CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Utilizing Large Language Models for Logged Data Anlysis in Industrial Contexts

Investigating the Adaptation of Large Language Models for Logged Data Analysis

Master's thesis in Computer science and engineering

Eric Le
Himanshu Sahni

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Utilizing Large Language Models for Logged Data Anlysis in Industrial Contexts

Investigating the Adaptation of Large Language Models for Logged Data Analysis

ERIC LE
HIMANSHU SAHNI

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Utilizing Large Language Models for Logged Data Anlysis in Industrial Contexts
Investigating the Adaptation of Large Language Models for Logged Data Analysis
ERIC LE
HIMANSHU SAHNI

© ERIC LE, HIMANSHU SAHNI, 2024.

Supervisor: Morteza Haghir Chehreghani, Department of Computer Science and Engineering
Advisor: Masoom Kumar, Volvo Group AB
Examiner: Moa Johansson, Departement of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Utilizing Large Language Models for Logged Data Anlysis in Industrial Contexts
Investigating the Adaptation of Large Language Models for Logged Data Analysis
ERIC LE
HIMANSHU SAHNI
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The rapid advancements in large language models (LLMs), such as OpenAI's GPT series and Meta's Llama series, hold significant potential to revolutionize data analysis processes in industrial contexts. This thesis explores the adaptation of LLMs for logged data analysis within an industrial context, specifically at Volvo Group. The objective is to customize and integrate an LLM-based system to automate the resource-intensive data analysis process and enhance efficiency. The research involves constructing a multi-component agent system that incorporates prompt engineering techniques, specifically targeting the analysis of MF4 files, a format optimized for efficient storage of time-stamped measurement data. Our methodology includes developing a planner to break down complex analysis tasks into manageable subtasks, a selector to categorize user requests into computation and plotting tasks, and a code interpreter to generate and execute Python code for these tasks. The system's performance was evaluated using custom-designed quantitative metrics and qualitative assessments from human evaluators to ensure accuracy and efficiency. The LLM-based agent outperformed the current Python-based workflow in terms of efficiency, reducing the need for manual coding and making the analysis process substantially faster and more convenient. The model achieved an accuracy of 93.89% for computations and 90.36% for plots. The reliability of the model was confirmed through rigorous evaluations, and it showed a consistent performance with high accuracy scores and positive feedback from the human evaluators. Key findings show the model handles complex data analysis and generates diverse, tailored plots, highlighting its utility and transformative potential for scalable, effective industrial data processing workflows.

Keywords: Large Language Models (LLMs), Logged Data Analysis, Automation, Open-Source Models, Prompt Engineering, Agent-Based Analysis, MF4 File Analysis

Acknowledgements

We want to express our deepest gratitude to all those who have supported us throughout the completion of this thesis. The journey has been enriching and informative, and we could only have navigated it with the invaluable guidance and assistance of many.

Firstly, we are immensely grateful to Masoom Kumar, our supervisor at Volvo Group, for his expert guidance, encouragement, and insights throughout our research. His expertise and thorough understanding of the automotive sector, large language models, and software engineering have been instrumental in shaping our approach and methodology. His practical insights into the problem area were invaluable.

We also extend our heartfelt thanks to Morteza Haghir Chehreghani, our academic supervisor at Chalmers University of Technology. His profound knowledge and academic rigor have greatly enhanced the quality of our work. His constructive feedback and unwavering support have been crucial in refining our thesis.

Special thanks go to Moa Johansson, our examiner, for her critical insights and rigorous evaluation of our work. Her constant feedback has been vital in pushing our thesis to its highest standard.

We thank Behrooz Razaznejad, our manager at Volvo Group, for allowing us to engage in this project. His support has been fundamental to this thesis.

Furthermore, we would like to acknowledge our peers and colleagues at the Volvo Group and the Department of Computer Science and Engineering for their collaborative spirit and helpful discussions. We are also grateful to the administrative and technical staff at Volvo Group for helping us set up the proper setup to pursue this thesis.

Lastly, we thank our families and friends, whose endless encouragement and support have sustained us through this intense academic endeavor.

Eric Le, Himanshu Sahni, Gothenburg, 2024-06-04

Contents

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Background at Volvo Group . . . 2
1.1.1 Data Collection . . . 2
1.1.2 Data Preparation . . . 3
1.1.3 Data Analysis with Python . . . 3
1.1.4 Problem . . . 4
1.2 Project Aim . . . 4
1.3 Limitations . . . 5
1.4 Thesis Outline . . . 5

2 Related Work **7**
2.1 Integrating LLM for Data Analysis . . . 7
2.2 LLM Powered Agents . . . 8
2.2.1 CI Agent . . . 8
2.2.2 Planner and CI Agent . . . 9
2.3 Motivation . . . 9

3 Theory **11**
3.1 Large Language Models . . . 11
3.1.1 Terminology . . . 11
3.1.2 Generation Parameters . . . 12
3.1.2.1 Determinism-Randomness Settings . . . 12
3.1.2.2 Token Settings . . . 13
3.2 Prompt Engineering . . . 13
3.2.1 Basics of Prompting . . . 13
3.2.2 New Tasks . . . 14
3.2.2.1 Zero-Shot . . . 14
3.2.2.2 Few-Shot . . . 14
3.2.3 Reasoning and Logic Prompting . . . 14
3.2.3.1 Chain-of-Thought . . . 15
3.2.3.2 Self-Consistency . . . 15
3.3 Agents . . . 16

4 Methods **19**
4.1 Python Workflow . . . 19
4.2 Agent Construction . . . 20
4.2.1 Planner . . . 20
4.2.2 Selector . . . 21
4.2.3 Code Interpreter . . . 21
4.2.4 Choice of LLMs . . . 22
4.3 Agent Interface . . . 22
4.3.1 Volvo Framework . . . 22
4.3.2 GUI Layout . . . 23
4.4 Dataset . . . 24
4.4.1 Request Categories . . . 24
4.4.2 Sample Distribution . . . 26
4.4.3 Crafting User Requests . . . 26
4.5 Evaluation . . . 27
4.5.1 Computation Metric . . . 27
4.5.2 Figure Metric . . . 27
4.5.3 Human Evaluators . . . 27

5 Results **29**
5.1 Model Performance . . . 29
5.1.1 Computations . . . 29
5.1.2 Plots . . . 30
5.2 Case Studies . . . 33
5.2.1 Event Detection Plot . . . 34
5.2.2 Power Temperature 2D Histogram . . . 35
5.2.3 Speed and Accumulated Distance Subplot . . . 36

6 Discussion **39**
6.1 Effective Practices . . . 39
6.1.1 Multi-Agent System . . . 39
6.1.2 Prompting Techniques . . . 39
6.2 Agent Model . . . 40
6.2.1 Computation Performance . . . 40
6.2.2 Plot Performance . . . 40
6.2.3 Case Studies . . . 41
6.2.4 Limitations . . . 41
6.3 Comparison Between Workflow and Agent . . . 41
6.4 Future Works . . . 42

7 Conclusion **43**

Bibliography **45**

A Appendix - Example Code **I**
A.1 Python Code For Generating Subplot . . . I

B Appendix - Dataset **III**
B.1 Computational Requests III
B.2 Plot Requests IV

C Appendix - System prompt **VII**
C.1 Planner Prompt VII
C.2 Selector Prompt VIII
C.3 CI Computation Prompt IX
C.4 CI Plot Prompt X

D Appendix - Computation Case Studies **XIII**
D.1 Computing Max, Min and Average XIII
D.2 Computing Total Distance Traveled XIV
D.3 Computing Correlation Coefficient XIV
D.4 Calculating Total Hydrogen Consumption XV
D.5 Computing Event Detection XV
D.6 Linear Regression XVI

E Appendix - Plot Case Studies **XVII**
E.1 Line plot XVII
E.2 Scatter plot XVIII
E.3 1D Histogram plot XIX
E.4 2D histogram XX
E.5 Sub plot XXI
E.6 Heatmap plot XXIII
E.7 Event detection plot XXIV
E.8 Bar plot XXV

F Appendix - Model Parameters **XXVII**

List of Figures

1.1 Data collection process during testing. 3
1.2 Workflow for preparing and analyzing logged data. 4

2.1 Pipeline of AgentCoder. Source: Adapted from [32]. 8
2.2 Agent architecture of Taskweaver [15]. Source: Adapted from [33]. . . 9

3.1 Comparison of model sizes among popular LM and LLMs. Source: Adapted from [36]. 12
3.2 Example of a chain of thought CoT prompting. Source: Adapted from [43]. 15
3.3 Example of the self-consistency technique. Source: Adapted from [44]. 16
3.4 A conceptual framework of LLM-based agent. Source: Adapted from [11]. 16

4.1 Schematic chart of the planner. 20
4.2 Schematic chart of the selector. 21
4.3 Schematic chart of the code interpreter. 22
4.4 Overview of the Lagent framework. Source: Adapted from [46]. . . . 23
4.5 Layout of the GUI, showcasing the sidebar on the left, and the chat page on the right. 24
4.6 Samples distribution. 26

5.1 Bar plot of accuracy scores for each subcategory request in the computation dataset. The error bars represent the standard deviation, averaged over four evaluators and the samples within each subcategory. 31
5.2 Bar plot of accuracy scores for each subcategory request in the first half of the plot dataset. The error bars represent the standard deviation, averaged over four evaluators and the samples within each subcategory. 32
5.3 Bar plot of accuracy scores for each subcategory request in the second half of the plot dataset. The error bars represent the standard deviation, averaged over four evaluators and the samples within each subcategory. 33
5.4 Case study of a event detection plot request. 34
5.5 Case study of a 2D histogram request. 36
5.6 Case study of a subplot request. 37

E.1 Line plot. XVII

E.2 Scatter plot. XVIII
E.3 Histogram plot 1D. XIX
E.4 Histogram plot 2D. XX
E.5 Sub plot. XXII
E.6 Heatmap. XXIII
E.7 Event detection plot. XXIV
E.8 Bar plot. XXV

List of Tables

4.1 Example plan. 21
4.2 Computation types and examples of different computation requests. . 25
4.3 Plot types and examples of different kinds of plot requests. 25

5.1 Evaluation scores for computation requests. 30
5.2 Evaluation scores for plot requests. 32

D.1 Comparison of obtained, actual values, and delta errors for power-level calculations. XIV
D.2 Comparison of computed, actual values, and delta errors for total distance travelled. XIV
D.3 Comparison of obtained, actual values, and delta errors for the correlation coefficient. XV
D.4 Comparison of computed, actual values, and delta errors for total hydrogen consumption. XV
D.5 Comparison of the number of detected events below 40% SOC, with actual observations and delta errors. XVI
D.6 Linear regression results between PowerOutput_CellUnitA and CurrentStorageCapacity_EPUnit, showing slope, intercept, and R-squared values. XVI

F.1 Model parameters of the thinking and coding agent. XXVII

1

Introduction

The recent advancements and developments in Large Language Models (LLMs) such as OpenAI's GPT series [1, 2] Meta's Llama series [3, 4], and Google's Gemini [5] have revolutionized the field of artificial intelligence (AI). For instance, LLMs like GPT and Llama have demonstrated remarkable capabilities in understanding, interpreting, and generating human-like text, [6] making them invaluable tools across various sectors [7, 8]. Generative AI has expanded horizons in both research and practical use, particularly within the field of Software Engineering (SE) [9].

In the domain of quantitative data analysis in SE, LLMs play an important role. Traditional methods heavily rely on manual coding, which is both time-consuming and hinders automation and the effective extraction of insights [10]. However, with LLM agents, there is a notable paradigm shift towards more automated, intelligent systems [10, 11]. LLM agents offer significant advantages by automating insight extraction and interpretation from extensive data volumes, boosting efficiency in tasks like code generation, comprehension, and natural language reasoning [12] [13]. Their use enhances qualitative data interpretation depth and nuance, which was previously challenging to achieve.

In recent studies, researchers [10, 14, 15] have explored the potential of integrating LLM agents in data analysis, primarily focusing on leveraging OpenAI's GPT series [1, 2]. However, the practicality of deploying GPT-series models in commercial settings presents challenges, especially in industries where access requires a subscription to OpenAI, leading to potential scalability issues and accumulating costs associated with token usage. This limitation highlights the necessity of exploring alternatives, particularly open-source LLMs such as the Llama series [4, 16] and Mixtral [17], which are licensed for commercial use. Despite the acknowledged potential of open-source language models [18], research on their application in this area of data analysis remains relatively scarce.

Additionally, the diverse data analysis workflows across different industries further complicate the integration of LLMs into domain-specific contexts, posing an ongoing research question. For instance, data formats like MF4 [19] have received limited attention in existing literature. Existing third-party tools for file analysis, including Open AI Interpreter [20], are limited in their capabilities. Specifically, they fail to support MF4 files, a format optimized for efficient storage of time-stamped measurement data [19]. Even when available, uploading data to third-party tools raises privacy concerns, particularly when dealing with sensitive information. This

motivates the development of LLM-powered applications adapted for industry-specific logged data analysis.

In this master's thesis, we explore a practical scenario focusing on adapting LLMs for logged data analysis within the Volvo Group. Due to the industry context and lack of attention to the application of open-source models, our emphasis lies on open-source LLMs licensed for commercial use, specifically investigating the Llama series [4, 16] and Mixtral-8x7b [17]. The primary focus will be on analyzing files in MF4 formats. Our methodology involves prompt engineering [21] and developing agents [11] capable of reasoning and code synthesis. We evaluate the agents' performance quantitatively using custom-designed metrics, which are then assessed by human evaluators to determine the accuracy of the computed data and the generated plots. The proposed agent aims to extract pertinent information and generate visualizations from logged data using natural language.

1.1 Background at Volvo Group

Volvo Group AB is a Swedish multinational manufacturing company founded in 1927 and headquartered in Gothenburg [22]. It is widely recognized for its manufacturing of commercial vehicles, encompassing trucks, buses, and construction equipment [22]. Within the company, the Powertrain Strategic Development (PSD) group is actively exploring future powertrain solutions, particularly assessing the viability of Hydrogen (H2) solutions within the context of H2 ICE (Internal Combustion Engine) and H2 Fuel cell propulsion solutions. The Fuel cell propulsion group, comprised of software (SW) developers, testing, and simulation engineers, is responsible for developing and evaluating the fuel cell system with the supplier.

1.1.1 Data Collection

The trucks undergo extensive testing to validate software functions, identify faults, and develop innovative fuel cell control concepts. The testing phase involves collecting extensive data through various trials, including hot, high-altitude, and cold climate assessments. Each test trip can yield up to 240 hours of logged vehicle data, covering various aspects of vehicle performance and software functions.

The data collection workflow is illustrated in Figure 1.1. Trucks are equipped with CAN (Controller Area Network) bus interfaces, which serve as central devices for connecting and logging data from electronic control units (ECUs) [23]. In the context of automobiles, ECUs are used for regulating various functions such as engine control, braking control, and other critical vehicle operations [23]. During each test trip, CAN bus interfaces record over 8,000 signals from various ECUs, capturing data related to vehicle speed, fuel consumption, hydrogen charge and more.

The data output from CAN bus interfaces is commonly known as CAN bus frames or messages. These frames encapsulate one or more raw signal values representing various physical parameters such as temperature, speed, power, etc. A software tool like CANalyzer [24] is utilized to extract meaningful information from these CAN

bus frames. CANalyzer is designed to analyse network communication, enabling users to intercept and decode data transmitted through CAN frames.

Processing with CANalyzer produces BLF files (binary logged format) [25]. BLF files are notable for their efficiency in file size and read/write performance, making them ideal for handling large volumes of data generated by CAN bus interfaces. Moreover, each entry within a BLF file is timestamped, providing precise chronological information about events occurring on the CAN bus network. These BLF files are typically stored on a network disk or other storage mediums for further analysis and review.

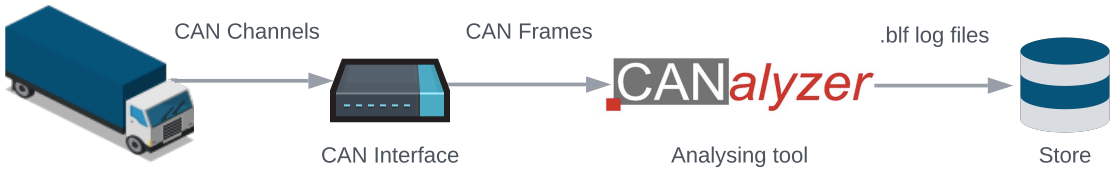


Figure 1.1: Data collection process during testing.

1.1.2 Data Preparation

After collecting logged data from a test trip, the next step is to analyze and extract insights from the trip's data. This process follows a structured workflow, as illustrated in Figure 1.2.

The workflow starts by converting Binary Logged Format (BLF) files, which typically totalling around 200 GB per trip, into Measurement Data Format (MDF). The latest MDF version, MF4, optimizes high-speed data capture and efficiently manages large data volumes. It supports lossless data compression, reducing file size without compromising data integrity, making it valuable for handling substantial data with minimized storage requirements. Once the BLF files are converted to MF4 format, the next step in the workflow involves preprocessing and data cleaning. This step aims to address any inconsistencies, noise, or missing signals in the data, ensuring that the data used for subsequent analysis is reliable and accurate.

Finally, all the processed MF4 files are merged and synchronized to a common timestamp. The synchronized data is then consolidated into a final MF4 file, which is ready for in-depth analysis and extraction of insights regarding the truck's performance, system behaviour, anomalies, and other relevant metrics.

1.1.3 Data Analysis with Python

The final MF4 file contains a comprehensive list of relevant signals. Each signal is linked with corresponding values and timestamps. The analysis phase is primarily centred on a Python-based workflow designed to calculate quantitative measures and produce plots. The computations encompass standard metrics such as maximum, minimum, average, median, and integration, alongside statistical measures like correlation coefficients and linear regression. These metrics provide a comprehensive

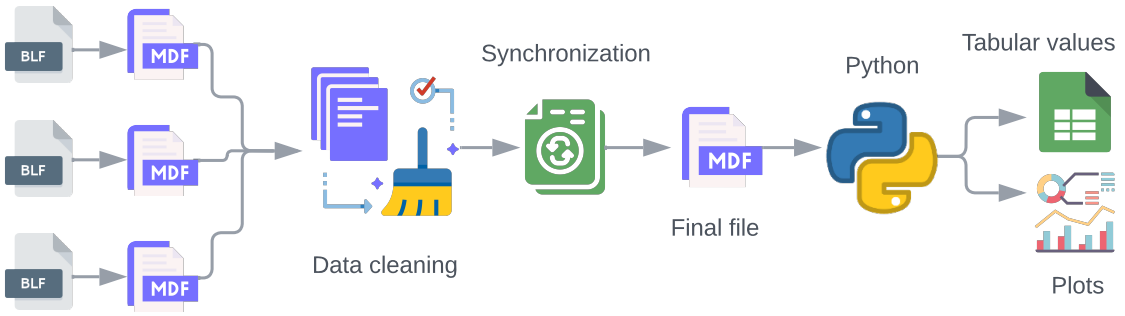


Figure 1.2: Workflow for preparing and analyzing logged data.

overview of the performance. Plots, such as line plots, scatter plots, histograms, heatmaps, and bar plots, complement this by offering insights into data distribution, aiding in anomaly detection, and enabling a deeper comprehension of system behavior.

1.1.4 Problem

The challenge currently lies in analyzing the final MF4 files generated during testing efficiently and comprehensively. While the current Python-based workflow has successfully computed quantitative measures and generated plots, it heavily relies on manual coding. This approach demands resource-intensive scripting for every potential scenario and requires frequent adjustments due to the dynamic nature of the analysis, which offers no universal solution.

Another drawback of the present Python workflow concerns the generation of plots. Since analysts often have unique visual style preferences, accommodating diverse needs through manual scripting is challenging. Moreover, requiring programming expertise hinders several team members from participating in the analysis process, limiting crucial data insights to just a few individuals. Such limitations pose challenges for collaboration efforts as well.

There is growing interest in integrating LLMs into Volvo's data analysis infrastructure to address these challenges. Adopting an LLM-driven tool for logging data analysis would provide a scalable and flexible solution, automating the analysis process and reducing the resources required for manual coding. Such a tool has the ability to enhance the current workflow through its natural language capabilities.

1.2 Project Aim

The primary objective of this project is to explore effective methods, such as prompt engineering and integrating agents, for customizing open-source LLMs and conducting data analysis on the logged data. Additionally, we aim to assess the performance of the LLM-based agent in comparison to the current methodology for logged data analysis, focusing on factors such as time efficiency and accuracy. Finally, we will evaluate the model's reliability to determine its potential as a replacement for the existing workflow.

To concretize the objectives, we frame a comprehensive research question:

- *How can LLMs be adapted for logged data analysis?*

This overarching research question can be subdivided into three specific inquiries, aligning with the project's structured approach:

1. *What are the most effective methods for customizing LLMs to conduct logged vehicle data analysis?*
2. *How does the performance of the LLM-based agent compare to the current methodology for logged data analysis in terms of time and accuracy?*
3. *What is the reliability of the LLM-based agent in comparison to the existing workflow for logged data analysis?*

1.3 Limitations

While this project aims to explore effective methods to adapt open-source LLMs for logged vehicle data analysis applications, it is important to acknowledge certain limitations:

1. **Scope:** The evaluation of the LLM-based agent is limited to examining recorded vehicle data only and may not cover inquiries outside of this specific area. As a result, the applicability of the results to other domains could be restricted.
2. **Domain-specific knowledge:** The knowledge base of the LLM will be adapted to the specific jargon and data formats commonly used in Volvo's logged vehicle data. Consequently, its expertise may be limited to the context of this particular domain.
3. **Performance evaluation:** The project aims to evaluate the performance of the LLM-based agent compared to the current methodology for analyzing logged data. However, it's important to acknowledge that the evaluation might not encompass all facets of data analysis. Certain subtleties of human analysis may not be completely replicated by the LLM-based agent.

1.4 Thesis Outline

In this section, we outline the project's structure. Chapter 2 covers previous approaches where researchers adapted LLMs for data analysis. Chapter 3 introduces the theory of LLMs, prompt engineering, and agents. Chapter 4 presents the methodology used in this study. Subsequently, Chapter 5 presents the experiment's findings, followed by a discussion of the observed outcomes and insights into future work and improvements in Chapter 6. Finally, Chapter 7 concludes the project by summarizing findings and implications of the study.

2

Related Work

The related work chapter of this thesis investigates how previous studies have utilized LLMs for data analysis. One common approach involves prompt engineering [26], where specific prompts or queries are designed to effectively utilize the inherent capabilities of LLMs. In contrast, a more comprehensive strategy integrates intelligent agents [27] into the analysis system. These agents mimic human reasoning and action, leading to a more human-like approach to data analysis. Finally, we discuss how our work differs from previous research and the contributions it brings to this area of study.

2.1 Integrating LLM for Data Analysis

Integrating Large Language Models (LLMs) for data analysis has recently gained significant attention. In a study titled [14], researchers explored the question, "Is GPT-4 a good data analyst?" The study empowered GPT-4 as a data analyst, enabling it to perform end-to-end data analysis using databases from various domains. This was achieved through meticulous, prompt design for GPT-4 to conduct experiments and generate key summaries and figures.

The experimental findings indicated that GPT-4 can generally achieve performance levels comparable to humans in data analysis tasks. However, it was acknowledged that there is still room for improvement to meet human standards, as the model encountered challenges related to hallucination problems, as discussed in the research article [28] and also noted in the GPT-4 technical report [2].

Despite these challenges, the study emphasized the significant time savings achieved by GPT-4 compared to a human analyst when performing the same tasks, as highlighted in [14]. This underscores the potential of LLMs like GPT-4 in automating and streamlining data analysis processes. However, the study also identified several limitations, such as difficulty in handling ambiguous queries, limited contextual understanding, etc. To address these issues, ongoing improvements are focused on enhancing the model's contextual understanding, refining its ability to interpret complex data accurately, and developing mechanisms to handle ambiguous queries more effectively.

2.2 LLM Powered Agents

The traditional approach involves giving instructions to LLMs to perform specific tasks. However, a recent progressive strategy proposed in current research suggests integrating intelligent agents powered by LLMs into systems, as outlined in [29]. This strategy aims to automate and enhance system complexity by introducing agents capable of reasoning, executing appropriate actions and utilizing provided tools as needed [27]. In this context, an agent acts as the central intelligence and is responsible for effectively carrying out data analysis tasks. These agents systematically break down a query into sub-queries and assign distinct tasks to specialized agents, as described in [30].

2.2.1 CI Agent

Typical data analysis tasks involve a complex interplay between data files and code execution to extract valuable insights and solutions. This demands harmonising natural language and code, a capability well-suited to LLMs [11]. Their unique ability to understand and generate both natural language and code positions them as ideal solutions for tackling such challenges [31].

In their recent work [32], the authors present a comprehensive approach that integrates test case generation and execution into the code creation process, aiming for a balanced and efficient development workflow. They introduce a novel solution called Multiagent-Code Generation (AgentCoder) [32], which employs a multi-agent framework comprising three specialized agents: the programmer agent, the test designer agent, and the test executor agent. The framework of AgentCoder and its pipeline are shown in Figure 2.1.

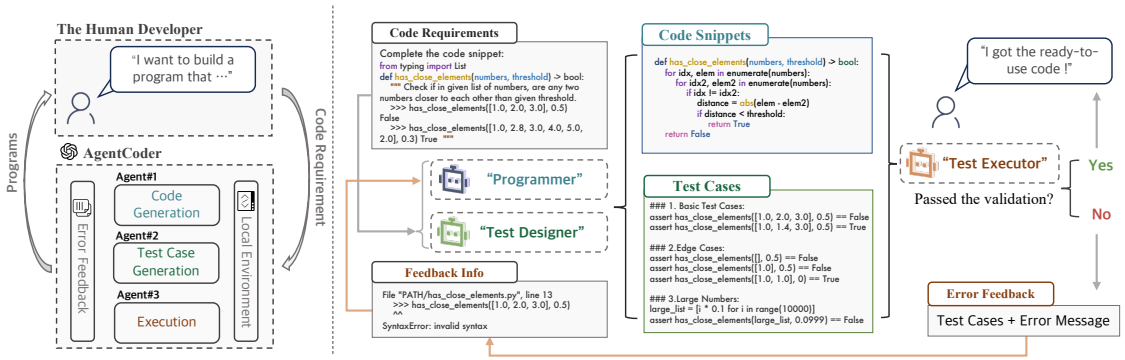


Figure 2.1: Pipeline of AgentCoder. Source: Adapted from [32].

During the coding process, the programmer agent handles code generation and refinement, taking into account feedback from the test executor agent. Simultaneously, the test designer agent focuses on generating relevant test cases for the code created by the programmer agent. As this collaborative system operates, the test executor agent executes the code with the generated test cases and provides feedback to the programmer agent. This coordinated approach leverages the strengths of each

specialized agent, ensuring a more effective code generation process and improving overall code quality and test coverage [32].

2.2.2 Planner and CI Agent

Expanding upon this approach for data analysis, Qiao et al. [15], discussed and implemented introduced the integration of a planner and code interpreter. The agent architecture is illustrated in Figure 2.2.

The planner acts as the central intelligence of the agent, breaking down user requests into smaller sub-tasks to formulate an initial plan. It further refines this plan based on specific requirements, generating the final plan before delegating each sub-task to the code interpreter for execution. The primary function of the code interpreter is to generate code using a variety of plugins and examples, which can encompass diverse forms like a web API call, a software module, or a customized algorithm [15].

Following code generation, the code undergoes validation and execution by the interpreter. The output includes the code itself, along with logs capturing stdout/stderr from the program and various outputs and images. Depending on the observed results and their alignment with the code interpreter's output, the planner may need to update its plan accordingly [15].

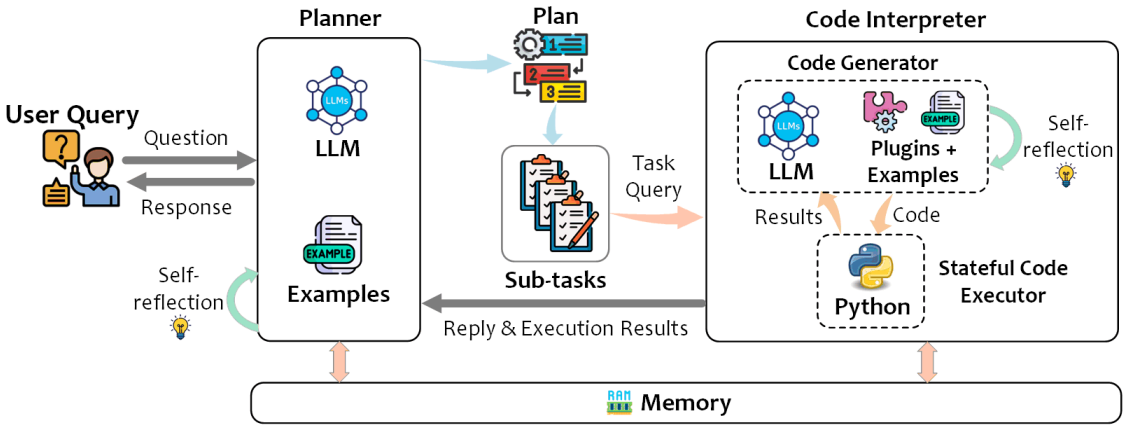


Figure 2.2: Agent architecture of Taskweaver [15]. Source: Adapted from [33].

2.3 Motivation

Previous research in the field has mainly concentrated on the extensive OpenAI GPT series [1, 2], known for its remarkable reasoning and code generation prowess. While its large size and adaptability make it appealing for various applications, such as data analysis, its closed-source nature presents challenges in industrial contexts, particularly due to paid access requirements.

In contrast to prior studies, our focus lies on the potential of smaller open-source alternatives for data analysis, which has received less attention in current research. By investigating these alternatives, we aim to bridge the research gap regarding

their suitability for industrial applications. Furthermore, there is a noticeable lack of attention in the current research landscape towards non-conventional file formats like MF4 files, which are widely used in specific industries. Hence, there is a pressing need to explore how these alternatives, particularly LLMs, can be adapted to analyze MF4 files. This represents an active research domain with substantial opportunities for impactful contributions.

3

Theory

The theory chapter of this thesis establishes the theoretical foundation supporting the research by summarizing fundamental concepts and principles that guide the study. It commences with an overview of LLMs. It further investigates prompt engineering techniques crucial for enhancing LLM performance. Finally, it discusses the role of agents within LLMs, emphasizing their impact on model functionality and decision-making. This foundational analysis sets the stage for the research and subsequent chapters.

3.1 Large Language Models

This section provides essential background information LLMs. LLMs represent advanced AI models that undergo extensive unsupervised pre-training on vast text datasets to comprehend, generate, and manipulate human-like text [34]. They can write articles, answer questions, summarize texts, translate languages, and even create content like poetry or code [34]. Examples of LLMs include OpenAI's GPT-series [1, 2], Meta's Llama families [3, 4], and Google's Gemini family [5].

3.1.1 Terminology

Model Size: When discussing LLMs, the term "model size" typically refers to the number of parameters within the LLM [35]. A larger model size means the model has more parameters, which allows it to capture more intricate patterns in the data and understand more complex relationships. However, it's important to note that larger models also require more computational resources for both training and deployment.

A comparison chart illustrating the model size of various common Language Models (LM) and LLMs is presented in Figure 3.1.

Context Window: The context window of an LLM refers to the maximum amount of text the model can consider at once when generating response elements [35].

Base model / foundational model: A pre-trained language model that serves as the starting point for building more specific models for downstream tasks [35].

Fine-tuning: Fine-tuning, a form of transfer learning, involves continuing the training of a pre-trained model on a specific dataset to optimize its performance for a particular task or domain [35].

Hallucinations: When an LLM generates text that is nonsensical or unfaithful to the provided source content, we often say that content is an LLM hallucination [28].

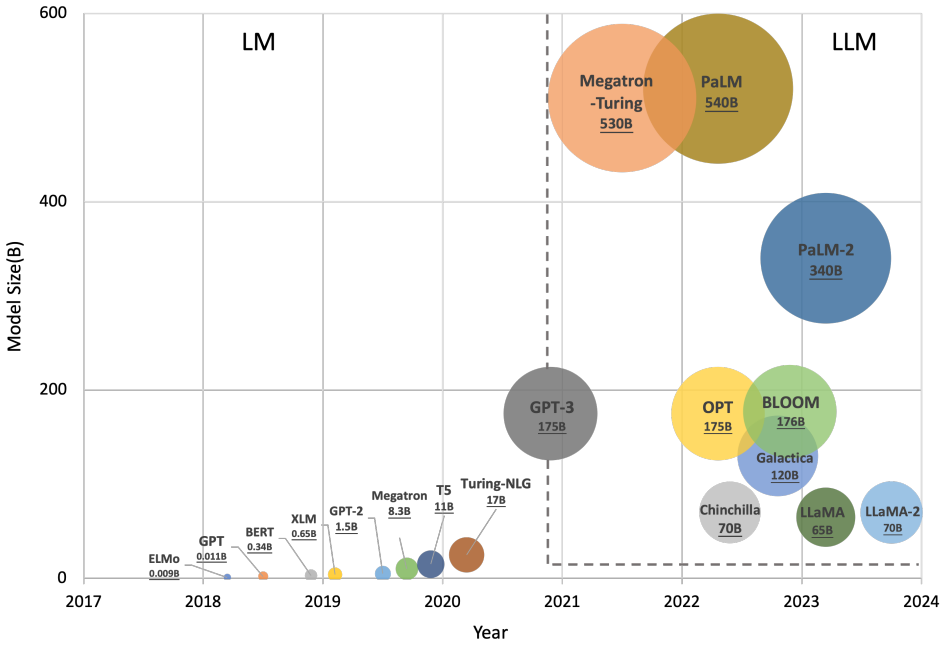


Figure 3.1: Comparison of model sizes among popular LM and LLMs. Source: Adapted from [36].

3.1.2 Generation Parameters

LLMs are designed to generate output based on probabilities, selecting the most likely next token or word based on their training, input, and settings. These models offer various settings that influence text generation, and adjusting them can improve response reliability and quality. Finding the best configurations often involves experimentation for specific applications. The key parameters for manipulating LLM outputs fall into two main groups: determinism-randomness settings and token settings.

3.1.2.1 Determinism-Randomness Settings

These parameters govern the balance between determinism and randomness in the output. They include temperature, top-p, and top-k, each playing a distinct role in shaping the generated content.

Temperature: The temperature parameter in LLMs ranges from 0 to 1, affecting response randomness. Higher values increase diversity and creativity but may reduce coherence, while lower values yield conservative and predictable results.

Zero temperature generates repetitive outputs, while higher temperatures enhance creativity but risk nonsensical outputs [35].

Top-P: Top-P sampling, or nucleus sampling, is a method that controls output randomness by choosing tokens until their cumulative probability exceeds a set threshold. Lower values result in precise responses, while higher values introduce more randomness for a diverse output [35].

Top-K: Top-K sampling is a technique used in language models to limit the number of tokens considered during the sampling process. It selects the top K most likely tokens based on their probabilities and samples from this restricted set [35].

3.1.2.2 Token Settings

These parameters are associated with the output tokens of the Language Model. They encompass maximum length, stop sequences, frequency penalty, and presence penalty.

Maximum length: The maximum length setting in language models determines the total number of tokens in the output, helping users control response length and avoid overly long or irrelevant outputs [35].

Stop Sequences: Stop sequences in language models signal when to end output generation, allowing control over content length and structure [35].

Frequency Penalty: Frequency penalty reduces repetition in the generated text by penalizing tokens based on their frequency of occurrence. The more a token is repeated in the text, the less likely it is for the model to use it again [35].

Presence Penalty: The presence penalty is similar to the frequency penalty but flatly penalizes tokens based on whether they have occurred or not, instead of proportionally [35].

3.2 Prompt Engineering

Prompt engineering has become a crucial method for enhancing the capabilities of large language models (LLMs) by providing task-specific instructions, called prompts, to guide model output without requiring extensive retraining or parameter adjustments [37]. In this section, we explore both foundational prompting techniques and more advanced strategies.

3.2.1 Basics of Prompting

By strategically incorporating key elements into a prompt, one can construct a foundational structure that enables LLMs to generate high-quality responses. A prompt in LLMs guides the model's output and can range from simple queries to complex tasks [38]. It typically includes instructions, questions, input data, and examples. In this section, we delve into some of the fundamental components of a well-crafted prompt.

Assigning Roles: Assigning roles guides text generation by providing specific contexts or perspectives to the LLM [39]. For example, using a role prompt like "You are a travel guide. Describe the top attractions in Paris" directs the model to adopt a travel guide's perspective, improving response accuracy for related content.

Be clear and precise: Creating clear and specific prompts helps the model produce desired outputs accurately [40]. Vague prompts can lead to generic responses, while precise prompts align better with unique requirements, guiding the model towards the correct response. For instance, a detailed prompt like "I want to understand the cutting edge of technology, specifically related to artificial intelligence and machine learning..." yields more relevant content compared to a broad request.

3.2.2 New Tasks

LLMs have undergone extensive training on vast corpora and possess pre-existing knowledge. This capability enables them to generate predictions when presented with new tasks, allowing them to perform tasks they haven't encountered before [41]. Two techniques that enable the capability to address new tasks without extensive training are zero-shot and few-shot prompting methods.

3.2.2.1 Zero-Shot

Zero-shot prompting is a prompting technique to guide LLMs towards new tasks [1]. LLMs can answer queries they haven't encountered before, such as generating summaries without task-specific training. For instance, a prompt like "Summarize a recent news article concerning renewable energy advancements in Asia" directs the model to produce a summary using its broad knowledge base, even though it has not been trained particularly in text summarization.

3.2.2.2 Few-Shot

Few-shot prompting involves providing models with a limited number of input-output examples to help them understand a given task, unlike zero-shot prompting, which provides no examples [42]. Even providing a few high-quality examples has been shown to enhance model performance on complex tasks compared to having no examples. However, few-shot prompting requires additional tokens to include these examples, which can be a challenge for longer text inputs.

3.2.3 Reasoning and Logic Prompting

LLMs are capable of reasoning even in zero-shot scenarios and can be stimulated to produce responses related to logical problems, task planning, critical thinking, and other tasks requiring reasoning abilities. The process of generating reasons is achievable only through the utilization of various prompting styles such as various prompting styles such as Chain-of-Thought (CoT) [43] prompting and Self-Consistency prompting [44].

3.2.3.1 Chain-of-Thought

LLMs often struggle when faced with complex reasoning tasks, which limits their potential effectiveness. To address this challenge, Chain-of-Thought (CoT) prompting was introduced as a method to guide LLMs through coherent and step-by-step reasoning processes [43]. CoT prompting involves presenting demonstrations that combine reasoning information with inputs and outputs, helping the model generate outcomes through systematic reasoning steps. Unlike few-shot prompting, which provides the model with examples of inputs and outputs to help it understand the task, CoT prompting includes detailed reasoning steps within the examples. This structured approach helps the model improve its reasoning by breaking down complex tasks into manageable steps, enhancing its ability to tackle tasks with structured logic. An example demonstrating chain of thought prompting is shown in Figure 3.2.

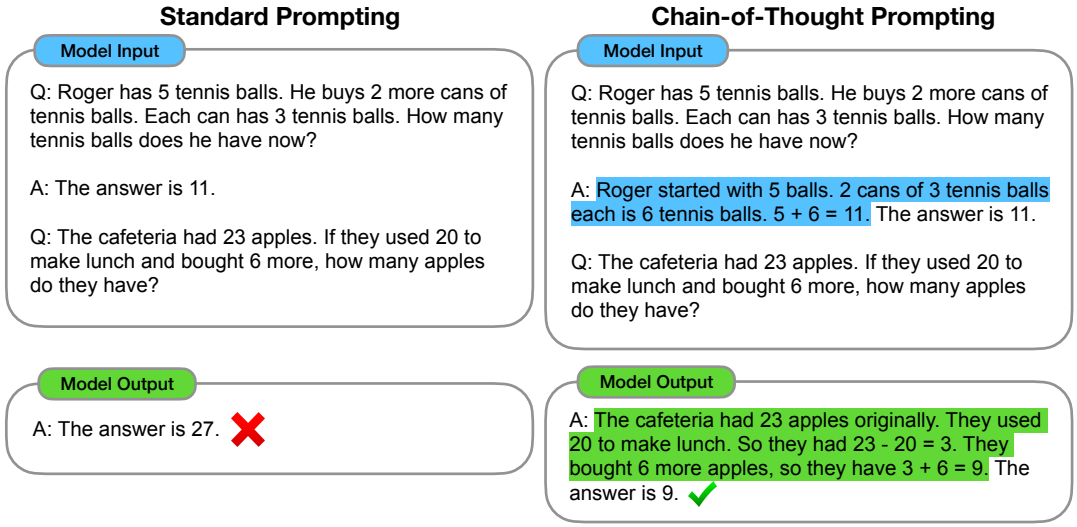


Figure 3.2: Example of a chain of thought CoT prompting. Source: Adapted from [43].

3.2.3.2 Self-Consistency

Self-consistency prompting improves CoT performance by generating diverse responses and selecting the most common answer from them [44]. It operates by generating various reasoning chains using the language model's decoder and then evaluating these chains to identify the most consistent final answer. This strategy is based on the idea that problems needing deep analysis often have multiple valid ways to solve them, leading to a more effective solution. For a practical illustration of the self-consistency technique, see to Figure 3.3.

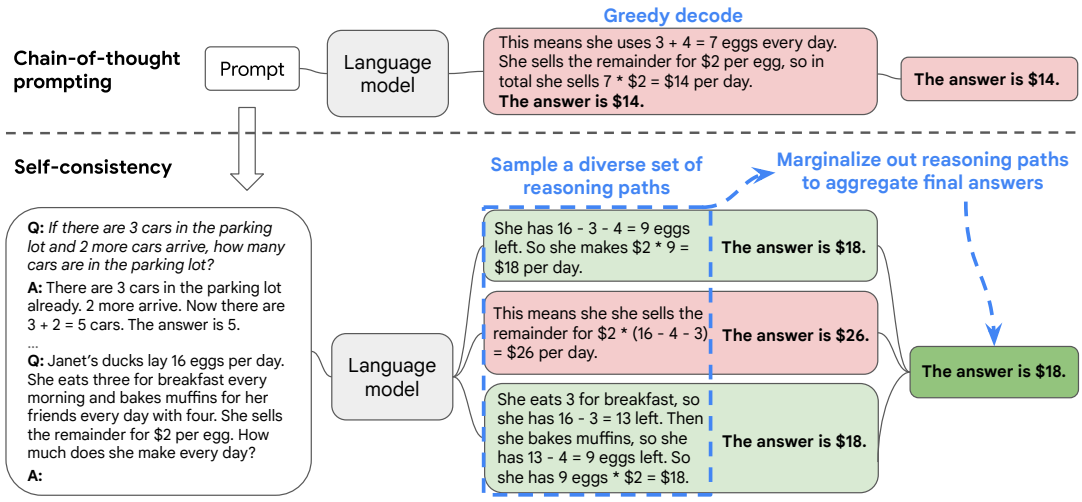


Figure 3.3: Example of the self-consistency technique. Source: Adapted from [44].

3.3 Agents

In AI, an agent is an artificial entity that perceives its environment, makes decisions, and takes actions [11]. Similar to the human brain's role, an AI agent's core is responsible for storing memories, information, and knowledge. It also handles tasks such as information processing, decision-making, reasoning, and planning. LLMs have emerged as primary components serving as the brain controller for these agents [11]. A conceptual framework of LLM-based agent with three components: brain, perception, and action, see Figure 3.4.

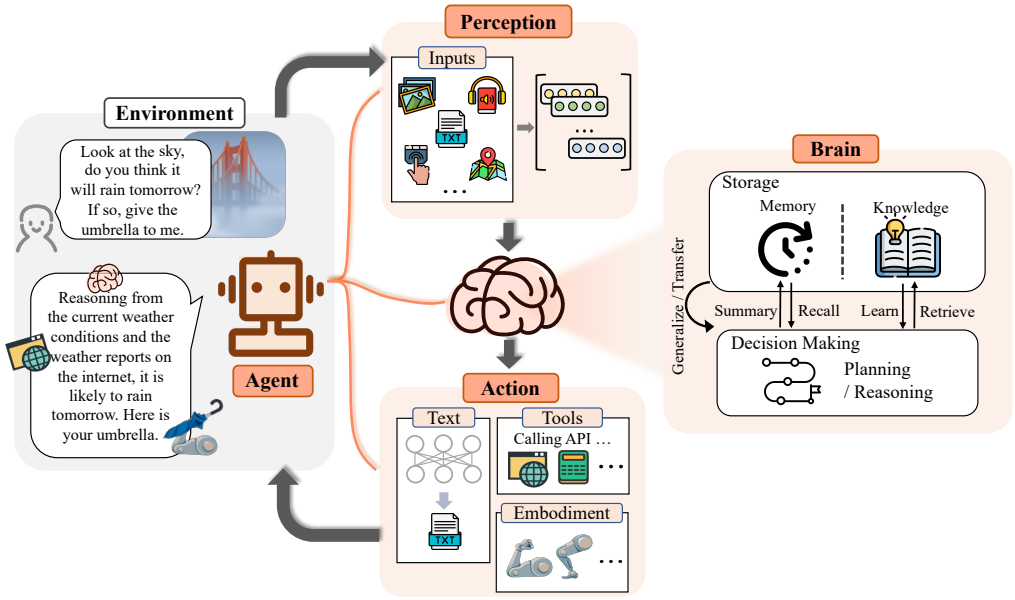


Figure 3.4: A conceptual framework of LLM-based agent. Source: Adapted from [11].

LLMs serve as brain controllers with key attributes like autonomy and proactiveness

reformulation and task decomposition [11]. They demonstrate autonomy by generating human-like text, engaging in conversations, and performing tasks without detailed instructions. Additionally, LLMs exhibit proactiveness, initiating goal-oriented actions and showcasing reasoning and strategic planning capabilities. Prompting LLMs with phrases like "let's think step by step" can stimulate their reasoning skills [43]. These models also show emergent planning abilities, including goal reformulation and task decomposition [11].

LLM-based agents exhibit limitations in certain areas, and the integration of tools can augment their capabilities and broaden their action space. Employing tools allows these agents to efficiently tackle complex tasks by systematically breaking them down into subtasks and applying appropriate tools to each. This results in more effective problem-solving with higher-quality outcomes. Specialized tools enable LLMs to enhance their expertise, adapt domain knowledge, and be more suitable for domain-specific needs [11].

Agents can learn to use tools effectively by leveraging the robust zero-shot and few-shot learning capabilities of LLMs [26]. Zero-shot prompts describe tool functionalities and parameters, while few-shot prompts provide demonstrations of specific tool usage scenarios and methods. This learning approach enhances agents' adaptability and proficiency in tool utilization, contributing to their overall problem-solving capabilities [11].

4

Methods

The methods chapter of this thesis outlines the methodology used for this project. It begins by exploring the current Python analysis workflow and explaining how the agent was constructed to replicate it. Next, it covers the Volvo framework and agent interface, detailing how the agent was built. Finally, it describes the process of crafting the dataset and customizing evaluation metrics for the experiment.

4.1 Python Workflow

The agent's primary goal is to reproduce the Python analysis on the final MF4 file. To better understand the analysis process, let's review the typical immediate steps involved:

1. **Load MF4 File:** Load the MF4 file for analysis into an MDF object. This object will contain all the signals along with their values and properties.
2. **Extract Signals:** Extract specific signals from the MDF object. Each signal contains its name, physical values, timestamps, and units. Examples of signals include fuel cell power, vehicle speed, and temperature.
3. **Implement and Apply Filter Conditions:** Analysis often focuses on signals within a specific range. For instance, it may be necessary to obtain values of the fuel cell power when the vehicle speed is between 10 and 70 km/h. To achieve this, construct a filter condition using the signal, logical operators, and values (e.g., `speed > 10 & speed < 70`). Apply this filter to isolate and extract the relevant portions of the signals that match these conditions.
4. **Recompute Time:** Following signal filtering, recompute the time signal to generate virtual timestamps. This step is essential because the original timestamps may become unevenly spaced after applying filter conditions, which can lead to inaccurate results.
5. **Perform Computations:** Conduct computations on the filtered signals using `numpy` and `scipy`. This could involve tasks like computing statistics, integrating data, and detecting events when the signal crosses a specific threshold.
6. **Plot Results:** Visualize signal distributions using functions from the `matplotlib` library. This can involve creating various plot types such as line plots, scatter plots, histograms, heat maps, and bar plots.

The analysis process consists of six steps, with steps 1-4 remaining consistent across all analyses. In contrast, steps 5 and 6 are adaptive, varying depending on the specific analysis requirements. The agent's task is to faithfully reproduce all six steps. For illustration, an example of the source code is provided in Appendix A.1.

4.2 Agent Construction

This section presents the overarching architecture of our MF4 agent, which draws inspiration from TaskWeaver [15] with a slight modification. Our architecture is composed of three core components: a planner, a selector, and a code interpreter (CI). Each of these components will be elaborated upon in subsequent sections.

4.2.1 Planner

The planner acts as the central processing unit of the agent architecture. Its primary function is to generate a comprehensive plan based on the user's request, including signal details such as signal names, units, and filtering conditions. This plan breaks down the user query into smaller subtasks for analysis, serving as a roadmap for the code interpreter (CI). The CI then executes the necessary steps based on this plan to generate the code accordingly.

The system prompt provides comprehensive instructions and guidelines to ensure the planner can generate relevant and actionable plans; see Appendix C.1. This uses the Chain-of-Thought (CoT) approach described in Section 3.2.3.1. A schematic chart detailing the Planner is illustrated in Figure 4.1.

We present a sample plan to illustrate its potential appearance in the following example 4.1.

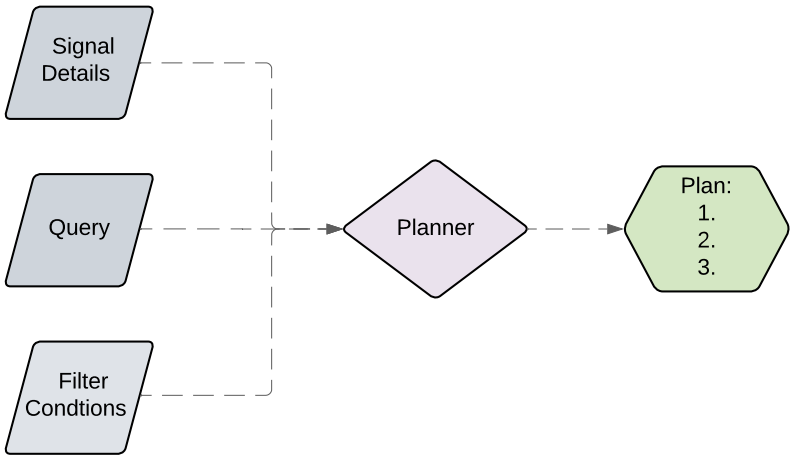


Figure 4.1: Schematic chart of the planner.

User request: Compute max and median of the signal.
Plan
1. Extract the signal.
2. Apply filter condition on the signal.
3. Compute max and mean of the filtered signal.

Table 4.1: Example plan.

4.2.2 Selector

The selector determines the code interpreter's role. User queries can be broadly categorized into two types: computation and plotting. If a user query involves only computation, the CI is designated the computation role. On the other hand, if the query includes plotting requests, the Code Interpreter is assigned the plot role. In cases where both computation and plot requests are present, the plot role is assigned.

The system prompt for the selector is found in Appendix C.2. To aid the selector in categorizing the user query, keywords are provided within each category for identification in the system prompt. The self-consistency method described in Section 3.2.3.2 is used to ensure consistent role assignment. A schematic chart detailing the selector is illustrated in Figure 4.2.

The motivation behind having a selector assign roles to the CI is due to the different requirements of the roles to be fulfilled. The plotting role requires more customized configurations. The plot role can do everything the computation does, not vice versa.

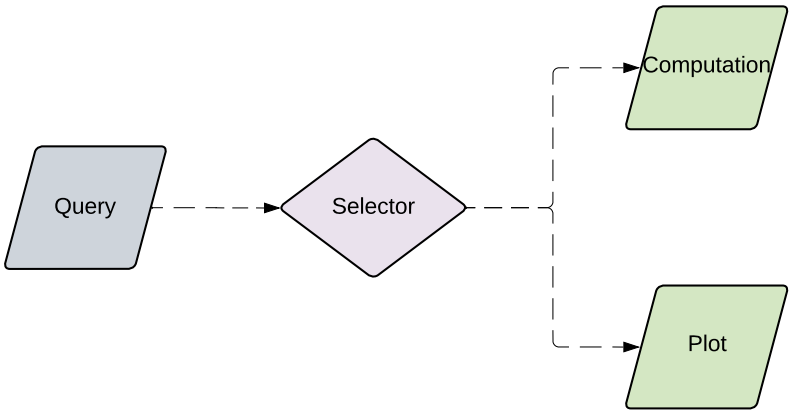


Figure 4.2: Schematic chart of the selector.

4.2.3 Code Interpreter

The primary function of the code interpreter (CI) is to generate and execute code based on the provided plan and selected role, producing output in the form of numerical values or plots. To ensure that the CI produces correct and runnable code relevant for MF4 analysis, described in Section 4.1, comprehensive instructions and

guidelines are provided in the system prompt. For example, few-shot examples as described in Section 3.2.2.2 are provided in the system prompt to help the agent use functions relevant for the request. The system prompts for the computation and plot are provided in Appendix C.3 and Appendix C.4, respectively. A schematic chart detailing the CI is illustrated in Figure 4.3.

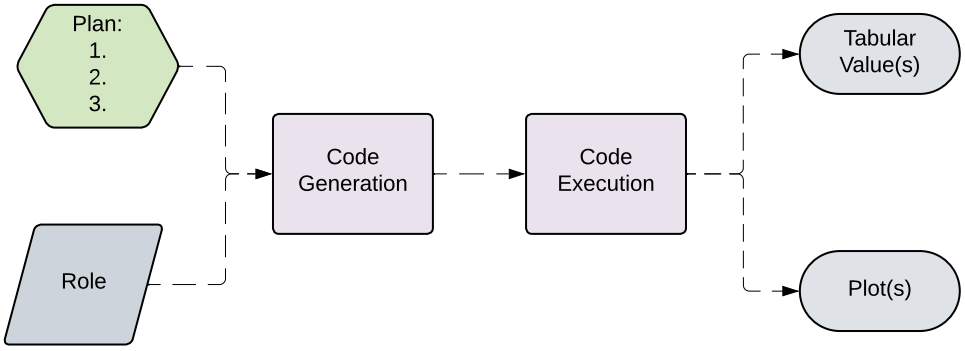


Figure 4.3: Schematic chart of the code interpreter.

4.2.4 Choice of LLMs

Open-source models like Code-Llama-2 series [16], and Mixtral 8x7b [17] demonstrate competitive performance against closed-source GPT-3.5-turbo benchmarks despite having fewer parameters [18]. Their accessibility and licenses, allowing free usage, modification, and distribution, make them preferable for this thesis. Open-source models also provide greater customization options than closed-source alternatives, often constrained by licensing agreements. Additionally, one of the important concerns in enterprise environments is data privacy when choosing closed-source models, as there are uncertainties about how the data might be used down the line. However, we expect the situation regarding data privacy and the performance of open-source models to improve soon.

Our approach selects open-source LLMs relevant to our task needs, covering language comprehension, generation, and code synthesis. Specifically, we utilize Mixtral-8x7b for language tasks [17] and the Code-Llama-34b for code synthesis [16]. Mixtral-8x7b fulfils reasoning functions in planner and selector roles, while the Code-Llama-34b generates code as the interpreter.

4.3 Agent Interface

In this section, we explore the framework for building the agent and the construction of our agent interface.

4.3.1 Volvo Framework

Running LLMs locally on a personal computer can be slow and resource-intensive, especially when dealing with models that have billions of parameters. An alternative

approach is quantization, which involves converting the model's weights from higher to lower precision [45]. However, this can compromise accuracy and results, making quantization less than ideal for fully realizing the potential of LLMs. To maximize effectiveness and run these models at their original precision levels, the Volvo Server is equipped with Graphics Processing Units (GPUs) and Central Processing Units (CPUs).

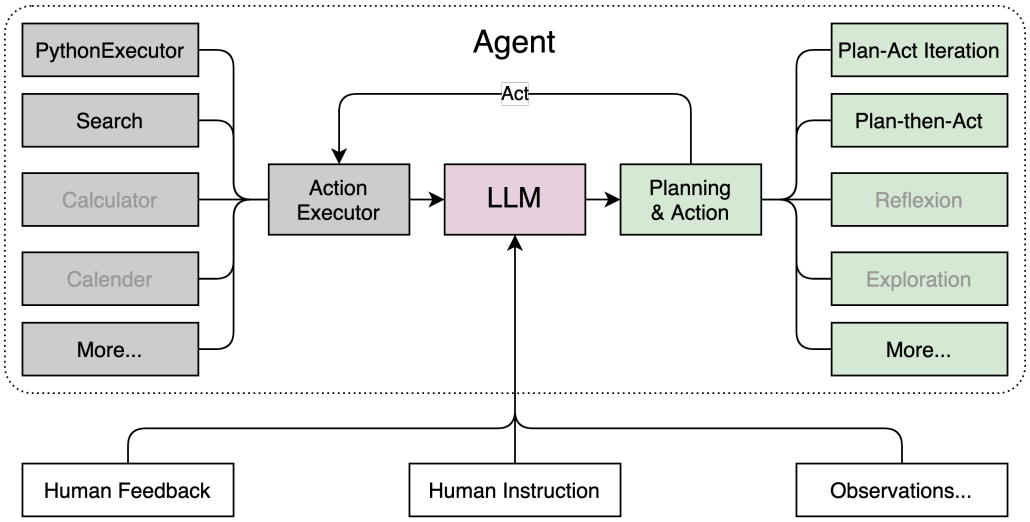


Figure 4.4: Overview of the Lagent framework. Source: Adapted from [46].

In developing the LLM application, we rely on the Volvo Framework. This framework is mainly built upon Lagent [46]. The Lagent framework is a lightweight, open-source tool designed to facilitate the creation of LLM-based agents. It offers various features to augment LLM functionality efficiently. An overview of the Lagent framework is shown in Figure 4.4.

This framework is connected to the server, which directly accesses the LLMs Code-Llama-34b and Mixtral 8x7b decoders through an IP address and port. Currently, these models are the only ones available.

4.3.2 GUI Layout

To facilitate interaction with the agent, a Graphical User Interface (GUI) was designed using the Python library `streamlit`. The GUI layout, illustrated in Figure 4.5, consists of a sidebar containing multiple tabs and the chat page.

In the "Setup" tab in the left sidebar, users can choose an MF4 file for analysis. Once a file is selected, the system extracts all signals and their units. Users can also apply custom filter conditions if needed. Users can input their requests on the right-hand side of the chat page. These requests, along with signal details and filter conditions, serve as inputs to the planner, as illustrated in Figure 4.1.

When a request is sent to the agent, the final response will undergo processing as detailed in Section 4.2. Subsequently, the generated code will be executed in the

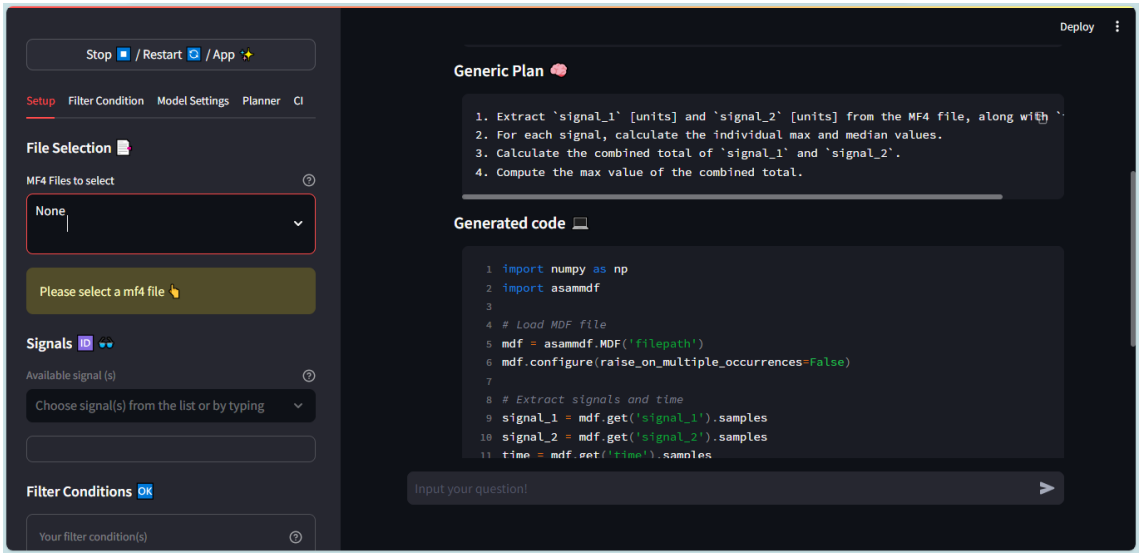


Figure 4.5: Layout of the GUI, showcasing the sidebar on the left, and the chat page on the right.

background, producing outputs such as numerical computations or figure plots, which will then be displayed on the chat page.

4.4 Dataset

This section explores the types of analyses performed on the experiment under various request categories. Additionally, we present the dataset and detail how it was crafted.

4.4.1 Request Categories

In the domain of MF4 analysis at Volvo, user inquiries are structured into two primary domains: computations and plots. These categories can be subdivided into subcategories covering the most common use cases for MF4 analysis.

Computations can be broadly categorized into five groups, as illustrated in Table 4.2. Examples of user requests for performing these different computations are presented in the same table.

Plots can be broadly categorized into eight groups, as illustrated in Table 4.3. Examples of user requests for performing these different computations are presented in the same table.

The model is capable of performing a variety of requests beyond those mentioned above. For example, depending on the user prompt, the agent can perform a combination of analyses rather than confining the request to a specific category. However, for the sake of transparency in the experiment, user queries are structured into the mentioned categories.

Computation Type	**Example**
Max, min, average, median	Compute individual max, individual median and combined max of signal_1 and signal_2.
Integration	Compute the individual energies consumed by integrating signal_1 and signal_2 over time.
Event detection	Count the number of events when signal_1 crosses the threshold.
Linear Regression	Do a linear regression between signal_1 and signal_2.
Correlation	Calculate the correlation coefficient between signal_1 and signal_2.

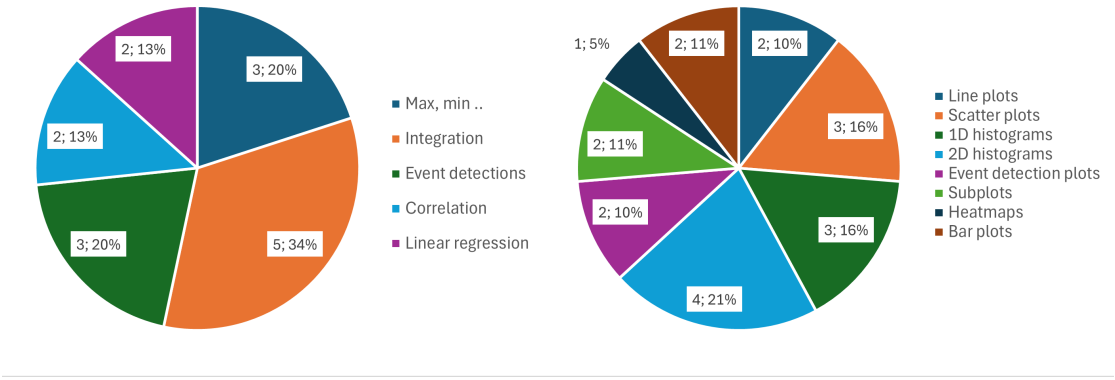
Table 4.2: Computation types and examples of different computation requests.

Plot Type	**Example**
Line plots	Plot signal_1 over time.
Scatter plots	Make a correlation scatter plot between signal_1 and signal_2.
1D Histograms	Make a histogram plot of signal_1.
2D Histograms	Do a 2d histogram plot of signal_1 and signal_2.
Event detection plots	Plot the signal_1 crosses over time and the detected events when the signal crosses above the threshold.
Subplots	Based on the signal_1 and signal_2. Make a 2x1 subplot with the following configuration: 1. The first subplot contains signal_1 over time. 2. The second subplot contains the accumulated signal_2
Heatmaps	Plot a heatmap of signal_1 on the x-axis, signal_2 on the y-axis and signal_3 on the z-axis.
Bar plots	Make a barplot of max min average of signal_1.

Table 4.3: Plot types and examples of different kinds of plot requests.

4.4.2 Sample Distribution

The samples of the computation and plot dataset can be found in Appendix B. The distribution of request types for computation and plot analysis is shown in Figure 4.6. The aim is to balance the number of samples across all subcategories to prevent the model's performance from being biased toward any particular one.



(a) Computation.
Total of 15 samples.

(b) Plot.
Total of 19 samples.

Figure 4.6: Samples distribution.

4.4.3 Crafting User Requests

When crafting user queries for constructing the dataset, there are guidelines, standards, and templates to follow for designing tests that ensure optimal performance.

The guidelines for crafting the user prompts are as follows:

1. **Use Exact Signal Names from the MF4 File:** Refer to the signals exactly as they are named in the MF4 file. This is because generating the code and extracting the signals must be correct, or else get a "no signal found" error.
2. **Be Specific using Keywords:** Include clear keywords such as "integration," "accumulation," "event detection," "line plot," or "heat map" to indicate the purpose of the request.
3. **State Unit Conversion and Specification:** Mention any required unit conversions or specific units in the request to guide the agent accurately. If no unit conversion is needed, the agent will work with the units as they are.
4. **Explicit Plot Customizations:** Specify any plot customizations explicitly, such as "set the font size to 14" or "plot this in red."
5. **Use Diverse Prompts for Each Category:** Frame requests in various ways for each category or increase complexity by involving multiple signals to ensure the model can handle different prompts with the same objective.

4.5 Evaluation

This section outlines how we've designed our own metrics to evaluate each aspect for assessing the agents' performance in analyzing MF4 files. The analysis can be divided into two main categories: computation and plots. To evaluate the model, we will employ human evaluators.

4.5.1 Computation Metric

We establish three evaluation metrics for computation:

1. Numerical Correctness: Does the numerical data and information presented align with the correct values and context?
2. Unit Correctness: Does the unit data and information presented align with the correct values and context?
3. Alignment: Does the computation align with the request? Or did it include additional elements?

We assess numerical, unit correctness, and alignment using a scale from 0 to 2: 0 for incorrect, 1 for partially correct, and 2 for correct.

4.5.2 Figure Metric

We define four evaluation metrics for the figures:

1. Data Correctness: Does the data and information presented in the figure align with the correct values and context?
2. Chart Type: Does the chosen chart type correspond to the requirements outlined in the request?
3. Aesthetics: Does the figure exhibit an appealing visual presentation, ensuring clarity and absence of formatting errors in labels, names, and units?
4. Alignment: Does the plot align with the request? Or did it include additional elements?

We assess correctness, aesthetics, and alignment using a scale from 0 to 2: 0 for incorrect, 1 for partially correct, and 2 for correct. The scale ranges from 0 to 1 for the chart type: 0 for incorrect and 1 for correct.

4.5.3 Human Evaluators

To evaluate the dataset, we employed human evaluators. This decision was motivated by the necessity for subjective judgment, the complexity of evaluation criteria, and the capability to discern nuances that automated systems might struggle with and cannot be solely relied upon. In total, four human evaluators participated: two who were involved in the thesis and two external individuals to ensure evaluation transparency. The evaluators were provided with a sheet containing the dataset,

which comprised user queries, filter conditions applied to the requests, ground truth answers for comparison with the model output, and an evaluation form to fill in the metric score.

5

Results

In the results chapter of this thesis, we present the outcomes of several conducted experiments. First, we will present and discuss the model's performance, evaluated by human evaluators using custom metrics on the dataset specifically crafted for this study. We will then showcase several case studies to demonstrate the model's output.

5.1 Model Performance

This section presents the scores on custom metrics for computation and plot quality, evaluated by the evaluators for the custom dataset.

5.1.1 Computations

The evaluation scores and the total accuracy for each subcategory measured in percent for the computation request are presented in Table 5.1. The computation request displays the number of samples, numerical correctness, unit correctness, and alignment for each subcategory. The scores have been averaged from four human evaluator scores and the number of samples for each subcategory. The dataset for the computation request can be found in Appendix B.1. To better visualize the scores in terms of accuracy, a box plot for each subcategory of computation requests are shown in Figure 5.1.

When evaluating model performance across various subcategories, it becomes evident that the model excelled in the max, min, average, and median categories, achieving a perfect score of 100% accuracy across all metrics. However, in subcategories such as integration and event detection, the performance was slightly less robust, with an total accuracy of 94-96%. Integration tasks involve calculating the total accumulated value of a signal over time, for example, determining the total energy consumption by integrating power output. In this subcategory, inaccuracies arose due to the model's failure to use recomputed time, resulting in minor deviations from the ground truth values. Event detection tasks require identifying and counting specific events, like threshold crossings where a signal exceeds a set limit. The model sometimes struggled to accurately compute these threshold crossings within the generated code, affecting its overall performance in this subcategory.

Moving on to the correlation coefficient subcategory, there was a noticeable decline

in numerical correctness, thus affecting its overall performance reaching 87.5%. This was primarily due to wrong function called when computing correlation coefficients for scenarios with no correlation. Finally, for linear regression, which obtained the total accuracy of 91.67%, the shortcomings were unexpected. Interestingly, the inaccuracies were not rooted in numerical precision or unit discrepancies but rather in alignment issues. By "alignment issues," we mean that the model's output did not match the specific requirements or expectations set by the user prompt. This discrepancy was largely influenced by additional analyses performed beyond the standard linear regression metrics such as computation of p-values and correlation coefficients. However, focusing solely on the slope and intercept would have sufficed for the given requirements.

Subcategory	Samples	Numerical Correctness	Units Correctness	Alignment	Total
Max, min, average, median	3	2.0	2.0	2.0	6.0 (100%)
Integration	5	1.75	2.0	2.0	5.75 (95.83%)
Event detection	3	1.67	2.0	2.0	5.67 (94.44%)
Correlation coefficient	2	1.38	2.0	2.0	5.25 (87.5%)
Linear regression	2	2.0	2.0	1.5	5.5 (91.67%)
Total	15	8.67 (86.67%)	10.0 (100%)	9.5 (95%)	28.17 (93.89%)

Table 5.1: Evaluation scores for computation requests.

Another noteworthy observation is that the model achieved 100% accuracy in unit correctness. This success can be attributed to the utilization of signal units provided by the MF4 file, with the model defaulting to work in these units unless instructed otherwise by the user. Additionally, it's worth mentioning that the alignment score metric across all subcategories was consistently high at 100%, except for linear regression. This indicates that the model predominantly provides responses aligned with the user's requests, albeit linear regression occasionally diverging from this trend. Considering the computation metrics and the model's performance across each subcategory, the total accuracy for the entire dataset amounted to 93.89%.

5.1.2 Plots

The evaluation scores and the total accuracy for each subcategory measured in percent for the plot request are presented in Table 5.2. For each subcategory in the plot request, the number of samples, numerical correctness, units, and alignment are

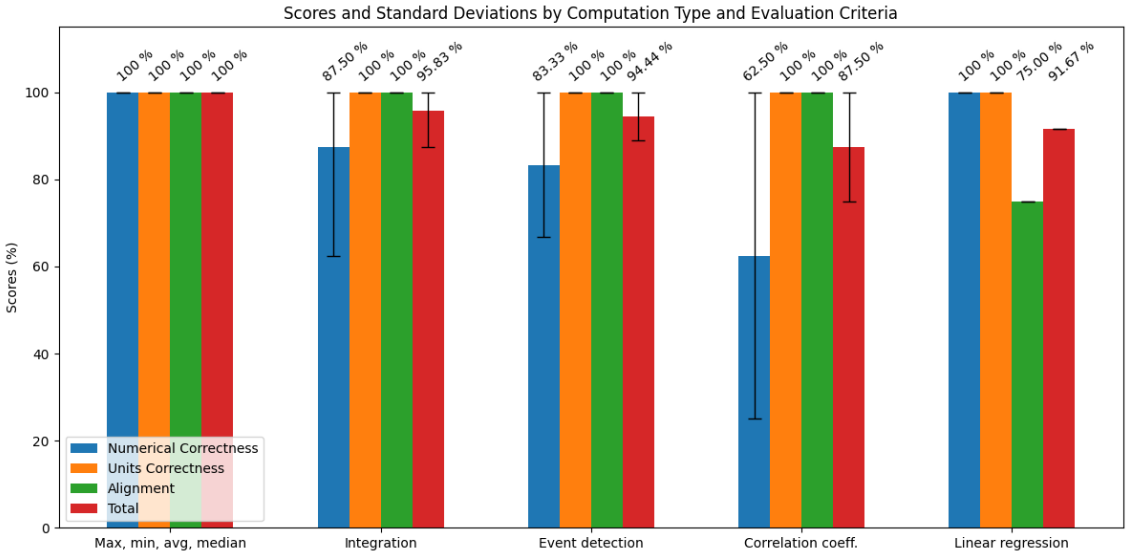


Figure 5.1: Bar plot of accuracy scores for each subcategory request in the computation dataset. The error bars represent the standard deviation, averaged over four evaluators and the samples within each subcategory.

displayed. The scores have been averaged from four human evaluator scores and the number of samples for each subcategory. The dataset for the plot requests can be found in Appendix B.2. To better visualize the scores in terms of accuracy, box plots for the first half and the second half of the plot requests are shown in Figure 5.2 and Figure 5.3, respectively.

The model achieved 100% accuracy in data correctness for scatter plots, 1D histograms, 2D histograms, and heatmaps, highlighting the model's strong performance in these areas. However, for other subcategories such as line plots, event detection, subplots, and bar plots, the accuracy was notably lower, around 50-75%. The lower scores in these subcategories were primarily due to the model's inability to correctly recompute the time when plotting signals against time. According to the analysis standards, the time data is considered incorrect in these cases, which explains why the model performed so well on histograms that do not involve time. Another contributing factor to the lower data correctness score is the incorrect usage of functions during the computation phase. For instance, when accumulation over time of a signal is requested before plotting, the model sometimes accumulates only over the signal and not together with time, leading to incorrect values.

Moving on to the chart type metric score, the model received 100% in every category, which implies that the model always generated the correct chart type per the user's request. The alignment metric also achieved a similar score, where the model nearly attained a perfect score in each subcategory. This indicates that the given responses were aligned with the user requests. The one exception was for subplots; the model did not follow the instructions to make well-separated subplots.

The results are mixed for the last metric score, which measures aesthetics. Half of the time, the model can generate necessary labels, names, and units in the plot; the

Subcategory	Samples	Data Correctness	Chart Type	Aesthetics	Alignment	Total
Line plots	2	1.38	1.00	2.00	2.00	6.38 (91.07%)
Scatter plots	3	2.00	1.00	1.33	2.00	6.33 (90.48%)
1D histograms	3	2.00	1.00	2.00	2.00	7.00 (100%)
2D histograms	4	2.00	1.00	1.63	2.00	6.63 (94.64%)
Event detection plots	2	1.00	1.00	1.50	2.00	5.5 (78.57%)
Subplots	2	1.50	1.00	1.75	1.75	6.00 (85.71%)
Heatmaps	1	2.00	1.00	1.00	2.00	6.00 (85.71%)
Bar plot	2	1.50	1.00	1.50	2.00	6.00 (85.71%)
Total	19	13.38 (83.59%)	8.00 (100%)	12.71 (79.43%)	15.75 (98.44%)	49.83 (90.36%)

Table 5.2: Evaluation scores for plot requests.

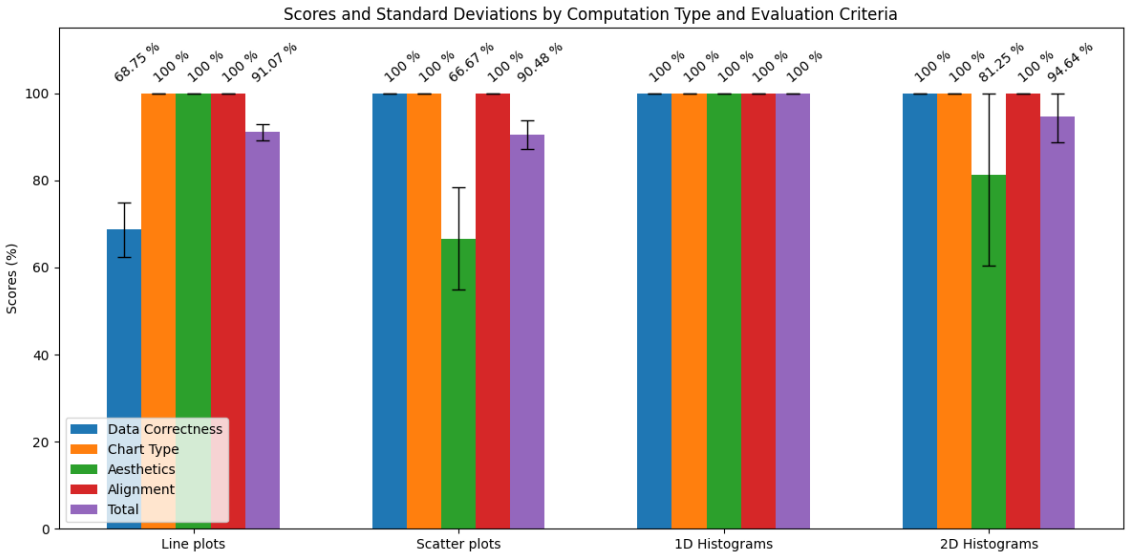


Figure 5.2: Bar plot of accuracy scores for each subcategory request in the first half of the plot dataset. The error bars represent the standard deviation, averaged over four evaluators and the samples within each subcategory.

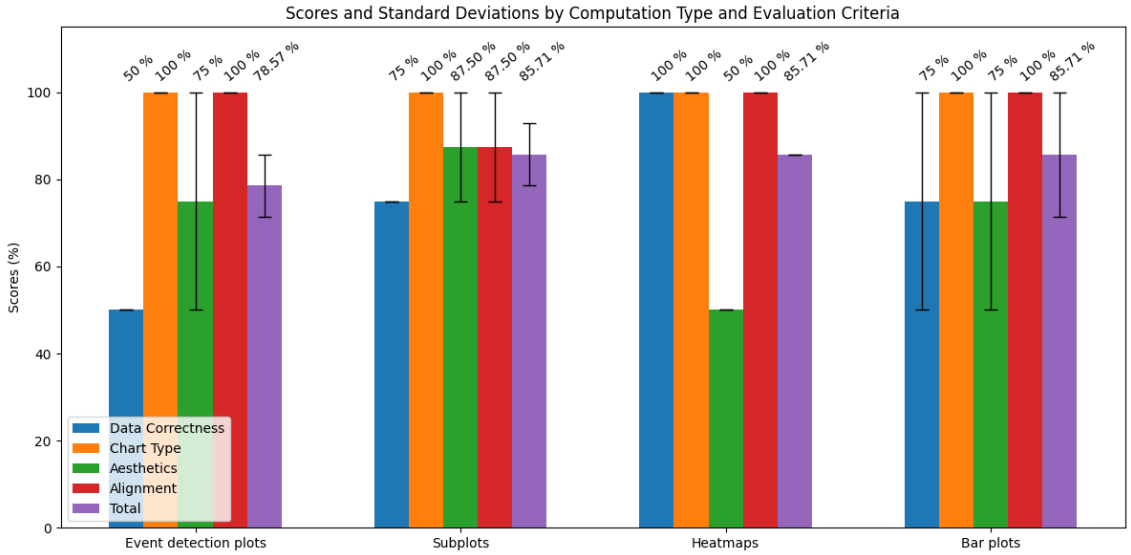


Figure 5.3: Bar plot of accuracy scores for each subcategory request in the second half of the plot dataset. The error bars represent the standard deviation, averaged over four evaluators and the samples within each subcategory.

other half does not perform as well. Issues include forgetting legends in the plot, placing legends on top of the plot, not adding or naming color bars for heat maps and 2D histograms, or making subplots too tight, causing labels to overlap. It is worth mentioning that the user requests in the dataset have been kept generic about plot customization. The results can be inconsistent when left to the model, as seen in the aesthetics scores. However, the model often performs as expected when specific instructions are provided at the user's request.

The total accuracy varied across the subcategories. The model achieved high total accuracy scores of approximately 90-100% for line plots, scatter plots, 1D histograms, and 2D histograms. On the other hand, the total accuracy for event detection plots, subplots, heatmaps, and bar plots was lower, around 78.57-87.71%. Considering the plot metrics and the model's performance across each subcategory, the overall total accuracy for the entire dataset amounted to 90.36%.

5.2 Case Studies

In this section, several selected case studies will be presented and analyzed. Each case study includes the request, the generated plan, and the output in the form of computations or plots. Additional case studies for each subcategory of the computation and plot dataset can be found in Appendix D and Appendix E, respectively. Due to the sensitivity of the data, dummy signal names are used throughout this chapter and in the appendix. For computation case studies, the obtained and actual values will be denoted by x and y, respectively. Additionally, the delta error is provided to verify the accuracy of the results. For the plot case studies, the signal values have been normalized to maintain confidentiality while allowing for accurate

visual representation.

5.2.1 Event Detection Plot

In this case study, we focus on detecting specific events within the logged vehicle data. Event detection is crucial in identifying and analyzing critical moments that may indicate anomalies or significant changes in the vehicle's performance.

User Prompt: Calculate total_power = PowerOutput_CellUnitA + PowerOutput_CellUnitB. Compute and print the number of detected events when the total_power crosses 0.91 kW. In the same figure, do the following: Scatter plot the events in red. Ensure the keyword label and set the order of the scatter plot to 3. Plot total power vs timeline plot. For the line plot, ensure linestyle "-", marker "o" and keyword label.

Generated Plan:

1. Extract ‘PowerOutput_CellUnitA‘ [kW], ‘PowerOutput_CellUnitB‘ [kW], and ‘time‘ [s] from the MF4 file.
2. Calculate ‘total_power‘ [kW] = ‘PowerOutput_CellUnitA‘ [kW] + ‘PowerOutput_CellUnitB‘ [kW].
3. Detect events when ‘total_power‘ [kW] crosses 0.91 kW.
4. Compute and print the number of detected events.
5. Scatter plot the events in red with keyword label and zorder of 3.
6. Plot ‘total_power‘ [kW] vs ‘time‘ [s] with linestyle "-", marker "o", and keyword label.

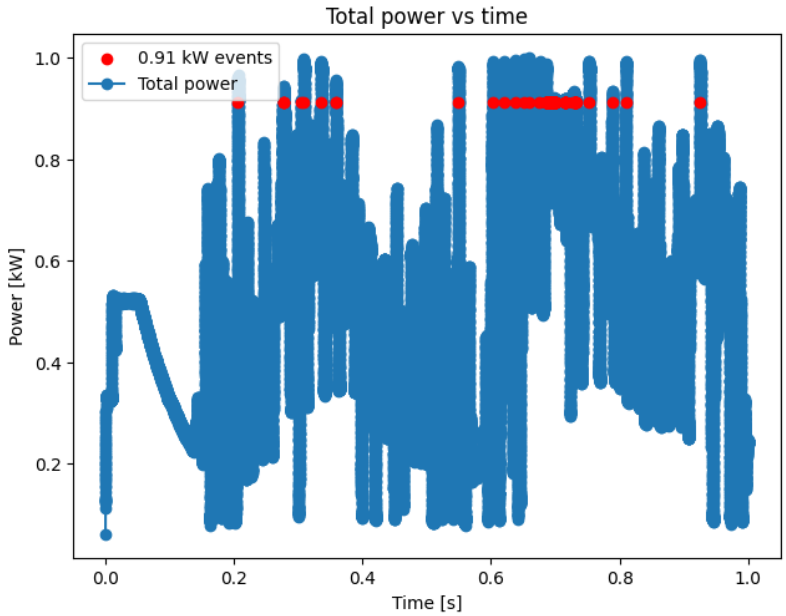


Figure 5.4: Case study of a event detection plot request.

Generated Figure: See Figure 5.4. This figure illustrates an event detection plot where the model first computes the events when the signal crosses the threshold and then plots these events on the same graph as the signal. The plan includes all necessary steps to perform the analysis properly. The CI can execute computations well, such as extracting the two power signals, adding them, and applying filter conditions. However, when generating the figure, it is not guaranteed that the CI will align with the user's intended specifications if they are not stated explicitly in the user query. For instance, the CI might inadvertently plot the events and signal in the same color or place the events behind the signal, whereas it's preferable for the events and the signal to be in different colors, with the events plotted in front of the power signal. To ensure aesthetic quality, additional instructions regarding colors, z-order, and line and marker styles were included in the user request.

5.2.2 Power Temperature 2D Histogram

In this case study, we examine the relationship between power output and temperature by creating a 2D histogram. Analyzing the relationship between power output and temperature is essential for understanding how different operational conditions affect the vehicle's performance.

User Prompt: Compute proportion = frequency / total frequency * 100 for each bin. Plot a 2d histogram of PowerOutput_CellUnitA, CoolantExitTemp_CellUnitA and the proportion. Display the proportion rounded to an integer as text for each bin in the plot Add colorbar with label "%time in each bin. Make sure to set the colormap to "jet" and bins to 15.

Generated Plan:

1. Extract ‘time‘ [s], ‘PowerOutput_CellUnitA‘ [kW], and ‘CoolantExitTemp_CellUnitA‘ [°C] from the MF4 file.
2. Compute frequency for each bin.
3. Compute total frequency.
4. Compute proportion for each bin: proportion = frequency / total frequency 100.
5. Create a 2D histogram of ‘PowerOutput_CellUnitA‘ [kW], ‘CoolantExit-Temp_CellUnitA‘ [°C], and the proportion.
6. Display the proportion as text for each bin in the plot, rounded to integer.
7. Add a colorbar with label "%time in each bin".
8. Set the colormap to "jet".
9. Set the bins to 15.
10. (Optional) Detect events.

Generated Figure: See Figure 5.5. This figure illustrates a 2D histogram plot showing the percentage of time spent in each bin of power and coolant exit tempera-

ture. The analysis plan covers all necessary steps, with an additional step, (step 10) added by the planner to detect events. It happens occasionally that the planner does that and it most to the system prompt. Despite the system prompt's instruction not to include optional steps (refer to Appendix C.1), the planner occasionally adds them, reflecting a limitation of LLMs.

However, when the plan is sent to the CI, it disregards optional steps, which typically doesn't cause issues. To ensuring aesthetics, specific plot requirements were once again provided to the model. These include displaying the proportion of time in each bin, setting the number of bins to 15, and configuring the colormap to 'jet'.

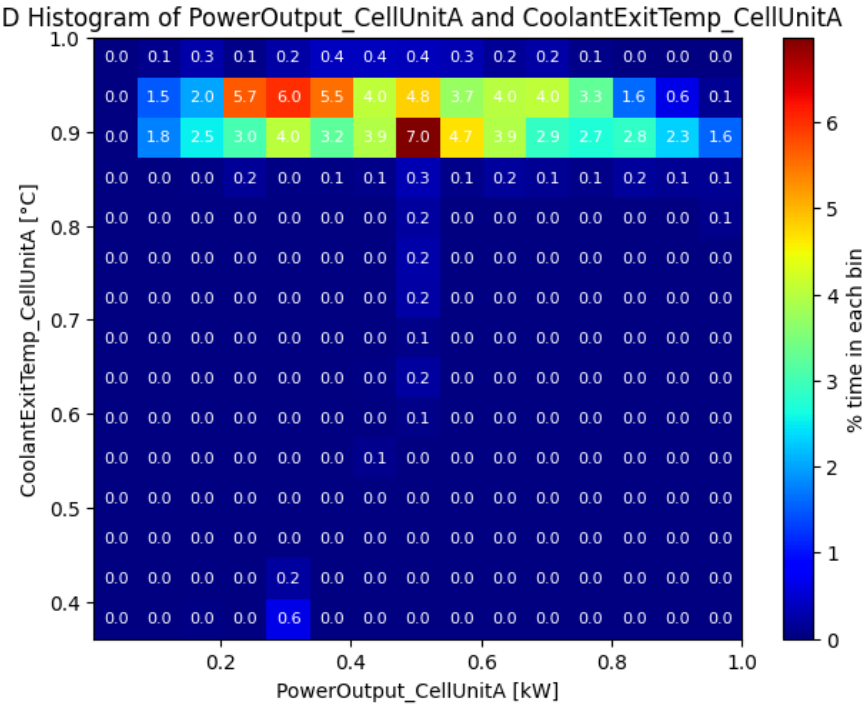


Figure 5.5: Case study of a 2D histogram request.

5.2.3 Speed and Accumulated Distance Subplot

In this case study, we analyze the relationship between vehicle speed and accumulated distance over time by creating subplots. Understanding the relationship between vehicle speed and accumulated distance over time is crucial for assessing the vehicle's efficiency and performance. By creating subplots that display these variables, we can gain insights into how speed influences the total distance traveled.

User Prompt: Based on the vehicle speed signal - CalculatedTravelVelocity, make a 2x1 subplot with the following configuration

1. The first subplot contains vehicle speed with respect to time
2. The second subplot contains the accumulated distance

Convert the vehicle speed in m/s before integrating Keep separation between the subplots.

Generated Plan:

1. Extract signals ‘CalculatedTravelVelocity‘ [Km/h] and ‘time‘ [s] from the MF4 file.
2. Convert ‘CalculatedTravelVelocity‘ from Km/h to m/s.
3. Integrate the converted ‘CalculatedTravelVelocity‘ [m/s] to obtain accumulated distance.
4. Create a 2x1 subplot with separate figures for ‘CalculatedTravelVelocity‘ [m/s] and accumulated distance.
5. Plot ‘CalculatedTravelVelocity‘ [m/s] against ‘time‘ [s] in the first subplot.
6. Plot accumulated distance against ‘time‘ [s] in the second subplot.

Generated Figure: See Figure 5.6. This is a case where the model did not align well with the user request, resulting in poor aesthetics performance. The user requested adequate separation between the subplots, but the model failed to account for that. The x-label of the first plot is beneath the second subplot due to the tight separation. Additionally, the legend of the first subplot is in the middle of the line plot, which is not an ideal location; a better placement would be the top left corner. It is important to note that it can be difficult for the model to know these preferences using the default capabilities and settings of Matplotlib. To improve aesthetics, specific requirements need to be provided in the user request, as the model cannot always get it right on its own.

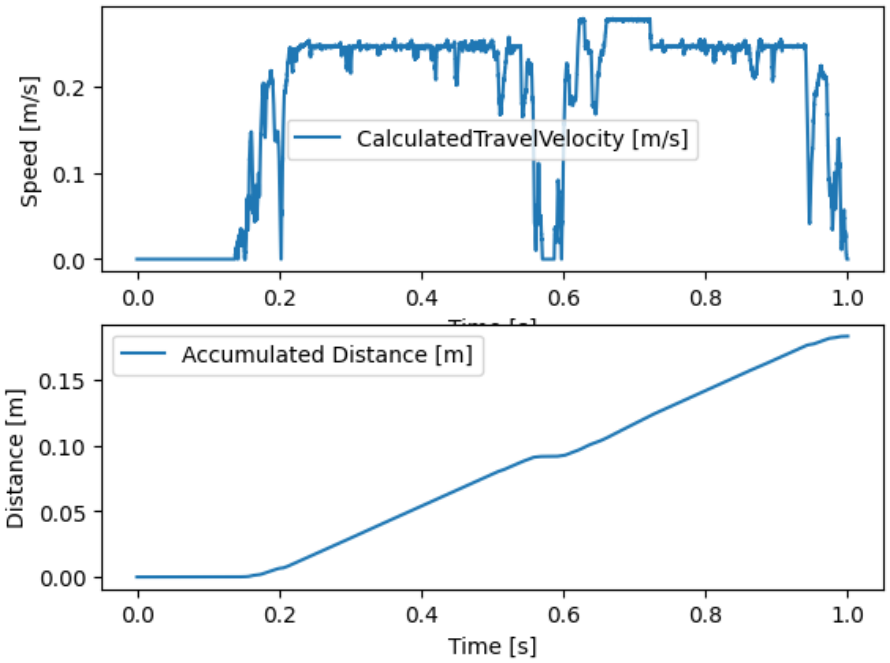


Figure 5.6: Case study of a subplot request.

6

Discussion

In the discussion chapter of this thesis, we thoroughly analyze the results obtained from the evaluation of the MF4 agent created in this study. We explore the implications of these findings, assess the strengths and weaknesses of our methodology, and propose directions for future investigation.

6.1 Effective Practices

In this section, we discuss the effective practices implemented in our study. We focus on utilizing a multi-agent system and the strategies employed in prompt engineering. These methods have proven effective in adapting an LLM-powered agent for logged data analysis.

6.1.1 Multi-Agent System

The multi-agent system we implemented, consisting of the planner, selector, and code interpreter, proved highly effective in managing complex tasks. The planner's role is to break down user queries into smaller, clear, and manageable tasks. For instance, it determines which signals to extract, which computations to perform, and how to handle unit conversions. This systematic approach ensures clear instructions for the code interpreter, reducing the risk of overlooking details and enhancing overall efficiency compared to scenarios without a planner. The selector was crucial in assigning the code interpreter role, selecting either the computation or plot role. This distinction was particularly beneficial for computation requests, as it allowed the CI to focus solely on computation without being distracted by plotting settings, thereby avoiding getting lost in context. Moreover, it prevented the model from generating unwanted plots that were not requested by the user, an issue that arose when a single CI role with both computation and plot instructions were employed.

6.1.2 Prompting Techniques

Another effective practice involves employing prompt engineering techniques when crafting system instructions for the planner, selector, and CI. Utilizing various prompt techniques, such as chain of thought, self-consistency, zero-shot, and few-shot learning, helps guide the expected behaviour. For the planner, a chain of thought was applied to generate step-by-step thinking. Self-consistency was employed for the

selector to ensure consistent role selection, supplemented by few-shot learning to distinguish between roles. Zero-shot prompting enabled the models to handle tasks without prior examples. In contrast, few-shot prompting provided limited examples, primarily for the CI, to enhance the model's understanding of the task. The system prompts also included prohibitions to prevent the model from executing unwanted actions. Emphasizing instructions with capital letters and symbols, such as quotes, could potentially impact the model's performance.

6.2 Agent Model

The model's performance was evaluated using two customized datasets: computations and plots. Each evaluation result will be thoroughly discussed alongside several case studies. Finally, we'll examine and discuss the limitations of the agent encountered during the evaluation process.

6.2.1 Computation Performance

The results for the entire computation dataset, see Section 5.1.1, were as follows: numerical correctness was 86.67%, unit correctness was 100%, and alignment was 95%. Regarding numerical correctness, the model often generated and computed the correct values. Incorrect values were mainly due to issues like not recomputing time or using an unsuitable function for the specific request. The model showed a high level of consistency in alignment, correctly aligning most of the time. Misalignment usually occurred when the model computed extra details but did not completely diverge from the request. The perfect score for unit correctness indicates that the model consistently computed in the correct units, whether by staying in the current units or performing unit conversions as needed.

The subcategories max, min, average, and median appeared to be the simplest in complexity, achieving perfect scores across the three metrics. In contrast, the subcategories for integration, event detection, and correlation coefficient struggled somewhat with numerical correctness, while linear regression faced challenges with the alignment metric. The total accuracy score for the dataset was computed to be 93.89%.

6.2.2 Plot Performance

The results for the entire plot dataset, see Section 5.1.2, were as follows: numerical correctness was 86.67%, chart type 100%, aesthetics 79.43%, and alignment was 98.44%. The model generated the requested chart types exceptionally well, achieving a perfect score. It also scored high in alignment, meaning the responses generally aligned with the user's requests. In the rare cases where it did not align, issues included insufficient separation between subplots despite the user's request. However, there was a noticeable drop in data correctness and aesthetic accuracy. Data correctness issues often stemmed from not recomputing the time variable or using incorrect functions, resulting in incorrect plotted data. For aesthetics, the model

sometimes failed to account for plot settings not explicitly mentioned in the user request. This included missing legends, unsuitable label names, missing colour bars in heat maps and 2D plots, and overly tight subplots causing label overlap despite instructions in the system prompt.

The subcategory of 1D histograms was the only one that achieved a perfect score. Other subcategories fell short either in data correctness or aesthetics. Additionally, there seems to be a correlation between the total accuracy of each subcategory and the complexity of the request for the model to execute. Histograms were the easiest, followed by line and scatter plots, then subplots, heatmaps, and bar plots. The most complex request subcategory appears to be event detection. The overall accuracy score for the dataset was 90.36%.

6.2.3 Case Studies

The case studies, as shown in Section 5.2 demonstrate the utilization of user prompts, generated plans, and figures to showcase the model's capabilities. We observed that the planner generates generic yet detailed plans in each use case, which should be sufficient for the CI to generate the code. Occasionally, the planner might generate optional steps, which are not ideal behaviour but does not pose a direct issue if the CI ignores them. To generate figures with satisfactory aesthetics, explicit specifications are often needed in the user prompt. While the models can usually handle this automatically, they sometimes miss these details.

6.2.4 Limitations

One notable limitation that we have is the Mixtral's and Code Llama's ability to follow instructions exactly or forget to adhere to them. As we've observed, the planner sometimes includes optional steps even though it has been instructed not to do so. Although this often affects the output for the CI, it can sometimes lead to unwanted results because the CI does what it is told to do. Additionally, the CI sometimes forgets to recompute time or add legends and colour bars, even though it has been instructed to always do so. This poses a problem in obtaining consistent and reliable results if we cannot ensure they always follow the instructions. It sometimes requires the user to be more specific than necessary. Another limitation of the agent is code generation. Occasionally, it hallucinates and produces code that is non-functional and incorrect.

6.3 Comparison Between Workflow and Agent

Our model has shown strong performance on our customized dataset, achieving 93.89% on computations and 90.36% on plots based on our specific evaluation metrics. Compared to human performance, where a human can generate the desired output most of the time with an accuracy of 100%, our model demonstrates significant effectiveness but still leaves room for improvement.

Using the current workflow with manual script templates that provide a set of defined

functions requires writing all the Python code for computations and plots. This process can be time-consuming, typically taking an experienced programmer 10-15 minutes or even longer. In contrast, agent inference takes only 10-15 seconds to generate and execute the code. The primary effort in utilizing the agent lies in writing the prompt in natural language, which is arguably faster and easier than writing Python code. Even if the generated code is not entirely correct, it provides a skeleton that can be modified as needed. This approach can significantly enhance productivity and scalability when conducting multiple analyses.

In summary, in an industrial context where 100 percent accuracy is desired, the agent may not yet meet human standards of reliability and accuracy to entirely replace human involvement. However, proper usage and integration can markedly accelerate productivity and save time compared to our current workflow.

6.4 Future Works

Improving the accuracy of our agent is a primary goal for future work. Although our study involved crafting the system instructions for the planner selector and code interpreter modules to the best of our abilities, further prompt tuning is necessary. We have observed how the prompts affect the model's behaviour and outputs and believe that fine-tuning the system prompts further could enhance the agent's performance and accuracy.

Another refinement to the model is the introduction of error handling and validation mechanisms between the agent's modules. Specifically, the planner can be enhanced with a refinement module to ensure the generated plan is accurate and feasible. Similarly, the code interpreter can be augmented with a validation component to verify the correctness of the generated code. This would significantly improve the robustness and reliability of the agent.

The models used in this study, including Mixtral 8x7b and Code-Llama-34b, have shown some limitations, particularly in their attention to detail within system instructions. Future research should consider adopting more advanced models to improve performance and outcomes. Utilizing state-of-the-art LLMs could enhance the system's capabilities, enabling more accurate and efficient data analysis. Investigating these newer models will likely significantly enhance the agent's overall functionality.

The current agent responds to user requests without considering or remembering previous interactions. To improve interaction and handle more complex tasks effectively, it's crucial to integrate memory and take a proactive approach. This means the agent should recall information and context from past conversations. By doing so, it can engage in multi-round discussions with users and work together to reach final conclusions. This ability could greatly enhance its effectiveness in long-term conversations and detailed investigations.

7

Conclusion

To conclude the findings of this thesis the approach of designing the model in an agentic manner, comprising a planner, selector, and code interpreter, along with the integration of well-known prompting techniques, proved to be an effective method for adapting LLMs to logged data analysis at Volvo Group AB. The model's performance was evaluated using custom metrics on our specific dataset, which includes common user requests for computations and plots. The model achieved a robust accuracy of 93.89% for computations and 90.36% for plots. While these scores don't quite reach the 100% accuracy level achievable by human analysts, the model showcases reliability that's suitable for integration into the current workflow with proper usage. This integration holds promise for enhancing productivity and time efficiency in logged data analysis tasks.

Bibliography

[1] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[2] OpenAI et al. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[3] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[4] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[5] Gemini Team et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2023. arXiv: 2312.11805 [cs.CL].

[6] Jingfeng Yang et al. *Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond*. 2023. arXiv: 2304.13712 [cs.CL].

[7] Muhammad Usman Hadi et al. "Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects". In: (July 2023). DOI: 10.36227/techrxiv.23589741.v1.

[8] Humza Naveed et al. *A Comprehensive Overview of Large Language Models*. 2023. arXiv: 2307.06435 [cs.CL].

[9] Junjie Wang et al. *Software Testing with Large Language Models: Survey, Landscape, and Vision*. 2024. arXiv: 2307.07221 [cs.SE].

[10] Zeeshan Rasheed et al. *Can Large Language Models Serve as Data Analysts? A Multi-Agent Assisted Approach for Qualitative Data Analysis*. 2024. arXiv: 2402.01386 [cs.SE].

[11] Zhiheng Xi et al. *The Rise and Potential of Large Language Model Based Agents: A Survey*. 2023. arXiv: 2309.07864 [cs.AI].

[12] Ziang Xiao et al. "Supporting Qualitative Analysis with Large Language Models: Combining Codebook with GPT-3 for Deductive Coding". In: *28th International Conference on Intelligent User Interfaces*. IUI '23. ACM, Mar. 2023. DOI: 10.1145/3581754.3584136. URL: http://dx.doi.org/10.1145/3581754.3584136.

[13] Ankit Yadav and Mayank Singh. *PythonSaga: Redefining the Benchmark to Evaluate Code Generating LLM*. 2024. arXiv: 2401.03855 [cs.CL].

[14] Liying Cheng, Xingxuan Li, and Lidong Bing. *Is GPT-4 a Good Data Analyst?* 2023. arXiv: 2305.15038 [cs.CL].

[15] Bo Qiao et al. *TaskWeaver: A Code-First Agent Framework*. 2023. arXiv: 2311.17541 [cs.AI].

[16] Baptiste Rozière et al. *Code Llama: Open Foundation Models for Code*. 2023. arXiv: 2308.12950 [cs.CL].

[17] Albert Q. Jiang et al. *Mixtral of Experts*. 2024. arXiv: 2401.04088 [cs.LG].
[18] Hailin Chen et al. *ChatGPT's One-year Anniversary: Are Open-Source Large Language Models Catching up?* 2024. arXiv: 2311.16989 [cs.CL].
[19] Vector. *MDF: Efficient Storage of Measurement Data with MDF Format*. https://www.vector.com/at/en/products/application-areas/ecu-calibration/measurement/mdf/. Accessed: 09/03/2024. 2024.
[20] OpenAi. *Chatgpt plugins*. URL: https://openai.com/blog/chatgpt-plugins.
[21] Pranab Sahoo et al. *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*. 2024. arXiv: 2402.07927 [cs.AI].
[22] Volvo Group. *Volvo Group*. https://www.volvogroup.com/en/about-us.html. Accessed: 13/03/2024. 2024.
[23] Saqib Hakak et al. "Autonomous vehicles in 5G and beyond: A survey". In: *Vehicular Communications* 39 (2023), p. 100551. ISSN: 2214-2096. DOI: https://doi.org/10.1016/j.vehcom.2022.100551. URL: https://www.sciencedirect.com/science/article/pii/S2214209622000985.
[24] Vector. *Analyzing ECUs and Networks with CANalyzer*. https://www.vector.com/int/en/products/products-a-z/software/canalyzer/#c4237. Accessed: 13/03/2024. 2024.
[25] Vector. *Logging Formats*. https://support.vector.com/kb?id=kb_article_view&sysparm_article=KB0011536&sys_kb_id=b6d21110870d4550b9f233770cbb3523&spa=1. Accessed: 13/03/2024. 2024.
[26] Banghao Chen et al. *Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review*. 2023. arXiv: 2310.14735 [cs.CL].
[27] Zhiheng Xi et al. *The Rise and Potential of Large Language Model Based Agents: A Survey*. 2023. arXiv: 2309.07864 [cs.AI].
[28] Y. Zhang et al. "Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models". In: (2023). arXiv: 2309.01219 [cs.CL].
[29] Taicheng Guo et al. *Large Language Model based Multi-Agents: A Survey of Progress and Challenges*. 2024. arXiv: 2402.01680 [cs.CL].
[30] Xueyu Hu et al. *InfiAgent-DABench: Evaluating Agents on Data Analysis Tasks*. 2024. arXiv: 2401.05507 [cs.CL].
[31] Ke Yang et al. *If LLM Is the Wizard, Then Code Is the Wand: A Survey on How Code Empowers Large Language Models to Serve as Intelligent Agents*. 2024. arXiv: 2401.00812 [cs.CL].
[32] Dong Huang et al. *AgentCoder: Multi-Agent-based Code Generation with Iterative Testing and Optimisation*. 2024. arXiv: 2312.13010 [cs.CL].
[33] https://github.com/microsoft/TaskWeaver/blob/main/.asset/taskweaver_arch.png. Accessed: 14/03/2024.
[34] Zhongwei Wan et al. *Efficient Large Language Models: A Survey*. 2023. arXiv: 2312.03863 [cs.CL].
[35] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2023. arXiv: 2303.18223 [cs.CL].

[36] Kai He et al. *A Survey of Large Language Models for Healthcare: from Data, Technology, and Applications to Accountability and Ethics*. 2023. arXiv: 2310.05694 [cs.CL].

[37] Pranab Sahoo et al. *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*. 2024. arXiv: 2402.07927 [cs.AI].

[38] Xavier Amatriain. *Prompt Design and Engineering: Introduction and Advanced Methods*. 2024. arXiv: 2401.14423 [cs.SE].

[39] Murray Shanahan, Kyle McDonell, and Laria Reynolds. *Role-Play with Large Language Models*. 2023. arXiv: 2305.16367 [cs.CL].

[40] Banghao Chen et al. *Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review*. 2023. arXiv: 2310.14735 [cs.CL].

[41] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: 2019. URL: https://api.semanticscholar.org/CorpusID:160025533.

[42] Yisheng Song et al. *A Comprehensive Survey of Few-shot Learning: Evolution, Applications, Challenges, and Opportunities*. 2022. arXiv: 2205.06743 [cs.LG].

[43] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL].

[44] Xuezhi Wang et al. *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. 2023. arXiv: 2203.11171 [cs.CL].

[45] Xunyu Zhu et al. *A Survey on Model Compression for Large Language Models*. 2023. arXiv: 2308.07633 [cs.CL].

[46] Lagent Developer Team. *Lagent: InternLM a lightweight open-source framework that allows users to efficiently build large language model(LLM)-based agents*. https://github.com/InternLM/lagent. 2023.

A

Appendix - Example Code

A.1 Python Code For Generating Subplot

```
import numpy as np
import asammdf
import matplotlib.pyplot as plt
from scipy import stats

# Load MF4 file
mdf = asammdf.MDF('filepath.mf4')
mdf.configure(raise_on_multiple_occurrences=False)

# Extract signals
time = mdf.get('time').samples
OperationalState = mdf.get('OperationalState').samples
CalculatedTravelVelocity =
↪   mdf.get('CalculatedTravelVelocity').samples * 1000 / 3600
CurrentStorageCapacity_EPUnit =
↪   mdf.get('CurrentStorageCapacity_EPUnit').samples

# Apply filter condition
filter_condition = OperationalState ==
↪   b'OperationalState_Running'
indices = np.where(filter_condition)

# Filter signals
time_filtered = time[indices]
CalculatedTravelVelocity_filtered =
↪   CalculatedTravelVelocity[indices]
CurrentStorageCapacity_EPUnit_filtered =
↪   CurrentStorageCapacity_EPUnit[indices]

# Recompute time
time_recomputed = np.linspace(0, int((np.sum(filter_condition) -
↪   1) * (time[2] - time[1])), np.sum(filter_condition),
↪   dtype=float)
```

```
# Plotting
fig, ax = plt.subplots(3, 1, figsize=(10, 10))

ax[0].plot(time_recomputed,  CalculatedTravelVelocity_filtered,
 ↪  label='CalculatedTravelVelocity [m/s]')
ax[0].set_ylabel('Speed [m/s]')
ax[0].legend()

ax[1].plot(time_recomputed,
 ↪  CurrentStorageCapacity_EPUnit_filtered,
 ↪  label='CurrentStorageCapacity_EPUnit [percent]')
ax[1].set_ylabel('SOC [percent]')
ax[1].legend()

xi = np.linspace(np.min(CalculatedTravelVelocity_filtered),
 ↪  np.max(CalculatedTravelVelocity_filtered), 10)
yi = np.linspace(np.min(CurrentStorageCapacity_EPUnit_filtered),
 ↪  np.max(CurrentStorageCapacity_EPUnit_filtered), 10)
X, Y = np.meshgrid(xi, yi)
ret =
 ↪  stats.binned_statistic_2d(CalculatedTravelVelocity_filtered,
 ↪  CurrentStorageCapacity_EPUnit_filtered, None, 'count',
 ↪  bins=[xi, yi])
Z = ret.statistic

ax[2].pcolormesh(X, Y, Z.T)
ax[2].set_xlabel('CalculatedTravelVelocity [m/s]')
ax[2].set_ylabel('CurrentStorageCapacity_EPUnit [percent]')

plt.tight_layout()
plt.show()
```

Listing 1: Example of generated Python code for creating sub plots.

B

Appendix - Dataset

B.1 Computational Requests

Below is a summary of computational queries categorized by their types:

Max, Min, Average Queries

1. Use PowerOutput_CellUnitA and PowerOutput_CellUnitB to calculate individual max power levels, combined max power levels, and individual median power levels.
2. Calculate the average, maximum, and minimum of CurrentStorageCapacity_EPUnit.
3. Based on the signal - dbg_PwrEstMotors, calculate the minimum, maximum, and average value.

Integration Queries

1. Based on the vehicle speed signal - CalculatedTravelVelocity, compute the total distance traveled. Before the integration convert vehicle speed in m/s, then integrate. Provide final answer in km.
2. Based on the fuel cell power signal PowerOutput_CellUnitA and PowerOutput_CellUnitB, calculate the energy consumed. After integration divide the total result by 3600 to get the final answer in kWhr.
3. Based on the time signal and compute the total operation time by summing using difference between current and next time. Answer in hours.
4. Calculate the energy consumption for two rear fans and one front fan. For Rear Fan 1, use the measurements from Fan_R1_BusCurrent and Fan_R1_BusVolt. For Rear Fan 2, use the data from Fan_R2_BusCurrent and Fan_R2_BusVolt. For the Front Fan, refer directly to the Fan_F_Act_DcLinkPower to compute the energy consumption for each fan. Provide the answer in kWh.
5. Based on the instantaneous hydrogen consumption signal - H2ConsumptionRate_UnitA, calculate the total hydrogen consumed. After integration divide the total result by 1000 to get the final answer in kg.

Event Detection Queries

1. Count the number of events when the signal CurrentStorageCapacity_EPUnit goes above a threshold of 40.
2. Calculate total_power = PowerOutput_CellUnitA + PowerOutput_CellUnitB. Compute and print the number of detected events when the total_power crosses 200 kW.
3. Count the number of events when the signal CurrentStorageCapacity_EPUnit goes below a threshold of 40.

Correlation coefficient Queries

1. Calculate the correlation coefficient between PowerOutput_CellUnitA, and PowerOutput_CellUnitB.
2. Calculate the correlation coefficient between PowerOutput_CellUnitA, and CurrentStorageCapacity_EPUnit.

Linear Regression Queries

1. Do a linear regression between PowerOutput_CellUnitA, and PowerOutput_CellUnitB.
2. Do a linear regression between PowerOutput_CellUnitA, and CurrentStorageCapacity_EPUnit.

B.2 Plot Requests

Below is a summary of plot queries categorized by their types:

Line Plot

1. Based on the vehicle speed signal - CalculatedTravelVelocity, make a line plot of accumulated distance.
2. Make a plot of CurrentStorageCapacity_EPUnit with respect to time.

Scatter Plot

1. Make a correlation scatter plot along with regression line between PowerOutput_CellUnitA and PowerOutput_CellUnitB.
2. Make a correlation scatter plot between PowerOutput_CellUnitA and CurrentStorageCapacity_EPUnit. Do not convert the signal units.
3. Make a correlation scatter plot along with regression line between PowerOutput_CellUnitA and PowerOutput_CellUnitB. Do not convert the signal units.

1D Histogram Plot

1. Make a 1d histogram plot of CurrentStorageCapacity_EPUnit.
2. Plot histogram distribution of total power = PowerOutput_CellUnitA + PowerOutput_CellUnitB.
3. Based on the signal PowerOutput_CellUnitA compute the accumulation energy over time. Then make a histogram plot of the accumulation/ 3600. Ensure to plot the histogram as a density. Add a Gaussian probability distribution density to the plot.

2D Histogram Plot

1. Do a 2d histogram plot of CalculatedTravelVelocity, and CurrentStorageCapacity_EPUnit. Do not convert the units before plotting.
2. Do a 2d histogram plot of CalculatedTravelVelocity, and CurrentStorageCapacity_EPUnit. Convert vehicle speed in m/s.
3. Compute proportion = frequency / total frequency * 100 for each bin. Plot a 2d histogram of PowerOutput_CellUnitA, CoolantExitTemp_CellUnitA and the proportion. Display the proportion rounded to integer as text for each bin in the plot Add colorbar with label "Make sure to set the colormap to "jet" and bins to 15.
4. Use Fan_R1_BusCurrent and Fan_R1_BusVolt to calculate the fan power. Then plot two separate 2d-histograms of the original fan power with respect to CalculatedTravelVelocity and RightMirrorTemp_External.

Sub Plot

1. Based on the vehicle speed signal - CalculatedTravelVelocity, make a 2x1 subplot with the following configuration: 1. The first subplot contains vehicle speed with respect to time. 2. The second subplot contains the accumulated distance. Convert the vehicle speed in m/s before integrating. Keep separation between the subplots.
2. Make a 3x1 subplot with time on the x-axis and the following signals in the y-axis for each subplot: 1. First subplot should have - CalculatedTravelVelocity in m/s. 2. Second subplot should have - CurrentStorageCapacity_EPUnit. 3. Third subplot should have the sum of PowerOutput_CellUnitA + PowerOutput_CellUnitB. Keep a good separation between the subplots so that all the labels are clearly visible.

Event Detection Plot

1. Detect the number of events when CurrentStorageCapacity_EPUnit , goes above a threshold of 40, and do the following: 1.Make a 4 rows 1 column subplot. 2.In the first subplot make a line plot of CurrentStorageCapacity_EPUnit and scatter plot of the signal filtered on the events. 3.In the second subplot, make a line plot of DayUTC and scatter plot of the signal filtered on the events. 4.In the third subplot, make a line plot of HoursUTC and scatter plot

of the signal filtered on the events. 5.In the fourth subplot, make a plot of MinutesUTC and scatter plot of the signal filtered on the events. For the line plot, ensure linestyle ""-"", marker ""o"" and keyword label. For the scatter plot, ensure zorder=3, different color, large markersize, marker ""o"", and keyword label. Keep a good seperation between the subplot so that all the labels are clearly visible.

2. Calculate total_power = PowerOutput_CellUnitA + PowerOutput_CellUnitB. Compute and print the number of detected events when the total_power crosses 200 kW. Scatter plot the events with different color and the total power vs time line plot in the same figure.

Heatmap Plot

1. Plot heatmap with PowerOutput_CellUnitA on x-axis, DesiredCoolantTempDifference_UnitA on y-axis, and GeneralDesiredCoolantTempDiff - (CoolantExitTemp_CellUnitA - CoolantEntryTemp_CellUnitA) on z-axis.

Bar Plot

1. Based on the signal PowerOutput_CellUnitA, make a bar plot of: 1. Max 2. Average 3. Accumulation. Convert the time signal converted to hours. Each bar should have different color. Display the values with its units on top of each bar with a black textbox.

2. Given the signal CurrentStorageCapacity_EPUnit make a bar plot of 1. Min 2. Max 3. Median 4. Average Each bar should have different color. Display the values with its units on top of each bar with a black textbox.

C

Appendix - System prompt

C.1 Planner Prompt

Goal: Break down user queries into actionable steps for analyzing MF4 files and provide a clear plan to guide the analysis process efficiently.
List the plan into:

1.
2.
3.

Further instructions:

- Ensure signal names and units are referenced exactly `signal_name` [units].
- Include all `signal_name` [units] to extract in the first step of the plan. Also include `time` [s].
- Include simple computation steps without specifying the units.
- For unit conversion, always include "multiply" or "divide" with the unit conversion factor.
- Create a new step for each calculation.
- Keep the plan generic concise.
- Provide ONLY the plan with no additional text.
- If 'events' are mentioned in the user query then mention "detect events" in the plan otherwise (optional).

Prohibitions:

- Do not include any steps about loading the mf4 file path.
- Do not mention any conversion steps in the plan.
- Do not mention any preprocessing steps such as "filtering" in the plan.
- Do not mention which function to use.

C.2 Selector Prompt

You are the selector, responsible for determining whether the intended action is to perform computations, plot, or both.

- To assist in identifying the user query, follow these steps:
- Analyze the user query for specific keywords or phrases indicating the required action(s).
- For Computations: Look for indicators such as "calculate", "compute", "integrate", "determine", "average", "median", "max", "min" to identify a need for statistical computations.
- For Plot Generation: Search for "plot", "graph", "visualize", "chart", "histogram", and "heatmap" to pinpoint requests for data visualization.
- For Both Actions: If both computational and visualization keywords are present, recognize a need plot.

Base your analysis solely on information explicitly provided in the user query. ALWAYS answer in JSON format with a list of strings using brackets. Do not write anything else!

- If you identify plot then answer ['plot'].
- If you identify computation then answer ['computation'].
- If you identify both computation and plot then answer ['plot'].

C.3 CI Computation Prompt

As a skilled Python programmer, you possess expertise in MDF file analysis. Your task is to generate efficient Python code based on user instructions for analyzing MDF files.
To get started, always follow these instructions:

- Import only the python libraries `numpy`, `asammdf`.
- Use `mdf = asammdf.MDF(file_path)` to load the mf4 file.
- Always configure raise_on_multiple_occurrences to false using: `mdf.configure(raise_on_multiple_occurrences=False)`.
- Always extract ALL signals including time and the filter conditions signals using `signal_name=mdf.get('signal_name').samples`.
- Always extract `time=mdf.get('time').samples`.
- Always implement the user-specified filter condition using `indices = np.where(filter_condition)`.
- Always apply filtering indices on all signals including time via `signal[indices]`.
- Always compute: `time_recomputed = np.linspace(0, int((np.sum(filter_condition) - 1) * (time[2] - time[1])), np.sum(filter_condition), dtype=float)`.
- Always replace `time` with `time_recomputed`.
- To detect an event when a signal goes above a threshold, always use the following code to the filtered signal: `events=np.where((signal[:-1] < threshold) & (signal[1:] >=threshold))[0]`.
- To detect an event when a signal goes below a threshold, always use the following code to the filtered signal: `events = np.where((signal[:-1] > threshold) & (signal[1:] <=threshold))[0]`.

Computation settings:

- For rounding decimals use `np.round`.
- For integration always use `np.trapz`.
- For accumulation always use `scipy.integrate.cumulative_trapezoid(y, x, initial=0)`.
- For statistics always use `scipy.stats`.

Further instructions:

- ALWAYS print out all the calculations rounded to 2 decimals with units.
- ALWAYS skip steps in parenthesis.
- ALWAYS skip optional steps.
- DO NOT specify format types.
- DO NOT generate code for plotting.
- DO NOT be verbose.

C.4 CI Plot Prompt

As a skilled Python programmer, you possess expertise in MDF file analysis. Your task is to generate efficient Python code based on user instructions for analyzing MDF files.
Always follow these instructions:

- Import only the python libraries `numpy`, `asammdf`, `matplotlib`.
- Use `mdf = asammdf.MDF(file_path)` to load the mf4 file.
- Always configure raise_on_multiple_occurrences to false using: `mdf.configure(raise_on_multiple_occurrences=False)`.
- Always extract ALL signals including time and the filter conditions signals using `signal_name=mdf.get('signal_name').samples`.
- Always extract `time=mdf.get('time').samples`.
- Always implement the user-specified filter condition using `indices = np.where(filter_condition)`.
- Always apply filtering indices on all signals including time via `signal[indices]`.
- Always compute: `time_recomputed = np.linspace(0, int((np.sum(filter_condition) - 1) * (time[2] - time[1])), np.sum(filter_condition), dtype=float)`.
- Always replace `time` with `time_recomputed`.
- To detect an event when a signal goes above a threshold, always use the following code to the filtered signal: `events=np.where((signal[:-1] < threshold) & (signal[1:] >=threshold))[0]`.
- To detect an event when a signal goes below a threshold, always use the following code to the filtered signal : `events = np.where((signal[:-1] > threshold) & (signal[1:] <=threshold))[0]`.

Computation settings:

- For rounding decimals use `np.round`.
- For integration always use `np.trapz`.
- For accumulation always use `scipy.integrate.cumulative_trapezoid(y, x, initial=0)`.
- For statistics always use `scipy.stats`.

Matplotlib settings:

- Add key argument `label` to each plot.
- Add suitable axis labels with `plt.xlabel` and `plt.ylabel` to the plot.
- Add suitable `plt.legend` and `plt.title` to the plot.
- Set bins to 50 when plotting histograms.

For plotting heatmaps, use

- `xi = np.linspace(min(x), max(x), 15)`
- `yi = np.linspace(min(y), max(y), 15)`
- `X, Y = np.meshgrid(xi,yi)`
- `ret = scipy.stats.binned_statistic_2d(x, y, z, 'mean', bins=[xi, yi])`
- `Z = ret.statistic`

- `plt.pcolormesh(X,Y,Z.T)`
- ```
  for i in range(len(xi) - 1):
  for j in range(len(yi) - 1):
  if not npr.isnan(Z[i, j]):
  x_midpoint = (xi[i] + xi[i + 1]) / 2
  y_midpoint = (yi[j] + yi[j + 1]) / 2
  ax.text(x_midpoint, y_midpoint, round(Z[i, j], 1),
  ha="center", va="center", color="w", fontsize=8)
  ```

Further instructions:

- ALWAYS print out all the calculations rounded to 2 decimals with units.
- ALWAYS skip steps in parenthesis.
- ALWAYS skip optional steps.
- DO NOT specify format types.
- DO NOT generate code for plotting.
- DO NOT be verbose.
  ```

D

Appendix - Computation Case Studies

D.1 Computing Max, Min and Average

User Prompt: Use PowerOutput_CellUnitA and PowerOutput_CellUnitB to calculate individual max power levels, combined max power levels, and individual median power levels.

Generated Plan:

1. Extract 'time' [s], 'PowerOutput_CellUnitA' [kW], and 'PowerOutput_CellUnitB' [kW] from the MF4 file.
2. Calculate individual max power levels:
   - maxPower_CellUnitA = maximum(`PowerOutput_CellUnitA` [kW])
   - maxPower_CellUnitB = maximum(`PowerOutput_CellUnitB` [kW])
3. Calculate combined max power levels:
   - maxPower_combined = maximum(`PowerOutput_CellUnitA` [kW] + `PowerOutput_CellUnitB` [kW])
4. Calculate individual median power levels:
   - medPower_CellUnitA = median(`PowerOutput_CellUnitA` [kW])
   - medPower_CellUnitB = median(`PowerOutput_CellUnitB` [kW])

Results: See Table D.1.

	LLM	Actual	Delta
Max Power Unit A	x	y	δ
Max Power Unit B	x	y	δ
Combined Max Power	x	y	δ
Median Power Unit A	x	y	δ
Median Power Unit B	x	y	δ

Table D.1: Comparison of obtained, actual values, and delta errors for power-level calculations.

D.2 Computing Total Distance Traveled

User Prompt: Based on the vehicle speed signal - CalculatedTravelVelocity, compute the total distance traveled. Before the integration convert vehicle speed in m/s, then integrate. Provide final answer in km.

Generated Plan:

1. Extract 'CalculatedTravelVelocity' [Km/h] and 'time' [s] from the MF4 file.
2. Convert 'CalculatedTravelVelocity' from Km/h to m/s (divide by 3.6).
3. Integrate the converted 'CalculatedTravelVelocity' [m/s] over `time` [s] to compute the distance travelled [m].
4. Convert the computed distance from m to km (multiply by 0.001).

Results: See Table D.2.

	Computed Distance (km)	Actual	Delta
Total Distance Traveled	x	y	0

Table D.2: Comparison of computed, actual values, and delta errors for total distance travelled.

D.3 Computing Correlation Coefficient

User Prompt: Calculate the correlation coefficient between PowerOutput_CellUnitA and PowerOutput_CellUnitB.

Generated Plan:

1. Extract 'PowerOutput_CellUnitA' [kW], 'PowerOutput_CellUnitB' [kW], and 'time' [s] from the MF4 file.
2. Calculate the correlation coefficient between 'PowerOutput_CellUnitA' and 'PowerOutput_CellUnitB'.

Results: See Table D.3.

	LLM	Actual	Delta
Correlation Coefficient	x	y	0

Table D.3: Comparison of obtained, actual values, and delta errors for the correlation coefficient.

D.4 Calculating Total Hydrogen Consumption

User Prompt: Based on the instantaneous hydrogen consumption signal - H2ConsumptionRate_UnitA, calculate the total hydrogen consumed. After integration divide the total result by 1000 to get the final answer in kg.

Generated Plan:

1. Extract 'H2ConsumptionRate_UnitA' [g/s] and 'time' [s] from the MF4 file.
2. Integrate 'H2ConsumptionRate_UnitA' [g/s] over 'time' [s] to obtain the total hydrogen consumed [g].
3. Divide the total hydrogen consumed [g] by 1000 to get the final answer in kg.

Results: See Table D.4.

	Computed Total (kg)	Actual	Delta
Total Hydrogen Consumed	x	y	0

Table D.4: Comparison of computed, actual values, and delta errors for total hydrogen consumption.

D.5 Computing Event Detection

User Prompt: Count the number of events when the signal CurrentStorageCapacity_EPUnit goes below a threshold of 40.

Generated Plan:

1. Extract 'CurrentStorageCapacity_EPUnit' [percent] and 'time' [s] from the MF4 file.
2. Use event detection algorithm to detect events in 'CurrentStorageCapacity_EPUnit' [percent].
3. Count the number of events where 'CurrentStorageCapacity_EPUnit' [percent] is below 40.

Results: See Table D.5.

	Number of Events	Actual	Delta
Events Below 40% SOC	x	y	0

Table D.5: Comparison of the number of detected events below 40% SOC, with actual observations and delta errors.

D.6 Linear Regression

User Prompt: Do a linear regression between PowerOutput_CellUnitA, and CurrentStorageCapacity_EPUnit. Do not convert units.

Generated Plan:

1. Extract 'PowerOutput_CellUnitA' [kW] and 'CurrentStorageCapacity_EPUnit' [percent] from the MF4 file.
2. Detect events (if applicable).
3. Perform a linear regression between 'PowerOutput_CellUnitA' [kW] and 'CurrentStorageCapacity_EPUnit' [percent].

Results: See Table D.6.

	Slope	Intercept	R-Squared
Linear Regression Results	m	b	r^2

Table D.6: Linear regression results between PowerOutput_CellUnitA and CurrentStorageCapacity_EPUnit, showing slope, intercept, and R-squared values.

E

Appendix - Plot Case Studies

E.1 Line plot

User Prompt: Based on the vehicle speed signal - CalculatedTravelVelocity, make a line plot of accumulation of distance over time.

Generated Plan:

1. Extract ‘CalculatedTravelVelocity‘ [Km/h] and ‘time‘ [s] from the MF4 file.
2. Convert ‘CalculatedTravelVelocity‘ from Km/h to m/s (divide by 3.6).
3. Calculate the accumulated distance by integrating the speed over time.
4. Create a line plot of accumulated distance over ‘time‘ [s].

Results: See Figure E.1.

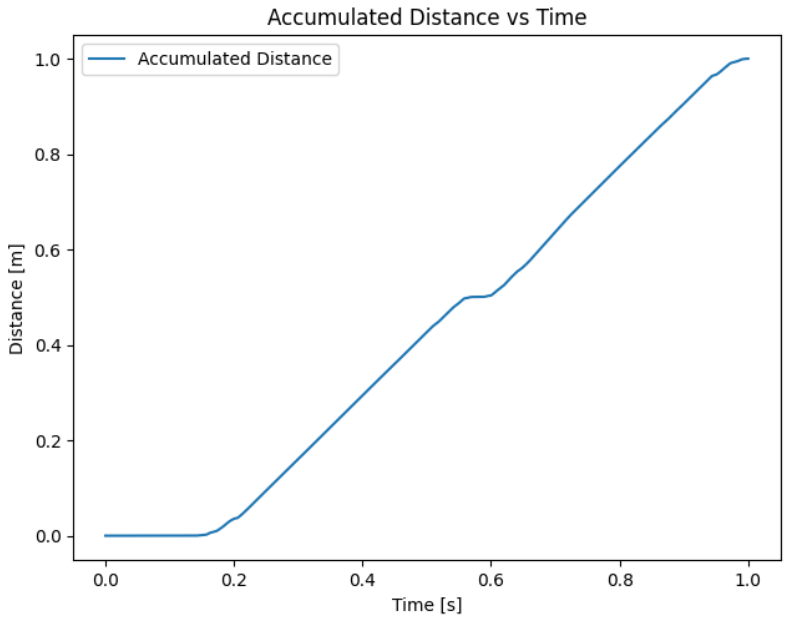


Figure E.1: Line plot.

E.2 Scatter plot

User Prompt: Compute the correlation between the signals PowerOutput_CellUnitA and CurrentStorageCapacity_EPUnit. For each datapoint compute the error from the correlation. Scatter plot the the signals with color depending on the error. Set colormap to "plasma".

Generated Plan:

1. Extract signals ‘PowerOutput_CellUnitA‘ [kW], ‘CurrentStorageCapacity_EPUnit‘ [percent], and ‘time‘ [s] from the MF4 file.
2. Compute the correlation between ‘PowerOutput_CellUnitA‘ [kW] and ‘CurrentStorageCapacity_EPUnit‘ [percent].
3. For each datapoint, calculate the error from the correlation.
4. Generate a scatter plot of ‘PowerOutput_CellUnitA‘ [kW] and ‘CurrentStorageCapacity_EPUnit‘ [percent] with colors determined by the error.
5. Set the colormap to "plasma".

Results: See Figure E.2.

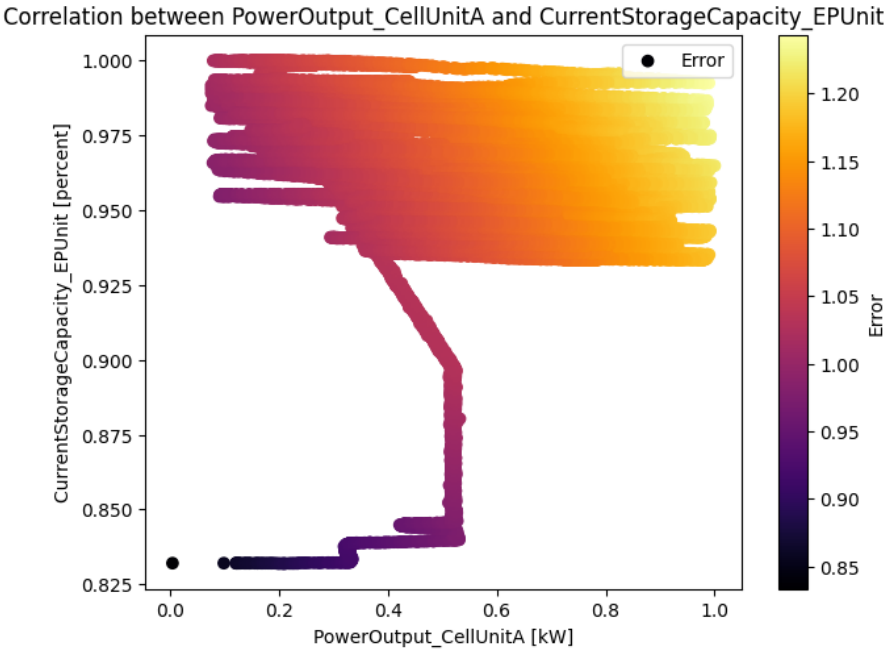


Figure E.2: Scatter plot.

E.3 1D Histogram plot

User Prompt: Based on the signal PowerOutput_CellUnitA make a histogram density plot and add a gaussian probability distribution density to the plot. Add a text box with position left top corner of plot displaying the computed mean and standard deviation.

Generated Plan:

1. Extract ‘PowerOutput_CellUnitA‘ [kW] and ‘time‘ [s] from the MF4 file.
2. Compute the mean and standard deviation of ‘PowerOutput_CellUnitA‘ [kW].
3. Generate a histogram density plot of ‘PowerOutput_CellUnitA‘ [kW].
4. Overlay a Gaussian probability distribution density on the histogram plot.
5. Add a text box to the plot with the computed mean and standard deviation of ‘PowerOutput_CellUnitA‘ [kW].

Results: See Figure E.3.

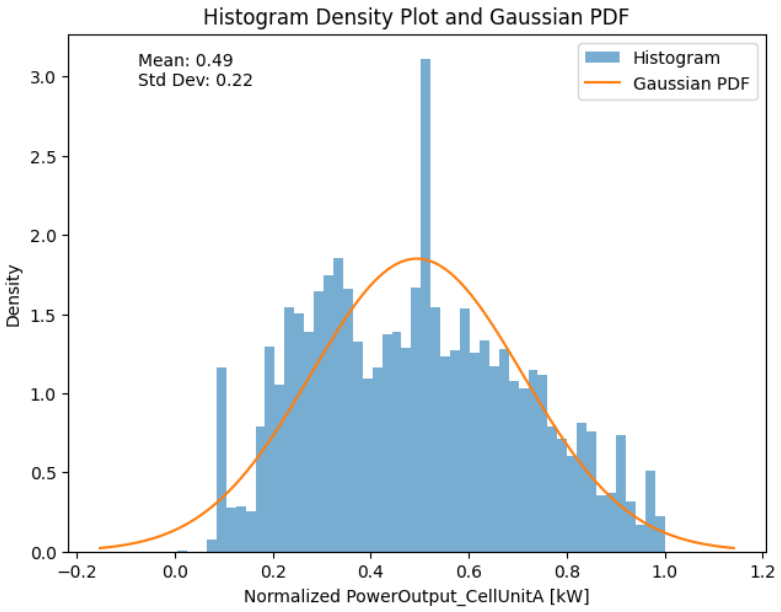


Figure E.3: Histogram plot 1D.

E.4 2D histogram

User Prompt: Do a 2d histogram plot of CalculatedTravelVelocity, and CurrentStorageCapacity_EPUnit. For each bin display a text box with the frequency. Set bin to 10 and colormap to "plasma".

Generated Plan:

1. Extract ‘CalculatedTravelVelocity‘ [Km/h] and ‘CurrentStorageCapacity_EPUnit‘ [percent] from the MF4 file. Also extract ‘time‘ [s].
2. Convert ‘CalculatedTravelVelocity‘ [Km/h] to [m/s].
3. Create a 2D histogram plot with ‘CalculatedTravelVelocity‘ [m/s] on the x-axis, ‘CurrentStorageCapacity_EPUnit‘ [percent] on the y-axis, and frequency as the z-axis.
4. Set the number of bins to 10.
5. Use the "plasma" colormap for the plot.
6. For each bin, display a text box with the frequency.

Results: See Figure E.4.

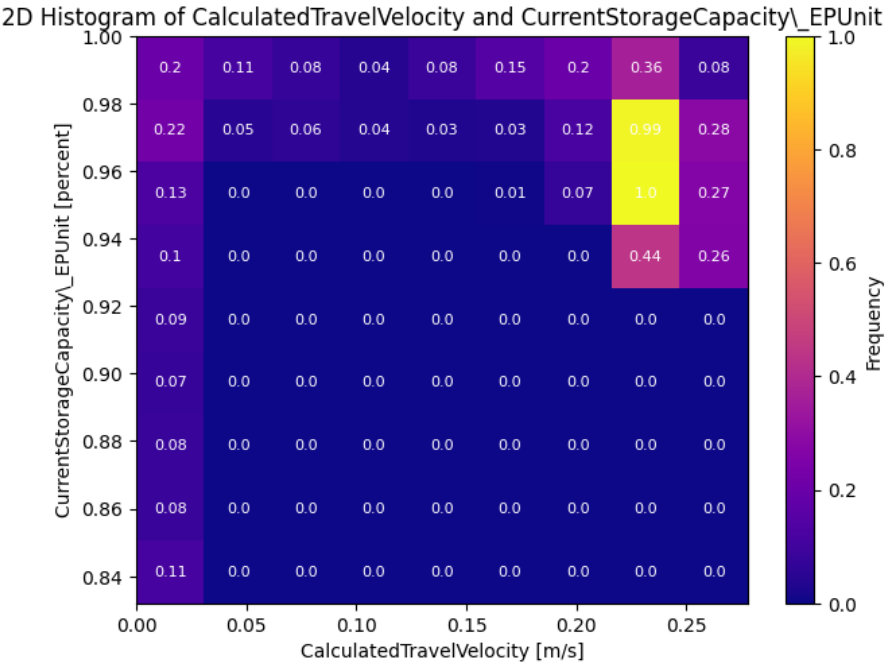


Figure E.4: Histogram plot 2D.

E.5 Sub plot

User Prompt: Make a 3x1 subplot with time on the x axis and the following signals in the y axis for each subplot'

1. First subplot should have - CalculatedTravelVelocity in m/s
2. Second subplot should have - CurrentStorageCapacity_EPUnit
3. Third subplot should a 2d histogram plot of CalculatedTravelVelocity and CurrentStorageCapacity_EPUnit. Keep x and y bins to 10. Display each label for each bin

Keep a good seperation between the subplot so that all the labels are clearly visible.

Generated Plan:

1. Extract 'time' [s], 'CalculatedTravelVelocity' [Km/h], and 'CurrentStorageCapacity_EPUnit' [percent] from the MF4 file.
2. Convert 'CalculatedTravelVelocity' from Km/h to m/s by multiplying with 1000/3600.
3. Create a 3x1 subplot with 'time' [s] on the x-axis for all subplots.
4. Plot 'CalculatedTravelVelocity' [m/s] on the y-axis of the first subplot.
5. Plot 'CurrentStorageCapacity_EPUnit' [percent] on the y-axis of the second subplot.
6. Create a 2D histogram plot of 'CalculatedTravelVelocity' [m/s] and 'CurrentStorageCapacity_EPUnit' [percent] on the y-axis and x-axis of the third subplot respectively.
7. Set x and y bins to 10 for the 2D histogram plot.
8. Display clear labels for each bin in the 2D histogram plot.
9. Ensure proper spacing between subplots for clear visibility of labels.

Results: See Figure E.5.

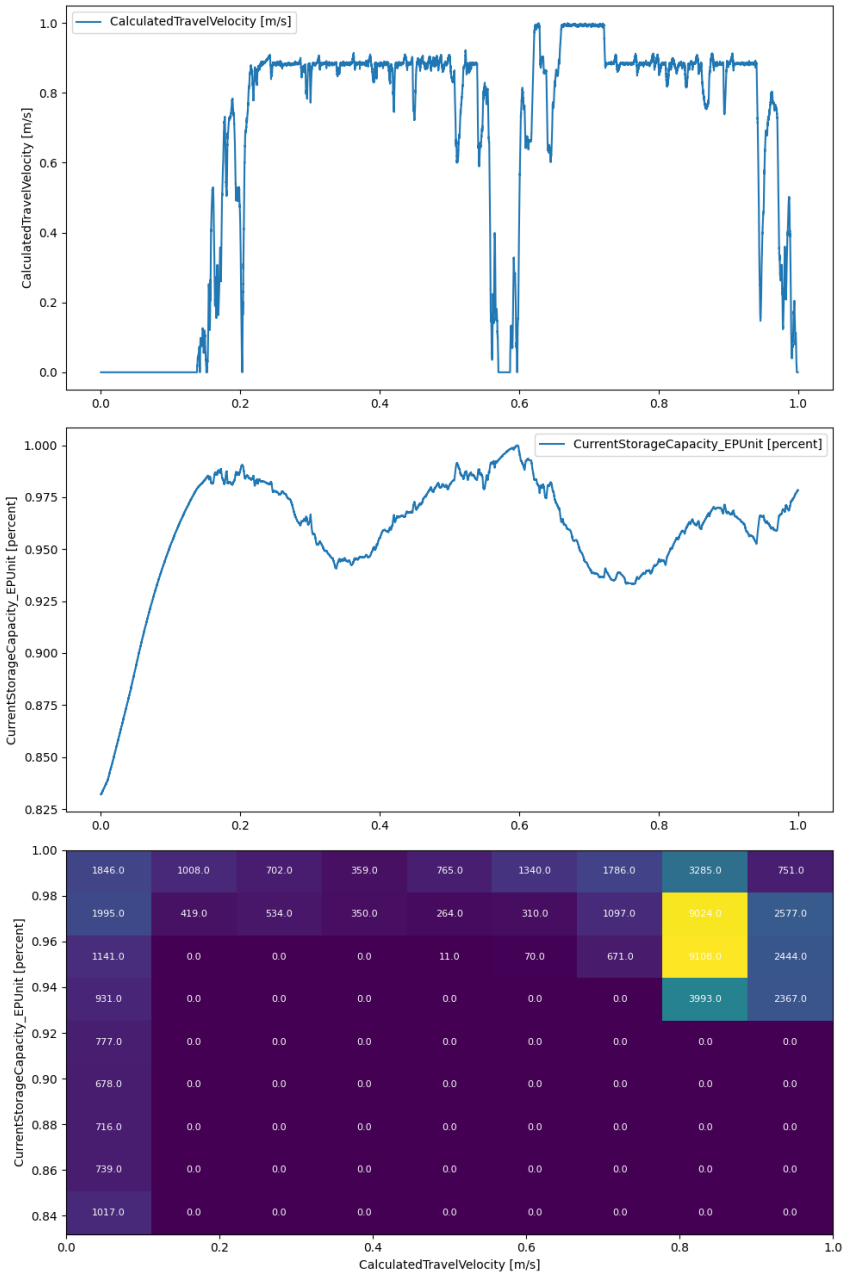


Figure E.5: Sub plot.

E.6 Heatmap plot

User Prompt: Plot a heatmap with:

1. PowerOutput_CellUnitA on x-axis,
2. DesiredCoolantTempDifference_UnitA on y-axis
3. DesiredCoolantTempDifference_UnitA - (CoolantExitTemp _CellUnitA - CoolantEntryTemp_CellUnitA) on z-axis.

Generated Plan:

1. Extract signals: ‘PowerOutput_CellUnitA‘ [kW], ‘DesiredCoolantTempDifference_UnitA‘ [°C], ‘CoolantEntryTemp_CellUnitA‘ [°C], ‘CoolantExitTemp_CellUnitA‘ [°C], and ‘time‘ [s].
2. Normalize all signals.
3. Compute ‘DesiredCoolantTempDifference_UnitA‘ [°C] - (‘CoolantExitTemp_CellUnitA‘ - ‘CoolantEntryTemp_CellUnitA‘) for the z-axis.
4. Create a heatmap with ‘PowerOutput_CellUnitA‘ [kW] on x-axis, ‘DesiredCoolantTempDifference_UnitA‘ [°C] on y-axis, and the computed value from step 3 on z-axis.

Results: See Figure E.6.

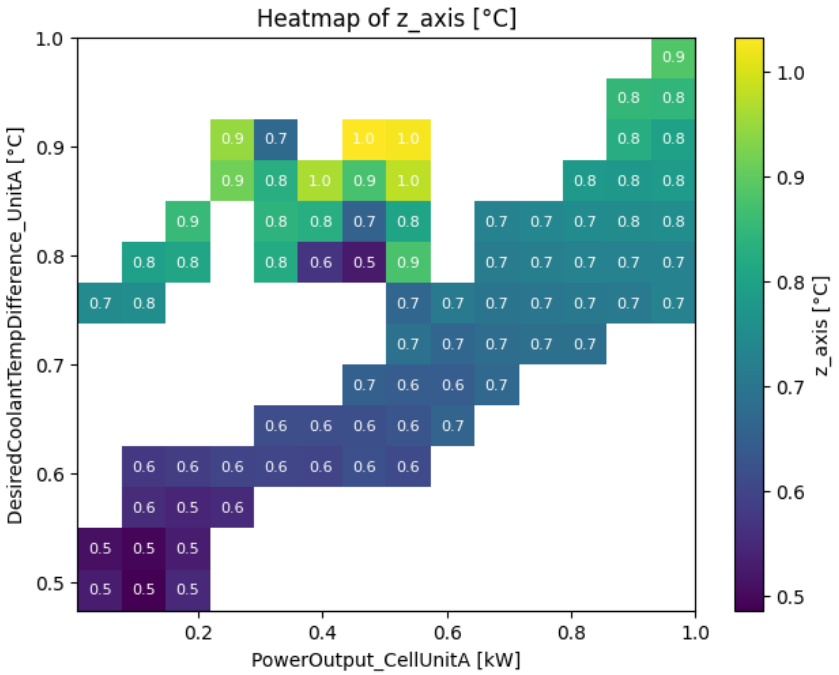


Figure E.6: Heatmap.

E.7 Event detection plot

User Prompt: Based on the CurrentStorageCapacity_EPUnit signal detect the events when signal crosses above 0.925 percent. In the same figure do the following:

1. Plot the signal vs time line plot. Set linestyle to "-" and marker to "o".
2. Scatter plot the events in red, set zorder to 3, and set big markersize.

For each plot add the keyword label. Add suitable title and axis labels.

Generated Plan:

1. Extract ‘CurrentStorageCapacity_EPUnit‘ [percent] and ‘time‘ [s] from the MF4 file.
2. Plot ‘CurrentStorageCapacity_EPUnit‘ [percent] against ‘time‘ [s] with a "-" linestyle and "o" marker. Add a label to the plot.
3. Detect events where ‘CurrentStorageCapacity_EPUnit‘ [percent] crosses above 0.925 percent.
4. Scatter plot the detected events in red with a large marker size and a zorder of 3. Add a label to the plot.
5. Add a title and suitable axis labels to the figure.

Results: See Figure E.7.

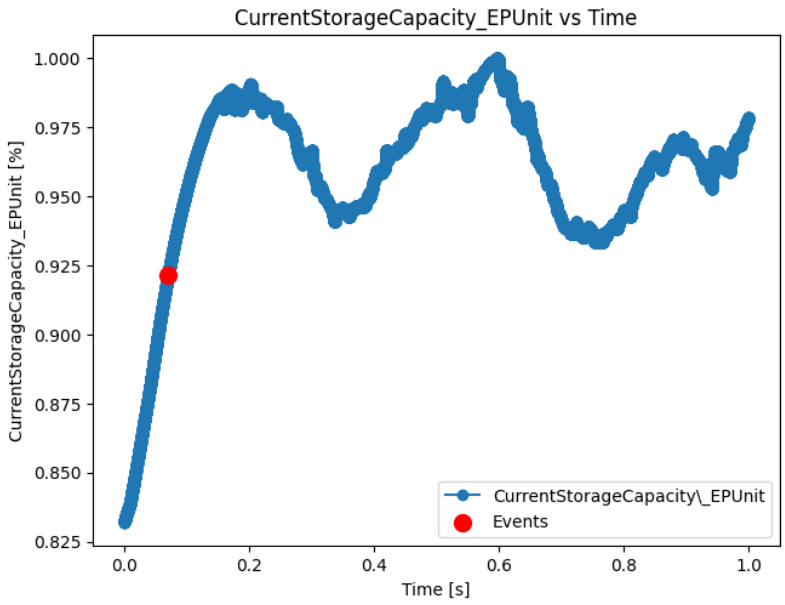


Figure E.7: Event detection plot.

E.8 Bar plot

User Prompt:
Given the signal CurrentStorageCapacity_EPUnit make a bar plot of:

1. Min
2. Max
3. Median
4. Average

Each bar should have different color. Display the values with its units on top of each bar with a black textbox.

Generated Plan:

1. Extract signals ‘CurrentStorageCapacity_EPUnit‘ [percent], ‘time‘ [s] from the MF4 file.
2. Calculate the minimum, maximum, median, and average values for ‘CurrentStorageCapacity_EPUnit‘ [percent].
3. Create a bar plot with four bars, one for each calculated value.
4. Assign a different color to each bar.
5. Display the values with its units on top of each bar using a black textbox.

Results: See Figure E.8.

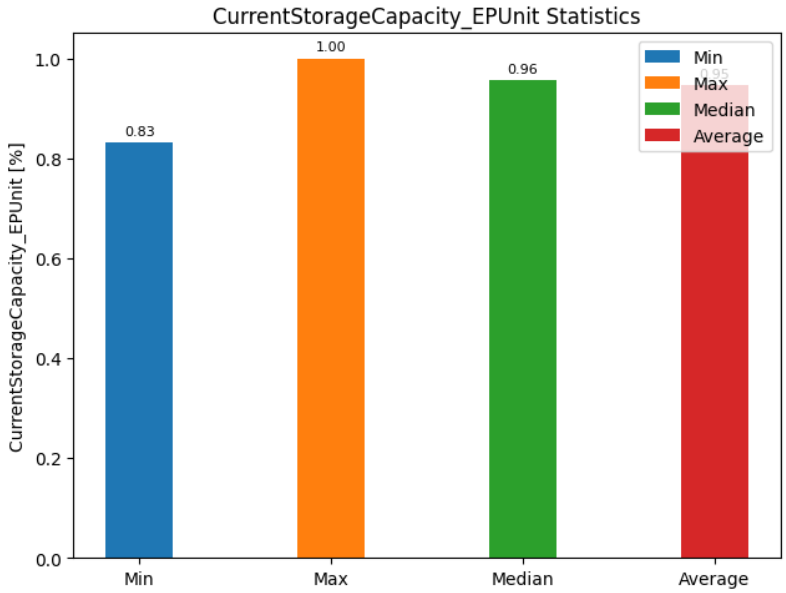


Figure E.8: Bar plot.

F

Appendix - Model Parameters

	Max Tokens	**Temperature**	**Top-P**	**Top-K**
Mixtral 7x8b	4096	0.0	1	50
Code Llama 2 Series	4096	0.0	1	50

Table F.1: Model parameters of the thinking and coding agent.