression algorithm GB. One example is the **X-val weighted** and **X-val avg. R^2** metric for the relatively complex BCF ECU of Vehicle B which—even after feature engineering and hyperparameter tuning—still show a significant negative value which means that the model performs even worse than by just predicting the average. Additionally, the corresponding dispersions are much higher compared to other ECUs.
From this fact, a limitation can be derived that even with a considerable amount of training data the quality aspirations cannot be met in certain cases. It can therefore be concluded that either more training data is needed or a more specific feature engineering or hyperparameter tuning is necessary. Another influential factor for low-quality prediction results can be that the relevant information for the energy consumption is not present on the data bus(es).

5.4.2 Hardware and Data Resources

In this study, CAN, LIN and FlexRay buses are evaluated, with the latter containing a high number of features. Other data buses—such as the automotive ethernet—could not be processed within the scope of this study since the number of features could not be handled by the workstations used as the RAM was exceeded (cf. Section 4.1.8) by the large amount of data points.
Some feature engineering methods like OHE contribute to the rise in the amount of data. Equipping the physical workstation with more RAM or using a more scalable computing approach (e.g. cloud computing) could solve this limitation in the future.

5.4.3 Methodological Limitations

With regard to the methodology applied for the selection of a suitable ML modeling algorithm, it can be stated that the number of ECUs (which is fixed to 18 in this case, distributed over two test vehicles) could still be extended to a higher number and to more diverse vehicles (e.g. to different E/E architecture platforms) in the future. This way, it is possible to obtain more significant and more substantiated results in the WSA. However, during this study the number and availability of vehicles equipped with the necessary measurement systems was limited to the two vehicles in set $\mathcal{T}$.

5.5 Conclusion

This research takes a deep dive into the currently uncharted territory of predicting the energy consumption of in-vehicle ECUs and their peripherals (low-voltage power net) using high-dimensional network communication data. Through the development and implementation of a comprehensive and automated ML processing pipeline, it could be demonstrated that inter-ECU communication data is a valuable resource for predicting low-voltage energy consumption within certain accuracy boundaries. This research's scientific novelty lies in several key contributions:

(i) **New Way of Data Utilization**
It was demonstrated that high-dimensional and complex network communication data, typically underutilized in automotive data science applications, can be systematically exploited to predict energy consumption. This marks a significant step further from traditional methods that rely heavily on conventional direct measurement systems.

(ii) **Advanced Feature Engineering and Selection**
The conducted research introduced a robust and automated feature engineering and selection pipeline which significantly reduced the number of input features for the prediction while enhancing the predictive performance of the respective models. This core element of the entire ML pipeline ensures that the resultant models are both efficient and effective for the given task.

(iii) **Systematic Algorithm Selection via WSA**
By employing a WSA to evaluate a given set of multiple diverse ML regression algorithms, it could be objectively identified that gradient boosting is the "most suitable" algorithm for predicting the ECU energy consumption. Using WSA as a systematic approach, it adds objectivity and reproducibility to the model selection process which is however influenced by the expert's estimations incorporated in the KPIs' weights.

(iv) **Explainable AI Integration**
The integration of XAI methods into the pipeline ensures that the predictions made and that the models themselves are interpretable. This is essential for practical real-world applications, as it allows energy efficiency engineers to understand and trust the model outputs, thereby facilitating better decision-making with regard to optimization measures towards energy efficiency.

(v) **Development of an Integrated Pipeline**
The creation of an end-to-end automated pipeline, which can be used by energy efficiency engineers as an integrated application without requiring in-depth ML knowledge, is another decisive contribution. This pipeline not only simplifies the process of model generation but also democratizes access to advanced predictive tools within the automotive industry, enabling domain experts to leverage ML and AI methods who otherwise might not have had the opportunity due to a lack of specialized knowledge. Consequently, this approach bridges the gap between data science and domain expertise, fostering broader adoption of data-driven methodologies.

As a whole, the implications of this research are considerable. For energy efficiency engineers, the developed pipeline and the results achieved offer a new way of working, reducing the need for extensive and costly measurement systems during the development of E/E architecture platforms. It permits the utilization of existing data bus traces to make accurate offline predictions, thereby optimizing the use of available data. For the automotive industry, this research contributes to the development of more efficient vehicles by providing a deeper understanding of the energy consumption behavior of ECUs. However, it is not without its limitations. The predictive quality and power of the ML models vary across different ECUs and vehicles, indicating the need for additional training data or more refined feature engineering and hyperparameter tuning. Additionally, the computational resources required to handle high-dimensional (training) data, particularly from automotive ethernet and FlexRay, pose a challenge which could be addressed with more powerful computing hardware in the future.
In conclusion, this research fills a critical gap in the state-of-the-art literature by presenting a novel and comprehensive approach to predicting ECU energy consumption using network communication data. It offers practical tools and insights for the automotive industry, paving the way for more efficient and data-driven vehicle development.
Future work on this topic is eventually assessed in the following and last chapter of this thesis.

6 Final Summary and Future Work

To conclude on this thesis and research project, a final summary and an outlook for further work in its context are given.

6.1 Final Summary

The main goal of this dissertation project was to exploit in-vehicle inter-ECU communication data transmitted over the various communication networks to predict the energy consumption of these control units and their associated actuators. The scope was therefore deliberately limited to the low-voltage power net of passenger vehicles. With the communication data harvested, a holistic ML processing pipeline was developed with the respective steps needed to reduce the high number of bus signals to those contributing to an energy consumption prediction. Other necessary parts of the pipeline were identified as an adequate data importing mechanism, and a process step in which an algorithm's hyperparameters are tuned with respect to the actual training data. To do so random search was selected for the latter.
Due to the numeric nature of the energy consumption prediction problem, a set of possibly suitable ML modeling algorithms was determined from which the "most suitable" was identified utilizing a weighted sum analysis (WSA). The criteria for this analysis were determined and weighted with domain and ML experts from the automotive industry. Applying a selection of 18 ECUs with training data collected from two test vehicles to the set of regression algorithms produced a considerable amount of results, enabling the assessment of each algorithm's suitability. From this procedure, the gradient boosting regressor (GB) was chosen as the "most suitable" algorithm since it provided the highest average and overall utility, given the WSA results.
Ultimately, three XAI methods as well as a Pearson correlation analysis were applied to selected power consumers, their respective ML models and input data. The associated results demonstrated the capability to "x-ray" the power consumer thereby revealing physical coherences which can then further be used by energy efficiency engineers to develop optimization measures.

6.2 Future Work

6.2.1 Complementary Methodologies

The presented methodology, which is to say the ML processing pipeline for energy consumption prediction, is built in a generic way with the goal to provide a tooling to handle all the different kinds of ECUs present in a vehicle. Therefore, future work could still comprise certain methodical specifications in a way that individual power consumers of interest can be more particularly fine-tuned with regard to the actual algorithm's hyperparameters. Hence, one could narrow down the hyperparameter

search ranges even more with certain manual effort if an even higher precision is needed. The same applies to the feature engineering and selection process step.
With regard to the evaluation of the generic pipeline—as it was conducted during this dissertation project—more trial runs could be conducted to average out the parameter selection through the random search. In the scope of this thesis one run per selected ECU was made. With more runs, more RS trials are conducted ultimately reaching a better statistical significance of the results which could lead to a more precise statement of the WSA, too. With a standard distribution anticipated, a rule of thumb states that 25 to 30 trials (per ECU) would return more robust results [199].

6.2.2 Application and Scaling of the ML Pipeline

Since the ML processing pipeline developed in the course of this dissertation project not only targets real-world data usage but also real-world application, it has been designed to ensure practical usability and scalability. Nonetheless, it must currently be executed as a python application requiring a certain knowledge of how to run scripts in a terminal, how to set-up virtual environments and how to incorporate the corresponding training data correctly. Therefore, a subsequent step in the development could be to wrap the ML processing pipeline compiled here in a stand-alone tool with a user-friendly interface so that engineers can access and use it more easily. Additionally, it is imaginable to display the validation and XAI results directly in the tool as well. This way, the pipeline logic is easily distributable and scalable, therefore providing an economic benefit eventually surpassing its development effort.
Furthermore, according to the CRISP-ML(Q) process model continuous monitoring and validation of the ML models and their predictive quality for the supported use cases is to be ensured, too. However, as the world of AI and ML evolves rapidly and especially with the emerging trends in the field of GenAI applications, enhancements to the methods incorporated in this project could be achieved. Additionally, generative AI could assist users of the pipeline in running and parameterizing it more efficiently while also speeding up the analysis of training data for missing driving situations. It could further be used to generate synthetic driving scenarios to augment training datasets, thus improving model generalization and robustness. Moreover, GenAI could support automated feature engineering by generating new candidate features or by suggesting transformations to enhance model performance. This would reduce manual effort while optimizing predictive power. Furthermore, the newly emerging multi-agent-based LLM technology—where LLMs have access to productive (engineering) tools and systems [200]—could be leveraged to automatically access data sources and documentation systems to store results, retrieve training data, or even to book and reserve test vehicles for training data collection independently. This would give engineers more time and capacity for the actual interpretation of the results. Ultimately, GenAI could enhance explainability by generating natural language reports that summarize model decisions, making the complex ML outputs more accessible to non-experts or to ECU owners to discuss optimization measures more thoroughly. This would facilitate better collaboration between data scientists and domain and ECU experts, driving more informed decision-making.

Bibliography

[1] Precedence Research, Internal combustion engine market volume, report by 2033, Tech. rep.
URL https://www.precedenceresearch.com/internal-combustion-engine-market (2024), last accessed: July 30, 2024.

[2] European Parliament, Euro 7 deal on new eu rules to reduce road transport emissions
URL https://www.europarl.europa.eu/news/en/press-room/20231207IPR15740/euro-7-deal-on-new-eu-rules-to-reduce-road-transport-emissions (2023), last accessed: July 30, 2024.

[3] McKinsey & Company, Five trends shaping tomorrow's luxury car market
URL https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/five-trends-shaping-tomorrows-luxury-car-market (2022), last accessed: July 30, 2024.

[4] European Commission, Eu climate action
URL https://climate.ec.europa.eu/eu-action_en (2023), last accessed: July 30, 2024.

[5] U.S. Environmental Protection Agency, Regulations for greenhouse gas emissions from passenger cars and light trucks
URL https://www.epa.gov/regulations-emissions-vehicles-and-engines/regulations-greenhouse-gas-emissions-passenger-cars-and (2024), last accessed: July 30, 2024.

[6] World Economic Forum, Fostering effective energy transition
URL https://www.weforum.org/reports/fostering-effective-energy-transition-2022 (2022), last accessed: July 30, 2024.

[7] R. Hausmann, R. Rigobon, An alternative interpretation of the 'resource curse': Theory and policy implications, Working Paper 9424, National Bureau of Economic Research
URL http://www.nber.org/papers/w9424 (January 2003).

[8] F. Mohammadi, M. Saif, A comprehensive overview of electric vehicle batteries market, e-Prime-Advances in Electrical Engineering, Electronics and Energy 3 (2023) 100–127.

[9] M. Farghali, A. I. Osman, Z. Chen, A. Abdelhaleem, I. Ihara, I. M. Mohamed, P.-S. Yap, D. W. Rooney, Social, environmental, and economic consequences of integrating renewable energies in the electricity sector: a review, Environmental Chemistry Letters 21 (3) (2023) 1381–1418.

[10] European Commission, European union (eu27) vehicles and fleet
URL https://alternative-fuels-observatory.ec.europa.eu/

transport-mode/road/european-union-eu27/vehicles-and-fleet (2024), last accessed: July 30, 2024.

[11] Bundesministerium für Wirtschaft und Klimaschutz, Umweltbonus endet mit Ablauf des 17. Dezember 2023
URL https://www.bmwk.de/Redaktion/DE/Pressemitteilungen/2023/12/20231216-umweltbonus-endet-mit-ablauf-des-17-dezember-2023.html (12 2023), last accessed: August 25, 2024.

[12] The Federal Council, Federal council decides to remove exemption from duty for e-vehicles
URL https://www.admin.ch/gov/en/start/documentation/media-releases.msg-id-98500.html (2023), last accessed: September 06, 2024.

[13] France Info, Baisse du bonus écologique pour les voitures électriques : un "coup de couteau dans le dos" des constructeurs, estime le président de l'association 40 millions d'automobilistes
URL https://www.francetvinfo.fr/economie/automobile/baisse-du-bonus-ecologique-pour-les-voitures-electriques-un-coup-de-couteau-dans-le-dos-des-constructeurs-estime-le-president-de-l-association-40-millions-d-automobilistes_6268560.html (2021), last accessed: August 01, 2024.

[14] International Energy Agency, Global ev outlook 2024: Trends in electric cars
URL https://www.iea.org/reports/global-ev-outlook-2024/trends-in-electric-cars (2024), last accessed: August 01, 2024.

[15] W. Zhou, C. J. Cleaver, C. F. Dunant, J. M. Allwood, J. Lin, Cost, range anxiety and future electricity supply: A review of how today's technology trends may influence the future uptake of BEVs, Renew. Sustain. Energy Rev. 173 (113074) (2023) 113074.

[16] M. Weiss, K. C. Cloos, E. Helmers, Energy efficiency trade-offs in small to large electric vehicles, Environmental Sciences Europe 32 (1) (2020) 46. doi:10.1186/s12302-020-00307-8.

[17] I. Kabashkin, L. Kurpniece, Investigating the impact of electric vehicle integration on the power system frequency response, RTU Environmental and Climate Technologies 24 (1) (2020) 41–51. doi:10.2478/rtuect-2020-0041.

[18] I. Miri, A. Fotouhi, N. Ewin, Electric vehicle energy consumption modelling and estimation—a case study, International Journal of Energy Research 45 (1) (2021) 501–520. doi:10.1002/er.5700.

[19] W.-H. Hucho, Chapter 1 - introduction to automobile aerodynamics, in: W.-H. Hucho (Ed.), Aerodynamics of Road Vehicles, Butterworth-Heinemann, 1987, pp. 1–46. doi:10.1016/B978-0-7506-1267-8.50005-1.

[20] Transportation Research Board, Special Report 286: Tires and Passenger Vehicle Fuel Economy: Informing Consumers, Improving Performance,
URL https://onlinepubs.trb.org/onlinepubs/sr/sr286.pdf Transportation Research Board, Washington, D.C., 2006.

[21] M. Trzesniowski, Powertrain, Springer Vieweg Wiesbaden, 2023. doi:10.1007/978-3-658-39885-9.

[22] U. Beyer, O. Ullrich, Organizational complexity as a contributing factor to underperformance, Businesses 2 (1) (2022) 82–96. doi:10.3390/businesses2010005.

[23] J. Hughes, A. Hetrick, Lost in the matrix: How to overcome the daily grind of organizational complexity, California Management Review
URL https://cmr.berkeley.edu/2021/12/lost-in-the-matrix-how-to-overcome-the-daily-grind-of-organizational-complexity (2021), last accessed: July 31, 2024.

[24] J. G. Proakis, Digital signal processing: principles, algorithms, and applications, 4/E, Pearson Education India, 2007.

[25] J. Mueller, J. Sprave, G. Frey, The effect of target variable rescaling on energy consumption prediction in a vehicle powernet using multi-target regression trees: A study from the automotive industry, in: Proceedings of the 2023 8th International Conference on Machine Learning Technologies, ICMLT '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 144–150. doi:10.1145/3589883.3589905.

[26] J. Mueller, L. Czekalla, F. Schuchter, G. Frey, Illuminating the black box: A comparative study of explainable ai for interpreting time series data in vehicle power net consumption models, in: 2023 International Conference on Machine Learning and Applications (ICMLA), 2023, pp. 166–173. doi:10.1109/ICMLA58977.2023.00031.

[27] J. Mueller, F. Schuchter, D. Brauneis, G. Frey, Enhancing power net efficiency with data-driven consumption prediction - a machine learning approach, in: 2024 IEEE Intelligent Vehicles Symposium (IV), 2024, pp. 2896–2903. doi:10.1109/IV55156.2024.10588508.

[28] J. Mueller, F. Schuchter, G. Frey, Licht in der blackbox – eine vergleichsstudie unterschiedlicher explainable ai methoden zur erklärung von stromverbräuchen im niedervolt-bordnetz von kraftfahrzeugen, in: Automation 2024 - VDI-Berichte, VDI Verlag, Baden-Baden, Germany, 2024, pp. 339–350.

[29] J. Mueller, F. Schuchter, G. Frey, Verbrauchsprognose im kfz-bordnetz mit erklärbarer ki: Vorschlag einer datenbasierten methodik, atp magazin 8 (2024) 56–64. doi:10.17560/atp.v66i8.2735.

[30] J. Mueller, X. Liu, J. Maier, G. Frey, Transferability of machine learning models for energy consumption prediction of in-vehicle low-voltage equipment: A case study, in: 2024 IEEE 20th International Conference on Automation Science and Engineering (CASE), 2024, pp. 1889–1896. doi:10.1109/CASE59546.2024.10711740.

[31] J. Mueller, J. Wagenknecht, F. Schuchter, G. Frey, Proposal of an automated feature engineering pipeline for high-dimensional in-vehicle bus data using reinforcement learning, in: Proceedings of the 2024 International Conference on

Robotics, Automation, and Artificial Intelligence (RAAI), IEEE, Singapore, 2024, to be published in IEEE Xplore.

[32] W. Nerreter, Grundlagen der Elektrotechnik, 4th Edition, URL http://www.content-select.com/index.php?id=bib_view&ean=9783446481596 Carl Hanser Verlag, 2024.

[33] H. Bumiller, K. Tkotz, Fachkunde Elektrotechnik, Europa-Fachbuchreihe für elektrotechnische Berufe, URL https://books.google.de/books?id=5UMwoAEACAAJ Verlag Europa-Lehrmittel, Nourney, Vollmer GmbH & Company KG, 2014.

[34] J. Heintze, Der elektrische Strom: Die Grundlagen, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 75–86. doi:10.1007/978-3-662-48451-7_6.

[35] E. Hering, K. Bressler, J. Gutekunst, Elektronik für Ingenieure, 4th Edition, Springer-Lehrbuch, Springer, Berlin, Germany, 2001.

[36] R. Hoffmann, A. Bergmann, Betrieb von elektrischen Anlagen, 11th Edition, URL http://www.content-select.com/index.php?id=bib_view&ean=9783800743230 VDE Verlag, 2017.

[37] ZVEI - German Electrical and Electronic Manufacturers' Association, Voltage classes for electric mobility URL https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2014/april/Voltage_Classes_for_Electric_Mobility/Voltage_Classes_for_Electric_Mobility.pdf (2014), last accessed: August 06, 2024.

[38] International Organization for Standardization, Electrically propelled road vehicles – safety specifications – part 1: Rechargeable energy storage system (ress) URL https://www.iso.org/standard/68665.html (2019), last accessed: February 10, 2025.

[39] M. Weber, T. Mundinger, F. von Kageneck, M.-F. Buciuman, F. Häuser, Mercedes "s-class" powernet architecture, in: M. Bargende, H.-C. Reuss, A. Wagner (Eds.), 21. Internationales Stuttgarter Symposium, Springer Fachmedien Wiesbaden, Wiesbaden, 2021, pp. 3–12.

[40] D. Scheer, O. Glodd, H. Günther, Y. Duhr, A. Schmid, Star3 - eine neue generation der e/e-architektur, Sonderprojekte ATZ/MTZ 25 (1) (2020) 72–79. doi:10.1007/s41491-020-0056-5.

[41] S. Lu, W. Shi, Vehicle computing: Vision and challenges, Journal of Information and Intelligence 1 (1) (2023) 23–35. doi:10.1016/j.jiixd.2022.10.001.

[42] International Organization for Standardization, Iso 26262: Road vehicles – functional safety (2018).

[43] H. Zeng, P. Joshi, D. Thiele, J. Diemer, P. Axer, R. Ernst, P. Eles, Networked Real-Time Embedded Systems, Springer Netherlands, Dordrecht, 2017, pp. 1–40. doi:10.1007/978-94-017-7358-4_25-1.

[44] International Organization for Standardization, Road vehicles – controller area network (can) – part 1: Data link layer and physical coding sublayer
URL https://www.iso.org/standard/86384.html (2024), last accessed: August 10, 2024.

[45] International Organization for Standardization, Road vehicles – flexray communications system – part 1: General information and use case definition
URL https://www.iso.org/standard/59806.html (2013), last accessed: August 10, 2024.

[46] SAE International, Lin network for vehicle applications j2602/1_200408
URL https://set-doi.cir-mcs.i.mercedes-benz.com/J2602/1_200408 (Aug. 2004).

[47] D. Paret, Medium, Implementation and Physical Layers in CAN, John Wiley & Sons, Ltd, 2007, Ch. 4, pp. 125–177. doi:10.1002/9780470511770.ch4.

[48] D. Paret, CAN: Its Protocol, Its Properties, Its Novel Features, John Wiley & Sons, Ltd, 2007, Ch. 2, pp. 25–82. doi:10.1002/9780470511770.ch2.

[49] K. Borgeest, Elektronik in der Fahrzeugtechnik: Hardware, Software, Systeme und Projektmanagement, 2023. doi:10.1007/978-3-658-41483-2.

[50] D. Paret, The CAN Physical Layer, John Wiley & Sons, Ltd, 2007, Ch. 3, pp. 83–123. doi:10.1002/9780470511770.ch3.

[51] J. Wagenbach, CAN bus topology and bus termination, https://support.maxongroup.com/hc/en-us/articles/360009241840-CAN-bus-topology-and-bus-termination (2024), last accessed: August 29, 2024.

[52] D. Paret, LIN – Local Interconnect Network, John Wiley & Sons, Ltd, 2007, Ch. 7, pp. 285–311. doi:10.1002/9780470511770.ch7.

[53] E. Hackett, Lin protocol and physical layer requirements, Tech. Rep. SLLA383A, Texas Instruments
URL https://www.ti.com/lit/pdf/SLLA383 (February 2018), last accessed: September 15, 2024.

[54] D. Paret, Time-Triggered Protocols – FlexRay, John Wiley & Sons, Ltd, 2007, Ch. 6, pp. 231–272. doi:10.1002/9780470511770.ch6.

[55] M. Rausch, FlexRay: Grundlagen, Funktionsweise, Anwendung, 2007.

[56] National Instruments, Flexray automotive communication bus overview, Tech. rep., National Instruments
URL https://www.ni.com/en.html (2024), last accessed: July 1, 2024.

[57] IEEE Standards Association, IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks
URL https://ieeexplore.ieee.org/document/10029106 (2022), last accessed: January 31, 2025.

[58] L. Lo Bello, G. Patti, L. Leonardi, A perspective on ethernet in automotive communications—current status and future trends, Appl. Sci. (Basel) 13 (3) (2023) 1278.

[59] International Organization for Standardization, Road vehicles — media oriented systems transport (most) — part 6: Application layer
URL https://www.iso.org/standard/72294.html (2020), last accessed: January 31, 2025.

[60] MOST Cooperation, AAR THM 3V2, Tech. rep.
URL https://www.mostcooperation.com/specifications (Version 3.2), last accessed: January 20, 2025.

[61] Robert Bosch GmbH, Kraftfahrtechnisches Taschenbuch, Springer-Verlag, 2024.

[62] G. Babiel, M. Thoben, Bordnetze und Powermanagement, 3rd Edition, Springer Vieweg, Wiesbaden, Germany, 2022.

[63] M. Krüger, Grundlagen der Kraftfahrzeugelektronik, Carl Hanser Verlag GmbH & Co. KG, München, 2020.

[64] E. Commission, Commission regulation (eu) 2017/1151 of 1 june 2017 supplementing regulation (ec) no 715/2007 of the european parliament and of the council on type-approval of motor vehicles with respect to emissions from light passenger and commercial vehicles (euro 5 and euro 6) and on access to vehicle repair and maintenance information
URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32017R1151 (2017), last accessed: August 25, 2024.

[65] C. Xiao, L. Zhao, T. Asada, W. G. Odendaal, J. D. van Wyk, An overview of integratable current sensor technologies, in: 38th IAS Annual Meeting on Conference Record of the Industry Applications Conference, 2003, IEEE, 2004.

[66] IPETronik GmbH & Co. KG, IPEshunt 1 - Präzisionsmesswiderstand (Datenblatt)
URL https://www.ipetronik.com/files/products/IPEshunt%201_datasheet_de.pdf (2025), last accessed: January 22, 2025.

[67] Texas Instruments, An Engineer's Guide to Current Sensing (Rev. B),
URL https://www.ti.com/lit/eb/slyy154b/slyy154b.pdf Texas Instruments, 2019, last accessed: January 20, 2025.

[68] P. Weßkamp, J. Melbert, High-accuracy current measurement with low-cost shunts by means of dynamic error correction, J. Sens. Sens. Syst. 5 (2) (2016) 389–400.

[69] T. Wickramasinghe, S. Azzopardi, B. Allard, C. Buttay, C. Joubert, C. Martin, J.-F. Mogniotte, H. Morel, P. Bevilacqua, T.-L. Le, A study on shunt resistor-based current measurements for fast switching GaN devices, in: IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, IEEE, 2019.

[70] IPETRONIK GmbH & Co. KG, IPEshunt 1 - Automotive Current Measurement Shunt
URL https://www.ipetronik.com/de/sensoren/ipeshunt-1.html (2024), last accessed: January 21, 2025.

[71] IPETRONIK GmbH & Co. KG, M-SENS 2: 4-Channel Analog Measurement Module
URL https://www.ipetronik.com/en/products-services/sensors/m-sens-2.html (2023), last accessed: February 02, 2025.

[72] MAGNA Telemotive GmbH, BLUEPIRAT Rapid Benutzerhandbuch, MAGNA Telemotive GmbH
URL https://sc.telemotive.de/bluepirat (2020), last accessed: September 06, 2024.

[73] Vector Informatik GmbH, Logging formats - vector knowledge portal
URL https://support.vector.com/kb?id=kb_article_view&sysparm_article=KB0011536 (2024), last accessed: September 06, 2024.

[74] Vector Informatik GmbH, Vector - vsignalyzer, https://www.vector.com/int/en/products/products-a-z/software/vsignalyzer/#c340722 (2024), last accessed: September 08, 2024.

[75] Vector Informatik GmbH, CANdb++ Manual, Version 3.1, Vector Informatik GmbH, Stuttgart, Germany
URL https://cdn.vector.com/cms/content/products/candb/Docs/CANdb_Manual_EN.pdf (2019), last accessed: October 03, 2024.

[76] ASAM e.V., Asam mdf - measurement data format, https://www.asam.net/standards/detail/mdf/wiki/ (2024), last accessed: September 08, 2024.

[77] A. M. Turing, Computing machinery and intelligence, Mind LIX (236) (1950) 433–460. doi:10.1093/mind/LIX.236.433.

[78] R. T. Kreutzer, Künstliche Intelligenz verstehen, Springer Fachmedien Wiesbaden, Wiesbaden, 2023.

[79] C. Stryker, M. Scapicchio, What is generative ai?, https://www.ibm.com/topics/generative-ai (March 2024), last accessed: September 13, 2024.

[80] R. Egger, Applied data science in tourism: Interdisciplinary approaches, methodologies, and applications, Springer Nature, Cham, Switzerland, 2022.

[81] H. R. Arabnia, K. Daimi, R. Stahlbock, C. Soviany, L. Heilig, K. Brüssau (Eds.), Principles of data science, 1st Edition, Transactions on Computational Science and Computational Intelligence, Springer Nature, Cham, Switzerland, 2020.

[82] H. Sheikh, C. Prins, E. Schrijvers, Mission AI: The New System Technology, Springer Nature, Cham, Switzerland, 2023.

[83] P. Perrotta, Machine Learning programmieren: Von der Codezeile zum ML-System, 2020.

[84] W. Ertel, Grundkurs Künstliche Intelligenz, 5th Edition, Computational Intelligence, Springer Vieweg, Wiesbaden, Germany, 2021.

[85] C. Bishop, Pattern Recognition and Machine Learning, 1st Edition, Information Science and Statistics, Springer, New York, NY, 2006.

[86] K. P. Murphy, Probabilistic machine learning, MIT Press, London, England, 2023.

[87] Y. Bengio, Deep Learning, Adaptive Computation and Machine Learning series, MIT Press, London, England, 2016.

[88] U. Kamath, J. Liu, Explainable artificial intelligence: An introduction to interpretable machine learning, Springer Nature, Cham, Switzerland, 2021.

[89] J. Saltz, What is the ai life cycle?
URL `https://www.datascience-pm.com/ai-lifecycle/` (2024), last accessed: September 01, 2024.

[90] D. Kutzias, C. Dukino, F. Kötter, H. Kett, Comparative analysis of process models for data science projects, in: Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 3, SCITEPRESS - Science and Technology Publications, 2023, pp. 1052–1062. `doi:10.5220/0011951910521062`.

[91] S. Studer, T. B. Bui, C. Drescher, A. Hanuschkin, L. Winkler, S. Peters, K.-R. Müller, Towards crisp-ml(q): A machine learning process model with quality assurance methodology, Machine Learning and Knowledge Extraction 3 (2) (2021) 392–413. `doi:10.3390/make3020020`.

[92] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[93] J. Hao, T. K. Ho, Machine learning made easy: A review of *Scikit-learn* package in python programming language, J. Educ. Behav. Stat. 44 (3) (2019) 348–361.

[94] K. Ramasubramanian, A. Singh, Deep learning using keras and TensorFlow, in: Machine Learning Using R, Apress, Berkeley, CA, 2019, pp. 667–688.

[95] N. L. Rane, S. K. Mallick, Ö. Kaya, J. Rane, Tools and frameworks for machine learning and deep learning: A review, in: Applied Machine Learning and Deep Learning: Architectures and Techniques, Deep Science Publishing, 2024.

[96] G. A. F. Seber, A. J. Lee, Linear Regression Analysis, 2nd Edition, Wiley Series in Probability and Statistics, John Wiley & Sons, Nashville, TN, 2012.

[97] S. Chatterjee, A. S. Hadi, Regression Analysis by Example, 5th Edition, Wiley Series in Probability and Statistics, Wiley-Blackwell, Hoboken, NJ, 2012.

[98] G. James, D. Witten, T. Hastie, R. Tibshirani, An Introduction to Statistical Learning: With applications in R, Springer Nature, Cham, Switzerland, 2021.

[99] G. De'ath, Multivariate regression trees: A new technique for modeling species–environment relationships, Ecology 83 (4) (2002) 1105–1117.

[100] J. R. Quinlan, Induction of decision trees, Machine Learning 1 (1) (1986) 81–106. doi:10.1007/BF00116251.

[101] R. V. McCarthy, M. M. McCarthy, W. Ceccucci, L. Halawi, R. McCarthy, M. McCarthy, W. Ceccucci, L. Halawi, Applying predictive analytics, Springer, 2022.

[102] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140. doi:10.1007/BF00058655.

[103] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32. doi:10.1023/A:1010933404324.

[104] E. W. Steyerberg, F. E. Harrell, G. J. Borsboom, M. Eijkemans, Y. Vergouwe, J. F. Habbema, Internal validation of predictive models: Efficiency of some procedures for logistic regression analysis, Journal of Clinical Epidemiology 54 (8) (2001) 774–781. doi:10.1016/S0895-4356(01)00341-9.

[105] Z.-H. Zhou, Ensemble methods: foundations and algorithms, CRC press, 2012.

[106] L. Mason, J. Baxter, P. Bartlett, M. Frean, Boosting algorithms as gradient descent, in: S. Solla, T. Leen, K. Müller (Eds.), Advances in Neural Information Processing Systems, Vol. 12,
URL https://proceedings.neurips.cc/paper_files/paper/1999/file/96a93ba89a5b5c6c226e49b88973f46e-Paper.pdf MIT Press, 1999, last accessed: January 30, 2025.

[107] J. H. Friedman, Greedy function approximation: A gradient boosting machine., The Annals of Statistics 29 (5) (2001) 1189 – 1232. doi:10.1214/aos/1013203451.

[108] T. Hastie, R. Tibshirani, J. Friedman, The elements of statistical learning, 2nd Edition, Springer series in statistics, Springer, New York, NY, 2009.

[109] A. Jain, Complete guide to parameter tuning in gradient boosting (gbm) in python
URL https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python (2024), last accessed: December 14, 2024.

[110] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics 5 (4) (1943) 115–133.

[111] K. Gurney, An Introduction to Neural Networks, Routledge, London, England, 1997.

[112] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (6) (1958) 386–408.

[113] R. Rojas, Neural Networks, Springer, Berlin, Germany, 1996.

[114] I. A. Basheer, M. Hajmeer, Artificial neural networks: fundamentals, computing, design, and application, Journal of microbiological methods 43 (1) (2000) 3–31.

[115] B. Karlik, A. V. Olgac, Performance analysis of various activation functions in generalized mlp architectures of neural networks, International Journal of Artificial Intelligence and Expert Systems 1 (4) (2011) 111–122.

[116] J. Lederer, Activation functions in artificial neural networks: A systematic overview (2021). arXiv:2101.09957.

[117] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, PMLR, 2011,
URL https://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf pp. 315–323, last accessed: January 30, 2025.

[118] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10, Omnipress, Madison, WI, USA, 2010, p. 807–814.

[119] L. Lu, Y. S. Yeonjong Shin, Y. S. Yanhui Su, G. E. K. George Em Karniadakis, Dying relu and initialization: Theory and numerical examples, Communications in Computational Physics (2020) 1671–1706doi:10.4208/cicp.oa-2020-0165.

[120] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks (2013). arXiv:1211.5063.

[121] C. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall, Activation functions: Comparison of trends in practice and research for deep learning, CoRR abs/1811.03378 (2018). arXiv:1811.03378.

[122] F. Murtagh, Multilayer perceptrons for classification and regression, Neurocomputing 2 (5-6) (1991) 183–197.

[123] Y. A. LeCun, L. Bottou, G. B. Orr, K.-R. Müller, Efficient BackProp, in: Lecture Notes in Computer Science, Lecture notes in computer science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 9–48.

[124] Y. Goldberg, A primer on neural network models for natural language processing (2015). arXiv:1510.00726.

[125] M. Riedmiller, Advanced supervised learning in multi-layer perceptrons—from backpropagation to adaptive learning algorithms, Computer Standards & Interfaces 16 (3) (1994) 265–278.

[126] D. E. Rumelhart, J. L. McClelland, Learning Internal Representations by Error Propagation, 1987, pp. 318–362.

[127] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Transactions on Neural Networks 5 (2) (1994) 157–166. doi:10.1109/72.279181.

[128] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2017). arXiv:1412.6980.

[129] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning,
URL http://www.deeplearningbook.org MIT Press, 2016, last accessed: September 05, 2024.

[130] Y. Bengio, A. Courville, P. Vincent, Representation learning: A review and new perspectives (2012). doi:10.48550/arXiv.1206.5538.

[131] A. Sherstinsky, Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network, Physica D: Nonlinear Phenomena 404 (2020) 132306. doi:10.1016/j.physd.2019.132306.

[132] F. A. Gers, N. N. Schraudolph, J. Schmidhuber, Learning precise timing with lstm recurrent networks, Journal of Machine Learning Research 3 (2002) 115–143.

[133] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[134] F. Chollet, et al., Keras
URL https://github.com/keras-team/keras (2015), last accessed: January 30, 2025.

[135] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15 (56) (2014) 1929–1958.

[136] A. Botchkarev, Evaluating performance of regression machine learning models using multiple error metrics in azure machine learning studio, SSRN Electron. J. (2018).

[137] V. Plevris, G. Solorzano, N. Bakas, M. Ben Seghier, Investigation of performance metrics in regression analysis and machine learning-based prediction models, 2022. doi:10.23967/eccomas.2022.155.

[138] Z. Yang, W. Gao, C. Luo, L. Wang, F. Tang, X. Wen, J. Zhan, Quality at the tail of machine learning inference (2024). arXiv:2212.13925.

[139] L. Baier, F. Jöhren, S. Seebacher, Challenges in the deployment and operation of machine learning in practice, in: Proceedings of the 27th European Conference on Information Systems (ECIS),
URL https://aisel.aisnet.org/ecis2019_rp/163 2019, last accessed: January 30, 2025.

[140] S. Khalid, T. Khalil, S. Nasreen, A survey of feature selection and feature extraction techniques in machine learning, in: 2014 Science and Information Conference, IEEE, 2014.

[141] K. Maharana, S. Mondal, B. Nemade, A review: Data pre-processing and data augmentation techniques, Global Transitions Proceedings 3 (1) (2022) 91–99.

[142] J. Waring, C. Lindvall, R. Umeton, Automated machine learning: Review of the state-of-the-art and opportunities for healthcare, Artificial Intelligence in Medicine 104 (2020) 101822. doi:10.1016/j.artmed.2020.101822.

[143] T. Verdonck, B. Baesens, M. Óskarsdóttir, S. vanden Broucke, Special issue on feature engineering editorial, Mach. Learn. 113 (7) (2024) 3917–3928.

[144] J. Han, M. Kamber, J. Pei, 3 - data preprocessing, in: J. Han, M. Kamber, J. Pei (Eds.), Data Mining (Third Edition), third edition Edition, The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann, Boston, 2012, pp. 83–124. doi:10.1016/B978-0-12-381479-1.00003-4.

[145] H. Kang, The prevention and handling of the missing data, Korean J. Anesthesiol. 64 (5) (2013) 402–406.

[146] R. J. A. Little, D. B. Rubin, Statistical analysis with missing data, 3rd Edition, Wiley Series in Probability and Statistics, John Wiley & Sons, Nashville, TN, 2019.

[147] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, H. Liu, Feature selection, ACM Comput. Surv. 50 (6) (2018) 1–45.

[148] G. Wei, J. Zhao, Y. Feng, A. He, J. Yu, A novel hybrid feature selection method based on dynamic feature importance, Applied Soft Computing 93 (2020) 106337. doi:10.1016/j.asoc.2020.106337.

[149] J. T. Hancock, T. M. Khoshgoftaar, Survey on categorical data for neural networks, J. Big Data 7 (1) (Dec. 2020).

[150] C. Rudin, C. Chen, Z. Chen, H. Huang, L. Semenova, C. Zhong, Interpretable machine learning: Fundamental principles and 10 grand challenges, Stat. Surv. 16 (Jan. 2022).

[151] A. Adadi, M. Berrada, Peeking inside the black-box: A survey on explainable artificial intelligence (xai), IEEE Access 6 (2018) 52138–52160. doi:10.1109/ACCESS.2018.2870052.

[152] J. Amann, A. Blasimme, E. Vayena, D. Frey, V. I. Madai, Precise4Q consortium, Explainability for artificial intelligence in healthcare: a multidisciplinary perspective, BMC Med. Inform. Decis. Mak. 20 (1) (2020) 310.

[153] E. Union, Regulation (eu) 2024/1689 of the european parliament and of the council
URL https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32024R1689 (2024), last accessed: November 01, 2024.

[154] T. W. House, Blueprint for an ai bill of rights: Making automated systems work for the american people.
URL https://www.whitehouse.gov/ostp/ai-bill-of-rights/ (2022), last accessed: November 01, 2024.

[155] C. Molnar, Interpretable Machine Learning: A Guide for Making Black Box Models Explainable,
URL https://christophm.github.io/interpretable-ml-book/ Independent, 2020, last accessed: December 08, 2024.

[156] D. W. Apley, J. Zhu, Visualizing the effects of predictor variables in black box supervised learning models (2016). arXiv:1612.08468.

[157] S. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions (2017). arXiv:1705.07874.

[158] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, S.-I. Lee, Explainable AI for trees: From local explanations to global understanding (2019). arXiv:1905.04610.

[159] C. Thornton, F. Hutter, H. H. Hoos, K. Leyton-Brown, Auto-weka: combined selection and hyperparameter optimization of classification algorithms, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 847–855. doi:10.1145/2487575.2487629.

[160] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, F. Hutter, Auto-sklearn 2.0: Hands-free automl via meta-learning (2022). arXiv:2007.04074.

[161] G. Dittmer, Nutzwertanalyse, in: Managen mit Methode, Gabler Verlag, Wiesbaden, 1995, pp. 43–56.

[162] P. C. Fishburn, Letter to the editor – additive utilities with incomplete product sets: Application to priorities and assignments, Operations Research 15 (3) (1967) 537–542.

[163] T. Glatte, Kompendium Standortstrategien für Unternehmensimmobilien, 1st Edition, Leitfaden des Baubetriebs und der Bauwirtschaft, Springer Fachmedien, Wiesbaden, Germany, 2018.

[164] L. Yang, A. Shami, On hyperparameter optimization of machine learning algorithms: Theory and practice, Neurocomputing 415 (2020) 295–316. doi:10.1016/j.neucom.2020.07.061.

[165] F. Hutter, L. Kotthoff, J. Vanschoren (Eds.), Automatic machine learning, 1st Edition, The Springer Series on Challenges in Machine Learning, Springer International Publishing, Basel, Switzerland, 2019.

[166] K. Eggensperger, F. Hutter, H. H. Hoos, K. Leyton-Brown, Efficient benchmarking of hyperparameter optimizers via surrogates, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI'15, AAAI Press, 2015, p. 1114–1120.

[167] D. Passos, P. Mishra, A tutorial on automatic hyperparameter tuning of deep spectral modelling for regression and classification tasks, Chemometr. Intell. Lab. Syst. 223 (104520) (2022) 104520.

[168] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (2012) 281–305.

[169] P. Liashchynskyi, P. Liashchynskyi, Grid search, random search, genetic algorithm: A big comparison for NAS (2019). arXiv:1912.06059.

[170] P. I. Frazier, A tutorial on bayesian optimization (2018). arXiv:1807.02811.

[171] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms (2012). arXiv:1206.2944.

[172] R. Isermann, Mechatronic Systems, Springer, London, England, 2005.

[173] A. Gonzalez-Briones, J. Prieto, F. De La Prieta, E. Herrera-Viedma, J. M. Corchado, Energy optimization using a case-based reasoning strategy, Sensors 18 (3) (2018). doi:10.3390/s18030865.

[174] Z. Wang, R. S. Srinivasan, A review of artificial intelligence based building energy use prediction: Contrasting the capabilities of single and ensemble prediction models, Renewable and Sustainable Energy Reviews 75 (2017) 796–808. doi:10.1016/j.rser.2016.10.079.

[175] J. Zhang, Z. Wang, P. Liu, Z. Zhang, Energy consumption analysis and prediction of electric vehicles based on real-world driving data, Applied Energy 275 (2020) 115408. doi:10.1016/j.apenergy.2020.115408.

[176] E. Kim, J. Lee, K. G. Shin, Real-time prediction of battery power requirements for electric vehicles, in: Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems, ICCPS '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 11–20. doi:10.1145/2502524.2502527.

[177] Y. Fan, Z. Wang, S. Pashami, S. Nowaczyk, H. Ydreskog, Forecasting auxiliary energy consumption for electric heavy-duty vehicles (2023). arXiv:2311.16003.

[178] Z. Feng, J. Zhang, H. Jiang, X. Yao, Y. Qian, H. Zhang, Energy consumption prediction strategy for electric vehicle based on lstm-transformer framework, Energy 302 (2024) 131780. doi:10.1016/j.energy.2024.131780.

[179] K. Liu, J. Wang, T. Yamamoto, T. Morikawa, Exploring the interactive effects of ambient temperature and vehicle auxiliary loads on electric vehicle energy consumption, Appl. Energy 227 (2018) 324–331.

[180] L. Schäfers, K. Franke, R. Savelsberg, S. Pischinger, Auxiliaries' power and energy demand prediction of battery electric vehicles using system identification and deep learning, IET Intell. Transp. Syst. 18 (4) (2024) 743–754.

[181] C. De Cauwer, J. Van Mierlo, T. Coosemans, Energy consumption prediction for electric vehicles based on real-world data, Energies 8 (8) (2015) 8573–8593. doi:10.3390/en8088573.

[182] Q. Zhu, Y. Huang, C. F. Lee, P. Liu, J. Zhang, T. Wik, Predicting electric vehicle energy consumption from field data using machine learning, IEEE Transactions on Transportation Electrification (2024) 1–1doi:10.1109/TTE.2024.3416532.

[183] D. Hrisca, Fast python reader and editor for asam mdf / mf4 (measurement data format) files
URL https://github.com/danielhrisca/asammdf (2024), last accessed: October 05, 2024.

[184] W. McKinney, Data structures for statistical computing in python, in: S. van der Walt, J. Millman (Eds.), Proceedings of the 9th Python in Science Conference, 2010, pp. 56–61. doi:10.25080/Majora-92bf1922-00a.

[185] ONNX Community, Open neural network exchange (onnx), https://onnx.ai/ (2024), last accessed: November 24, 2024.

[186] I. Corporation, Intel® xeon® gold 6226r processor specifications, https://www.intel.de/content/www/de/de/products/sku/199347/intel-xeon-gold-6226r-processor-22m-cache-2-90-ghz/specifications.html (2020), last accessed: January 02, 2025.

[187] J. Heaton, An empirical analysis of feature engineering for predictive modeling, in: SoutheastCon 2016, IEEE, 2016.

[188] A. Altmann, L. Toloşi, O. Sander, T. Lengauer, Permutation importance: a corrected feature importance measure, Bioinformatics 26 (10) (2010) 1340–1347.

[189] P. Probst, Hyperparameters, tuning and meta-learning for random forest and other machine learning algorithms,
URL https://api.semanticscholar.org/CorpusID:201710457 2019, last accessed: January 30, 2024.

[190] T. M. Oshiro, P. S. Perez, J. A. Baranauskas, How many trees in a random forest?, in: Machine Learning and Data Mining in Pattern Recognition, Lecture notes in computer science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 154–168.

[191] N. Jermain, Optimizing hyperparameters for random forest algorithms in scikit-learn
URL https://opendatascience.com/optimizing-hyperparameters-for-random-forest-algorithms-in-scikit-learn/ (2019), last accessed: December 02, 2024.

[192] K. Madhusudhanan, S. Jawed, L. Schmidt-Thieme, Hyperparameter tuning MLPs for probabilistic time series forecasting (Mar. 2024). arXiv:2403.04477.

[193] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, F. Hutter, Neural architecture search: Insights from 1000 papers (2023). arXiv:2301.08727.

[194] S. Wang, C. Ma, Y. Xu, J. Wang, W. Wu, A hyperparameter optimization algorithm for the LSTM temperature prediction model in data center, Sci. Program. 2022 (2022) 1–13.

[195] R. Jozefowicz, W. Zaremba, I. Sutskever, An empirical exploration of recurrent network architectures, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, JMLR.org, 2015, p. 2342–2350.

[196] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, I. Stoica, Tune: A research platform for distributed model selection and training, arXiv preprint arXiv:1807.05118 (2018).

[197] S. Raschka, V. Mirjalili, Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition, Packt Publishing, Birmingham, 2020.

[198] J. M. Zhang, M. Harman, L. Ma, Y. Liu, Machine learning testing: Survey, landscapes and horizons, IEEE Transactions on Software Engineering 48 (1) (2022) 1–36. doi:10.1109/TSE.2019.2962027.

[199] J. A. Rice, Mathematical Statistics and Data Analysis, 3rd Edition, Duxbury Press, Belmont, CA, 2006.

[200] C. Sypherd, V. Belle, Practical considerations for agentic llm systems (2024). arXiv:2412.04093.

Appendices

A Test Vehicle Options Lists

This appendix presents a comprehensive inventory of the key options and configurations available on both test Vehicle A and Vehicle B, as specified by the OEM.

A.1 Options List Vehicle A

Table A.1: Option codes and descriptions list for Vehicle A (excerpt).

Code	Option Description
H29	ZIERELEMENTE-HOLZ WURZELNUSS GLAENZEND (2B29)
K32	AKTIVER FAHRSPURWECHSELASSISTENT
L	LINKS-LENKUNG
L2B	LUXUS-LENKRAD LEDER
M005	FAHRZEUGE MIT 4-MATIC-/ALLRAD-ANTRIEB
M656	R6-DIESELMOTOR OM656
PBG	KONNEKTIVITAETSPAKET PREMIUM
P09	SONNENSCHUTZ-PAKET
P20	FAHRASSISTENT-PAKET PLUS
P21	LUFTQUALITAETSPAKET
P43	SITZKOMFORT-PAKET IM FOND
P47	EINPARK-PAKET HIGH
P64	MEMORY PAKET VORNE
P69	KOMFORT-HEIZPAKET VORNE U.HINTEN
P82	DIEBSTAHLSCHUTZ-PAKET PLUS
U01	FONDGURTSTATUSANZEIGE
U10	BEIFAHRERSITZ MIT GEWICHTSERKENNUNG
U19	AUGMENTED REALITY VIDEO
U25	EINSTIEGSCHIENE BELEUCHTET
U60	FUSSGAENGERSCHUTZ - AKTIVE MOTORHAUBE
01U	MBCONNECT - DIENSTE FUER NAVIGATION
049U	DESIGNO-MAGNO-KASCHMIRWEISS LACKIERUNG
1B3	KOMMUNIKATIONSMODUL ECE-AUSFUEHRUNG
2U1	KUEHLERJALOUSIE
2XXL	BUNDESREPUBLIK DEUTSCHLAND
223	FONDLEHNENVERSTELLUNG UND KOPFSTUETZEN ELEKTRISCH
233	ABSTANDSREGELTEMPOMAT PLUS (DISTRONIC PLUS)
235	AKTIVER PARK-ASSISTENT
243	AKTIVER SPURHALTE-ASSISTENT (KAMERA)
249	INNEN- UND AUSSENSPIEGEL AUTOMATISCH ABBLENDBAR
266	DISTRONIC PLUS QUERUNTERSTUETZUNG (DTR+Q)
275	MEMORY-PAKET (FAHRERSITZ, LENKSAEULE U. SPIEGEL)
276	MEMORY IM FOND
292	PRE-SAFE IMPULS SEITE

Continued on next page

Table A.1 – continued from previous page

Code	Option Description
293	SIDEBAG IM FOND LINKS UND RECHTS
297	SONNENSCHUTZROLLO ELEKTRISCH IN FONDTUER LI.U.RE
322	NACKENWAERMER IM FOND
325	MITTENAIRBAG
33U	MBCONNECT - ERWEITERTE KONNEKTIVITAET
351	ECALL-NOTRUFSYSTEM
365	FESTPLATTEN-NAVIGATION
367	LIVE TRAFFIC - FAEHIGKEIT
380	AUTO-TELEFONIE-PAKET
401	SITZKLIMATISIERUNG VORNE
402	SITZKLIMATISIERUNG HINTEN
406	MULTIKONTURSITZ IM FOND
413	PANORAMA-SCHIEBEDACH/-GLASDACH
421	GETRIEBE AUTOMATISCH 9-GANG
432	FAHRDYNAMISCHER SITZ LINKS UND RECHTS
436	KOMFORTKOPFSTUETZE FAHRER UND BEIFAHRER
443	LENKRAD HEIZBAR
447	BERUEHRBILDSCHIRM IM FOND (TOUCHSCREEN REAR)
475	REIFENDRUCKKONTROLLE (RDK)
489	AIRMATIC DC / LUFTFEDERUNG SEMIAKTIV
501	360°-KAMERA
513	VERKEHRSZEICHENERKENNUNG
534	CONNECT 20 PREMIUM (NTG7)
540	ROLLO ELEKTRISCH FUER HECKFENSTER
546	AUTOMATISCHE GESCHWINDIGKEITSREGELUNG
551	EINBRUCH- UND DIEBSTAHLWARNANLAGE BASIS (EDW)
581	KLIMATISIERUNGSAUTOMATIC
582	KLIMAANLAGE IM FOND
587	LOGO-PROJEKTION UEBER SPIEGEL
628	AUTOMATISCHE FERNLICHTSCHALTUNG PLUS (IHC+)
642	SCHEINWERFER LED DYNAMISCH RECHTSVERKEHR
682	FEUERLOESCHER
72B	USB-PAKET
8U0	BEHEIZTER WISCHWASSERBEHAELTER
804A	LEDER / NAPPA / SEMIANILIN - BRAUN
810	SOUNDSYSTEM PREMIUM
856	TELEFONBEDIENHOERER IN FONDARMLEHNE
868	ZENTRALDISPLAY GROESSE L
874	MAGIC VISION CONTROL INKL.SCHEIBENWASCHANL.BEHEIZT
883	SERVOSCHLIESSUNG
889	KEYLESS - GO
891	AMBIENTENBELEUCHTUNG PREMIUM
897	DRAHTLOSE TELEFONAUFLADUNG VORNE
898	DRAHTLOSE TELEFON-AUFLADUNG IM FOND
902	KOMFORTSITZHEIZUNG VORNE
903	KOMFORTSITZHEIZUNG HINTEN
906	AUFLAGEN VORNE BEHEIZT
907	AUFLAGEN HINTEN BEHEIZT

A.2 Options List Vehicle B

Table A.2: Option codes and descriptions list for Vehicle B (excerpt).

Code	Option Description
B53	AUSSENGERAEUSCH-SOUNDGENERATOR FUER HYBRIDE UND EV
GA	GETRIEBE AUTOMATISCH
H08	ZIERELEMENTE - HOLZ FINELINE ANTHRAZIT (2C09)
K32	AKTIVER FAHRSPURWECHSELASSISTENT
K33	ERWEITERTES WIEDERANFAHREN BEI STOP-AND-GO-VERKEHR
K34	STRECKENBASIERTE GESCHWINDIGKEITSANPASSUNG
L	LINKS-LENKUNG
L3E	SPORT-LENKRAD - SPALTLEDER GLATT
M005	FAHRZEUGE MIT 4-MATIC-/ALLRAD-ANTRIEB
PAG	DISPLAY-PAKET HIGH
PAX	LICHT-PAKET-PREMIUM
PBG	KONNEKTIVITAETSPAKET PREMIUM
PBH	REMOTE PARKING-PAKET
PBR	PAKET FIT UND GESUND - HIGH
P20	FAHRASSISTENT-PAKET PLUS
P21	LUFTQUALITAETSPAKET
P24	KEYLESS-GO-PAKET HIGH
P49	SPIEGEL-PAKET
P53	LUFTREINIGUNGS-PAKET
P82	DIEBSTAHLSCHUTZ-PAKET PLUS
U19	AUGMENTED REALITY VIDEO
U23	SITZBELEGUNGSERKENNUNG IM FOND
U25	EINSTIEGSCHIENE BELEUCHTET
01U	MBCONNECT - DIENSTE FUER NAVIGATION
13U	MBCONNECT - EV-FUNKTIONEN (HERMES)
16U	SMARTPHONE INTEGRATION APPLE CARPLAY
17U	SMARTPHONE INTEGRATION ANDROID AUTO
19R	LM-RAD 5-SPEICHEN DESIGN 21” RUNDUM
2S0	PAKET - MULTIMEDIALE / DIGITALE INHALTE
2XXL	BUNDESREPUBLIK DEUTSCHLAND
211A	LEDER - SCHWARZ / ANTHRAZIT
215	ADAPTIVES-DAEMPFUNGS-SYSTEM PLUS (ADS+)
216	HINTERACHSLENKUNG MIT GROSSEM WINKEL
22U	MB-CONNECT MBUX ENTERTAINMENT
231	GARAGENTOROEFFNER
233	ABSTANDSREGELTEMPOMAT PLUS (DISTRONIC PLUS)
241	FAHRERSITZ LINKS ELEKTRISCH VERSTELLBAR MIT MEMORY
242	FAHRERSITZ RECHTS ELEKTRISCH VERSTELLBAR M. MEMORY
243	AKTIVER SPURHALTE-ASSISTENT (KAMERA)
249	INNEN- UND AUSSENSPIEGEL AUTOMATISCH ABBLENDBAR
262	HINTERE SENSOREN FUER SPURHALTEASSISTENT
266	DISTRONIC PLUS QUERUNTERSTUETZUNG (DTR+Q)
272	VORDERE SENSOREN ZUR KREUZUNGSUEBERWACHUNG

Continued on next page

Table A.2 – continued from previous page

Code	Option Description
275	MEMORY-PAKET (FAHRERSITZ, LENKSAEULE U. SPIEGEL)
290	WINDOWBAG
292	PRE-SAFE IMPULS SEITE
294	KNIEBAG
299	PRESAFE
318	SCHEINWERFER LED DIGITAL RECHTSVERKEHR
32U	KLANGPERSONALISIERUNG
321	BIOMETRISCHE BENUTZERIDENTIFIKATION-FINGERABDRUCK
324	LICHTBAND VORNE
325	MITTENAIRBAG
351	ECALL-NOTRUFSYSTEM
36U	MBCONNECT KONNEKTIVITAET U. ERWEITERTES LADEN PLUS
383	KOMMUNIKATIONSMODUL RAMSES ENTRY (4G)
399	MULTIKONTURSITZ VORN MIT MASSAGEFUNKTION
401	SITZKLIMATISIERUNG VORNE
443	LENKRAD HEIZBAR
444	HEAD-UP-DISPLAY ADVANCED (HUD)
475	REIFENDRUCKKONTROLLE (RDK)
489	AIRMATIC DC / LUFTFEDERUNG SEMIAKTIV
500	AUSSENSPIEGEL ELEKTRISCH ANKLAPPBAR
501	360°-KAMERA
503	FERNGESTEUERTES PARKEN
507	FERNGESTEUERTES PARKEN PREMIUM
513	VERKEHRSZEICHENERKENNUNG
546	AUTOMATISCHE GESCHWINDIGKEITSREGELUNG
551	EINBRUCH- UND DIEBSTAHLWARNANLAGE BASIS (EDW)
553	ANHAENGERASSISTENT
581	KLIMATISIERUNGSAUTOMATIC
587	LOGO-PROJEKTION UEBER SPIEGEL
596	IR WIRKSAME VERGLASUNG
597	WINDSCHUTZSCHEIBE HEIZBAR
628	AUTOMATISCHE FERNLICHTSCHALTUNG PLUS (IHC+)
67B	AUTOMATISCHE FAHRERTUER
72B	USB-PAKET
8U4	PTC-BATTERIE-ZUHEIZER FUER HYBRID/EV
810	SOUNDSYSTEM PREMIUM
83B	DC-LADEFUNKTION
84B	AC-LADEFUNKTION EIN-/MEHRPHASIG HIGH (6KW-22KW)
860	BEIFAHRER-DISPLAY
865	TV TUNER DIGITAL
871	SENSORIK F. HECKDECKELOEFFNUNG/SCHLIESSUNG
875	SCHEIBENWASCHANLAGE BEHEIZT
876	INNENRAUM-LICHTPAKET
878	AMBIENTEBELEUCHTUNG CONNECTED LIGHT
890	AUTOMATISCHE RUECKWANDTUER
891	AMBIENTENBELEUCHTUNG PREMIUM
897	DRAHTLOSE TELEFONAUFLADUNG VORNE
902	KOMFORTSITZHEIZUNG VORNE

B Python Virtual Environment

The list below shows all Python dependencies required to run the entire code base of this research project. They come along with their specific versions as defined in the associated `requirements.txt` file. They ensure a reproducible environment set-up for running any part of the associated codebase.

absl-py==1.4.0
aiosignal==1.3.1
asammdf==7.4.2
astunparse==1.6.3
attrs==22.2.0
black==23.1.0
brokenaxes==0.5.0
cachetools==5.3.0
canmatrix==1.0
certifi==2022.12.7
charset-normalizer==3.0.1
click==8.1.3
cloudpickle==2.2.1
coloredlogs==15.0.1
contourpy==1.0.7
cycler==0.11.0
dash==2.8.1
dash-core-components==2.0.0
dash-html-components==2.0.0
dash-table==5.0.0
distlib==0.3.6
dm-tree==0.1.8
eli5==0.13.0
et-xmlfile==1.1.0
Farama-Notifications==0.0.4
filelock==3.9.1
Flask==2.2.3
flatbuffers==2.0.7
fonttools==4.38.0
frozenlist==1.3.3
future==0.18.3
gast==0.4.0
google-auth==2.16.0
google-auth-oauthlib==1.0.0
google-pasta==0.2.0
graphviz==0.20.1
grpcio==1.51.1
gymnasium==0.29.1
h5py==3.8.0
humanfriendly==10.0
idna==3.4
imageio==2.31.1
imbalanced-learn==0.11.0
importlib-metadata==6.0.0
isal==1.1.0
itsdangerous==2.1.2
jax==0.4.10
Jinja2==3.1.2
joblib==1.2.0
jsonschema==4.17.3
kaleido==0.2.1
keras==2.12.0
kiwisolver==1.4.4
lazy-loader==0.2
libclang==15.0.6.1
lime==0.2.0.1
llvmlite==0.39.1
lxml==4.9.3
lz4==4.3.2
Markdown==3.4.1
MarkupSafe==2.1.2
matplotlib==3.6.3
ml-dtypes==0.1.0
mpmath==1.3.0
msgpack==1.0.5
mypy-extensions==1.0.0
networkx==3.1
numba==0.56.4
numexpr==2.8.4
numpy==1.23.5
oauthlib==3.2.2
onnx==1.14.0
onnxconverter-common==1.13.0
onnxruntime==1.15.0
openpyxl==3.0.10
opt-einsum==3.3.0

packaging==23.0
pandas==1.5.3
pathspec==0.11.0
patsy==0.5.3
Pillow==9.4.0
platformdirs==3.0.0
plotly==5.13.1
protobuf==4.23.2
PyALE==1.1.3
pyasn1==0.4.8
pyasn1-modules==0.2.8
pyparsing==3.0.9
pyrsistent==0.19.3
python-dateutil==2.8.2
pytz==2022.7.1
PyWavelets==1.4.1
PyYAML==6.0
ray==2.3.0
requests==2.28.2
requests-oauthlib==1.3.1
rsa==4.9
scikeras==0.10.0
scikit-image==0.21.0
scikit-learn==1.2.1
scipy==1.10.0
seaborn==0.12.2
setuptools==69.5.1
shap==0.41.0
six==1.16.0
skfeature-chappers==1.1.0
skl2onnx==1.14.1
slicer==0.0.7
statsmodels==0.14.0
sympy==1.12
tabulate==0.9.0
tenacity==8.2.1
tensorboard==2.12.3
tensorboard-data-server==0.7.0
tensorboard-plugin-wit==1.8.1
tensorflow==2.12.0
tensorflow-estimator==2.12.0
tensorflow-io-gcs-filesystem==0.30.0
termcolor==2.2.0
tf2onnx==1.16.1
threadpoolctl==3.1.0
tifffile==2023.4.12
tomli==2.0.1
tqdm==4.64.1
typing-extensions==4.4.0
urllib3==1.26.14
virtualenv==20.21.0
Werkzeug==2.2.2
wrapt==1.14.1
zipp==3.11.0
keras_nlp==0.6.1

C Detailed Results of the Pairwise Comparison

Detailed results of the pairwise comparison carried out together with domain as well as ML research experts from the automotive industry. The goal is to identify a relative ranking for several suggested metrics and KPIs. The template has been granted for use by the Mercedes-Benz Design for Six Sigma Blackbelt Program.

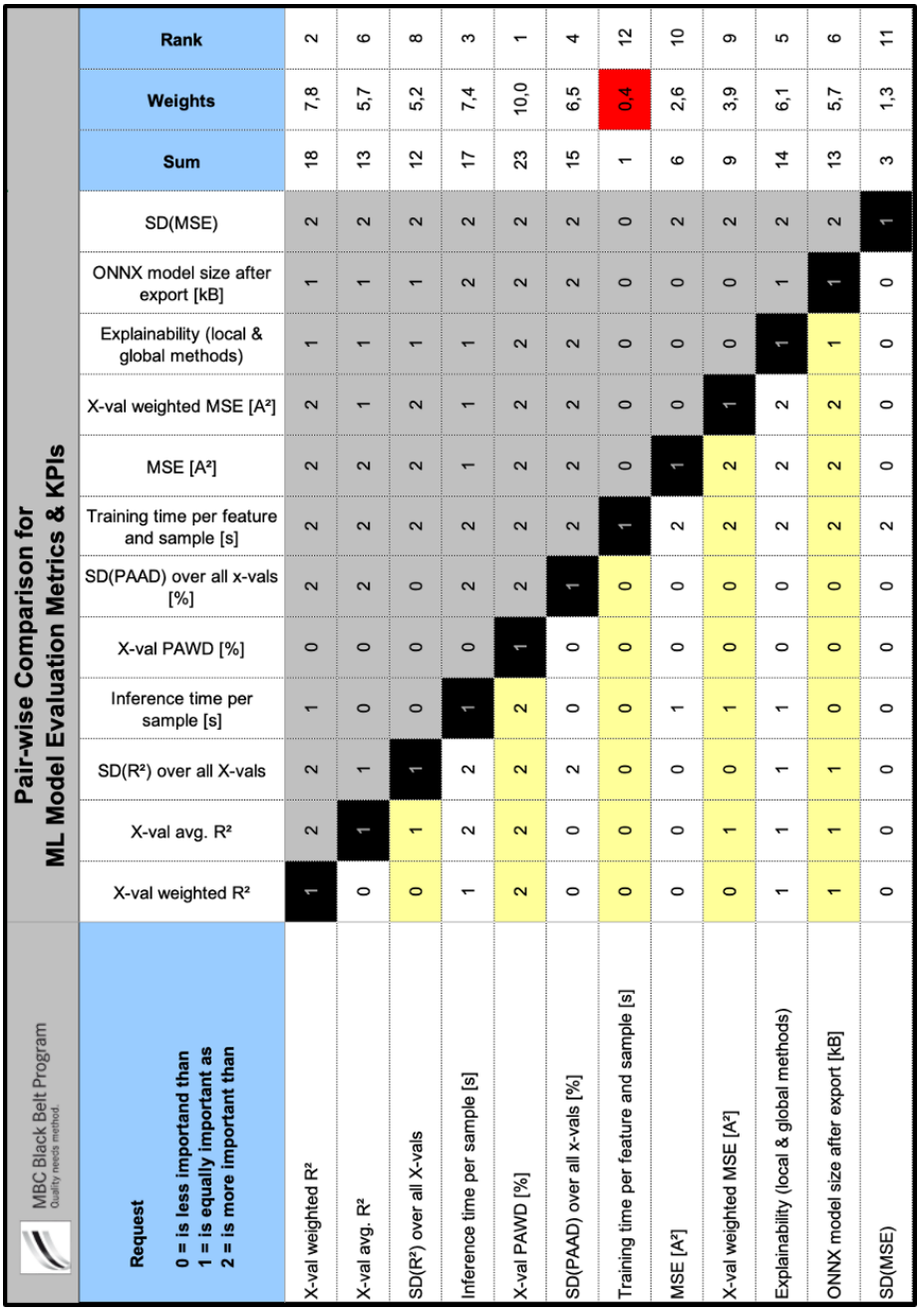

Figure C.1: Pairwise comparison of the 12 suggested ML model assessment metrics. Each KPI is compared to each other criterion.

D Holistic Pipeline Overview

The following figure shows the entire ML pipeline developed during this research. It comprises all steps and "detailed views" described in Chapter 4.
Disclaimer: *Train** in Fig. D.2 refers to the subset of the entire training data which is used for model induction during one specific cross-validation split.

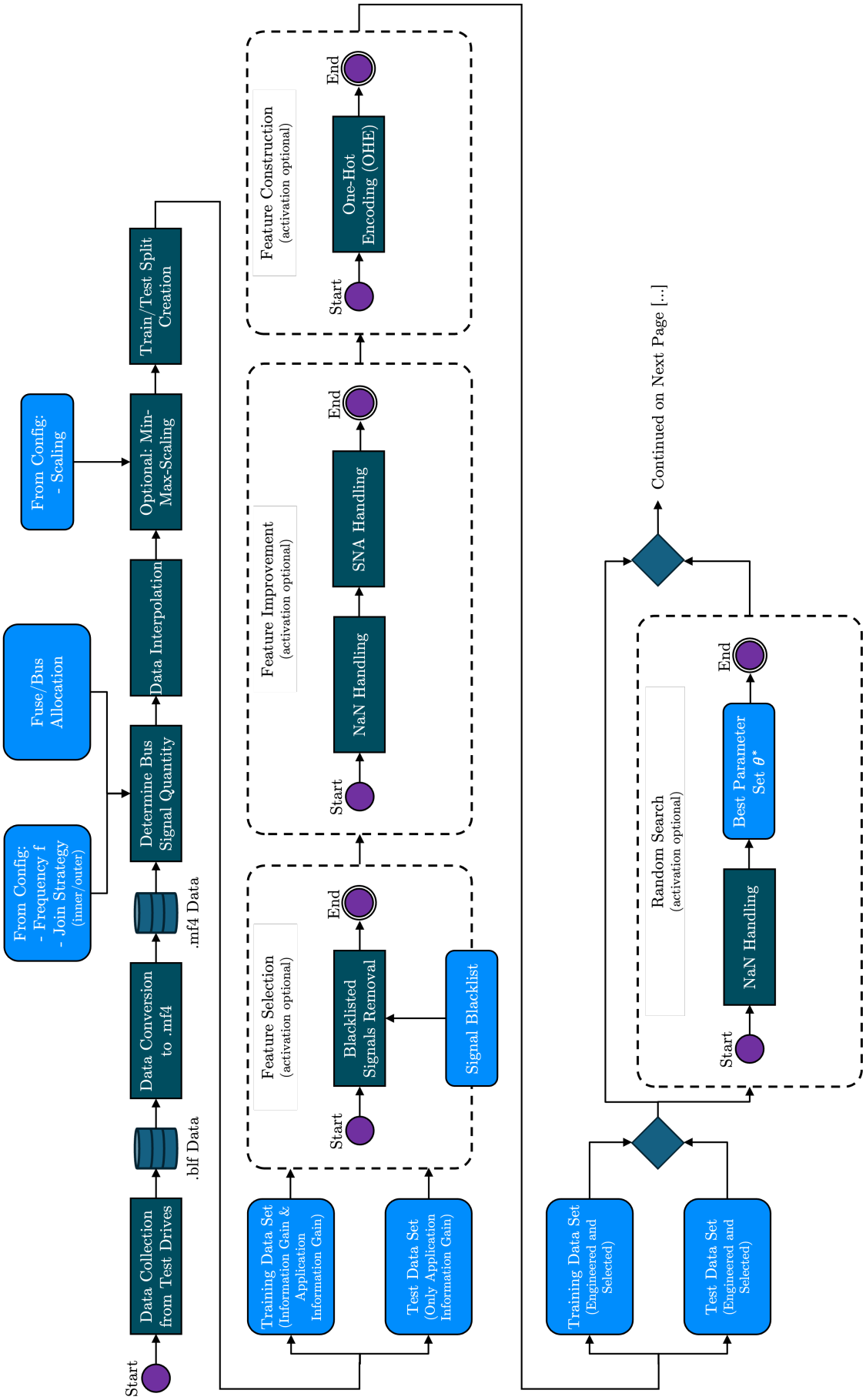

Figure D.1: Detailed steps of the ML pipeline - part 1.

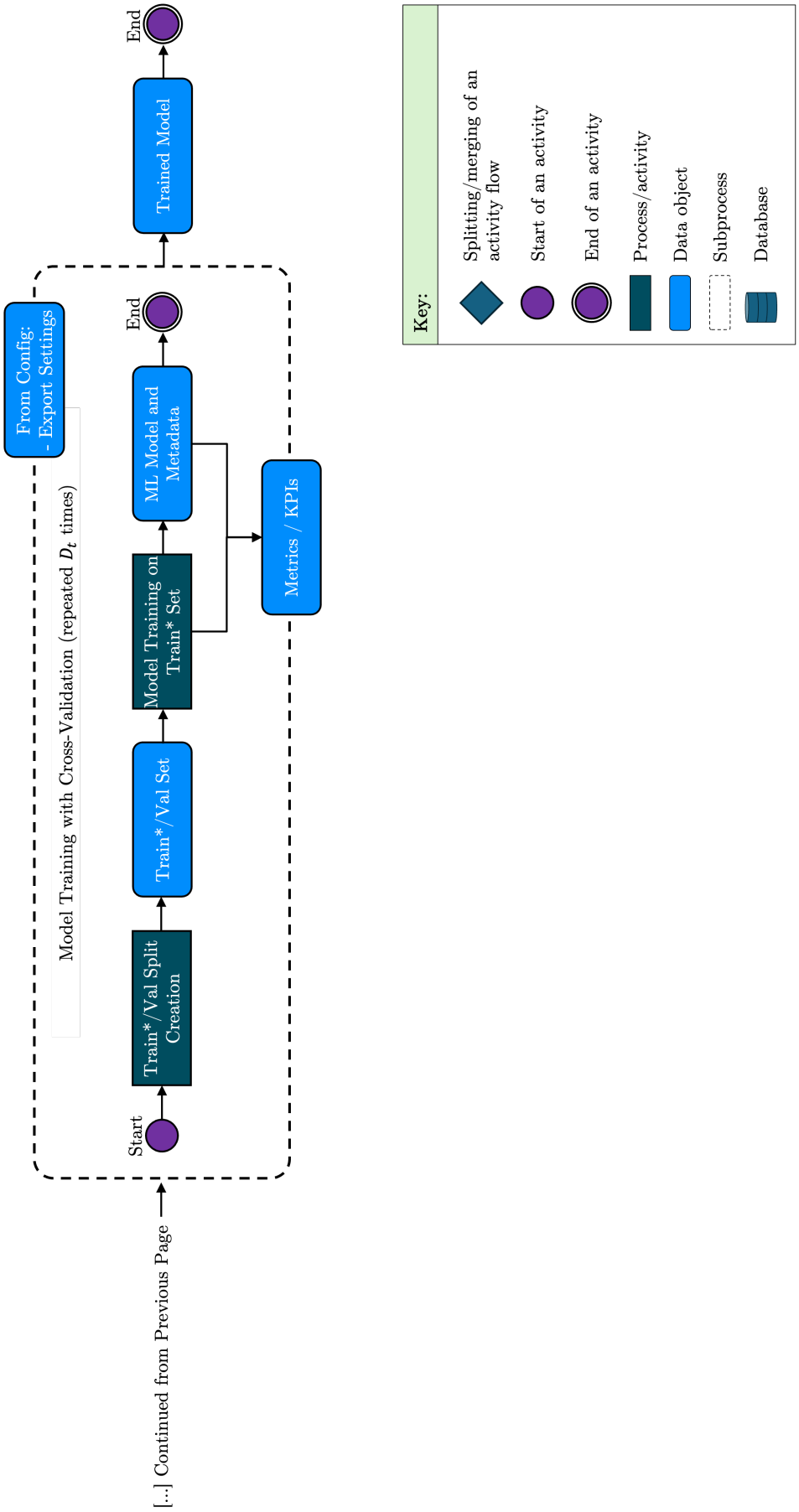

Figure D.2: Detailed steps of the ML pipeline - part 2.

E Relevant Listings for Feature Engineering and Selection

E.1 Zero Variance Elimination

The Python implementation in Listing E.1 shows the implementation of the two-step process for filtering out low-variance features in the (training) dataset: `filter_by_variance_fit(self, df, th=0.0)` identifies numeric columns with variance below a given threshold (`th`, in this case it is set to zero) and compiles a list of these zero-variance columns. It ensures that at least 5 columns are retained, even if low-variance columns have to be added back to meet this minimum. The `df.var(numeric_only=True)` (cf. Listing E.1, lines 10 and 17) implements the variance calculation according to Equation 3.20 ensuring only numeric columns are used.
Then, the class method `filter_by_variance_transform(self, df, drop_col_var=None)` removes the identified columns from the DataFrame while ignoring any errors caused by non-existent ones.

```
1  import pandas as pd
2  from typing import List, Optional
3
4  class FeatureSelection:
5      def filter_by_variance_fit(self, df: pd.DataFrame, th: float =
           0.0) -> List[str]:
6          '''
7          Identify columns with variance below the threshold.
8          '''
9          # Identify numeric columns with variance less than or equal
               to the threshold.
10         drop_col = (
11             df.select_dtypes(include=["number"])
12             .columns[df.var(numeric_only=True) <= th]
13             .tolist()
14         )
15
16         # Calculate variances for numeric columns.
17         variances = df.select_dtypes(include=["number"]).var()
18         # Sort variances in descending order.
19         variances_sorted = variances.sort_values(ascending=False)
20         # Determine how many columns to keep, ensuring at least 5
               columns are retained.
21         num_columns_to_keep = max(len(variances_sorted) - len(
               drop_col), 5)
22         # Select columns to drop, adding extra columns from the
               sorted variances if needed.
23         if num_columns_to_keep < 5:
24             drop_col_var = drop_col + variances_sorted.index[: 5 -
                   num_columns_to_keep].tolist()
25         else:
```

```
            drop_col_var = drop_col

        return drop_col_var

    def filter_by_variance_transform(
        self,
        df: pd.DataFrame,
        drop_col_var: Optional[List[str]] = None
    ) -> pd.DataFrame:
        '''
        Drop columns with low variance from the DataFrame.
        '''
        df = df.drop(columns=drop_col_var, axis=1, errors="ignore")
        return df
```

Listing E.1: Fit and transform methods for (zero) variance elimination as part of the `FeatureSelection.py` class.

E.2 Feature Importance Selection

The class method `drop_unimportant_features_fit(self, df, th, model, fuse, path, X_trn, y_trn)` is designed to calculate feature importances using either a GB or an RF model, depending on the specified input (`model` variable). The feature importance is computed on the training dataset `X_trn` and `y_trn`. As a result, features with an importance below a pre-defined threshold (here: 0.0009999 is used) are marked for removal. They are eventually returned in `drop_cols_fi`—a list of columns (features) to be dropped. The aforementioned threshold has proven to be satisfactory in previous experiments with different values. The relatively low threshold used here does justice to the size and complexity of the input data, as the feature importances potentially have to be split across many features and—when using IFI—the sum of all importances is one.
The `drop_unimportant_features_transform(self, df, drop_cols_fi=None)` function then removes these unimportant features from the dataset, logging the number of features before and after the process. If no features remain, an error is raised to ensure the integrity of the dataset needed for subsequent steps of the ML pipeline.

```
import pandas as pd
from typing import List, Optional

class FeatureSelection:
    def drop_unimportant_features_fit(
        self,
        df: pd.DataFrame,
        th: float,
        model: str,
        fuse,
        path: str,
        X_trn,
        y_trn
    ) -> List[str]:

```

```
        #Initialize feature selection with provided config.
        feature_selection = FeatureSelection(self.config)
        '''
        Determine global feature importance on train data.
        '''
        if model == "gradientboosting":
            '''
            Calculate feature importance using gradient boosting.
            '''
            importances = feature_selection.
               calc_globalFeatureImportance(
                "gradientboosting", fuse, path, X_trn, y_trn
            )
        else:
            '''
            Calculate feature importance using random forest.
            '''
            importances = feature_selection.
               calc_globalFeatureImportance(
                "randomforest", fuse, path, X_trn, y_trn
            )

        '''
        Create boolean mask to select columns to keep.
        '''
        keep_cols_fi = ~df.columns.isin([col for _, col in
           importances if _ < th])
        drop_cols_fi = df.columns[~keep_cols_fi]
        drop_cols_fi = [col for col in drop_cols_fi if col != "file
           "]

        return drop_cols_fi

    def drop_unimportant_features_transform(
        self,
        df: pd.DataFrame,
        drop_cols_fi: Optional[List[str]] = None
    ) -> pd.DataFrame:

        print("Number of features in dataset:", len(df.columns))

        '''
        Select columns to keep and assign back to DataFrame.
        '''
        df = df.drop(columns=drop_cols_fi, axis=1, errors="ignore")
        print("All unimportant features dropped.")
        print("Number of features in dataset:", len(df.columns))

        #Raise error if no important features are found.
        if len(df.columns) == 0:
            raise ValueError("#ERROR#! No important features could
               be found!")

        return df
```

Listing E.2: ML algorithm-dependent selection of feature importance calculation. A precursor to determine which "unimportant" feature columns are to be removed from the entire dataset (`df`) based on a threshold `th`.

E.3 Feature Correlation Filtering

The feature correlation filtering mechanism is implemented as part of the `FeatureSelection.py` class, designed to identify and remove highly correlated features from a given training dataset. The implementation, detailed below, is structured into three principal components: filtering irrelevant columns, identifying correlated features, and preserving the most important features based on predefined feature importance scores. The filtering process begins by excluding the helper columns which are irrelevant for the analysis. For the underlying use case, the columns `file`, `timestamps`, or `I_` (the latter identifying the target columns, as I stands for the electric current) are identified and removed. Next, the pairwise PCCs between the remaining features are computed, resulting in a correlation matrix. Features with a PCC greater than or equal to the threshold `th` are marked as highly correlated. For each pair of highly correlated features, the feature to keep is identified based on its importance score. The feature with the lower overall importance is removed to preserve the feature considered more valuable ("important") for model training. If a feature does not have an explicitly defined importance score, it is assigned a default value of zero, ensuring robust handling of missing information.

- `_filter_target_columns([...])`: A method that removes the helper columns.
- `_identify_highly_correlated_features([...])`: A method calculating a correlation matrix and identifying pairs of features with correlation coefficients above the specified threshold `th`.
- `_drop_features([...])`: This method examines the identified highly correlated feature pairs and removes the less important feature from each pair based on the importance scores.
- `drop_correlated_features_fit([...])`: A method that returns the list of features to be dropped.
- `drop_correlated_features_transform([...])`: A method that applies the identified feature drops to a new dataset.

```
1  import pandas as pd
2  from typing import List, Tuple
3
4  class FeatureSelection:
5
6      def _filter_target_columns(df: pd.DataFrame) -> pd.DataFrame:
7
8          '''
9          Drop helper columns from the DataFrame.
10         '''
11         target_prefixes = ("file", "timestamps", "I_")
12         filtered_columns = [col for col in df.columns if not col.
               startswith(target_prefixes)]
13         return df[filtered_columns]
14
15     def _identify_highly_correlated_features(df: pd.DataFrame, th:
           float) -> List[Tuple[str, str]]:
```

```
16
17          '''
18          Identify pairs of features that are highly correlated based
                on the threshold th.
19          '''
20          corr_matrix = df.corr()
21          correlated_pairs = [
22              (corr_matrix.columns[i], corr_matrix.columns[j])
23              for i in range(len(corr_matrix.columns))
24              for j in range(i + 1, len(corr_matrix.columns))
25              if abs(corr_matrix.iloc[i, j]) >= th
26          ]
27          return correlated_pairs
28
29      def _drop_features(
30          df: pd.DataFrame,
31          correlated_pairs: List[Tuple[str, str]],
32          importances: List[Tuple[float, str]]
33      ) -> List[str]:
34
35          '''
36          Drop the less important feature from each correlated
                feature pair.
37          '''
38          feature_importance_dict = {feature: importance for
                importance, feature in importances}
39          drop_list: List[str] = []
40
41          for feature1, feature2 in correlated_pairs:
42              if feature1 in drop_list or feature2 in drop_list:
43                  continue
44
45              if feature_importance_dict.get(feature1, 0) <
                    feature_importance_dict.get(feature2, 0):
46                  drop_list.append(feature1)
47              else:
48                  drop_list.append(feature2)
49
50          return drop_list
51
52      def drop_correlated_features_fit(
53          cls,
54          df: pd.DataFrame,
55          th: float,
56          importances: List[Tuple[float, str]]
57      ) -> List[str]:
58
59          '''
60          Fit function that identifies and stores the features to
                drop.
61          '''
62          df = cls._filter_target_columns(df)
63          correlated_pairs = cls._identify_highly_correlated_features
                (df, th)
64          return cls._drop_features(df, correlated_pairs, importances
                )
65
66      def drop_correlated_features_transform(
```

```
        df: pd.DataFrame,
        dropped_features: List[str]
    ) -> pd.DataFrame:

        '''
        Drop the identified features from the DataFrame.
        '''
        return df.drop(columns=dropped_features, errors="ignore")
```

Listing E.3: Set of functions to drop correlated features from a Pandas DataFrame with a correlation threshold.

E.4 Median Imputation

Listing E.4 shows the Python implementation of the median imputation fit and transform functions. Here, a generic threshold is implemented that defines the number of unique values that a feature column must have so that the functions are applicable which in this case is 32. The `fill_sna_median_fit([...])` method calculates and stores filler values for each column, either as the column's median (if the column contains sufficiently many unique values and is numeric) or as a placeholder string `"sna"`. The `fill_sna_median_transform([...])` method applies this filler information to another DataFrame (here: the actual training data), replacing invalid or missing values in accordance with the calculated fillers.

```
import pandas as pd
import warnings
from typing import Dict, Union

class SnaHandling:
    def fill_sna_median_fit(self, df_train: pd.DataFrame, th: int)
        -> Dict[str, Union[float, str]]:
        '''
        Initialize dictionary to store median or "sna" for each
            column.
        '''
        sna_dict: Dict[str, Union[float, str]] = {}
        '''
        Iterate through each column in the training DataFrame.
        '''
        for col in df_train.columns:
            '''
            Check if column has enough unique values.
            '''
            if len(df_train[col].value_counts()) >= th:
                '''
                Convert column to numeric, coercing errors to NaN.
                '''
                col_numeric = pd.to_numeric(df_train[col], errors="
                    coerce")
                '''
                Calculate median while ignoring runtime warnings.
                '''
                with warnings.catch_warnings():
```

```
                    warnings.filterwarnings("ignore", category=
                        RuntimeWarning)
                    median = col_numeric.median(skipna=True)
                '''
                Check if median is a valid float and not NaN.
                '''
                if self._is_float(median) and not pd.isnull(median)
                    :
                    sna_dict[col] = median
                else:
                    sna_dict[col] = "sna"
            else:
                sna_dict[col] = "sna"

        return sna_dict

    def fill_sna_median_transform(self, df: pd.DataFrame, sna_dict:
        Dict[str, Union[float, str]]) -> pd.DataFrame:
        '''
        Iterate through each column in the DataFrame.
        '''
        for col in df.columns:
            try:
                '''
                Check if the sna_dict value of that column is a
                    float.
                '''
                if self._is_float(sna_dict[col]):
                    '''
                    Fill missing and invalid values with the median
                        .
                    '''
                    df[col] = [
                        (
                            float(val) if self._is_float(val)
                            else sna_dict[col] if "sna" in str(val)
                                .lower() else 0
                        )
                        for val in df[col]
                    ]
                    df[col].fillna(sna_dict[col], inplace=True)
                else:
                    '''
                    Fill missing and invalid values with "sna".
                    '''
                    df[col] = [
                        sna_dict[col] if "sna" in val.lower() else
                            str(val)
                        for val in df[col].astype(str)
                    ]
                    df[col] = df[col].astype(str).fillna("nan")
            except KeyError:
                continue

        return df
```

Listing E.4: Fit and transform methods for median imputation with variable threshold.

E.5 One-Hot Encoding

The code shown in Listing E.5 provides the corresponding Python implementation of the one-hot encoding part of the feature engineering pipeline. The class `FeatureEncoding.py` class encapsulates the two key methods for fitting and transforming categorical data of the passed dataset. The `one_hot_encoding_fit([...])` method fits a `OneHotEncoder` object to categorical columns identified in that DataFrame, while `one_hot_encoding_transform([...])` ensures compatibility with the encoder and transforms the data into a one-hot encoded format.
To handle variations in input DataFrames, the helper functions `combine_dfs` and `remove_duplicate_cols` manage the merging and cleaning of DataFrames, ensuring consistent structure and avoiding duplicate columns.

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from typing import List, Optional
import numpy as np

class FeatureEncoding:
    def one_hot_encoding_fit(self, df: pd.DataFrame) ->
        OneHotEncoder:
        '''
        Initialize OneHotEncoder with parameters to handle unknown
            categories.
        '''
        ohe_regr = OneHotEncoder(handle_unknown="ignore", dtype=int
            )

        '''
        Select columns with object or category data types.
        '''
        string_columns = list(df.select_dtypes(include=["object", "
            category"]).columns)

        '''
        Fit the OneHotEncoder on the selected columns.
        '''
        ohe_regr.fit(df[string_columns].astype(str))

        return ohe_regr

    def one_hot_encoding_transform(self, df: pd.DataFrame, ohe_regr
        : Optional[OneHotEncoder] = None) -> pd.DataFrame:
        '''
        Identify columns present in encoder but missing in the
            DataFrame.
        '''
        diff = list(
            set(ohe_regr.feature_names_in_)
            - set(list(df.select_dtypes(include=["object", "
                category"]).columns))
        )

        '''
        Add missing columns to dataframe and fill with zeros if any
```

```
            are found.
        '''
        if len(diff) > 0:
            df_test = pd.DataFrame(columns=np.array(diff))
            df_test.fillna(value=0, inplace=True)
            df = pd.concat([df, df_test], axis="columns")

        '''
        Transform the dataframe using the fitted OneHotEncoder.
        '''
        df_out = ohe_regr.transform(df[ohe_regr.feature_names_in_].
            astype(str))
        df_out = df_out.toarray()
        df_out = pd.DataFrame(df_out, columns=ohe_regr.
            get_feature_names_out())

        '''
        Drop original columns that were one-hot encoded.
        '''
        df = df.drop(ohe_regr.feature_names_in_, axis=1)

        '''
        Reset index of both dataframes to ensure alignment.
        '''
        df_out.reset_index(drop=True, inplace=True)
        df.reset_index(inplace=True)

        '''
        Combine the transformed dataframe with the original
            DataFrame.
        '''
        df = combine_dfs([df_out, df])

        return df

##########################
# Helper functions
##########################

def combine_dfs(dfs: List[pd.DataFrame]) -> pd.DataFrame:
    '''
    Reset index for all dataframes and concatenate them.
    '''
    dfs_modified = [df.reset_index(drop=True) for df in dfs]
    combined_df = remove_duplicate_cols(pd.concat(dfs_modified,
        axis=1))
    combined_df.index = dfs[0].index
    return combined_df

def remove_duplicate_cols(df: pd.DataFrame) -> pd.DataFrame:
    '''
    Remove duplicate columns from the DataFrame.
    '''
    return df.loc[:, ~df.columns.duplicated()]
```

Listing E.5: Fit and transform methods for OHE applicable to a Pandas DataFrame - including necessary helper functions.

F Miscellaneous Python Listings

F.1 Implementation of Random Search

The RS implementation is divided into two parts: a scikit-learn wrapper and a Ray Tune wrapper depending on the regression model in use.

F.1.1 Scikit-learn Wrapper for RF, GB and MLP

Detailed Python implementation of the hyperparameter tuning for RF, GB and MLP. The private `_get_best_param_RS([...])` method implements randomized hyperparameter optimization for the aforementioned algorithms using the scikit-learn `RandomizedSearchCV` class. As parameters it accepts a configuration object and training data, then performs randomized search with 5-fold internal cross-validation over 100 iterations. Finally, the best hyperparameters θ_a^* are stored in the `regr.best_params_` variable, and the best score (mean loss) is stored in the `regr.best_score_` variable for further processing in the actual training operation.

```
from models.model import Model
from sklearn.model_selection import RandomizedSearchCV

class RandomSearch:

[...]

def _get_best_param_RS(self, config, X_trn, y_trn) -> object:

    ''' Get model type from configuration object. '''
    model_type = config.model.lower()

    ''' Create regressor object (raw). '''
    model = Model(config, None, X_trn, y_trn)
    regr = model.regressor

    [...]

    if model_type in {"randomforest", "gradientboosting", "mlp"}:
        regr = RandomizedSearchCV(
            regr,
            param_distributions=self._get_search_parameters(
                model_type),
            verbose=5,
            cv=5,
            [...],
            n_iter=100,
            random_state=10,
        )

    [...]

```

```
    ''' Fit regressor with tuned parameters. '''
    regr.fit(X_trn, y_trn)

    ''' Store the best score and parameters. '''
    best_score = regr.best_score_
    best_params = regr.best_params_

    return regr

[...]
```

Listing F.1: Python wrapper class `RandomSearch.py` which implements `RandomizedSearchCV` for hyperparameter tuning (used for both RF and GB as well as MLP).

F.1.2 Ray Tune for LSTM

Detailed implementation of the Ray Tune-based hyperparameter optimization which is used for the LSTM algorithm within the `RandomSearch` class. In the wrapper function `_get_best_param_ray([...])`, the type of the regressor `regr` is stored in the `self.model_type` variable via the `config` object and a regressor object is created. Next, a `tune.Tuner` object is instantiated (among others with a function-based Ray Tune Trainable[32], a configuration and the respective parameter space.. Subsequently the `tuner.fit()` method is called to start the tuning process. The latter runs the tuning trials and returns the respective trial results. The best one is obtained by calling `results.get_best_result([...])` which originates from the trial with the lowest `"mean_loss"`. The best hyperparameters θ_a^* are stored in `regr.best_params_`, and the best score (mean loss) is stored in `regr.best_score_` for further processing in the actual training operation.

```
from models.model import Model
from ray import tune
from ray.tune.schedulers import AsyncHyperBandScheduler

class RandomSearch:

[...]

def _get_best_param_ray(self, config, X_trn, y_trn, num_samples
    =100) -> object:
    '''Define a search space, run_config and return results object
    num_samples = number of samples for RandomSearch.
    '''
    ''' Get model type from configuration object. '''
    model_type = config.model.lower()

    ''' Create regressor object (raw). '''
    model = Model(config, None, X_trn, y_trn)
    regr = model.regressor

    ''' Create a Tuner object for hyperparameter tuning. '''
    tuner = tune.Tuner(
```

[32] `https://docs.ray.io/en/latest/tune/api/trainable.html`, last accessed: Jan 04, 2025

```
22          self._objective,  # RayTune function-based Trainable
23          tune_config=tune.TuneConfig(
24              scheduler=AsyncHyperBandScheduler(metric="mean_loss",
                    mode="min"),  # Scheduler for managing trials
25              search_alg=self._get_search_alg(),  # Search algorithm
                    for hyperparameter tuning, here: RS
26              num_samples=num_samples,  # Number of samples for the
                    search
27          ),
28          param_space=regr.param_space,  # Hyperparameter search
                space
29      )
30
31      ''' Run the tuning process. '''
32      results = tuner.fit()
33
34      ''' Get the best result based on the mean_loss metric. '''
35      best_result = results.get_best_result(metric="mean_loss", mode=
            "min")
36
37      ''' Store the best hyperparameters and score in the regressor
            object. '''
38      regr.best_params_ = best_result.config
39      regr.best_score_ = best_result.metrics["mean_loss"]
40
41      return regr
42
43  [...]
```

Listing F.2: Python wrapper class `RandomSearch.py` implementing hyperparameter tuning with Ray Tune (used for LSTM).

F.2 Cross-Validation Python Implementation

When a CV object is created, the `CrossValidation.py` class accepts a Pandas DataFrame (`dataframe`) which consists of the imported and feature engineered training data. The `__init__` method then establishes several instance variables:

- `self.df` retains the original input DataFrame `df`.
- `self.df_id` and `self.df_ts` designate the column names for the test drive IDs and timestamps.
- `self.delta_time` computes the time interval between consecutive rows, derived from the DataFrame's index.
- `self.cv_splits` is initialized as an empty list intended to eventually store the CV splits.
- `self.ls_testdrives` is populated with a list of unique test drive IDs (cf. above) extracted from the DataFrame via the `get_testdrive_ids(self)` method.

The function `create_splits(self)` actually realizes the creation of the CV splits by iterating over each unique test drive ID (`self.ls_testdrives`). For each test drive ID:

- `df_train_folds` is generated by excluding the current test drive from the DataFrame, thus forming the respective model fit's training dataset ($D_t - 2$ drives).
- `df_val_folds` is created by including only the data which forms the validation set for one fit (one dedicated test drive per CV iteration).
- The timestamps within both `df_train_folds` and `df_val_folds` are recalibrated using `self.delta_time` to ensure that they start with 0 and the that new resulting timestamps are evenly spaced as in the original (non-split) dataset to maintain consistency among shuffled drives.
- Each resultant split includes the test drive ID, the validation set, and training set for a given fit and is eventually appended to `self.cv_splits` as shown in Listing F.3, line 34.

```
1  import pandas as pd
2
3  class CrossValidation:
4      def __init__(self, dataframe):
5          ''' Initialize with a Pandas DataFrame containing
               timestamps, features, and IDs of test drives. '''
6          self.dataframe = dataframe
7          ''' Column name for the ID of test drives (was added during
               data import). '''
8          self.df_id = "file"
9          ''' Column name for the timestamps. '''
10         self.df_ts = "timestamps"
11         ''' Time delta based on the index of the DataFrame. '''
12         self.delta_time = self.dataframe.index[1]
13         ''' List to store different splits with different folds. ''
               '
14         self.cv_splits = []
15         ''' List of unique IDs of test drives in the DataFrame. '''
16         self.ls_testdrives = self.get_testdrive_ids()
17
18     def get_testdrive_ids(self):
19         ''' Return a list of unique test drive IDs from the
               DataFrame. '''
20         return pd.unique(self.dataframe[self.df_id]).tolist()
21
22     def create_splits(self):
23         ''' Create cross-validation splits by iterating over each
               unique test drive ID. '''
24         for testdrive_id in self.ls_testdrives:
25             ''' Get training data excluding the current test drive.
                   '''
26             df_train_folds = self.df_train[self.df_train[self.df_id
                   ] != testdrive_id].copy()
27             ''' Get validation data for the current test drive. '''
```

```
28              df_val_folds = self.dataframe[self.dataframe[self.df_id
                    ] == testdrive_id].copy()
29              ''' Reset timestamps for the training set. '''
30              df_train_folds[self.df_ts] = [x * self.delta_time for x
                     in range(len(df_train_folds))]
31              ''' Reset timestamps for the validation set. '''
32              df_val_folds[self.df_ts] = [x * self.delta_time for x
                    in range(len(df_val_folds))]
33              ''' Append the split to the list. '''
34              self.cv_splits.append([testdrive_id, df_val_folds,
                    df_train_folds])
35          return self.cv_splits
36
37      [...]
```

Listing F.3: `CrossValidation.py` class for systematic train and test split creation within the LOOCV strategy (excerpt).

F.3 Model Export Python Implementation

The function `export_model(config, main_collector, model_collector)` exports the trained regressor in the ONNX format, while the function `export_model_information([...])` exports supplementary metadata regarding the model and data. This metadata includes information about the carline, E/E architecture, affected bus systems, ECU, and other relevant details, all stored in a JSON file. An example JSON for the driver display of Vehicle A can be found in Appendix G. These outputs ensure both human- and machine-readable accessibility. Moreover, the design ensures that key meta-information about the script's execution (`main_collector`) and the model-specific details (`model_collector`) which are compiled in previous steps of the training process (not mentioned for the sake of brevity) are seamlessly integrated into the export process.

```
1  from doc_utils import ModelExport
2  from data.data import logger
3
4  import config
5  import time
6
7  class Model:
8
9  [...]
10
11      if config.model_export:
12          exporter = ModelExport(export_dir=doku.get_fuse_path(fuse))
13          logger.info(f"Exporting model to {doku.get_fuse_path(fuse)}
                ")
14
15          print("--- Starting Model Export ---")
16          start_time = time.time()
17          exporter.export_model(
18              config,
19              main_collector,
20              model_collector
```

```
21          )
22          exporter.export_model_information(main_collector,
                model_collector)
23          model_export_time = time.time() - start_time
24          logger.info("--- %s seconds to export model---" % (
                model_export_time))
25          print("--- Regressor Export Finished ---")
26      else:
27          pass
```

Listing F.4: Excerpt of the model export functionality as part of a central `Model.py` class.

F.4 The Model Class - An Example of a Modular Paradigm

The `Model.py` class is an example of the modular structure pursued for the implementation of the overall ML pipeline designed in this research. Listing F.5 below illustrates this modular, object-oriented principle in greater detail for the regressor creation and for the explainer objects. In the example of XAI, a hierarchical structure that is realized by the dedicated classes located "below" `Explanation.py` in which the actual XAI logic is implemented and returned. Depending on whether XAI is activated in a training run or not, the respective object is created or the part of the code is skipped using a boolean flag (cf. Listing F.5, line 48).

```
1  from sklearn.ensemble import RandomForestRegressor,
       GradientBoostingRegressor
2  from sklearn.neural_network import MLPRegressor
3  from models.lstm_model import LSTMModel
4  from explanation.explanation import Explanation
5
6  ''' More imports omitted for brevity. '''
7  [...]
8
9  class Model:
10     def __init__(self, config, best_params, X_trn, y_trn):
11         self.model_type = config.model.lower()
12         ''' Retrieve configuration parameters. '''
13         self.config_dict = config.get_dict(
14             [...]
15         )
16
17         ''' Store the best parameters from the RS tuning. '''
18         self.best_params = best_params
19
20         ''' More class variable declarations omitted for brevity. '
               ''
21         [...]
22
23         ''' Model initialization. '''
24         if self.model_type == "randomforest":
25             self.regr = RandomForestRegressor(**self.config_dict)
26         elif self.model_type == "gradientboosting":
```

```
27              self.regr = GradientBoostingRegressor(**self.
                    config_dict)
28          elif self.model_type == "mlp":
29              self.regr = MLPRegressor(**self.config_dict)
30          elif self.model_type == "lstm":
31              self.regr = LSTMModel(X_trn.shape[1], self.config_dict)
32          pass
33
34      def generate_model(config, best_params, X_trn, y_trn, [...]):
35
36          # ******************************************************
37          # Model Creation
38          # ******************************************************
39          model = Model(config, best_parameters, X_trn, y_trn)
40          regr = model.regr
41
42          ''' Model training, evaluation, export, etc. omitted for
                brevity. '''
43          [...]
44
45          # ********************************
46          # XAI Methods
47          # ********************************
48          if config.exp_enabled:
49              explainer = Explanation(
50
51              ''' Initialize the Explanation class. '''
52              [...]
53              )
54
55              '''Feature Importance explanation. '''
56              feature_importances_trn = explainer.do_pfi_explanation(
57                  X_trn, y_trn, "training"
58              )
59
60              '''Explaining feature importances with PFI. '''
61              explainer.explain_signal_names_pfi()
62
63              '''ALE explanation. '''
64              explainer.do_ale_explanation(X_trn)
65              explainer.explain_signal_names_ale()
66
67              ''' SHAP explanation. '''
68              explainer.do_shap_explanation(X_trn)
69
70              return regr
```

Listing F.5: Shortened `Model.py` class focusing on the modularity of the XAI functionality and the `Model` object generation.

F.5 The Explanation Class Constructor

Listing F.6 presents the definition of the `Explanation.py` class which serves as a wrapper for encapsulating XAI functions (cf. above). As input, it takes training and validation datasets (`X_trn`, `X_val`, `y_trn`, `y_val`), a regression model (`regr`), and ECU (target) related parameters (`fuse`, `numFuses`). Additionally, it retains feature names via the `feature_columns` list to support interpretability. This structure facilitates seamless integration of explainability techniques within the overall ML workflow that might need this input data. As a result, all variables mentioned above are class variables.

```
class Explanation:
    def __init__(
        self,
        X_trn,
        X_val,
        y_trn,
        y_val,
        regr,
        fuse,
        feature_columns,
        numFuses,
    ):
        [...]
```

Listing F.6: Definition of an `Explanation` object. It is needed to call encapsulated XAI functions.

G Metadata for a Training Run and ECU Model

The example JSON file shown in Listing G.1 below provides detailed information on the first CV iteration of the driver display ECU using the GB modeling algorithm. From the provided JSON, the model can be identified as "00_DriverDisplay_20250105_153515_gradientboosting", and that it is trained on data from "Vehicle A", a "sedan" car line, and that it utilizes a single CAN bus system. Key features include illumination level, display sensor values, battery voltage and current, and sun elevation angle, with the target being the driver display's energy consumption (current).
This model's performance can be derived from the R^2 validation score (in this iteration: $R^2 = 0.91$). Further energy related KPIs are also stored but they are omitted for brevity. Additionally, the JSON file provides detailed metadata on the environmental and operational conditions during the training phase, such as outside and inside temperatures measured by the vehicle's onboard temperature sensors, brightness levels, vehicle speed, and supply battery voltage. The JSON also captures the day/night ratio, city driving ratio, and motorway driving ratio, all measured in percent, offering a comprehensive view of the conditions under which the model is trained.

```
{
  "model-ID": "00_F223_20250105_153515_gradientboosting",
  "carline": [
    "Sedan"
  ],
  "architecture": [
    "<non-disclosed>"
  ],
  "bussystems": [
    "1xCAN"
  ],
  "fuses": "F223",
  "component": [
    "Driver_Display"
  ],
  "test_vehicle": [
    "Vehicle A"
  ],
  "raster": 0.2, //Import frequency
  "features": [
    "Illumination Level",
    "Display Sensor Value All",
    "Head-Up Display Sensor Value",
    "Supply Battery Voltage",
    "Supply Battery Current",
    "Sun Elevation Angle"
  ],
  "targets": [
```

```
    "Driver Display Current"
  ],
  "model-type": "gradientboosting",
  "model-parameters": "{'n_estimators': 100, 'learning_rate': 0.1,
     'max_depth': 3, 'min_samples_split': 2, 'min_samples_leaf': 1,
      'max_features': None, 'subsample': 1.0, 'loss': '
     squared_error', [...],",
  "R^2 validation": [
    0.91
  ],

  //[...]

  //Further model and energy consumption metrics

  "training time [s]": 19.26,
  "traindata length [s]": 47502.0,
  "validationdata length [s]": 1034.0,
  "outside temperature (train)": {
    "max": 22.0,
    "avg": 9.56,
    "min": -0.45,
    "25%": 8.0,
    "50%": 10.0,
    "75%": 11.5
  },
  "inside temperature (train)": {
    "max": 27.40,
    "avg": 23.19,
    "min": 8.0,
    "25%": 22.70,
    "50%": 23.40,
    "75%": 24.60
  },
  "brightness (train)": {
    "max": 250.0,
    "avg": 6.91,
    "min": 1.0,
    "25%": 1.0,
    "50%": 1.0,
    "75%": 5.0
  },
  "vehicle speed (train)": {
    "max": 226.90,
    "avg": 74.31,
    "min": 0.0,
    "25%": 38.90,
    "50%": 69.90,
    "75%": 111.06
  },
  "LV-voltage (train)": {
    "max": 14.73,
    "avg": 13.83,
    "min": 10.69,
    "25%": 13.02,
    "50%": 14.32,
    "75%": 14.57
  },
```

```
84    "day night relation (train)": {
85      "day": 0.46,
86      "night": 0.54
87    },
88    "city relation (train)": {
89      "no city": 0.44,
90      "city": 0.56
91    },
92    "motorway (train)": {
93      "unknown": 0.38,
94      "no motorway": 0.24,
95      "motorway": 0.38
96    }
97  }
```

Listing G.1: JSON representation of evaluation and analysis metadata from a training run (one single cross-validation) of the ML pipeline.

H Implementation of the PFI Class

The `PFI.py` class, shown in Listing H.1, is the Python implementation for computing and visualizing Permutation Feature Importance (PFI) for an ML model. PFI was introduced in Section 3.6 and applied in Section 4.2.2.

Initialization: The class is initialized with an `explanation` object that provides the regression model (`regr`), feature names (`features`), and a unique model identifier (`model_id`). Additionally, the class ensures a directory is created at a specified path to store PFI-related outputs.

Key Functions:

- `calculate_importances([...])`: This function computes the feature importances using the given PFI algorithm for RF, GB, MLP, and LSTM models.
- `print_importances([...])`: This function writes the computed feature importances to a text file, documenting the mean importance and standard deviation for each feature.
- `plot_importances([...])`: This function generates a bar plot of the top features ranked by their importances for an efficient visual reference of the PFI results. The plot is saved as a PNG image in the designated directory.
- `get_feature_importances([...])`: This helper function returns a sorted list of feature importances (PFI) together with their feature names, enabling further analysis if needed.

```
1  import os
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import seaborn as sns
5  import sklearn
6  from sklearn.inspection import permutation_importance
7  from models.lstm_model import LSTMModel
8
9  class PFI:
10     def __init__(self, explanation) -> None:
11         ''' Initialize paths, model type and feature data. '''
12         self.pfi_path = os.path.join(explanation.
               get_explanation_path(), "PFI")
13         if not os.path.exists(self.pfi_path):
14             os.mkdir(self.pfi_path)
15         self.regr = explanation.regr
16         self.features = explanation.features
17         self.model_id = explanation.model_id
18         self.feature_importances = None
19         self.pfi_features = None
20
21     def calculate_importances(self, X, y) -> None:
```

```
        ''' Compute permutation importances if the model supports it. '''
        if (
            type(self.regr) == sklearn.ensemble._RandomForestRegressor
            or sklearn.ensemble.GradientBoostingRegressor
            or sklearn.neural_network.MLPRegressor
            or LSTMModel
        ):
            self.feature_importances = permutation_importance(
                self.regr, X, y, random_state=0
            )

    def print_importances(self, X, y) -> None:
        ''' Save feature importances to a text file. '''
        if self.feature_importances is None:
            self.calculate_importances(X, y)
        with open(
            os.path.join(
                self.pfi_path, f"{self.model_id}-PermutationFeatureImportance.txt"
            ),
            "w",
        ) as f:
            for i in self.feature_importances.importances_mean.argsort()[::-1]:
                f.write(
                    f"{self.feature_importances.importances_mean[i]:.3f}"
                    f" +/- {self.feature_importances.importances_std[i]:.3f}"
                    f" {self.features[i]:20}\n"
                )

    def plot_importances(self, max_display=6) -> None:
        ''' Plot and save a bar plot of feature importances. '''
        if self.feature_importances is None:
            raise ValueError("Feature importances not computed; run calculate_importances first.")
        pfi_data = {
            "features": self.features,
            "feature importances": self.feature_importances.importances_mean,
            "feature importances std": self.feature_importances.importances_std,
        }
        pfi_df = pd.DataFrame(pfi_data)
        pfi_df.sort_values(by=["feature importances"], ascending=False, inplace=True)
        pfi_df = pfi_df[0:max_display][:]
        self.pfi_features = pfi_df["features"].tolist()

        plt.figure(figsize=(12, 8))
        plt.title(
            f"{self.model_id} - Permutation Feature Importance",
            fontsize=12,
        )
        img = sns.barplot(
```

```
70              x=pfi_df["feature importances"],
71              y=pfi_df["features"],
72              palette="ch:s=-.25,rot=.25",
73          )
74          img.set_yticklabels(img.get_yticklabels(), fontsize=12)
75          img.set_xticklabels(img.get_xticklabels(), fontsize=12)
76          plt.tight_layout()
77          plt.savefig(
78              os.path.join(
79                  self.pfi_path,
80                  f"{self.model_id}-PermutationFeatureImportances.png
                        ",
81              ),
82              dpi=300,
83          )
84          plt.clf()
85
86      def get_feature_importances(self) -> list:
87          ''' Return a sorted list of feature importances and names.
                ''' 
88          return sorted(
89              zip(
90                  [float(x) for x in self.feature_importances.
                        importances_mean],
91                  self.features,
92              ),
93              reverse=True,
94          )
```

Listing H.1: `PFI.py` class to encapsulate the generation of permutation feature importances for a given ML modeling algorithm and dataset.

I Implementation of the ALE Class

The `ALE.py` class, defined in Listing I.1, provides the functionality to generate accumulated local effects (ALE) plots for analyzing ML models feature-based as described in Sections 3.6 and 4.2.2.

Initialization: The constructor of the class accepts training and validation datasets (`X_trn`, `X_val`), a trained model (`regr`), and the target names (here: `fuses`). It also automatically creates a directory to store ALE plots under the path provided.

Key Functions:

- `create_1D_ale([...])`: This function generates 1-dimensional ALE plots for each feature in the dataset and it skips features with near-zero variance to avoid the creation of too many plots that provide no information. The plots are saved as PNG images only if the ALE effect exceeds a defined threshold, reducing the number of plots and focusing on features with a minimum impact.
- `feature_variance_is_zero([...])`: This is a helper function that checks if the variance of a given feature is below a specified threshold (e.g. 0.001). If this is the case the feature is discarded for the ALE analysis (cf. above).

```
import os
import numpy as np
import matplotlib.pyplot as plt

class ALE:
    def __init__(self, explanation, X_trn, regr, fuses) -> None:
        ''' Initialize the ALE class with necessary datasets, model
            , and explanation parameters. '''
        self.X_trn_df = (X_trn)
        self.regr = regr
        self.fuses = fuses
        self.model_id = explanation.model_id

        self.ale_path = os.path.join(explanation.
            get_explanation_path(), "ALE")

        if not os.path.exists(self.ale_path):
            os.mkdir(self.ale_path)

        self.common_params = {"grid_size": 100, "include_CI": False
            , "C": 0.95}

    def create_1D_ale(self, X, path) -> None:
        ''' Create and save 1D ALE plots for each feature. '''
        _1D_exp_path = os.path.join(self.ale_path, path)
        if not os.path.exists(_1D_exp_path):
            os.mkdir(_1D_exp_path)

        plt.close()
```

```
27          num_features = len(self.preprocessor.getFeatureColumns())
28
29          for fuse_id in range(len(self.fuses)):
30              for feature_number in range(num_features):
31                  # Skip feature if its variance is nearly zero.
32                  if self.feature_variance_is_zero(feature_number, X)
                        :
33                      continue
34                  feature_name = str(self.preprocessor.
                        getFeatureColumns()[feature_number])
35                  ale_eff = ale(
36                      X=X,
37                      model=self.regr,
38                      feature=[feature_name],
39                      **self.common_params,
40                  )
41                  if (abs(ale_eff["eff"].max()) + abs(ale_eff["eff"].
                        min())) > 0.1:  # Plot only if ALE effect
                        exceeds 0.1.
42                      plt.gcf()
43                      plt.tight_layout()
44                      plt.savefig(
45                          os.path.join(
46                              _1D_exp_path,
47                              f"{self.model_id}-{feature_name}-1D-
                                    ALE_on_{path}.png",
48                          ),
49                          dpi=300,
50                      )
51                      plt.close()
52
53      def feature_variance_is_zero(self, feature_number, df) -> bool:
54          ''' Return True if the variance of the feature is near zero
                . '''
55          feature_name = self.preprocessor.getFeatureColumns()[
                feature_number]
56          array = df[str(feature_name)].values
57          if np.var(array) <= 0.001:
58              return True
59          else:
60              return False
```

Listing I.1: `ALE.py` class to encapsulate the generation of 1-dimensional accumulated local effects (ALE) plots for a given ML modeling algorithm and dataset.

J Implementation of the SHAP Class

The `SHAP.py` class, shown in Listing J.1, is the corresponding Python implementation for computing and visualizing SHAP values in this project as discussed in Section 3.6 and applied in Section 4.2.2.

Initialization: The class is initialized with the following parameters:

- `explanation`: An object providing specific information such as `model_id` from the `Explanation.py` class.
- `regressor`: The ML regression model for which SHAP values shall be calculated (e.g., `RandomForestRegressor`, `MLPRegressor`).
- `feature_columns`: A list of feature names used by the model.
- `X`: A matrix of input features used for SHAP calculations.
- `fuse`: Specifies the target variable name.
- `numFuses`: The number of target variables.

The class also initializes a path (`shap_path`) to store SHAP-related outputs. If the specified directory does not exist, it is created upon execution of the class constructor.

Key Functions:

- `create_shap_explainer([...])`: This method creates a SHAP explainer object depending on the type of regression model and returns an appropriate SHAP explainer.
- `generate_shap_values([...])`: This function calculates SHAP values for a set of indices or for all data points if no indices are provided. It uses the SHAP explainer created by the `create_shap_explainer([...])` function.
- `shap_waterfall_plot([...])`: This method generates and saves waterfall plots for the SHAP values. The plots are saved as PNG images in the SHAP directory corresponding to the (cross-validation) run of the ML pipeline. The number of features displayed can be controlled and customized using the `max_display` parameter.

```
1 import os
2 import numpy as np
3 import matplotlib.pyplot
4 import shap
5 import sklearn
6 import time
7 from models.lstm_model import LSTMModel
8
```

```
class SHAP:
    def __init__(
        self, explanation, regressor, feature_columns, X, fuse,
            numFuses
    ):
        ''' Initialize the SHAP class with the regression model and
            necessary data. '''
        self.regr = regressor
        self.feature_columns = feature_columns
        self.X_create = X
        self.fuse = fuse
        self.numFuses = numFuses
        self.model_id = explanation.model_id

        ''' Create the SHAP explainer object based on the model
            type. '''
        self.shap_explainer = self.create_shap_explainer()

        self.shap_values = None
        self.shap_features = None
        self.imp_features = []

        ''' Set the path where SHAP outputs will be stored and
            create the directory if it does not exist. '''
        self.shap_path = os.path.join(explanation.
            get_explanation_path(), "SHAP")
        if not os.path.exists(self.shap_path):
            os.mkdir(self.shap_path)
        self.dir = self.shap_path

    def create_shap_explainer(self):
        ''' Create an appropriate SHAP explainer based on the model
            type. '''
        if (
            type(self.regr) == sklearn.ensemble.
                RandomForestRegressor
            or sklearn.ensemble.GradientBoostingRegressor
        ):
            ''' For RandomForestRegressor, use SHAP Tree explainer.
                '''
            return shap.explainers.Tree(self.regr, feature_names=
                self.feature_columns)

        elif type(self.regr) == sklearn.neural_network.MLPRegressor
            :
            ''' For MLPRegressor, use SHAP Kernel explainer. '''
            return shap.KernelExplainer(
                self.regr, self.X_create, feature_names=self.
                    feature_columns
            )
        elif type(self.regr) == LSTMModel:
            ''' For LSTMModel, use SHAP Deep explainer. '''
            return shap.DeepExplainer(
                self.regr.model, self.X_create, feature_names=self.
                    feature_columns
            )
        else:
            pass  # No explainer available for other model types
```

```
55
56      def generate_shap_values(self, indices, X_exp):
57          ''' Generate SHAP values for given indices or all samples
                in X_exp. '''
58          if indices != None:
59              ''' If specific indices are provided, extract the
                    corresponding data points. '''
60              self.X_exp = np.take(X_exp, indices, 0)
61              self.indices = indices
62          else:
63              ''' Otherwise, use all the data points. '''
64              self.X_exp = X_exp
65              self.indices = list(range(0, len(X_exp), 1))
66
67          ''' Calculate SHAP values using the explainer. '''
68          if self.shap_explainer != None:
69              start_time = time.time()
70              self.shap_values = self.shap_explainer(self.X_exp)
71          else:
72              pass  # No SHAP explainer available, skipping value
                    generation
73
74      def shap_waterfall_plot(self, indices=None, X_exp=np.array([
            None])):
75          ''' Generate and save SHAP waterfall plots for the given
                indices or all data points. '''
76          self.imp_features = []
77
78          ''' Generate SHAP values if they are not already computed.
                '''
79          if X_exp.all() != None:
80              self.generate_shap_values(indices, X_exp)
81
82          max_display = 7  # Maximum number of features to display on
                the waterfall plot
83
84          ''' Create summary plot if the model has only one target
                fuse. '''
85          if self.numFuses == 1:
86              for num in range(len(self.indices)):
87                  ''' Clear the current plot '''
88                  matplotlib.pyplot.clf()
89
90                  ''' Create SHAP explanation object for the current
                        data point. '''
91                  exp = shap.Explanation(
92                      self.shap_values.values,
93                      self.shap_values.base_values[0][0],
94                      self.shap_values.data,
95                      feature_names=self.feature_columns,
96                  )
97
98                  ''' Extract feature names and SHAP values for the
                        current data point. '''
99                  self.shap_features = exp[num].feature_names
100                 self.shap_values_ = exp[num].values
101
102                 ''' Sort the features by the absolute value of
```

```
                their SHAP values. '''
103             order = np.argsort(-np.abs(self.shap_values_))
104             if len(order) < max_display:
105                 max_display = len(order)
106
107             ''' Ensure that features are in list form for easy
                iteration. '''
108             if not isinstance(self.shap_features, list):
109                 self.shap_features = [self.shap_features]
110
111             ''' Store the top contributing features for the
                current data point. '''
112             if len(self.shap_features) == 1:
113                 self.imp_features = self.shap_features
114             else:
115                 for i in range(max_display):
116                     self.imp_features.append(self.shap_features
                        [order[i]])
117
118             ''' Generate and save the waterfall plot for the
                current data point. '''
119             shap.plots.waterfall(exp[num], show=False,
                max_display=7)
120             matplotlib.pyplot.savefig(
121                 os.path.join(
122                     self.shap_path,
123                     f"{self.model_id}-SHAP-Waterfall-Timestamp
                        {self.indices[num]}.png",
124                 ),
125                 bbox_inches="tight",
126                 dpi=300,
127             )
128             matplotlib.pyplot.clf()
129     else:
130         pass
131
132     [...]
```

Listing J.1: Implementation of the `SHAP.py` class to encapsulate the generation of SHAP values and plots for the investigated ML modeling algorithms.

K Detailed Feature Engineering and Hyperparameter Tuning Results

This section of the appendix shows the detailed results of the training runs conducted with the 18 selected ECUs (from the set $\mathcal{E}$). First, the feature engineered results are depicted for the four ML modeling algorithms and subsequently those with additionally tuned hyperparameters (random search) are presented. As a baseline, the respective previous results are taken into account. This means, for feature engineering and selection the baseline is no data preparation at all (suffix "_raw"). Hence, for the hyperparameter tuning (suffix "_ht") the baseline is the results from the preceding feature engineering and selection step (denotation with the suffix "_en").
Remark: The results from the hyperparameter tuning are used for the utility value calculation (after the application of the respective capping strategies) in the scope of the weighted sum analysis (WSA) (cf. Section 4.3.3).

Table K.1: Detailed feature engineering results for RF. ECUs marked with "_raw" comprise the raw, whereas "_en" denotes the use of the modified dataset.

	ECU / Change	X-val weighted PAWD [%]	X-val weighted R^2	SD(PAAD)	X-val avg. R^2	SD(R^2)	X-val weighted MSE [A_2]	MSE [A_2]	SD(MSE) [A_2]
Vehicle A	Extractor Fan_raw	10.486	0.692	12.710	0.714	0.173	1.104	0.867	2.756
	Extractor Fan_en	10.019	0.710	11.955	0.731	0.162	1.019	0.780	2.537
	Change [%]	-4.45	2.39	-5.94	2.40	-6.54	-7.67	-7.75	-7.93
	Coolant Pump_raw	0.523	0.997	0.523	0.997	0.001	0.007	0.007	0.004
	Coolant Pump_en	0.356	0.998	1.187	0.998	0.000	0.004	0.004	0.001
	Change [%]	-31.98	0.12	126.91	0.17	-68.03	-48.57	-51.09	-65.11
	Right Pixel Headlamp_raw	4.013	0.783	3.801	0.805	0.151	0.139	0.107	0.085
	Right Pixel Headlamp_en	3.715	0.750	6.421	0.768	0.236	0.135	0.114	0.100
	Change [%]	-7.42	-4.16	68.93	-4.55	56.09	-2.47	5.97	18.01
	Left Pixel Headlamp_raw	3.390	0.769	3.734	0.789	0.192	0.103	0.089	0.068
	Left Pixel Headlamp_en	2.260	0.793	2.594	0.810	0.182	0.083	0.078	0.060
	Change [%]	-33.32	3.17	-30.51	2.00	-5.05	-19.47	-12.43	-12.00
	BCF_raw	15.529	-0.250	23.001	-0.113	0.712	1.361	0.985	1.535
	BCF_en	9.489	0.135	16.937	0.191	0.439	1.004	0.713	1.122
	Change [%]	-38.90	153.90	-26.37	268.45	-38.37	-26.20	-27.62	-26.94
	Adaptive Suspension_raw	4.316	0.666	6.142	0.655	0.127	0.114	0.136	0.136
	Adaptive Suspension_en	2.187	0.802	4.621	0.783	0.112	0.071	0.092	0.125
	Change [%]	-49.32	20.50	-24.78	19.44	-11.76	-37.81	-32.16	-8.07
	Driver Display_raw	9.067	-0.510	13.399	-1.225	2.579	0.006	0.005	0.005
	Driver Display_en	3.080	0.431	4.530	0.258	0.518	0.003	0.002	0.002
	Change [%]	-66.03	184.45	-66.19	121.06	-79.91	-53.13	-55.53	-56.61
	CID_raw	2.709	-0.624	4.662	-0.947	2.767	0.005	0.006	0.007
	CID_en	1.662	0.334	2.632	0.232	0.621	0.003	0.005	0.005
	Change [%]	-38.65	153.57	-43.54	124.51	-77.56	-27.21	-30.65	-25.77
	Fuel Supply ECU_raw	1.184	0.954	1.938	0.950	0.109	0.006	0.007	0.006
	Fuel Supply ECU_en	0.441	0.988	0.765	0.987	0.028	0.002	0.002	0.001
	Change [%]	-62.78	3.62	-60.52	3.84	-74.43	-73.33	-72.08	-81.44
Vehicle B	Seat ECU Driver_raw	22.663	-8.728	173.695	-362.408	1726.075	1.907	1.458	1.610
	Seat ECU Driver_en	13.415	-5.158	108.221	-63.115	205.587	1.577	1.201	1.380
	Change [%]	-40.81	40.90	-37.69	82.58	-88.09	-17.33	-17.67	-14.32
	Coolant Pump_raw	0.422	0.998	0.975	0.997	0.001	0.007	0.008	0.005
	Coolant Pump_en	0.672	0.998	1.117	0.998	0.001	0.006	0.007	0.002
	Change [%]	59.17	0.02	14.57	0.04	-47.34	-12.65	-19.80	-53.07
	Right Pixel Headlamp_raw	1.594	0.719	2.851	0.748	0.275	0.052	0.049	0.067
	Right Pixel Headlamp_en	3.060	0.410	4.646	0.478	0.525	0.105	0.092	0.118
	Change [%]	92.01	-43.03	62.95	-36.05	90.71	100.59	89.81	74.86
	Left Pixel Headlamp_raw	1.607	0.730	2.569	0.762	0.252	0.046	0.044	0.057
	Left Pixel Headlamp_en	2.609	0.184	3.501	0.375	0.748	0.088	0.081	0.089
	Change [%]	62.30	-74.82	36.28	-50.71	197.26	89.01	83.35	55.31
	BCF_raw	7.097	-19.449	13.077	-13.156	43.527	0.009	0.008	0.019
	BCF_en	7.135	-23.300	13.301	-14.825	50.125	0.010	0.008	0.021
	Change [%]	0.53	-19.80	1.71	-12.68	15.16	7.22	-0.15	8.44
	Door ECU Front Left_raw	17.333	0.022	13.549	0.033	0.143	0.748	0.742	0.095
	Door ECU Front Left_en	15.443	0.067	14.116	0.093	0.171	0.712	0.693	0.092
	Change [%]	-10.91	210.95	4.19	179.64	19.70	-4.88	-6.61	-3.16
	Active Air Suspension_raw	23.651	0.157	16.424	0.220	0.310	0.163	0.116	0.065
	Active Air Suspension_en	11.747	0.600	12.543	0.529	0.209	0.075	0.074	0.042
	Change [%]	-50.33	282.12	-23.63	139.75	-32.63	-54.27	-36.51	-34.66
	Steering Column ECU_raw	41.723	-279.959	121.579	-1619.102	4336.299	2.584	2.039	1.372
	Steering Column ECU_en	17.557	-20.701	24.978	-120.317	337.183	0.741	0.541	0.464
	Change [%]	-57.92	92.61	-79.46	92.57	-92.22	-71.33	-73.44	-66.21
	CID_raw	6.538	-4.861	8.659	-8.102	23.846	0.038	0.029	0.041
	CID_en	4.490	-3.028	7.839	-8.395	30.622	0.027	0.024	0.034
	Change [%]	-31.32	37.70	-9.47	-3.62	28.42	-27.95	-19.87	-15.63

Table K.2: Detailed feature engineering results for GB. ECUs marked with "_raw" comprise the raw, whereas "_en" denotes the use of the modified dataset.

	ECU / Change	X-val weighted PAWD [%]	X-val weighted R^2	SD(PAAD)	X-val avg. R^2	SD(R^2)	X-val weighted MSE [A_2]	MSE [A_2]	SD(MSE) [A_2]
Vehicle A	Extractor Fan_raw	10.177	0.723	11.546	0.743	0.149	0.956	0.751	2.372
	Extractor Fan_en	10.079	0.723	11.503	0.744	0.150	0.959	0.753	2.382
	Change	-0.97	0.10	-0.38	0.10	1.00	0.79	0.32	0.42
	Coolant Pump_raw	0.229	0.998	0.278	0.998	0.000	0.006	0.005	0.002
	Coolant Pump_en	0.233	0.998	0.291	0.998	0.000	0.006	0.005	0.002
	Change	1.79	0.00	4.73	0.00	-0.09	-1.31	-0.90	0.40
	Right Pixel Headlamp_raw	0.916	0.922	1.791	0.926	0.058	0.055	0.045	0.040
	Right Pixel Headlamp_en	1.423	0.857	1.668	0.861	0.101	0.085	0.073	0.047
	Change	55.28	-7.03	-6.88	-7.00	72.43	53.34	61.76	17.41
	Left Pixel Headlamp_raw	0.990	0.914	1.479	0.922	0.061	0.045	0.038	0.031
	Left Pixel Headlamp_en	1.367	0.860	1.622	0.871	0.102	0.064	0.057	0.037
	Change	38.06	-5.95	9.68	-5.55	65.97	42.76	48.91	21.16
	BCF_raw	5.999	0.390	6.542	0.454	0.483	1.046	0.632	1.333
	BCF_en	7.389	0.420	7.459	0.548	0.426	1.122	0.656	1.436
	Change	23.17	7.78	14.02	20.60	-11.81	7.19	3.80	7.76
	Adaptive Suspension_raw	4.026	0.729	5.113	0.720	0.150	0.101	0.117	0.131
	Adaptive Suspension_en	1.602	0.805	3.399	0.787	0.109	0.073	0.095	0.125
	Change	-60.20	10.40	-33.52	9.27	-27.74	-27.80	-19.27	-4.69
	Driver Display_raw	3.204	0.626	2.555	0.411	0.772	0.002	0.001	0.001
	Driver Display_en	1.389	0.709	1.286	0.627	0.309	0.001	0.001	0.001
	Change	-56.66	13.36	-49.67	52.59	-59.99	-30.02	-26.16	-25.26
	CID_raw	1.156	0.557	1.696	0.477	0.300	0.002	0.003	0.004
	CID_en	1.352	0.483	2.307	0.320	0.682	0.003	0.003	0.004
	Change	16.92	-13.30	36.01	-32.92	127.49	4.11	8.42	-4.39
	Fuel Supply ECU_raw	0.126	0.992	0.217	0.991	0.017	0.001	0.001	0.000
	Fuel Supply ECU_en	0.210	0.990	0.320	0.989	0.021	0.001	0.001	0.000
	Change	67.53	-0.15	47.43	-0.17	24.78	11.16	12.21	23.65
Vehicle B	Seat ECU Driver_raw	6.260	0.654	14.285	0.014	1.863	1.492	1.130	1.361
	Seat ECU Driver_en	4.838	0.581	17.940	-0.667	3.936	1.466	1.140	1.369
	Change	-22.72	-11.21	25.59	-4830.59	111.31	-1.70	0.91	0.63
	Coolant Pump_raw	0.207	0.998	0.396	0.998	0.000	0.007	0.007	0.002
	Coolant Pump_en	0.403	0.997	1.160	0.997	0.001	0.008	0.008	0.003
	Change	94.54	-0.03	192.70	-0.04	92.26	12.39	17.76	73.27
	Right Pixel Headlamp_raw	1.899	0.717	3.240	0.797	0.321	0.066	0.120	0.066
	Right Pixel Headlamp_en	1.247	0.692	2.042	0.731	0.304	0.049	0.057	0.049
	Change	-34.36	-3.49	-36.99	-8.23	-5.19	-26.23	-52.65	-26.23
	Left Pixel Headlamp_raw	1.346	0.787	2.321	0.833	0.235	0.045	0.037	0.076
	Left Pixel Headlamp_en	0.991	0.751	1.545	0.772	0.256	0.040	0.033	0.045
	Change	-26.42	-4.53	-33.45	-7.29	9.24	-11.47	-12.14	-41.25
	BCF_raw	9.112	-22.708	14.071	-13.663	40.796	0.013	0.010	0.021
	BCF_en	5.876	-15.872	11.776	-8.763	36.626	0.008	0.006	0.018
	Change	-35.51	30.10	-16.31	35.87	-10.22	-42.52	-37.77	-15.44
	Door ECU Front Left_raw	9.741	0.119	11.844	0.152	0.155	0.672	0.651	0.103
	Door ECU Front Left_en	11.753	0.117	15.566	0.149	0.190	0.673	0.650	0.096
	Change	20.66	-1.86	31.42	-1.61	22.86	0.13	-0.09	-6.50
	Active Air Suspension_raw	8.331	0.637	8.078	0.634	0.155	0.072	0.058	0.038
	Active Air Suspension_en	6.108	0.672	7.793	0.658	0.129	0.064	0.054	0.034
	Change	-26.68	5.40	-3.52	3.77	-16.72	-10.16	-5.71	-10.33
	Steering Column ECU_raw	3.845	-2.499	7.051	-19.149	58.663	0.262	0.206	0.204
	Steering Column ECU_en	2.531	-0.000	6.392	-4.239	13.494	0.313	0.229	0.207
	Change	-34.18	99.99	-9.35	77.86	-77.00	19.44	11.47	1.58
	CID_raw	4.424	-3.781	8.428	-10.116	44.322	0.025	0.024	0.038
	CID_en	4.310	-3.977	9.877	-13.142	59.500	0.026	0.027	0.049
	Change	-2.58	-5.18	17.19	-29.91	34.25	2.58	16.18	30.09

Table K.3: Detailed feature engineering results for MLP. ECUs marked with "_raw" comprise the rawm whereas "_en" denotes the use of the modified dataset.

	ECU / Change	X-val weighted PAWD [%]	X-val weighted R^2	SD(PAAD)	X-val avg. R^2	SD(R^2)	X-val weighted MSE [A_2]	MSE [A_2]	SD(MSE) [A_2]
Vehicle A	Extractor Fan_raw	11.620	0.723	13.780	0.735	0.115	0.585	0.470	1.318
	Extractor Fan_en	14.324	0.693	14.842	0.712	0.149	0.943	0.745	2.299
	Change [%]	23.26	-4.12	7.70	-3.12	29.48	61.13	58.27	74.42
	Coolant Pump_raw	0.348	0.998	0.573	0.998	0.001	0.006	0.005	0.002
	Coolant Pump_en	0.381	0.998	0.648	0.998	0.000	0.005	0.004	0.001
	Change [%]	9.59	0.03	13.13	0.03	-35.70	-14.39	-14.56	-28.83
	Right Pixel Headlamp_raw	1.775	0.907	2.640	0.911	0.084	0.058	0.050	0.047
	Right Pixel Headlamp_en	3.440	0.729	3.837	0.748	0.312	0.163	0.137	0.151
	Change [%]	93.76	-19.57	45.32	-17.91	269.75	180.12	174.87	223.50
	Left Pixel Headlamp_raw	1.874	0.898	2.626	0.901	0.112	0.043	0.041	0.038
	Left Pixel Headlamp_en	2.804	0.800	3.690	0.809	0.165	0.093	0.087	0.061
	Change [%]	49.58	-10.88	40.50	-10.24	47.66	115.84	110.09	62.05
	BCF_raw	32.764	-3.083	61.868	-2.942	7.340	4.298	3.620	7.064
	BCF_en	39.485	-7.804	90.946	-7.160	23.580	8.608	7.490	20.726
	Change [%]	20.51	-153.12	47.00	-143.37	221.26	100.27	106.88	193.41
	Adaptive Suspension_raw	4.316	0.666	6.142	0.655	0.127	0.114	0.136	0.136
	Adaptive Suspension_en	2.187	0.802	4.621	0.783	0.112	0.071	0.092	0.125
	Change [%]	-49.32	20.50	-24.78	19.44	-11.76	-37.81	-32.16	-8.07
	Driver Display_raw	7.235	-0.050	11.669	-0.686	2.196	0.004	0.004	0.005
	Driver Display_en	6.792	-0.650	7.696	-0.620	3.395	0.005	0.003	0.004
	Change [%]	-6.13	-1201.94	-34.04	9.72	54.65	10.94	-14.82	-15.72
	CID_raw	5.128	-2.623	6.627	-1.747	4.847	0.016	0.011	0.024
	CID_en	6.26295	-2.05014	7.87395	-1.33563	3.44764	0.01983	0.01595	0.02990
	Change [%]	22.12	21.84	18.81	23.54	-28.86	22.79	49.26	25.37
	Fuel Supply ECU_raw	0.535	0.989	0.654	0.988	0.021	0.002	0.002	0.001
	Fuel Supply ECU_en	0.204	0.991	0.337	0.991	0.018	0.001	0.001	0.000
	Change [%]	-61.87	0.26	-48.47	0.28	-14.58	-25.23	-27.24	-49.55
Vehicle B	Seat ECU Driver_raw	11.927	-0.483	36.497	-8.803	37.509	1.738	1.470	2.260
	Seat ECU Driver_en	11.202	-0.019	76.388	-22.720	76.478	1.602	1.206	1.456
	Change [%]	-6.08	96.04	109.30	-158.08	103.89	-7.83	-17.95	-35.57
	Coolant Pump_raw	0.422	0.998	0.975	0.997	0.001	0.007	0.008	0.005
	Coolant Pump_en	0.228	0.999	0.442	0.999	0.000	0.004	0.004	0.001
	Change [%]	-46.03	0.10	-54.69	0.12	-72.23	-44.75	-48.17	-74.73
	Right Pixel Headlamp_raw	1.972	0.677	2.942	0.770	0.365	0.067	0.049	0.118
	Right Pixel Headlamp_en	4.929	-0.480	5.812	0.092	1.690	0.177	0.128	0.237
	Change [%]	149.94	-170.79	97.55	-88.02	362.66	163.06	163.66	100.80
	Left Pixel Headlamp_raw	2.314	0.747	2.989	0.747	0.360	0.062	0.046	0.102
	Left Pixel Headlamp_en	3.260	0.349	4.305	0.349	0.861	0.130	0.103	0.168
	Change [%]	40.88	-53.26	44.03	-53.26	139.10	110.65	121.81	64.78
	BCF_raw	7.400	-27.461	14.181	-22.542	60.118	0.011	0.010	0.023
	BCF_en	6.382	-19.259	12.595	-10.870	43.955	0.009	0.007	0.021
	Change [%]	-13.76	29.87	-11.18	51.78	-26.89	-21.74	-25.17	-9.47
	Door ECU Front Left_raw	24.322	-0.120	33.336	-0.107	0.554	0.851	0.830	0.317
	Door ECU Front Left_en	23.659	-0.098	30.455	-0.122	0.254	0.849	0.871	0.220
	Change [%]	-2.72	18.20	-8.64	-14.87	-54.13	-0.19	4.88	-30.55
	Active Air Suspension_raw	11.282	0.506	11.523	0.541	0.153	0.101	0.073	0.047
	Active Air Suspension_en	7.715	0.619	17.496	0.484	0.410	0.060	0.078	0.076
	Change [%]	-31.62	22.30	51.84	-10.53	168.28	-40.82	7.56	63.06
	Steering Column ECU_raw	27.081	-0.033	29.272	-2.320	9.322	1.746	1.029	2.240
	Steering Column ECU_en	3.344	0.510	5.878	-0.928	3.539	0.360	0.258	0.234
	Change [%]	-87.65	1652.05	-79.92	60.01	-62.03	-79.36	-74.88	-89.56
	CID_raw	7.900	-4.950	11.021	-12.606	36.493	0.049	0.047	0.055
	CID_en	7.457	-6.580	12.344	-17.949	67.234	0.051	0.052	0.068
	Change [%]	-5.61	-32.92	12.00	-42.39	84.24	4.64	10.56	21.92

Table K.4: Detailed feature engineering results for LSTM. ECUs marked with "_raw" comprise the raw whereas "_en" denotes the use of the modified dataset.

	ECU / Change	X-val weighted PAWD [%]	X-val weighted R^2	SD(PAAD)	X-val avg. R^2	SD(R^2)	X-val weighted MSE [A_2]	MSE [A_2]	SD(MSE) [A_2]
Vehicle A	Extractor Fan_raw	11.743	0.724	16.706	0.726	0.136	0.309	0.265	0.538
	Extractor Fan_en	14.495	0.695	17.465	0.707	0.130	0.663	0.534	1.510
	Change [%]	23.44	-3.96	4.54	-2.57	-4.51	114.46	101.68	180.61
	Coolant Pump_raw	0.923	0.969	3.301	0.967	0.016	0.081	0.069	0.031
	Coolant Pump_en	0.686	0.969	1.586	0.970	0.009	0.081	0.067	0.033
	Change [%]	-25.67	0.06	-51.96	0.27	-42.70	-0.24	-2.97	5.53
	Right Pixel Headlamp_raw	1.867	0.799	3.240	0.803	0.179	0.103	0.092	0.050
	Right Pixel Headlamp_en	3.982	0.748	6.103	0.764	0.195	0.147	0.127	0.097
	Change [%]	113.22	-6.34	88.36	-4.84	9.00	43.00	38.00	93.00
	Left Pixel Headlamp_raw	2.253	0.779	2.921	0.792	0.204	0.084	0.079	0.050
	Left Pixel Headlamp_en	3.735	0.750	5.028	0.773	0.294	0.105	0.096	0.099
	Change [%]	65.79	-3.73	72.15	-2.37	44.12	25.49	22.02	97.30
	BCF_raw	56.809	-5.921	83.893	-5.124	9.854	8.568	5.978	10.259
	BCF_en	6.722	-0.161	9.025	-0.120	0.528	1.229	0.886	1.515
	Change [%]	-88.17	97.28	-89.24	97.66	-94.64	-85.66	-85.18	-85.24
	Adaptive Suspension_raw	3.947	0.642	7.708	0.582	0.302	0.125	0.163	0.199
	Adaptive Suspension_en	2.781	0.771	5.693	0.744	0.144	0.084	0.113	0.169
	Change [%]	-29.55	20.07	-26.14	27.82	-52.43	-32.71	-30.80	-15.12
	Driver Display_raw	9.471	-34.220	21.666	-68.536	245.508	0.006	0.007	0.011
	Driver Display_en	8.652	-0.818	19.196	-2.007	4.367	0.006	0.007	0.012
	Change [%]	-8.65	97.61	-11.40	97.07	-98.22	6.67	7.03	0.56
	CID_raw	7.670	-56.947	10.726	-74.215	263.203	0.016	0.017	0.022
	CID_en	5.662	-1.236	6.159	-1.425	2.320	0.008	0.009	0.011
	Change [%]	-26.18	97.83	-42.58	98.08	-99.12	-53.38	-47.20	-51.48
	Fuel Supply ECU_raw	1.418	0.908	1.942	0.894	0.270	0.010	0.011	0.007
	Fuel Supply ECU_en	0.850	0.963	1.943	0.959	0.040	0.009	0.011	0.010
	Change [%]	-40.01	6.12	0.09	7.33	-85.37	-5.20	-4.07	53.69
Vehicle B	Seat ECU Driver_raw	17.624	-0.407	48.494	-28.886	111.241	1.800	1.320	1.603
	Seat ECU Driver_en	13.707	-2.265	68.581	-30.172	99.623	1.698	1.258	1.534
	Change [%]	-22.22	-457.19	41.42	-4.45	-10.44	-5.65	-4.66	-4.34
	Coolant Pump_raw	0.360	0.998	0.472	0.998	0.000	0.006	0.006	0.002
	Coolant Pump_en	0.464	0.998	0.655	0.998	0.000	0.006	0.006	0.002
	Change [%]	28.74	0.01	38.88	0.01	5.37	-2.75	-2.98	-7.08
	Right Pixel Headlamp_raw	2.003	0.682	3.137	0.699	0.329	0.049	0.049	0.051
	Right Pixel Headlamp_en	5.145	-0.447	5.932	0.097	1.499	0.169	0.120	0.218
	Change [%]	156.79	-165.55	89.09	-86.07	356.09	246.53	143.64	324.13
	Left Pixel Headlamp_raw	1.838	0.685	2.744	0.704	0.343	0.040	0.049	0.072
	Left Pixel Headlamp_en	3.015	0.230	4.258	0.395	0.783	0.126	0.098	0.180
	Change [%]	64.07	-66.40	55.18	-43.94	128.65	219.09	99.68	150.49
	BCF_raw	4.806	-28.708	9.834	-24.321	69.226	0.007	0.008	0.014
	BCF_en	9.777	-48.966	17.241	-35.711	101.983	0.018	0.013	0.036
	Change [%]	103.42	-70.57	75.31	-46.83	47.32	137.08	58.59	153.23
	Door ECU Front Left_raw	28.238	-0.546	35.202	-0.143	1.028	1.235	0.899	0.899
	Door ECU Front Left_en	12.402	-0.025	12.891	0.039	0.172	0.781	0.737	0.111
	Change [%]	-56.08	95.45	-63.38	127.23	-83.28	-36.76	-18.04	-87.61
	Active Air Suspension_raw	13.630	0.464	20.277	0.390	0.287	0.100	0.095	0.064
	Active Air Suspension_en	15.239	0.372	26.963	0.220	1.098	0.122	0.124	0.196
	Change [%]	11.81	-19.69	32.97	-43.50	282.15	22.03	31.16	207.37
	Steering Column ECU_raw	19.066	-19.789	34.053	-119.731	477.087	1.084	0.826	1.012
	Steering Column ECU_en	9.579	-0.666	13.897	-4.053	9.621	0.669	0.529	0.727
	Change [%]	-49.76	96.63	-59.19	96.61	-97.98	-38.30	-35.96	-28.13
	CID_raw	6.898	-4.306	11.210	-11.338	38.252	0.048	0.042	0.059
	CID_en	11.167	-4.616	11.139	-7.243	14.453	0.072	0.049	0.062
	Change [%]	61.88	-7.19	-0.63	36.11	-62.22	48.15	16.19	5.15

Table K.5: Detailed hyperparameter tuning results for RF. ECUs marked with "_en" comprise the feature engineered "_ht" denotes the use of the tuned parameter set.

	ECU / Change	X-val weighted PAWD [%]	X-val weighted R^2	SD(PAAD)	X-val avg. R^2	SD(R^2)	X-val weighted MSE [A_2]	MSE [A_2]	SD(MSE) [A_2]
Vehicle A	Extractor Fan_en	10.019	0.709	11.955	0.731	0.162	1.019	0.799	2.537
	Extractor Fan_ht	9.872	0.728	11.582	0.750	0.154	0.968	0.758	2.413
	Change [%]	-1.47	2.68	-3.12	2.55	-4.68	-5.06	-5.18	-4.87
	Coolant Pump_en	0.274	0.998	0.356	0.998	0.000	0.004	0.004	0.001
	Coolant Pump_ht	0.242	0.998	0.327	0.998	0.000	0.004	0.004	0.001
	Change [%]	-11.39	0	-8.08	0.00	-15.32	0.49	0.29	-6.11
	Right Pixel Headlamp_en	3.715	0.750	4.246	0.768	0.236	0.135	0.114	0.100
	Right Pixel Headlamp_ht	3.197	0.796	3.761	0.808	0.157	0.116	0.099	0.068
	Change [%]	-13.94	6.14	-11.43	5.25	-33.45	-14.23	-13.04	-32.33
	Left Pixel Headlamp_en	2.261	0.793	2.595	0.805	0.182	0.083	0.078	0.060
	Left Pixel Headlamp_ht	1.899	0.828	2.067	0.835	0.142	0.072	0.070	0.051
	Change [%]	-15.98	4.41	-20.34	3.70	-22.23	-13.11	-10.67	-14.93
	BCF_en	9.489	0.135	16.937	0.191	0.439	1.004	0.713	1.122
	BCF_ht	5.620	0.549	5.512	0.633	0.308	0.869	0.540	1.202
	Change [%]	-40.78	307.77	-67.45	232.46	-29.79	-13.51	-24.35	7.17
	Adaptive Suspension_en	2.187	0.802	4.621	0.783	0.112	0.071	0.092	0.125
	Adaptive Suspension_ht	1.799	0.804	4.166	0.784	0.114	0.071	0.092	0.126
	Change [%]	-17.75	0.23	-9.84	0.16	1.82	-0.21	0.55	1.14
	Driver Display_en	3.080	0.431	4.530	0.258	0.518	0.003	0.002	0.002
	Driver Display_ht	2.576	0.652	2.650	0.588	0.287	0.002	0.002	0.002
	Change [%]	-16.38	51.42	-41.49	128.02	-44.60	-20.96	-27.89	-21.89
	CID_en	1.662	0.334	2.632	0.232	0.621	0.003	0.005	0.003
	CID_ht	1.431	0.524	2.180	0.442	0.399	0.003	0.005	0.003
	Change [%]	-13.88	56.86	-17.17	90.33	-35.68	-17.87	-6	-17.87
	Fuel Supply ECU_en	0.441	0.988	0.765	0.987	0.028	0.002	0.002	0.001
	Fuel Supply ECU_ht	0.154	0.993	0.218	0.992	0.015	0.001	0.001	0.000
	Change [%]	-64.93	0.47	-71.49	0.52	-44.83	-35.22	-37.10	-69.88
Vehicle B	Seat ECU Driver_en	13.415	-5.158	108.221	-63.115	205.587	1.577	1.201	1.380
	Seat ECU Driver_ht	9.211	0.478	40.419	-5.229	25.816	1.520	1.153	1.364
	Change [%]	-31.33	109.26	-62.65	91.72	-87.44	-3.61	-3.97	-1.10
	Coolant Pump_en	0.228	0.999	0.442	0.999	0.000	0.004	0.004	0.001
	Coolant Pump_ht	0.255	0.998	0.594	0.998	0.000	0.005	0.005	0.002
	Change [%]	11.93	-0.04	34.38	-0.04	35.19	26.13	26.85	29.99
	Right Pixel Headlamp_en	3.060	0.410	4.646	0.478	0.525	0.105	0.092	0.118
	Right Pixel Headlamp_ht	3.347	0.380	5.094	0.462	0.558	0.123	0.107	0.157
	Change [%]	9.39	-7.26	9.64	-3.39	6.31	17.18	16.03	33.04
	Left Pixel Headlamp_en	2.609	0.184	3.501	0.375	0.748	0.088	0.081	0.089
	Left Pixel Headlamp_ht	2.541	0.309	3.620	0.499	0.580	0.094	0.080	0.114
	Change [%]	-2.60	68.01	3.41	32.94	-22.48	7.36	-1.61	28.64
	BCF_en	7.135	-23.300	13.301	-14.825	50.125	0.010	0.008	0.021
	BCF_ht	7.956	-18.807	13.417	-13.302	38.174	0.010	0.009	0.020
	Change [%]	11.51	19.29	0.88	10.27	-23.84	-0.46	14.83	-6.74
	Door ECU Front Left_en	15.443	0.067	14.116	0.093	0.171	0.712	0.693	0.092
	Door ECU Front Left_ht	11.459	0.133	15.035	0.155	0.168	0.662	0.648	0.111
	Change [%]	-25.80	98.61	6.51	67.34	-1.74	-6.95	-6.55	20.25
	Active Air Suspension_en	11.747	0.600	12.543	0.529	0.209	0.075	0.074	0.042
	Active Air Suspension_ht	8.464	0.603	18.046	0.498	0.443	0.076	0.079	0.073
	Change [%]	-27.95	0.61	43.87	-5.69	112.36	1.93	7.59	71.25
	Steering Column ECU_en	17.557	-20.701	24.978	-120.317	337.183	0.741	0.541	0.464
	Steering Column ECU_ht	2.210	0.523	4.378	-1.751	7.504	0.311	0.224	0.201
	Change [%]	-87.41	102.52	-82.47	98.54	-97.77	-57.98	-58.58	-56.56
	CID_en	4.490	-3.028	7.839	-8.395	30.622	0.027	0.024	0.034
	CID_ht	4.119	-2.448	7.555	-7.769	32.205	0.026	0.022	0.032
	Change [%]	-8.26	19.15	-3.62	7.46	5.17	-5.39	-5.93	-5.16

Table K.6: Detailed hyperparameter tuning results for GB. ECUs marked with "_en" comprise the feature engineered "_ht" denotes the use of the tuned parameter set.

	ECU / Change	X-val weighted PAWD [%]	X-val weighted R^2	SD(PAAD)	X-val avg. R^2	SD(R^2)	X-val weighted MSE [A_2]	MSE [A_2]	SD(MSE) [A_2]
Vehicle A	Extractor Fan_en	10.079	0.723	11.503	0.744	0.150	0.959	0.753	2.382
	Extractor Fan_ht	9.962	0.724	11.694	0.745	0.149	0.931	0.730	2.304
	Change	-1.16	0.14	1.66	0.05	-0.73	-2.96	-2.96	-3.25
	Coolant Pump_en	0.233	0.998	0.291	0.998	0.000	0.006	0.005	0.002
	Coolant Pump_ht	0.237	0.999	0.339	0.999	0.000	0.004	0.003	0.001
	Change	1.75	0.08	16.38	0.09	-28.69	-36.25	-36.63	-30.42
	Right Pixel Headlamp_en	1.423	0.857	1.668	0.101	0.857	0.085	0.045	0.040
	Right Pixel Headlamp_ht	1.593	0.859	1.632	0.106	0.859	0.081	0.073	0.047
	Change	12.01	0.21	-2.13	5.09	0.21	-4.09	61.76	17.41
	Left Pixel Headlamp_en	0.990	0.914	1.479	0.922	0.061	0.045	0.038	0.031
	Left Pixel Headlamp_ht	1.599	0.859	1.832	0.873	0.096	0.071	0.059	0.046
	Change	61.52	-6.06	23.88	-5.26	55.95	58.07	54.82	49.18
	BCF_en	7.389	0.420	7.459	0.548	0.426	1.122	0.656	1.436
	BCF_ht	5.212	0.543	5.840	0.627	0.341	0.852	0.540	1.280
	Change	-29.46	29.19	-21.70	14.54	-20.00	-24.05	-17.70	-10.90
	Adaptive Suspension_en	1.602	0.805	3.399	0.787	0.109	0.073	0.095	0.125
	Adaptive Suspension_ht	1.823	0.817	3.271	0.801	0.106	0.065	0.084	0.109
	Change	13.79	1.52	-3.76	1.81	-2.25	-10.59	-11.78	-12.95
	Driver Display_en	1.286	0.709	1.389	0.627	0.309	0.001	0.001	0.001
	Driver Display_ht	1.335	0.707	1.368	0.629	0.320	0.001	0.001	0.001
	Change	3.85	-0.30	-1.51	0.36	3.71	5.65	-2.01	0.01
	CID_en	1.352	0.483	2.307	0.320	0.682	0.003	0.003	0.004
	CID_ht	0.809	0.592	1.472	0.519	0.338	0.003	0.003	0.004
	Change	-40.15	22.54	-36.16	62.27	-50.50	-1.53	-0.61	19.25
	Fuel Supply ECU_en	0.210	0.990	0.320	0.989	0.021	0.001	0.001	0.000
	Fuel Supply ECU_ht	0.193	0.991	0.283	0.990	0.021	0.001	0.001	0.000
	Change	-8.07	0.06	-11.32	0.06	-2.88	-9.61	-7.83	-9.63
Vehicle B	Seat ECU Driver_en	4.838	0.581	17.940	-0.667	3.936	1.466	1.140	1.369
	Seat ECU Driver_ht	5.315	0.657	16.962	0.249	1.161	1.455	1.124	1.373
	Change	9.86	13.15	-5.45	137.30	-70.51	-0.79	-1.35	0.27
	Coolant Pump_en	0.403	0.997	1.160	0.997	0.001	0.008	0.008	0.003
	Coolant Pump_ht	0.305	0.998	0.821	0.998	0.001	0.006	0.006	0.002
	Change	-24.52	0.07	-29.21	0.07	-26.32	-26.75	-25.03	-20.66
	Right Pixel Headlamp_en	0.207	0.998	0.396	0.998	0.000	0.007	0.007	0.002
	Right Pixel Headlamp_ht	0.403	0.997	1.160	0.997	0.001	0.008	0.008	0.003
	Change	94.54	-0.03	192.70	-0.04	92.26	12.39	17.76	73.27
	Left Pixel Headlamp_en	1.247	0.692	2.042	0.731	0.304	0.049	0.057	0.049
	Left Pixel Headlamp_ht	0.951	0.739	1.543	0.747	0.296	0.034	0.030	0.034
	Change	-23.75	6.91	-24.42	2.10	-2.72	-30.32	-47.92	-30.32
	BCF_en	5.876	-15.872	11.776	-8.763	36.626	0.008	0.006	0.018
	BCF_ht	6.588	-17.468	11.313	-12.555	36.437	0.008	0.007	0.015
	Change	12.11	-10.06	-3.94	-43.27	-0.52	3.58	2.20	-13.92
	Door ECU Front Left_en	9.741	0.119	11.844	0.152	0.155	0.672	0.650	0.096
	Door ECU Front Left_ht	11.753	0.117	15.566	0.149	0.190	0.673	0.653	0.125
	Change	20.66	-1.86	31.42	-1.61	22.86	0.13	0.36	30.15
	Active Air Suspension_en	6.108	0.672	7.793	0.658	0.129	0.064	0.054	0.034
	Active Air Suspension_ht	6.182	0.679	7.782	0.666	0.132	0.063	0.053	0.033
	Change	1.22	1.08	-0.15	1.27	2.07	-1.82	-2.88	-1.77
	Steering Column ECU_en	2.531	-0.666	6.392	-4.239	13.494	0.313	0.229	0.207
	Steering Column ECU_ht	2.717	-3.973	9.773	-29.742	124.698	0.311	0.231	0.202
	Change	7.37	-496.55	52.89	-601.58	824.08	-0.40	0.72	-2.38
	CID_en	4.310	-3.977	9.877	-13.142	59.500	0.026	0.027	0.049
	CID_ht	4.548	-3.492	9.812	-10.726	48.625	0.025	0.027	0.052
	Change	5.53	12.20	-0.66	18.38	-18.28	-5.36	-1.96	4.68

Table K.7: Detailed hyperparameter tuning results for MLP. ECUs marked with "_en" comprise the feature engineered "_ht" denotes the use of the tuned parameter set.

	ECU / Change	X-val weighted PAWD [%]	X-val weighted R^2	SD(PAAD)	X-val avg. R^2	SD(R^2)	X-val weighted MSE [A_2]	MSE [A_2]	SD(MSE) [A_2]
Vehicle A	Extractor Fan_en	14.324	0.693	14.842	0.712	0.149	0.943	0.745	2.299
	Extractor Fan_ht	23.181	-0.164	30.771	-0.157	0.359	1.798	1.532	3.549
	Change [%]	61.83	-123.65	107.33	-121.99	140.26	90.77	105.74	54.38
	Coolant Pump_en	0.381	0.998	0.648	0.998	0.000	0.005	0.004	0.001
	Coolant Pump_ht	2.522	0.961	3.410	0.965	0.012	0.109	0.083	0.051
	Change [%]	561.98	-3.76	426.07	-3.31	3572.37	2197.15	1989.73	3331.73
	Right Pixel Headlamp_en	3.440	0.729	3.837	0.748	0.312	0.163	0.137	0.151
	Right Pixel Headlamp_ht	7.299	0.409	11.191	0.456	0.415	0.410	0.298	0.272
	Change [%]	112.18	-43.90	191.67	-39.00	33.07	151.01	116.71	80.08
	Left Pixel Headlamp_en	2.804	0.800	3.690	0.809	0.165	0.093	0.087	0.061
	Left Pixel Headlamp_ht	6.030	0.323	12.306	0.401	0.498	0.318	0.277	0.227
	Change [%]	115.10	59.66	233.53	-50.45	202.04	243.40	218.91	272.19
	BCF_en	39.485	-7.804	90.946	-7.160	23.580	8.608	7.490	20.726
	BCF_ht	5.620	0.549	5.512	0.633	0.308	0.869	1.210	1.869
	Change [%]	-85.77	107.03	-93.94	108.85	-98.69	-89.91	-83.85	-90.98
	Adaptive Suspension_en	1.538	0.811	3.749	0.794	0.100	0.068	0.087	0.108
	Adaptive Suspension_ht	4.225	0.417	8.097	0.385	0.141	0.209	0.257	0.216
	Change [%]	174.76	-48.55	115.97	-51.48	41.79	208.24	195.53	98.91
	Driver Display_en	6.792	-0.650	7.696	-0.620	3.395	0.005	0.003	0.004
	Driver Display_ht	49.105	-51.312	52.575	-79.440	145.360	0.102	0.070	0.086
	Change [%]	622.98	-7789.20	583.14	-12718.41	4181.17	2115.01	2206.06	2033.36
	CID_en	6.263	-2.050	7.874	-1.336	3.448	0.020	0.016	0.030
	CID_ht	8.544	-3.160	9.958	-3.045	3.153	0.015	0.016	0.018
	Change [%]	36.42	-54.12	26.47	-127.96	-8.55	-25.90	1.34	-38.71
	Fuel Supply ECU_en	0.204	0.991	0.337	0.991	0.018	0.001	0.001	0.000
	Fuel Supply ECU_ht	6.034	-0.222	9.736	-0.264	0.425	0.370	0.441	0.325
	Change [%]	2858.22	-122.37	2790.30	-126.70	2311.86	26799.38	31282.92	82459.77
Vehicle B	Seat ECU Driver_en	11.202	-0.019	76.388	-22.720	76.478	1.602	1.206	1.456
	Seat ECU Driver_ht	62.320	-201.805	631.115	-3065.391	8115.845	5.884	4.458	4.977
	Change [%]	456.34	-1053758.20	726.19	-13391.87	10512.05	267.22	269.53	241.81
	Coolant Pump_en	0.464	0.998	0.655	0.998	0.000	0.006	0.006	0.002
	Coolant Pump_ht	1.937	0.954	2.650	0.951	0.015	0.140	0.158	0.080
	Change [%]	317.73	-4.43	304.67	-4.73	3252.21	2190.26	2416.19	4938.19
	Right Pixel Headlamp_en	4.929	-0.480	5.812	0.092	1.690	0.177	0.128	0.237
	Right Pixel Headlamp_ht	3.632	-0.230	6.642	-0.039	1.509	0.171	0.228	0.462
	Change [%]	-26.33	52.11	14.29	-142.63	-10.67	-3.40	77.67	94.74
	Left Pixel Headlamp_en	3.260	0.069	4.305	0.349	0.861	0.130	0.103	0.168
	Left Pixel Headlamp_ht	2.916	0.259	5.292	0.318	0.598	0.189	0.289	0.792
	Change [%]	-10.57	277.27	22.92	-8.84	-30.60	45.15	181.31	370.33
	BCF_en	6.382	-19.259	12.595	-10.870	43.955	0.009	0.007	0.021
	BCF_ht	7.181	-16.520	11.586	-13.441	29.124	0.007	0.007	0.014
	Change [%]	12.52	14.22	-8.01	-23.66	-33.74	-17.61	-5.75	-31.37
	Door ECU Front Left_en	23.659	-0.098	30.455	-0.122	0.254	0.849	0.871	0.220
	Door ECU Front Left_ht	10.421	-0.064	10.599	-0.032	0.086	0.817	0.805	0.144
	Change [%]	-55.95	34.89	-65.20	73.94	-65.99	-3.74	-7.60	-34.50
	Active Air Suspension_en	7.715	0.619	17.496	0.484	0.410	0.060	0.078	0.076
	Active Air Suspension_ht	30.419	-1.522	23.591	-0.347	1.102	0.648	0.228	0.362
	Change [%]	294.29	-346.01	34.84	-171.73	169.12	986.59	192.08	373.39
	Steering Column ECU_en	3.344	0.510	5.878	-0.928	3.539	0.360	0.258	0.234
	Steering Column ECU_ht	53.153	-324.967	133.497	-1414.937	2761.396	4.534	3.483	2.753
	Change [%]	1489.72	-63817.30	2171.19	-152444.29	77928.39	1158.13	1247.87	1076.62
	CID_en	7.457	-6.580	12.344	-17.949	67.234	0.051	0.052	0.068
	CID_ht	6.098	-4.210	9.605	-11.502	41.346	0.041	0.035	0.044
	Change [%]	-18.23	36.01	-22.19	35.92	-38.50	-19.71	-32.51	-35.02

Table K.8: Detailed hyperparameter tuning results for LSTM. ECUs marked with "_en" comprise the feature engineered "_ht" denotes the use of the tuned parameter set.

	ECU / Change	X-val weighted PAWD [%]	X-val weighted R^2	SD(PAAD)	X-val avg. R^2	SD(R^2)	X-val weighted MSE [A_2]	MSE [A_2]	SD(MSE) [A_2]
Vehicle A	Extractor Fan_raw	14.495	0.695	17.465	0.707	0.130	0.663	0.534	1.510
	Extractor Fan_en	19.902	0.637	26.296	0.643	0.203	0.960	0.769	2.285
	Change [%]	37.30	-8.44	50.57	-9.10	56.03	44.80	44.21	51.37
	Coolant Pump_raw	0.686	0.969	1.586	0.970	0.009	0.081	0.067	0.033
	Coolant Pump_en	4.765	0.968	1.800	0.967	0.011	0.085	0.071	0.035
	Change [%]	594.65	-0.17	13.53	-0.23	22.71	5.50	5.58	5.02
	Right Pixel Headlamp_raw	3.982	0.748	6.103	0.764	0.195	0.147	0.127	0.097
	Right Pixel Headlamp_en	2.601	0.788	2.718	0.800	0.174	0.113	0.095	0.050
	Change [%]	-34.67	5.40	-55.46	4.69	-10.44	-22.86	-24.62	-48.13
	Left Pixel Headlamp_raw	3.735	0.750	5.028	0.773	0.294	0.105	0.096	0.099
	Left Pixel Headlamp_en	1.457	0.805	2.451	0.812	0.158	0.079	0.078	0.049
	Change [%]	-60.99	7.37	-51.26	4.99	-46.19	-24.79	-18.78	-48.13
	BCF_raw	6.722	-0.161	9.025	-0.120	0.528	1.229	0.886	1.515
	BCF_en	13.608	-0.195	18.443	-0.120	1.028	1.505	1.127	1.757
	Change [%]	102.43	-21.13	104.36	-0.03	94.59	22.48	27.22	16.02
	Adaptive Suspension_raw	2.781	0.771	5.693	0.744	0.144	0.084	0.113	0.169
	Adaptive Suspension_en	2.585	0.775	4.986	0.755	0.125	0.082	0.107	0.152
	Change [%]	-7.06	0.52	-12.42	1.48	-13	-2	-5	-10
	Driver Display_raw	8.652	-0.818	19.196	-2.007	4.367	0.006	0.007	0.012
	Driver Display_en	8.717	-0.405	13.458	-0.734	1.644	0.009	0.007	0.009
	Change [%]	0.75	50.49	-29.91	63.41	-62.37	42.67	0.00	-25.00
	CID_raw	5.662	-1.236	6.159	-1.425	2.320	0.008	0.009	0.011
	CID_en	4.199	-0.234	7.204	-0.515	1.760	0.008	0.011	0.020
	Change [%]	-25.83	81.08	16.97	63.85	-24.13	3.50	25.81	81.21
	Fuel Supply ECU_raw	0.850	0.963	1.943	0.959	0.040	0.009	0.011	0.010
	Fuel Supply ECU_en	1.520	0.907	2.626	0.900	0.170	0.016	0.018	0.010
	Change [%]	78.76	-5.84	35.13	-6.17	329.31	67.16	60.62	0.13
Vehicle B	Seat ECU Driver_raw	13.707	-2.265	68.581	-30.172	99.623	1.698	1.258	1.534
	Seat ECU Driver_en	10.597	-0.956	50.099	-24.381	67.593	1.693	1.265	1.585
	Change [%]	-22.69	57.82	-26.95	19.19	-32.15	-0.28	0.50	3.33
	Coolant Pump_raw	0.839	0.960	0.945	0.957	0.012	0.117	0.133	0.045
	Coolant Pump_en	7.453	0.946	1.012	0.944	0.011	0.162	0.174	0.052
	Change [%]	788.54	-1.52	7.09	-1.36	-10.10	38.48	31.18	16.54
	Right Pixel Headlamp_raw	5.145	-0.447	5.932	0.097	1.499	0.169	0.120	0.218
	Right Pixel Headlamp_en	4.224	0.138	5.344	0.313	0.767	0.133	0.106	0.167
	Change [%]	-17.90	130.84	-9.93	221.53	-48.85	-21.66	-11.93	-23.04
	Left Pixel Headlamp_raw	3.015	0.230	4.258	0.395	0.783	0.126	0.098	0.180
	Left Pixel Headlamp_en	3.674	-0.147	3.966	0.329	0.960	0.117	0.090	0.121
	Change [%]	21.86	-163.89	-6.88	-16.68	22.48	-7.18	-7.75	-33.07
	BCF_raw	9.777	-48.966	17.241	-35.711	101.983	0.018	0.013	0.036
	BCF_en	7.593	-22.880	12.429	-16.025	43.382	0.009	0.008	0.019
	Change [%]	-22.34	53.27	-27.91	55.13	-57.46	-46.62	-39.05	-48.41
	Door ECU Front Left_raw	12.402	-0.025	12.891	0.039	0.172	0.781	0.737	0.111
	Door ECU Front Left_en	9.491	-0.002	10.617	0.054	0.154	0.762	0.726	0.097
	Change [%]	-23.48	92.80	-17.64	38.07	-10.29	-2.37	-1.48	-13.02
	Active Air Suspension_raw	15.239	0.372	26.963	0.220	1.098	0.122	0.124	0.196
	Active Air Suspension_en	11.612	0.447	11.758	0.511	0.164	0.120	0.080	0.057
	Change [%]	-23.80	19.95	-56.39	132.09	-85.06	-1.42	-35.31	-70.71
	Steering Column ECU_raw	9.579	-0.666	13.897	-4.053	9.621	0.669	0.529	0.727
	Steering Column ECU_en	14.244	-2.407	20.633	-7.649	17.244	0.660	0.525	0.954
	Change [%]	48.71	-261.44	48.48	-88.72	79.23	-1.25	-0.78	31.22
	CID_en	0.049	0.062	11.139	-7.243	14.453	0.072	0.049	0.062
	CID_ht	0.039	0.050	11.508	-9.492	33.057	0.039	0.039	0.050
	Change [%]	-20.68	-18.92	3.31	-31.04	128.72	-45.11	-20.68	-18.92

L Results of the Weighted Sum Analysis

Table L.1: Detailed results of the WSA. Utility values per ECU and ML modeling algorithm (all values are rounded to the third decimal place, maximum values in boldface).

	ECUs	RF	GB	MLP	LSTM
Vehicle A	Extractor Fan	18.157	21.776	11.732	11.092
	Coolant Pump	**54.910**	**48.500**	**45.111**	**40.112**
	Right Pixel Headlamp	33.610	44.641	14.227	31.437
	Left Pixel Headlamp	43.288	46.047	14.431	34.366
	BCF	22.735	22.225	10.653	4.918
	Adaptive Suspension	35.495	36.654	20.357	30.752
	Driver Display	38.829	40.117	15.968	14.812
	CID	32.810	38.880	19.697	20.911
	Fuel Supply ECU	53.771	45.920	15.965	38.327
Vehicle B	Seat ECU Driver	14.481	20.817	11.592	5.321
	Coolant Pump	53.470	48.015	43.939	34.668
	Right Pixel Headlamp	21.582	41.187	21.517	17.218
	Left Pixel Headlamp	26.543	44.147	22.814	19.049
	BCF	20.160	22.956	21.712	15.041
	Door ECU Front Left	8.195	12.418	14.520	6.445
	Active Air Suspension	20.261	31.528	11.914	12.427
	Steering Column ECU	27.790	23.469	11.102	10.784
	CID	22.578	24.818	21.859	14.743