Michael Neuhold

Python-Based Extension for Automated Evaluation of Charging Measurement Data in AVL Drive 5

Diploma Thesis

5AHME, 2025-26

HTBL Kapfenberg

Supervisor

Dipl.-Ing Messner, Thomas

Kapfenberg, April 2026

Berufsbildende Schulen

Erklärung

Die unterfertigten Kandidaten/Kandidatinnen haben gemäß den geltenden schulrechtlichen Bestimmungen die Ausarbeitung einer abschließenden Arbeit (Diplomarbeit bzw. Abschlussarbeit) mit folgender Aufgabenstellung gewählt:

Python-Based Extension for Automated Evaluation of Charging Measurement Data in AVL Drive 5

Individuelle Aufgabenstellungen im Rahmen des Gesamtprojektes:

- Michael Neuhold (5AHME_w): **Extend acquisition with user-story signals and automate analysis for DC/AC charging. Pull data via Python, store key signals, compute power/losses, and publish KPIs/graphs back to AVL Drive 5.**

Die Kandidaten/Kandidatinnen nehmen zur Kenntnis, dass die abschließende Arbeit in eigenständiger Weise und außerhalb des Unterrichtes zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichtes mit einbezogen werden können, die jedenfalls als solche entsprechend kenntlich zu machen sind.

Die Abgabe der vollständigen abschließenden Arbeit hat in digitaler und in zweifach ausgedruckter Form bis spätestens **08.03.2026** beim zuständigen Betreuer/der zuständigen Betreuerin zu erfolgen.

Die Kandidaten/Kandidatinnen nehmen auch zur Kenntnis, dass ein Abbruch der abschließenden Arbeit nicht möglich ist.

Kandidaten/Kandidatinnen:	**Datum und Unterschrift bzw. Handysignatur:**
Michael Neuhold (5AHME_w)	13.10.2025 

Statutory declaration

I, the undersigned student, hereby declare under oath that I have written the present diploma thesis entitled:

Python-Based Extension for Automated Evaluation of Charging Measurement Data in AVL Drive 5

independently and without any inadmissible external assistance. All parts of this thesis that are taken literally or in essence from publications or other sources have been clearly marked as such and properly referenced.

Furthermore, I declare:

- That no content was taken from unreferenced sources.
- That no unauthorized assistance were used.
- That all contributing individuals and their contributions have been appropriately acknowledged.

Declaration on the Use of Artificial Intelligence (AI)

I declare that the use of Artificial Intelligence (AI) in this diploma thesis is as follows:

- No use of AI for new, uncitet content creation.
- AI was used only as support, e.g., for research, grammar checking, or better formulating sentences. All resulting content has been independently reviewed, revised, and understood.

I am aware that any unacknowledged or unauthorized use of AI may be considered academic misconduct and may lead to appropriate consequences.

Michael Neuhold

Acknowledgments

I would like to express my sincere gratitude to AVL List GmbH for the opportunity to complete my internship and for their valuable support during this diploma thesis.
Special thanks go to Prof. Messner Thomas for supervising my work and to Prof. Dirl Matthias for connecting me with AVL.
I also want to thank all my co-workers at AVL for their continuous help and encouragement throughout the project.
Finally, I wish to thank my family and friends for their support, especially for reviewing and correcting this thesis.

Abstract

This diploma thesis addresses the limited evaluation of battery charging processes in AVL Drive 5, where previously only a few parameters were considered. To overcome this limitation, the existing Python-based analysis was extended to support both AC and DC charging sessions and to automatically calculate a broad range of key performance indicators.
The implemented solution standardizes input data, detects charging events, and generates comprehensive results that are directly integrated into AVL Drive 5 and its database. As a result, charging analyses are now more complete, accurate, and immediately available to engineers without additional manual effort.
The outcome is an efficient and scalable framework that significantly enhances benchmarking capabilities and provides a solid foundation for future extensions in charging analysis.

Contents

1. Introduction

Chapter 1 provides an overview of the background, goals, scope, and motivation for this diploma thesis. It describes the starting point of the project and defines the key objectives and boundaries of the work.

1.1. Initial situation

AVL Drive 5 is a AVL-developed tool used for the visualization and analysis of vehicle measurement data. It supports real-time display of test results, typically in the form of time-value graphs and numerical data. For automated analysis, Python scripts are already being used within the software to process specific driving scenarios, such as acceleration and deceleration events.
Although existing templates for evaluating battery charging processes are available, they are limited to reading and processing only a single voltage and current value, and displaying one calculated power value. However, modern vehicles provide significantly more relevant charging-related parameters.

1.2. Goals

The main goal of this diploma thesis is to improve the existing Python script used in AVL Drive 5 for analyzing battery charging processes.
The script will be extended to support a wide range of measurement channels, such as current, voltage, torque, and speed, depending on what is available in the test data. Based on this input data, the script calculates important values like power, energy, efficiency, and charging duration. These results are then saved to the database and shown in AVL Drive 5 as graphs and numerical values.
This work helps make the charging analysis more complete and accurate, especially for fast charging. The full list of required channels and output values is based on the user story provided by AVL and will be discussed later in this thesis.

1.3. Definition and scope of work

This diploma thesis covers the full extension of an existing Python-based evaluation script used in AVL Drive 5 for battery charging analysis.
The goal is to support both DC and AC charging scenarios and to calculate all relevant key performance indicators (KPIs) based on the measurement data.
The script reads multiple measurement channels such as current, voltage, torque, speed, State of Charge (SOC), and temperature. It uses a predefined JSON file to map different channel name variants to a standard format. All required input signals are processed automatically if present in the measurement.
The calculated output includes power and energy values for several subsystems,

charging durations based on SOC levels, efficiency metrics (e.g. DCDC, OBC, charging efficiency), and various statistical values. Calculations are only performed if the required signals are available in the measurement; optional values are handled accordingly.
The output is written to the AVL Drive 5 environment and partially to the internal database, using an updated Custom_tree.csv and internal transfer functions. Visualization is handled entirely by AVL Drive 5. No plots or GUI changes were made as part of this work.
The implementation was done entirely in Python using pandas, numpy, and scipy.

1.4. Project environment and motivation

This project was carried out during an internship at AVL List GmbH [1] in Graz as part of the diploma thesis in the 5th year at HTBL Kapfenberg. The work was directly related to the AVL Drive 5 analysis system, which is used for evaluating vehicle test data.
Battery charging is a critical part of electric vehicle testing, and precise analysis of charging data is essential for benchmarking and system validation. Although AVL Drive 5 already supported some basic charging analysis, many relevant signals and KPIs were missing or not yet fully automated.
The motivation for this project came from the need to extend the existing script to support a complete charging evaluation workflow for both Alternating Current (AC) and Direct Current (DC) charging sessions. This enables engineers to get detailed results directly in AVL Drive 5 without using external tools or calculations.
On a personal level, the project was an opportunity to combine my interest in programming with a real-world application inside a large company. I already enjoyed writing Python code during my studies at HTL and university, and this project allowed me to apply those skills in a structured and professional environment. AVL was a company I had heard many good things about, so I was especially motivated to work there and contribute to an actual production setup.

[1] https://www.avl.com/en

2. Theoretical background

Chapter 2 presents the theoretical background of the Python libraries that are commonly used for data analysis and numerical computations in scientific and engineering contexts.

2.1. Scripting and Data Handling with Python

Python[1] is a widely adopted language in scientific computing and data science due to its rich ecosystem of libraries tailored for numerical operations and structured data handling (1). Among the most foundational libraries in this area are *pandas*, *numpy*, and *scipy*, each of which builds on core Python functionality to enable efficient and expressive data workflows.

2.1.1. NumPy

NumPy[2] is a core library for numerical operations in Python. It introduces the *ndarray*, a fixed-type, homogeneous N-dimensional array object (2), which enables efficient computation on large datasets by avoiding Python-level loops.
Internally, NumPy arrays are stored in contiguous memory blocks and are optimized for vectorized operations, where mathematical functions like *mean*, *max*, or element-wise operations such as *np.multiply* are implemented in compiled code. This allows for significant performance improvements over native Python lists, especially when dealing with large datasets (3).
NumPy also provides mechanisms for handling missing or undefined values, such as the special constant *np.nan*, and logical operations like *np.isnan* to detect them.

2.1.2. Pandas

pandas[3] provides high-level data structures and functions designed to make working with structured data fast and intuitive. The central structure is the *DataFrame*, which represents a two-dimensional, labeled data table with columns of potentially heterogeneous data types (4). Operations such as sorting, indexing, filtering, and time-based slicing are implemented in a way that mimics relational database logic but remains embedded in Python.
A *DataFrame* is composed of one or more *Series*, which are one-dimensional labeled arrays (5). This internal structure allows columns to be accessed individually while maintaining alignments across a shared index. The *pandas* library is particularly suited for handling time-series or tabular sensor data due to its indexing capabilities and compatibility with NumPy.

[1] https://www.python.org
[2] https://numpy.org
[3] https://pandas.pydata.org

2.1.3. SciPy Interpolation

Built on top of NumPy, *scipy*[4] offers a broad collection of algorithms for scientific computing, including integration, optimization, and signal processing. The module *scipy.interpolate* provides tools for interpolation of data points.
One common approach is the use of the *interp1d* function, which creates a continuous function based on discrete data using linear or higher-order interpolation. This is useful in contexts where uniform sampling is required or values must be estimated at arbitrary time steps (6).

2.2. JSON in Combination with Python

JavaScript Object Notation (JSON) is a lightweight, text-based format for representing structured data. It is widely used due to its language independence and human-readability. In Python, JSON data is typically processed using the built-in *json* module (7).
JSON structures are composed of key-value pairs enclosed in curly braces {} (representing objects, i.e., Python dictionaries), and ordered collections in square brackets *[]* (representing arrays, i.e., Python lists). The *json.load()* function is commonly used to parse JSON files into Python-native data structures, while *json.dumps()* converts Python objects into JSON-formatted strings.

[4] https://scipy.org

3. Implementation concepts

Chapter 3 outlines the practical concepts and methods applied during the implementation of the analysis, describing the underlying user story, its functional requirements, and the tools and frameworks used to realise the solution.

3.1. Procedure

The starting point of the project was a proposal from the project manager at AVL, who identified the limited evaluation of charging processes in AVL Drive 5 as a relevant topic. During the subsequent internship, the project was officially defined and the user story was created together with the responsible managers and the internal client. This specification provided the functional scope of the implementation and described which KPIs and calculations had to be integrated.
Based on this definition, the first step was to study the theoretical background and to understand the relations between the required parameters. In parallel, I became familiar with the large existing codebase of AVL Drive 5, in order to learn how measurement data can be accessed, processed, and written back into the application. This was an essential step to prepare for the practical implementation.
The implementation itself was carried out incrementally, by programming the KPIs one after another. Each function was tested extensively within the AVL Drive 5 environment to ensure correct calculations and stable integration. In addition, documentation was created continuously to record which formulas, libraries, and design decisions were applied. This iterative workflow made it possible to validate intermediate results and to react quickly to issues such as inconsistent measurement data or missing channels.
After completing the internship at AVL, the focus shifted to writing the diploma thesis. The practical results and the developed solution were structured, documented, and embedded into the academic framework of this work, linking the implementation with theoretical background, detailed analysis, and evaluation.

3.2. User-Story & Requirements

The implementation is based on a user story originating from the benchmarking department, which defines the functional scope and expected outcome of the developed analysis. The goal is to enable a comprehensive evaluation of vehicle battery charging processes within AVL Drive 5, with support for *DC* fast charging and *AC* charging scenarios. The primary focus is on charging analysis, where accurate event detection, robust peak handling, and the automated calculation of a wide range of KPIs are required.
At the beginning of each analysis, the script must correctly detect the start and end of a charging session. The start is determined when the high-voltage battery current (*I_HV_Battery*) exceeds a defined threshold — set to 2 A —, while the end

is identified when it drops below this threshold towards the end of the measurement. Initial spikes in the signal (see Figure A.1), which are common at the start of a charging process, must be ignored to prevent distortion of the calculated results. A central requirement is the ability to handle variations in channel naming across different vehicle types and measurement setups. For this purpose, a configuration file *Config_battery_charging.json* (see Listing 7) defines standard channel names and their possible aliases. The script reads measurement data from AVL Drive 5, maps the raw channel names to their standardised counterparts, and uses the first available valid channel in the list for further processing. For example, the high-voltage battery voltage (*U_HV_Battery*) is combined with current (*I_HV_Battery*) to determine the high-voltage battery power (*PWR_HV_Battery*), while the low-voltage battery channels *I_LV_Battery* and *U_LV_Battery* are used to calculate *PWR_LV_Battery*. Optional channels, such as *AC2* or *PTC2*, may not be present in all measurements. In such cases, defined fallback channels are used, or the corresponding result is marked as unavailable and log messages are printed so that the user is informed.

Once the channel mapping is complete, the system calculates all KPIs defined in the user story. These include charging power values for multiple subsystems, efficiency metrics such as *Charging_efficiency*, *DCDC_efficiency* and *OBC_efficiency*, cumulative energy values obtained from the integration of power signals, as well as detailed time-based parameters describing charging durations for specific SOC ranges. Additional criteria track operation times for auxiliary components, time to maximum power, and the ratio between average and peak power during the session. In DC scenarios, the power at the charging port (*PWR_HV_DC_ChargePort*) and its cumulative energy transfer (*E_HV_DC_ChargePort*) are key outputs, while in AC scenarios, onboard charger performance (*PWR_HV_OBC_out*) and related losses are evaluated. Statistical parameters, including minimum, maximum, and mean values for relevant voltage, current, and temperature channels, are also part of the calculated outputs. All parameters calculated by the script are stored in an internal result tree within the Python implementation. This tree is then transferred to AVL Drive 5, where the values are displayed according to the predefined structure and parameter IDs. The file *Custom_tree.csv* (see Listing 8) serves as a reference definition for this structure, containing all possible channels, their unique parameter IDs, and their organisation within main and sub operation modes. While the file itself is not modified by the script, it ensures that the calculated results are correctly assigned and visualised within AVL Drive 5.

The requirements also demand that the system is robust against incomplete datasets, inconsistent channel naming, and varying test procedures. By adhering to the mapping configuration and implementing conditional fallbacks, the analysis can be applied across a wide range of vehicles and measurement conditions.

3.3. Used Tools & Frameworks

The implementation is fully integrated into the AVL Drive 5 environment, which provides the measurement data via its internal framework layer written in C[1]. The Python scripts operate within this embedded environment, retrieving measurement channels directly from AVL Drive 5 and preparing the results for visualisation in the application its native interface. The graphical plotting is handled entirely by AVL Drive 5, while the scripts focus on reading, processing, and writing the calculated values.
Data processing is primarily carried out using the Python libraries *pandas* (see Section 2.1.2) and *numpy* (see Section 2.1.1), as described in detail in Chapter2. The *pandas DataFrame* structure is used to store and manipulate time-series data from the measurement files, allowing efficient filtering and selection of relevant time intervals. Common operations include the use of *.max()* to determine the maximum value within the charging period and *.mean()* to compute average values over a defined range. These operations are applied to the standardised channel names after the mapping process, ensuring consistent results across all datasets.
The configuration of channel mappings is handled by the file *Config_battery_charging.json* (see Listing 7), which defines all relevant measurement channels, their aliases, and the selection order to be used during processing. The output is written to the *Custom_tree.csv* file (see Listing 8), which contains the parameter IDs and labels required by AVL Drive 5 to display the results numerically and graphically. This configuration-driven approach ensures that the analysis remains flexible and easily adaptable to new vehicle types, measurement conditions, and naming conventions without the need for code modifications.

[1] https://www.c-language.org

4. Data Analysis

Chapter 4 describes the strategy for analyzing battery charging measurements. It introduces the available input signals and their meaning in the automotive context, and explains the fundamental equations that form the technical background for the calculation of key performance indicators in later sections.

4.1. Available Data & Their Meaning

For the analysis of battery charging processes, several measurement channels play a central role. These signals originate from the electric powertrain of the vehicle and describe the behavior of the high-voltage battery as well as of the electric machine. They are essential for the detection of charging events, for defining the analysis boundaries, and for calculating KPIs in later steps.
The signal *I_HV_Battery* represents the electrical current of the high-voltage battery. It is directly linked to the amount of charge flowing into or out of the battery. In the context of charging analysis, this channel is also used to detect the start and end of the charging session, since the presence of a significant current marks the active charging phase. Therefore, *I_HV_Battery* is not only a physical quantity but also a trigger signal for session detection.
Closely related is the channel *U_HV_Battery*, which measures the voltage level of the high-voltage battery. Together, voltage and current describe the electrical operating state of the battery. While *U_HV_Battery* typically remains within a defined range depending on the SOC and battery design, *I_HV_Battery* is the dynamic indicator for charging or discharging.
From these two channels, higher-level quantities such as the instantaneous high-voltage battery power (*PWR_HV_Battery*) and the accumulated energy (*E_HV_Battery*) are derived. These represent some of the most important KPIs for evaluating charging sessions, as they quantify how much power was delivered to the battery at any given time and how much total energy was stored during the process.
Besides the battery-related channels, signals from the electric machine are also relevant in certain scenarios. These include the torque (*M_EM*) and the rotational speed (*N_EM*) of the motor. Even though they are not directly part of the charging process, they enable the calculation of the mechanical power output and thus provide a connection between the electrical input and the mechanical behavior of the vehicle. This is particularly relevant when analyzing combined driving and charging scenarios or when evaluating overall efficiency.

4.2. Technical Background for Calculations

Based on the signals introduced in Section 4.1, different physical quantities can be derived. These form the basis for calculating the KPIs defined in the user story. The following equations summarize the most relevant relations.
The instantaneous electrical power of the high-voltage battery is obtained from the product of current and voltage:

$$P_{\text{HV\_Battery}} = U_{\text{HV\_Battery}} \cdot I_{\text{HV\_Battery}} \tag{4.1}$$

This relation corresponds to the fundamental law of electrical power (8). In the context of charging analysis, it quantifies how much power is transferred to the battery at any given time.
For evaluating the overall energy that has been stored during the charging session, the power signal is integrated over time:

$$E_{\text{HV\_Battery}} = \int P_{\text{HV\_Battery}}\, dt \tag{4.2}$$

This accumulated value represents the total amount of energy supplied to the battery in watt-hours and is one of the key figures in benchmarking (9).
Besides electrical parameters, mechanical signals from the electric machine are relevant in certain scenarios. The mechanical output power can be expressed as a function of torque and rotational speed:

$$P_{\text{EM\_REAR}} = M_{\text{EM\_REAR}} \cdot 2\pi \cdot \frac{n_{\text{EM\_REAR}}}{60} \tag{4.3}$$

Here, $M_{\text{EM\_REAR}}$ denotes the torque in Newton-meters and $n_{\text{EM\_REAR}}$ the rotational speed in revolutions per minute. This equation describes how electrical energy delivered to the motor is converted into mechanical power (10).
All further KPIs defined in Chapter 4, such as charging efficiency or statistical indicators, are based on combinations or aggregations of these fundamental quantities. By applying these equations, the raw measurement data can be transformed into meaningful performance indicators that allow objective comparisons between different vehicles and charging scenarios.

5. Technical implementation

Chapter 5 focuses on the integrated logging functionality of the Python scripts. All major steps of the workflow up to the process from reading measurement channels, performing calculations, and storing results, up to writing data back to AVL Drive 5 and the database are continuously logged. This ensures full transparency of the data flow, supports debugging during development, and provides valuable diagnostic information for end users in operational environments.

5.1. Python Script Architecture

The Python implementation is structured in a modular way to ensure maintainability and clarity. The file structure is shown in Listing 5.1. Each file has a dedicated responsibility, following the principles of separation of concerns.

```
battery-charging/
    configurations.py
    data_provider.py
    dependencies.py
    main.py
    settings.py
```

Figure 5.1.: Python file structure

The entry point of the analysis system is *main.py*, which orchestrates the loading of measurement data, the execution of calculations, and the transfer of results back into AVL Drive 5. The file *configurations.py* contains static definitions and mappings, including references to the configuration JSON file (see Listing 7) used for channel alignment. Parameters such as thresholds and constants are defined in *settings.py*, whereas *dependencies.py* manages shared imports and ensures consistency across modules. Finally, *data_provider.py* offers the access layer to measurement channels from AVL Drive 5 and prepares them in a structured format for further processing.
For consistent formatting, the entire codebase was formatted with the *Black code formatter*[1]. Function and class documentation follow the NumPy/SciPy docstring convention, which ensures a clear and standardized structure for describing input parameters, return values, and examples (11).

[1] https://black.readthedocs.io/en/stable/

5.2. Data Transfer from & to AVL Drive 5 + Database

This section explains how measurement channels are retrieved from AVL Drive 5, how events and results are computed and written back into the internal result tree, and how standardized channels are persisted in the database for later reuse. The Python code below is shown verbatim and reflects the implementation used in the embedded Drive 5 environment.

5.2.1. Fetching Data from AVL Drive 5

The loading step iterates over a predefined list of possible channel names (*ch_names*). For each candidate, the embedded API *avl_drive.core.read_parameter* is called with the *Id.Measurement.Channel* identifier to fetch the raw channel meta and data for the current *measurement_id*. Wrapping the result in a *pandas.DataFrame* (*df_channel_info*) normalizes structure and typing so that downstream mapping and selection logic can operate on a uniform tabular representation. This design aligns with the configuration-driven mapping introduced earlier, where the first available alias is taken as the standard channel for further processing.

Listing 1 Fetching Data from AVL Drive 5

```
for name in ch_names:
        df_channel_info = pd.DataFrame(
            avl_drive.core.read_parameter(
                session=session,
                parameter_id=Id.Measurement.Channel,
                original_measurement_id=measurement_id,
                name=name
            )
        )
```

5.2.2. Writing Data to AVL Drive 5

After channels have been harmonized and transformed, *calculate_events(...)* derives all required outputs (KPIs, durations, integrals) for the active charging session. The call to *update_tree_for_measurement(...)* then transfers these results into the internal result tree (*self.tree*) (see Listing 5). This tree structure is consumed by AVL Drive 5's C-based framework, which takes care of persisting values with their parameter IDs and exposing them to the graphical user interface (GUI) (see Figure A.2).

Listing 2 Writing Data to AVL Drive 5

```
df_events = calculate_events(
    session=session,
    measurement_id=measurement_id,
    df_channels=df_channels,
    df_channel_soc=df_channel_soc,
    ch_defs=ch_defs,
)

update_tree_for_measurement(
        self.tree,
        df_events=df_events,
        measurement_id=measurement_id,
    )
```

This separation between calculation and transfer keeps responsibilities clear: Python focuses on data preparation and correctness, while the framework handles storage, visualization, and ID mapping according to the predefined *Custom_Tree.csv* (see Listing 8).

5.2.3. Writing Data to the Database

For template reuse and downstream analyses, selected input channels are also written back into the database under standardized names. The following snippet derives the sampling frequency from the measurement metadata, constructs a channel object via *futures.channel.create_channel(...)* with canonical naming (*"I_HV_Battery"*) and unit, and appends it to a batch. The batch is then committed with *trigger.write_channels(...)*. The *try/except* guard logs any I/O failure with full context and re-raises the exception so that calling layers can react appropriately.

Listing 3 Writing Data to the Database

```
frequency = 1 / df_channels["sample_time"].iloc[0]
channels = []

channel = futures.channel.create_channel(
    name="I_HV_Battery",
    unit="A",
    frequency=1 / ch_defs.ch_def_i_hv_battery.sample_time,
    measurement_id=measurement_id,
    channel_data=futures.get_array(
        df_channels,
        ch_defs.ch_def_i_hv_battery.name
    ),
)
channels.append(channel)

try:
    trigger.write_channels(session=session, channels=channels)
except Exception as err:
    log.exception(f"Error writing channel to the AVL-DRIVE database")
    raise err
```

Standardizing names and units at write time ensures that subsequent templates can rely on a stable interface regardless of OEM-specific aliases in the raw data. Persisting frequency explicitly documents the time base of the stored series and avoids ambiguity when combining channels from different sources.

5.3. Calculations

The calculation routines implement the theoretical equations introduced in Chapter 4. Based on the mapped input channels, physical quantities such as electrical power and mechanical output are computed. The following listing illustrates how high-voltage battery power and rear motor power are derived.

Listing 4 Calculating KPIs

```
ch_u_hv_battery = futures.get_array(
    df_channels,
    ch_defs.ch_def_u_hv_battery.name
)
ch_i = futures.get_array(df_channels, ch_defs.ch_def_i_hv_battery.name)
ch_power = np.multiply(ch_i, ch_u_hv_battery) / 1000
df_channels[CH_NAME_PWR_HV_BATTERY] = ch_power

ch_m = futures.get_array(df_channels, ch_defs.ch_def_m_em_rear.name)
ch_n = futures.get_array(df_channels, ch_defs.ch_def_n_em_rear.name)
ch_power = np.multiply(ch_m, ch_n) * 2 * np.pi / 60 / 1000
df_channels[CH_NAME_PWR_EM_REAR] = ch_power

# ...
```

The first block multiplies current and voltage to obtain instantaneous battery power (see Equation 4.1). By dividing by 1000, the values are normalized to kilowatts, ensuring consistency across all KPIs. The second block applies the mechanical relation between torque and speed (see Equation 4.3) to compute the power at the rear electric machine. These results are appended to the central *df_channels* DataFrame, which acts as the intermediate storage for all derived parameters. Further calculations follow the same principle, relying on the standardized channel names defined in *Config_battery_charging.json* (see Listing 7).

5.4. Storage and Data Structure

To make results accessible in AVL Drive 5, the computed KPIs are stored in hierarchical tree structures that follow the format of the predefined *Custom_Tree.csv* (see Listing 8). This ensures that each value is associated with a unique parameter ID and can be visualized consistently in the GUI.

Listing 5 Data storage

```
if ChargingMode.IS_AC_CHARGING:
    sub_dict[OP_MODE_SUB_AC] = trigger.Sub(
        criteria={
            CRITERION_CG_LOSSES: trigger.Criterion(
                parameter={
                    PARAM_E_HV_BTY: trigger.Parameter(),
                    PARAM_E_LV_BTY: trigger.Parameter(),
                    PARAM_E_HV_DCDC: trigger.Parameter(),
                    PARAM_E_LV_DCDC: trigger.Parameter(),
                    PARAM_E_HV_PTC: trigger.Parameter(),
                    PARAM_E_HV_PTC2: trigger.Parameter(),
                    PARAM_E_HV_AC: trigger.Parameter(),
                    PARAM_E_HV_AC2: trigger.Parameter(),
                    PARAM_E_HV_OBC_OUT: trigger.Parameter(),
                    PARAM_E_HV_INVERTER_EM_FRONT: trigger.Parameter(),
                    PARAM_E_HV_INVERTER_EM_REAR: trigger.Parameter(),
                    PARAM_E_EM_FRONT: trigger.Parameter(),
                    PARAM_E_EM_REAR: trigger.Parameter(),
                },
            ),
            # ...
custom_tree = trigger.Total(main={OP_MODE_MAIN: trigger.Main(sub=sub_dict)})
```

In this snippet, an alternating-current charging mode triggers the creation of a *trigger.Sub* object, which groups relevant criteria such as charging losses. Each criterion holds a set of parameters, declared via *trigger.Parameter()*, that correspond to the energy flows defined in the user story and mapped in the configuration file. Finally, the complete structure is assembled in *custom_tree*, which represents the top-level hierarchy (*OP_MODE_MAIN*) with nested sub-modes. This object mirrors the structure in *Custom_Tree.csv* and guarantees that AVL Drive 5 can automatically assign and display the calculated values in the correct category.

5.5. Testing, Logging & Error Handling

During implementation, continuous testing and extensive logging were used to verify correct data flow and detect edge cases. The following logs represent the actual output produced by the Python script when executed in the embedded AVL Drive 5 environment.

Listing 6 Snippet of log output

```
INFO     2025-07-16 10:42:33 [main]
    Starting analysis for measurement_id=4711
DEBUG    2025-07-16 10:42:34 [mapper]
    Mapping channel "I HV Battery CAN" -> "I_HV_Battery"
DEBUG    2025-07-16 10:42:34 [mapper]
    Mapping channel "U HV Battery" -> "U_HV_Battery"
INFO     2025-07-16 10:42:35 [calc]
    Calculated PWR_HV_Battery = 92.4 kW
INFO     2025-07-16 10:42:35 [calc]
    Calculated E_HV_Battery = 4.23 kWh
WARNING  2025-07-16 10:42:36 [calc]
    Optional channel "I_HV_AC2" not available -> skipped
ERROR    2025-07-16 10:42:37 [writer]
    Error writing channel to the AVL-DRIVE database
Traceback (most recent call last):
  File "data_provider.py", line 117, in write_channels
    trigger.write_channels(session=session, channels=channels)
Exception: Database write failed (timeout)
```

These log entries serve two distinct purposes. For the developer, they are essential for debugging and validation. Every step (loading, mapping, calculation, and writing) is documented with timestamps and log levels (*INFO*, *DEBUG*, *WARNING*, *ERROR*). That makes it possible to quickly identify missing channels, inspect intermediate values, and locate failures in the pipeline.
For later end users in the field, the logs provide transparency. Warnings indicate that optional channels were not present in a specific measurement but were handled gracefully according to the fallback rules defined in the configuration (see Listing 7). Errors are logged with context so that problems in data transfer (e.g., database connection issues) can be reported to support teams with actionable details.
By integrating structured logging and explicit error handling into the script, the analysis becomes more robust and maintainable, ensuring reproducibility and easier diagnostics both during development and in operational use at AVL.

6. Results

Chapter 6 presents the results of the developed Python-based analysis system and highlights its practical outcome and advantages for end users at AVL.

6.1. Outcome

The implemented Python scripts now allow for a comprehensive and automated evaluation of both AC and DC charging processes in AVL Drive 5. All relevant parameters defined in the user story (see Chapter 3) are correctly calculated and stored. Measurement data are processed in a memory-efficient and structured manner using a mapping system based on a predefined configuration JSON file.
The scripts dynamically detect relevant channel names, standardize them via a flexible mapping structure, detect peaks at the beginning of the measurement, and calculate various parameters.
All calculated KPIs are stored directly in the measurement database and are immediately visualized in AVL Drive 5. The implementation supports optional channels (e.g., AC2, PTC2) gracefully and ensures fallback values or omissions as per requirement. The result is more complete, reliable, and granular analysis of battery charging sessions.

6.2. GUI

Figure 6.1 shows the final result as it appears within the AVL Drive 5 user interface. The left-hand side illustrates the extended result tree (see Figure A.2) under the operation mode *Battery Charging*, which now contains multiple subcategories such as *Charging duration*, *Charging power*, *Energy*, *Efficiency*, and more. Each of these categories holds the newly calculated KPIs, automatically generated by the implemented Python scripts and written into the measurement database.
On the right side, the graph view demonstrates how selected channels (e.g., HV_battery voltage and temperature) are visualized alongside cursor data. Below, the result table lists all relevant time-based and energy-based parameters, including charging durations for different SOC intervals, energy flows, and operating durations of AC, PTC, and DC components.
This graphical integration enables immediate feedback and insight for engineers working within AVL Drive 5. All values are consistently calculated, visually accessible, and aligned with the requirements defined in the original user story (see Chapter 3) — providing a powerful tool for benchmarkers and analysts.

Figure 6.1.: insight into AVL Drive 5

6.3. Benefit for AVL

The developed solution provides several tangible benefits to AVL and its engineering teams. Most notably, it significantly increases efficiency by automating the extraction and calculation of KPIs, thereby reducing the need for manual post-processing. Furthermore, the use of a standardized and flexible channel mapping approach ensures high accuracy and reproducibility of results across different vehicles and measurement setups.
The evaluation has been extended to include a much broader range of measurement signals. This allows more detailed analysis of auxiliary power consumption, component-specific losses, and overall charging efficiency, both in DC and AC charging sessions. All calculated values are stored in the AVL Drive 5 database and are directly integrated into its graphical interface, making the results instantly available to end users without the need for additional tools or workflows.
Another important benefit lies in the scalability of the solution. Thanks to the configuration-based design of the scripts, new measurement variants or vehicle models can easily be supported without requiring changes in the program code. This ensures long-term maintainability and allows benchmarkers and analysts at AVL to perform more complete, accurate, and insightful evaluations of battery charging behavior in future projects.

7. Conclusions

Chapter 7 provides a reflection on the implementation process by highlighting one unresolved technical issue and outlining possible directions for future development based on the current state of the project.

7.1. Problems

During the implementation phase, a critical issue was identified when plotting graphs that contain *not a number* (nan) values at the beginning of a measurement. This problem originates deep within the framework layer used by AVL Drive 5 and is implemented in C, outside the scope of Python. As this bug could not be resolved on the script level, it has been reported to the responsible development team for further investigation and resolution.
Additionally, while the current version of the script performs robustly under standard conditions, certain edge cases — such as missing optional measurement channels — required special handling through fallback mechanisms or default values. These scenarios highlight the importance of future work focused on more dynamic error handling and broader input validation.

7.2. Outlook

The solution presented in this work lays a solid foundation for further developments in the automated evaluation of battery charging processes within AVL Drive 5. By standardizing the naming of key measurement channels and implementing a flexible configuration system, additional scripts can now be developed based on the established structure and logic.
Thanks to the clear code organization, detailed documentation, and modular design, other developers can build on this implementation to create reusable templates for graph generation or extend the analysis with additional KPIs. Furthermore, the script architecture allows for easy adaptation to different vehicle types and test cases, which will be particularly valuable for benchmarking use cases.

7.3. Reflection

The project showed that extending existing analysis scripts is not only a technical challenge, but also requires careful coordination between requirements, data availability, and implementation. One of the main challenges was handling inconsistent or incomplete datasets, which underlined the importance of flexible configuration and fallback mechanisms. Another lesson was the need for extensive logging and testing to ensure reproducibility in an industrial environment.
From a personal perspective, the work improved my skills in structuring larger

Python projects and strengthened my ability to translate user stories into technical implementations. If I were to repeat the project, I would plan more time for early validation with different datasets to avoid edge-case problems later.
Overall, the extension provided measurable improvements. Instead of only three evaluated values, the new implementation delivers a comprehensive set of KPIs, automatically calculated and directly stored in the database. This made the analysis faster, more complete, and directly useful for AVL engineers in their daily benchmarking work.

List of Tables

List of Figures

List of Listings

A. Appendix

This appendix provides supplementary material that complements the main text. It contains excerpts of configuration files used in the implementation, additional figures illustrating results, and a list of abbreviations for quick reference.

Source

The following snippets show the configuration files that were essential for the implementation of the analysis. They define the mapping of channel names and the structure of the result tree within AVL Drive 5.

Config_battery_charging.json

This JSON file defines all relevant channel aliases for the analysis. It ensures that different naming conventions in the measurement data are standardized before calculations are performed.

Listing 7 Snippet of *Config_battery_charging.json*

```
{
  "CHI_HV_BATTERY_NAMES": [
    "I_HV_Battery",
    "I_HV_battery_CAN",
    "I_HV_Battery_CAN",
    "I_BHV_CAN",
    "I_HV_BATT_CAN",
    "I_HV_BHV"
  ],
  "CH_U_HV_BATTERY_NAMES": [
    "U_HV_Battery_CAN",
    "U_HV_Battery",
    "U_HV_battery_CAN",
    "U_BHV_CAN",
    "U_HV_BATT_CAN"
  ],
  "CH_SOC_HV_BATTERY_NAMES": [
    "STA_HV_Battery_SOC_display_CAN",
    "STA_HVB_SOC_Display_CAN",
    "STA_HV_Battery_SOC"
  ],
  // ...
  "CH_T_Ambient_NAMES": [
    "T_Ambient",
    "T_Ambient_CAN"
  ]
}
```

Custom_Tree.csv

The Custom Tree specifies the hierarchical organization of all KPIs in AVL Drive 5. Each entry maps a parameter to a unique ID and defines its unit and placement within the software's interface.

Listing 8 Snippet of *Custom_tree.csv*

```
Driveability;10;Battery Charging;91;Battery Charging DC;01;Charging losses;01;
    E_HV_Battery;81007;kWh
Driveability;10;Battery Charging;91;Battery Charging DC;01;Charging losses;01;
    E_LV_Battery;81000;kWh
Driveability;10;Battery Charging;91;Battery Charging DC;01;Charging losses;01;
    E_HV_DCDC;81001;kWh
;;;;;;;;;;
Driveability;10;Battery Charging;91;Battery Charging DC;01;Charging Duration;02;
    0-50%;81014;s
Driveability;10;Battery Charging;91;Battery Charging DC;01;Charging Duration;02;
    0-80%;81015;s
Driveability;10;Battery Charging;91;Battery Charging DC;01;Charging Duration;02;
    0-100%;81016;s
// ...
Driveability;10;Battery Charging;91;Battery Charging AC;02;Charging efficiency;12;
    DCDC_efficiency;81160;%
;;;;;;;;;;
Driveability;10;Battery Charging;91;Battery Charging AC;02;Operation mode parameters;
    89;event duration;28021;s
```

For the full code, please write an email to michaelneuhold19@gmail.com and you may get access to a private repository containing all the files. This repository will be set to public, once this thesis is available to the public (2029).

Graphs & Diagrmas

Figure A.1 highlights how peaks at the start of the measurement can look like. The programm automaticly detects these peaks.

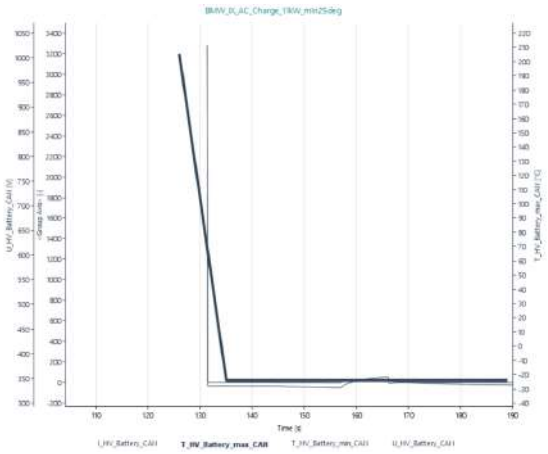

Figure A.1.: Peaks at the start

Figure A.2 shows the resulting tree structure that stores all KPIs.

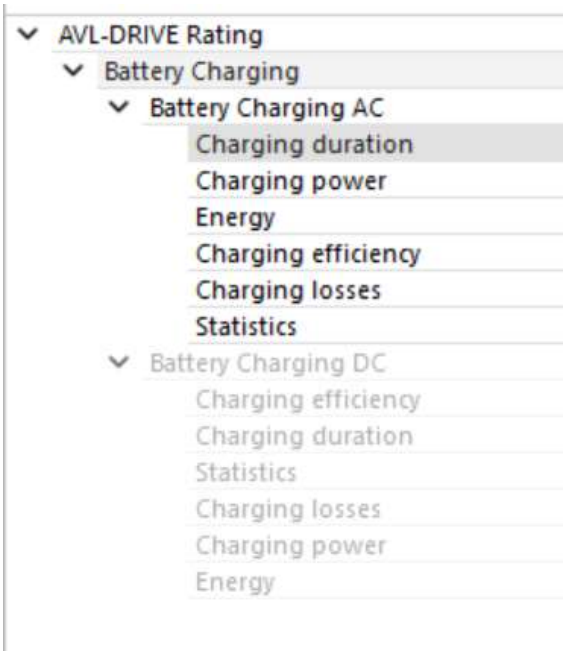

Figure A.2.: Resulting Tree

Abbreviations

Table A.1.: Abbreviations

Abbreviation	Meaning
AI	Artificial Intelligence
AC	Alternating Current
DC	Direct Current
GUI	Graphical User Interface
I/O	Input and Output
KPI	Key-Performance Indicator
SOC	State of Charge

Bibliography

[1] G. Van Rossum and F. L. Drake Jr, "Python programming language," *Python Software Foundation*, 2009, (accessed: 23.07.2025).

[2] numpy development team, "numpy.array," https://numpy.org/doc/stable/reference/generated/numpy.array.html, (accessed: 23.07.2025).

[3] ——, "numpy.ndarray," https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html, (accessed: 23.07.2025).

[4] pandas development team, "pandas.dataframe," https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html, (accessed: 27.07.2025).

[5] ——, "pandas.series," https://pandas.pydata.org/docs/reference/api/pandas.Series.html, (accessed: 27.07.2025).

[6] scipy development team, "scipy.interpolate.interp1d," https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html, (accessed: 27.07.2025).

[7] P. S. Foundation, "json — json encoder and decoder," https://docs.python.org/3/library/json.html, (accessed: 27.07.2025).

[8] International Electrotechnical Commission, "International standard iec 60050-131: International electrotechnical vocabulary – electric and magnetic circuits," https://webstore.iec.ch, 2017, (accessed: 04.09.2025).

[9] Y. Liu, H. Chen, and S. Müller, "Key performance indicators for electric vehicle charging benchmarking," 2020, (accessed: 04.09.2025).

[10] R. B. GmbH, "Bosch automotive handbook," 2021, (accessed: 04.09.2025).

[11] N. Community, "Numpy/scipy documentation style guide," https://numpydoc.readthedocs.io/en/latest/format.html, (accessed: 04.09.2025).