UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Intelligent transport system support for electric vehicle routing services

Trabajo Fin de Máster

Máster Universitario en Ingeniería Industrial-Màster Universitari en Enginyeria Industrial

AUTOR: Pintado Font, Fernando

Tutores: Alonso Ábalos, José Miguel; Ponci, Ferdinanda; Gümrükcü, Erdem

CURSO ACADÉMICO: 2023/2024

Intelligent Transport System Support for Electric Vehicle Routing Services

Verkehrssysteme zur Unterstützung von Routing-Diensten für Elektrofahrzeuge

Fernando Pintado Font
Matriculation Number: 449954

Master Thesis

The present work was submitted to
RWTH Aachen University
Faculty of Electrical Engineering and Information Technology
Institute for Automation of Complex Power Systems
Univ.-Prof. Ferdinanda Ponci, Ph. D.

Supervisor: Erdem Gümrükcü, M.Sc.

Eidesstattliche Versicherung

Ich, Fernando Pintado Font (Matrikelnummer 449954), versichere hiermit an Eides Statt, dass ich die vorliegende Masterarbeit mit dem Titel

Intelligent Transport System Support for Electric Vehicle Routing Services

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Valencia, 05/12/2023 — Ort, Datum

 — Unterschrift

Belehrung

§156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit einer Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des §158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Valencia, 05/12/2023 — Ort, Datum

 — Unterschrift

Kurzfassung

Die Elektrifizierung des Verkehrs ist für die Verringerung der Treibhausgasemissionen und die Eindämmung des Klimawandels zunehmend wichtig. Elektrofahrzeuge sind für diesen Übergang von zentraler Bedeutung, da sie keine Abgasemissionen verursachen und mit erneuerbaren Energiequellen kompatibel sind. Trotz dieser Vorteile wird die massenhafte Einführung von Elektrofahrzeugen durch Bedenken hinsichtlich der begrenzten Reichweite und der unzureichenden Ladeinfrastruktur (LIS) behindert.

Digitale Technologien sind der Schlüssel zur Lösung dieser Probleme und ermöglichen einen Übergang zu einem kohlenstoffarmen Verkehrssystem. Intelligente Verkehrssysteme, die digitale Technologien wie Datenanalyse, das Internet der Dinge (IoT), und Cloud Computing nutzen, können den Fahrern von Elektrofahrzeugen unschätzbare Dienste anbieten. Diese Dienste können Echtzeitinformationen über Ladestationen, dynamische Routenplanung und Energiemanagementlösungen umfassen.

Diese Arbeit konzentriert sich auf die Entwicklung eines Intelligente Verkehrssysteme, das auf Elektrofahrzeug-Routing-Dienste zugeschnitten ist. Das System wird Verkehrs-, Wetter- und LIS-Daten integrieren, um den Ladevorgang und die Fahrpläne unter Berücksichtigung der Batteriekapazität des Fahrzeugs, des Ladezustands und der Benutzerpräferenzen zu optimieren. Das vorgeschlagene System soll das Nutzererlebnis verbessern, indem es genaue und zuverlässige Informationen über Ladestationen, Kosten, geschätzte Fahrtzeiten und mögliche Energieeinsparungen bietet.

Um diese Ziele zu erreichen, beginnt die Arbeit mit einem Überblick über die vorhandene Literatur zur Rolle von intelligenten Verkehrssystemen für die Effizienz von Elektrofahrzeugen und die Optimierung von Routen, wobei die Stärken und Grenzen der derzeitigen Methoden hervorgehoben werden. Anschließend wird ein neuartiger Rahmen für die Integration verschiedener Datenquellen in den Routing-Prozess entwickelt. Ziel ist die Erkennung von Ladestationen innerhalb eines bestimmten Suchraums und die Vorhersage der geschätzten Ankunftszeiten und des Ladezustands. Dieser Rahmen wird eine Analyse von Benutzerszenarien beinhalten, um seine Effektivität zu testen, und er wird einer strengen Leistungsbewertung unter verschiedenen Betriebsbedingungen unterzogen.

Die Ergebnisse dieser Arbeit sollen zur Entwicklung effizienterer und umweltfreundlicherer Verkehrssysteme beitragen. Durch die Nutzung der Leistungsfähigkeit digitaler Technologien und der Datenanalyse wird das geplante intelligente Verkehrssystem die Entwicklung hin zu einem umweltfreundlicheren Verkehrsökosystem unterstützen, den ökologischen Fußabdruck verringern und gleichzeitig das Fahrerlebnis für Nutzer von Elektrofahrzeugen verbessern.

Stichwörter: Elektrofahrzeuge, intelligente Verkehrssysteme, Verkehr, Klimawandel, Elektrifizierung, Mobilität, Routenoptimierung, Energiemanagement, Digitalisierung, Ladeinfrastruktur, Ladestationen.

Abstract

The electrification of transport is increasingly vital for reducing greenhouse gas emissions and mitigating climate change. Electric vehicles (EVs) are pivotal in this transition, offering zero tailpipe emissions and compatibility with renewable energy sources Despite these advantages, the mass adoption of EVs is hindered by concerns over limited range and insufficient Charging Infrastructure (CI).

Digital technologies hold the key to resolving these challenges, enabling a transition to a low-carbon transport system. Intelligent Transport Systems (ITS) that utilise digital technologies such as data analytics, Internet of Things (IoT), and cloud computing can offer invaluable services to EV drivers. These services can include real-time information on Charging Stations (CSs), dynamic route planning, and energy management solutions.

This thesis focuses on developing an ITS tailored for EV routing services. The system will integrate traffic, weather, and CI data to optimise charging and schedules, considering the vehicle's battery capacity, State of Charge (SoC), and user preferences. The proposed system aims to enhance the user experience by offering accurate and reliable information on charging locations, costs, estimated travel times, and potential energy savings.

To achieve these objectives, the thesis will commence with a review of existing literature on the role of ITSs for EV efficiency and route optimisation, highlighting the strengths and limitations of current methodologies. Subsequently, a novel framework for integrating diverse data sources into the routing process will be developed. The purpose is to detect CSs within a specific search space and predict Estimated Arrival Times (ETA) and SoCs. This framework will incorporate an analysis of user scenarios to test its effectiveness and will be subjected to rigorous performance evaluation under various operational conditions.

The findings from this thesis are expected to contribute to the advancement of more efficient and environmentally friendly transportation systems. By leveraging the power of digital technologies and data analytics, the envisioned ITS will support the move towards a greener transportation ecosystem, reducing the environmental footprint while enhancing the driving experience for EV users.

Keywords: Electric Vehicles, Intelligent Transport Systems, Transport, Climate Change, Electrification, Mobility, Route Optimisation, Energy Management, Digitalisation, Charging Infrastructure, Charging Stations.

Table of Contents

List of Figures

List of Tables

List of Listings

1. Introduction

1.1 Background and Motivation

1.1.1 Adopting Low-Carbon Transport for Climate Mitigation

Transportation serves as the cornerstone for facilitating commerce, enabling trade, and enhancing communication. It provides individuals with the essential capability to move from one location to another. It is a complex but affordable system of interconnections between manufacturing facilities and global consumer markets. It is not easy to think of our industrialised world as developed in aspects such as healthcare or education and different social or economic factors without the critical infrastructure of transport that sustains the world [1].

The current transportation model, while efficient in meeting the demands of society, poses a significant threat to the environment and human health. It is responsible for a quarter of the European Union's (EU´s) greenhouse gas emissions and contributes to air and noise pollution and habitat fragmentation. In 2020, road transport was responsible for most greenhouse gas emissions from the transportation sector in the EU, accounting for 77% of total emissions, including domestic transport and international bunkers [2].

Despite efforts to improve the energy efficiency of vehicles, the overall increase in transportation activity has resulted in a rise in emissions, mainly nitrogen oxides, which have adverse effects on health and the environment.

Given the EU's goal of achieving climate neutrality by 2050 [3], developing a sustainable mobility system is imperative. This system must prioritise cleaner, more active modes of transportation and cleaner fuels while reducing the need for mobility where feasible. Addressing transportation's environmental and health impacts is essential to ensuring a sustainable future for Europe and the world.

The solution to this problem is de-carbonising the transport system as soon as possible and in the least harmful way for the current mobility network. Electric Vehicles (EVs) are one of the most visible solutions being implemented in the world right now. Transitioning to these vehicles is essential to meeting the climate goals of the EU. This move towards a sustainable future can help reduce emissions and improve air quality, leading to a healthier planet.

Governments and organisations across the EU are offering various incentives such as subsidies, tax exemptions, and the development of Charging Infrastructure (CI) to encourage the adoption of low-carbon transport systems and EVs. For example, the Spanish government has implemented a policy framework that offers financial incentives amounting to as much as 7,000 euros [4] for consumers who purchase EVs and concurrently decommission their older, less efficient combustion-engine vehicles.

The installation and purchase of private chargers is also granted up to 30-40% of its cost (up to 100,000 euros) [5]. Some cities, such as Madrid, have also implemented low-emission zones to prevent vehicles that do not meet emissions standards from entering certain areas, as illustrated in Figure 1 [6]. These measures are crucial in creating an environment that promotes using low-carbon transport systems and EVs.

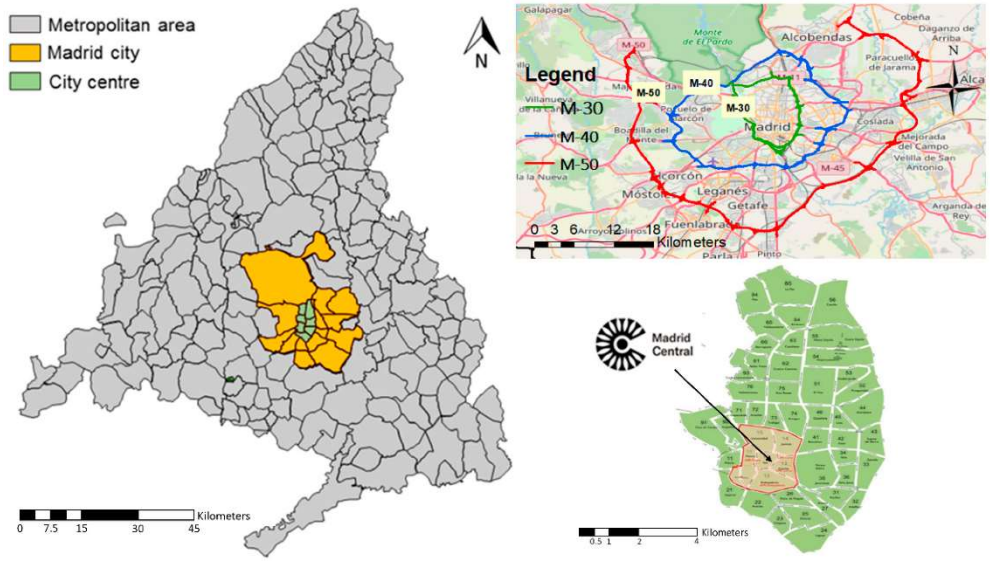

Figure 1: Map of Madrid's low emission zone area.

1.1.2 Challenges of EVs in Low-Carbon Transport

EVs have become more practical and cost-effective as technology advances, making them more accessible to the public. However, despite these positive developments, several challenges remain. The cost of EVs can be higher than traditional vehicles, even with incentives and subsidies. The availability of CI is another obstacle, particularly in rural areas. The development of CI requires significant investment to make the widespread adoption of EVs possible. Governments and private entities must invest in developing a robust and reliable CI to make EVs a more practical option for drivers.

Range anxiety is a psychological barrier that hinders the widespread adoption of EVs. This fear of running out of battery charge while driving on the road has become one of the primary concerns of potential EV buyers. The limited driving range of EVs compared to conventional Internal Combustion Engine (ICE) vehicles creates a sense of uncertainty and anxiety for many drivers. This issue can lead to a lack of trust in EVs and even deter potential buyers from considering them a viable option. Therefore, improving battery technology that allows for longer driving ranges is crucial in overcoming this barrier and promoting the mass adoption of EVs.

Charging time is also a significant consideration for long-distance journeys, where drivers may need to stop and recharge multiple times.

In Europe, the CI is often categorised by the type of connectors: Type 1, Type 2, and fast-charging options like Combined Charging System (CCS) and CHAdeMO [7]. Type 1 connectors are less common in Europe and are generally used for slower Alternate Current (AC) charging, taking up to 24 hours to fully charge an EV. Type 2 connectors are the European standard for AC charging and can fully charge an EV in 4-8 hours, depending on the power output of the Charging Station (CS).

For fast Direct Current (DC) charging, Europe primarily uses the CCS and CHAdeMO connectors. These can charge an EV up to 80% in a minimum of 30 minutes, making them ideal for long-distance travel. However, fast-CSs are less common than Type 2 stations and can be more expensive to use. The availability and type of CSs are crucial factors for potential EV buyers in Europe [8].

Figure 2 represents the different connectors for each market, showing the difference for AC and DC.

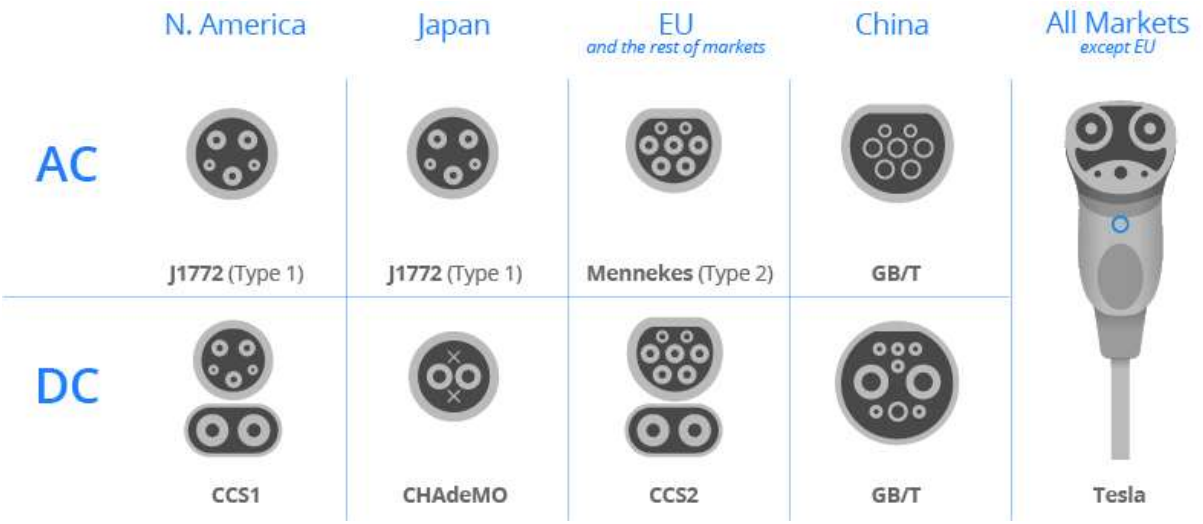

Figure 2: Comparison of EV charging connectors by region and current type [9].

Advances in battery technology are expected to reduce charging times further, making EVs even more convenient for users. Governments and private entities are investing in research and development to encourage the development of advanced battery technology. Some companies are also exploring alternative battery materials, such as solid-state batteries, which could offer even greater energy density and faster charging times [10].

1.2 Introduction to Intelligent Transport Systems

The transition towards a more sustainable transportation system is critical to mitigating the impacts of climate change. EVs have emerged as a promising solution, as they produce zero emissions and offer a cleaner alternative to traditional combustion engine vehicles. However, the success of EVs in achieving widespread adoption and usage depends on their usability, which includes factors such as range, CI, and driving experience.

To address these usability challenges, there is a need for innovative solutions that can enhance the usability and sustainability of EVs. Intelligent Transport Systems (ITSs) emerge as a significant player in this gap.

ITSs are a modern approach to road and traffic management by intertwining information, communication, and control technologies, see Figure 3. They aim for the integration of drivers, vehicles, and roads to support driving activities, therefore enhancing traffic efficiency, and reducing transit time. By delivering innovative services across different modes of transport, ITS enables better-informed decisions, promoting safer and more coordinated use of transport networks. Through core components like accurate location data, digital mapping, and seamless communication channels, ITSs improve real-time data collection, analysis, and distribution, helping in improved traffic management and user experience [11] [12].

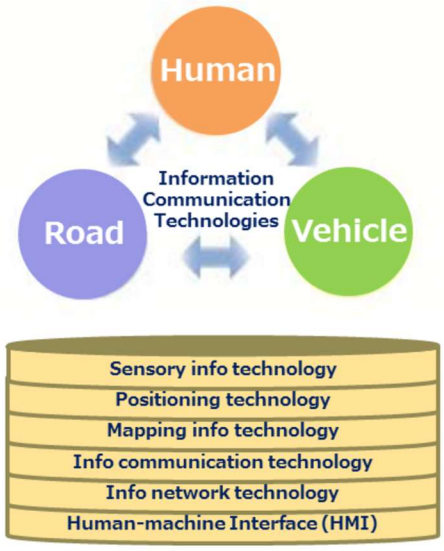

Figure 3: Components of ITSs [12].

They are expected to offer a future with safer roads and tackle Europe's rising emission and traffic congestion issues. By integrating in various Internet of Things (IoT) technologies across all passenger and freight transport modes, they help create a smoother, more efficient, and greener transport experience [13].

One key area where ITSs can enhance EVs' usability is the development of smarter CI. Using real-time data on traffic flow and energy demand, CI can be optimised to provide more efficient charging that minimises congestion and reduces energy consumption. Additionally, ITSs can be used to develop dynamic pricing models that incentivise off-peak charging, further reducing energy consumption and costs.

Another area where ITSs can improve EVs' sustainability is the development of more intelligent driving systems. By leveraging AI (Artificial Intelligence) and IoT technologies, intelligent driving systems can provide real-time data on traffic patterns and road conditions, allowing drivers to make more informed decisions about their driving routes and reducing energy consumption. Additionally, intelligent driving systems can be used to develop more efficient and eco-friendly driving techniques, such as regenerative braking and predictive speed control. Moreover, smart parking can be incorporated into ITSs, having control over the status of available space and monitoring parking data in real time [14].

ITSs can also play a crucial role in mitigating range anxiety, a key barrier to EV adoption. By utilizing algorithms and real-time data, ITS can determine an EV's maximum range based on its current State of Charge (SoC), considering variables like load, driving patterns, and environmental conditions. ITS can pinpoint CSs within this range that offer compatible connectors, providing drivers with detailed route information, including travel time, distance, and projected SoC at arrival. This is called Intelligent Trip Planning (ITP) and is explained in [15]. This capability not only enhances the driving experience but also reassures drivers of their ability to reach destinations with sufficient battery life, significantly reducing the anxiety associated with EV range limitations.

1.3 Goal & Objectives

The goal of this thesis is to develop an ITS that can provide charging guidance for EV users around the final destination of their journey.

The first objective of the ITS is to enable smart decision-making for users by delivering essential details that can alleviate range anxiety. By incorporating data on battery status, CSs, and estimated driving ranges into the ITS, the system can provide drivers with more confidence in their vehicle's capabilities and provide a set of useful CSs that fit the user’s criteria and vehicle’s range with an estimated value of SoC at each station.

It is crucial to note that all data provided to the user regarding range and final SoC is intended to be as conservative as possible. This approach is adopted to avoid instilling overconfidence in users and to ensure that the provided estimates do not risk users potentially not reaching a CS due to overestimated ranges, thereby maintaining user trust and system reliability.

The second goal is to optimise the current CI infrastructure. CSs, present a complex landscape of constraints that must be considered. For instance, individual CSs vary in the number of chargers available and the power levels they offer. These variations inevitably influence the charging duration, which in turn affects drivers' travel plans. Further complicating the scenario are the local grid conditions, which can sometimes alter the performance and efficiency of these CSs across regions.

Recognizing that choosing where EVs will charge can significantly influence the optimization of the existing CI, the ITS aims to make this selection more intuitive and efficient. To address these complexities, the ITS presented in this thesis seeks to verify compatibility and availability for the user, thereby preventing the scenario of directing users to CSs that lack compatible chargers, sufficient power, or are unavailable.

1.4 Extended Aims

In addition to the main objectives, this thesis also explores wider aims tied to the development of an ITS for EVs

Firstly, by improving the reliability and ease-of-use of EV routing systems, this work hopes to create a more positive view of EV technology, which could encourage more people to adopt EVs. While measuring the direct impact on EV adoption rates may go beyond the immediate scope of this thesis, the aim to contribute to a positive EV perception is central to the research objectives.

Secondly, this thesis supports a move towards a more sustainable and efficient transportation system by promoting better routing and charging strategies. Even though the direct contributions to sustainability may be incremental and unfold over the long term, aligning the ITS development with eco-friendly transport principles is a mindful approach to innovation in this field. This alignment resonates with the EU's goals of reducing carbon emissions and promoting cleaner transport options as part of its decarbonisation efforts.

Lastly, the framework designed in this thesis aims to emphasise inclusivity by serving a diverse user base with varying transportation needs and preferences. The system's aim for adaptability in meeting specific user needs not only improves user satisfaction but also expands the potential market reach of the system. This emphasis showcases the importance of designing technological advancements within the EV domain that can serve a wide spectrum of societal needs and preferences.

1.5 Research Questions

1.5.1 What Data Types Should be Considered for the ITS?

Various data types are essential for achieving the objectives outlined in the previous section. Routing and mapping data are indispensable for the system to provide accurate estimations of the time and distance needed to reach each CS from the original location. This information is vital not only for efficiently guiding the user but also for calculating the EV's driving range, considering the total distance to be travelled.

In addition, traffic data should be thoroughly considered, as offering the user the best possible route is key to enhancing their satisfaction. Accurate estimation of the Expected Time of Arrival (ETA) is essential, incorporating potential traffic delays or everyday situations that may lengthen or shorten the usual travel time.

Incorporating weather and temperature data into EV routing could have a profound impact. The system could suggest alternative routes or modify charging strategies by considering rain, snow, or extreme temperatures. By avoiding adverse weather conditions, EV drivers could experience safer journeys, minimising the risks of accidents or breakdowns. Additionally, the SoC of EVs is directly affected by the temperature of the route. By considering temperature in the calculations, the ITS could assess the impact of temperature on battery performance and deliver precise predictions on the actual range of the EV.

Another crucial aspect of an ITS is its ability to factor in the availability, proximity, and cost of CI. By integrating data on CS locations, availability, and compatibility with different EV models, the system could guide drivers to the most convenient charging points along their routes. This ensures EV drivers have access to reliable charging options, easing range anxiety and promoting widespread adoption of EVs.

Moreover, an optimum framework could personalise EV routing based on user preferences and vehicle characteristics. By considering factors such as preferred charging speeds, battery range, and desired charging levels, the system could provide tailored recommendations to optimise the driving experience. This high level of customisation enhances user satisfaction and ensures that EVs are employed with maximum efficiency and effectiveness.

1.5.2 How Can These Parameters Be Integrated?

As outlined in Section 1.5.1, integrating information related to EV characteristics, traffic, CI, pricing, and weather is mostly essential for a precise ITS. The question that arises now is how this diverse data can be effectively incorporated into the microservice.

1.5.2.1 Data Integration Technologies

External services use Streaming Application Programming Interfaces (APIs), WebSockets, and Webhooks to efficiently integrate both delayed and real-time data into an ITS [16].

- Streaming APIs provide continuous updates by connecting with a data provider, delivering immediate access to the latest information.

- Websockets enable bidirectional communication, allowing the system to receive traffic-related events and updates as they happen, eliminating the need for frequent API requests.

- Webhooks, on the other hand, offer a push-based mechanism where the system registers a callback URL with the traffic data provider, receiving updates as incidents occur without actively requesting data.

The ITS can seamlessly incorporate all the necessary data into its decision-making algorithms by leveraging these technologies, ensuring accurate route optimisation and energy efficiency.

Most external data providers, particularly those offering information on EV characteristics, traffic, CI, pricing, and weather, commonly present their data through APIs. Additionally, APIs offer a standardised, user-friendly interface, simplifying the process of connecting and retrieving diverse datasets. Many of these external services offer free trials through their APIs, which can be particularly valuable during the testing phases of ITS development. This allows for experimentation and evaluation of different data sources without incurring initial costs.

1.5.3 Is Real-Time Data Essential?

Many mapping and traffic services do not provide real-time information but generate it using aggregated and anonymised information from various sources, including GPS data and historical traffic patterns. As a result, it may not capture exact conditions or unexpected incidents. To ensure more precise and up-to-date real-time traffic data for immediate decision-making in an ITS, it could be beneficial to explore integration options with specialised real-time traffic data providers.

Real-time data on charger availability could also be incredibly valuable, enabling users to plan their charging sessions without unexpected delays, however it may incur in costs.

While real-time data may appear less critical for weather conditions compared to traffic or CS availability, it gains importance in regions where weather can change rapidly. Access to current weather information may be crucial for predicting storms, snow, or other conditions that might disrupt traffic or impact vehicle performance. In areas where such weather events are common, obtaining weather data from a reliable and frequently updated API is invaluable.

For a competitive framework, it is evident that real-time data could be valuable. Given the rapidly changing nature of current traffic conditions, it would be crucial to frequently update traffic information to provide accurate estimates of the SoC and the time required to reach CSs. Additionally, the fluctuating availability and pricing of CSs should be continuously monitored to prevent users from experiencing unexpected wait times or costs that deviate from the initial estimates provided by the ITSs.

However, as a cost-effective foundation for developing an ITS, initial implementation can be considered without relying on real-time data. This approach allows for the development and testing of essential functionalities and features of the ITS, providing a solid groundwork for future enhancements.

1.6 Outline

This thesis is structured into distinct chapters, each focusing on a different aspect of the study, encompassing research, implementation, and evaluation.

The examination begins with Chapter 2, 'Literature Review,' where various topics like behavioural aspects of EV drivers, routing optimization, SoC management, intention-aware systems, and environmental factors influencing EV performance are discussed. This chapter also reviews the state of the art of technology and identifies gaps in the existing literature and systems.

Chapter 3, 'Methodology and Approach,' details the conceptual framework for the ITS developed for EVs. This chapter starts by outlining the primary service goal of the system. The focus then shifts to the required inputs for the system, chosen development platform and discusses the planned deployments, showcasing the system's utility and integration potential.

Chapter 4, 'Pre-implementation: Analysis of External Data Platforms and Resources': focuses on the preliminary steps before the actual implementation of the ITS. It covers the analysis of various external services, libraries, and database management systems, setting the foundation for the system's infrastructure.

Chapter 5, 'Navigator: Microservice Implementation,' shifts focus to the practical aspects of implementing the ITS prototype, known as ‘Navigator’, as a microservice. This chapter offers an in-depth examination of the selected resources and platforms crucial for the system's functionality. It comprehensively details the implementation process of the Navigator, emphasising the core functions and error handling mechanisms within the system.

The narrative progresses to Chapter 6, 'Navigator: Standalone Deployment,' where the design, implementation, and system requirements for deploying the microservice independently are discussed. This discussion continues in Chapter 7, 'Evrich: Integrated Deployment,' where the deployment of the evrich service within the integrated Navigator is considered. Both chapters include a practical application and validation section, where user scenarios are explored.

Chapter 8, 'Navigator: Computational Analysis and Prospects,' thoroughly explores detailed performance evaluation of the Navigator microservice and its individual components, including the navigator logic, database, and API containers. Key performance metrics are examined, providing insights into Central Processing Unit (CPU) usage, memory usage, and API responsiveness. Scalability is assessed and results interpreted, providing a clear picture of the system's operational efficiency. Furthermore, the chapter discusses potential improvements, positioning the Navigator for future enhancements and applications.

The evaluation of the system is conducted in Chapter 9, 'Discussion and Conclusion.' This chapter encapsulates the research findings, discusses the contributions and impact of the research, presents the results and analysis of the system's performance, discusses possible improvements, and suggests future avenues for research and innovation within the Service-based Open-source Grid automation platform for Network Operation of the future (SOGNO) platform. The chapter concludes with some final remarks on the research undertaken, thus bringing the narrative to a close.

2. Literature Review

2.1 Introduction

The arrival of EVs has brought about a fundamental change in the transportation sector. However, the widespread adoption of EVs is hindered by several challenges, such as range anxiety, SoC management, and the impact of environmental factors like temperature. New models offer a promising solution to these challenges. This literature review explores the various dimensions of ITS for EVs, focusing on SoC prediction, routing optimisation, and the impact of driving behaviour and environmental conditions on EV performance.

In the following sections, a thorough analysis is carried out across several key areas related to ITS for EVs. Section 2.1 delves into the Behavioural Aspects of EV Drivers, shedding light on how their unique driving patterns influence routing preferences and traffic dynamics. Following this, Section 2.2 explores the complexities of Routing Optimization and EV SoC Management, crucial for enhancing the travel experience for EV users. In Section 2.3, the focus shifts to Intention-Aware Systems and Multi-Agent Considerations, aiming at optimising routing by understanding the intentions or policies of other road users. Section 2.4 probes into the Environmental Factors Affecting EV Performance, a critical aspect given the significant impact of variables like speed, slope, and temperature on EV's SoC and range. In Section 2.5, the discussion transitions to the Current State-of-the-Art, evaluating existing geospatial service APIs and their applicability in EV routing. Lastly, Section 2.6 identifies the Gaps in Existing Systems and Literature, highlighting the areas that require further exploration to bridge the gap between theoretical models and practical solutions, thereby contributing towards the broader objective of developing a more robust and comprehensive ITS for EVs.

2.2 Reference Studies

2.2.1 Behavioural Aspects of EV Drivers

Understanding the unique behaviours of EV drivers is foundational for any ITS system. The paper [17] "Routing aspects of electric vehicle drivers and their effects on network performance" investigates how EV drivers choose their routes compared to traditional ICE drivers. The study uses a mathematical model to understand these choices and finds that EV drivers often pick roads with lower speeds to save battery life. This behaviour can improve overall traffic flow under certain conditions.

In the pursuit of developing a comprehensive system for EVs that efficiently identifies nearby CSs, understanding the unique routing behaviours of EV drivers becomes indispensable. The study reveals that EV drivers often opt for routes with lower speeds to conserve battery life. This behavioural aspect could serve as a parameter in the system's algorithmic design, enabling it better to meet the specific needs of these drivers.

The paper also provides an in-depth understanding of how EV drivers weigh factors such as energy conservation and range anxiety when making routing decisions. By integrating these insights into the ITS, it can offer more personalised and effective routing and charging solutions, thereby elevating the overall user experience, and helping to fine-tune the algorithms of the future system, making it more effective in addressing range anxiety.

The study finally confirms the problem stated in Section 1.1.2: that range anxiety significantly influences the route choices of EV drivers. To alleviate this anxiety in an ITS it is critical to incorporate precise data and different data to have an accurate SoC estimation.

2.2.2 Routing Optimisation and SoC Management

In EV routing systems, optimising the travel experience for users remains a critical challenge. The paper [18] "Benefits of Multi-Destination Travel Planning for Electric Vehicles" offers a comprehensive examination of the complexities involved in EV travel planning, with a particular focus on multi-destination routes. This work serves as a valuable resource for understanding the unique challenges and considerations in enhancing the travel experience for EV users.

The paper delves into the intricacies of route optimisation, emphasising the benefits of multi-destination planning. This involves optimising routes that include multiple stops, considering various constraints such as the SoC, availability of CI, and time. Route optimisation is inherently complex and involves many variables, making this paper's focus on multi-destination planning a valuable contribution to the literature.

The paper's focus on accurate SoC estimation aligns with the considerations outlined in Section 1, specifically in Section 1.5.1. The authors explore how SoC can affect the choice of CSs and the overall travel plan. Understanding the complexities around battery management is crucial for any system that optimises EV travel, as it directly impacts both the feasibility of reaching a particular CS and the duration of the charging process.

While the paper does not explicitly discuss the technical aspects of data gathering, it does emphasise the importance of accurate and comprehensive data for effective route optimisation. This is particularly relevant for systems that rely on real-time or near-real-time data to provide route and CS recommendations.

2.2.3 Intention-Aware Systems and Multi-Agent Considerations

One of the influential papers in ITSs for EVs is [19] titled "Intention-Aware Routing of Electric Vehicles." This paper introduces a groundbreaking Intention-Aware Routing System (IARS) designed to optimise the routing for EVs. The primary objective of IARS is to minimise the expected journey time for EVs by considering the intentions or policies of other vehicles. This is particularly crucial for EVs that may need to recharge during route and could face significant queuing times if multiple vehicles opt for the same CSs. Through simulations using real-world data from The Netherlands, the paper demonstrates that IARS can lead to an over 80% improvement in waiting times at CSs and a more than 50% reduction in overall journey times.

The IARS system operates by predicting queuing times at CSs based on the current intentions of other EVs. It then suggests the most efficient route and CS to the driver. The system uses three sources of information to derive probabilistic travel times: known user goals (i.e., arrival time distributions at specific CSs), intentions from users who have participated in the past, and users whose wishes are unknown to the system. This multi-faceted approach allows for a more accurate congestion prediction at CSs, enabling more efficient route planning.

In the context of this thesis, which aims to create a comprehensive EV system that efficiently locates nearby CSs, the Intention-Aware Routing System (IARS) is a pertinent point of study and comparison. IARS optimises routing for EVs by leveraging real-time and historical data, thereby directly addressing the core focus of this research. Furthermore, the paper introduces the innovative concept of "intention-aware" systems. By understanding the intentions of other drivers—such as their preferred CSs or routes—search algorithms can be further refined to provide more personalised and efficient CS recommendations, enhancing the user experience. The paper's rigorous methodology and empirical results also offer a valuable benchmark for this thesis. It provides both validation for the effectiveness of advanced routing algorithms in reducing journey times and for the aim of the ITS to leverage CS availability and CS data is a key factor to avoid waiting times, mentioned in Section 1.1.5.

2.2.4 Environmental Factors Affecting EV Performance

One of the most pressing challenges facing the widespread adoption of EVs is the crucial issue of range anxiety, which stems from the uncertainty surrounding the vehicle's remaining range and the availability of CSs. This seminar paper investigates the factors affecting the SoC and range of EVs, offering valuable insights that are highly relevant to developing a comprehensive ITS for EVs.

The paper [20] " Routing systems to extend the driving range of electric vehicles " presents an in-depth study to understand the factors influencing EVs' range and SoC. Conducted over three years, the research utilises high-resolution real-world data from the SwitchEV trial, which involved 44 EVs covering over 400,000 miles across the Northeast of England. The study employs linear models to determine energy expenditure equations and uses Dijkstra's graph search algorithm to find the route minimising energy consumption.

The paper highlights how driving speed and traffic conditions affect energy consumption. For instance, higher highway speeds can lead to quicker battery drain, while stop-and-go traffic can be beneficial due to regenerative braking (Page 4, 10). These insights are crucial for confirming the need of traffic data and street type on Section 1.5.1 for developing ITS applications that can adapt to varying driving conditions and offer real-time guidance to drivers.

Moreover, the accuracy of predicting an EVs future efficiency and range is significantly improved by incorporating historical and speed data, depending on each specific vehicle model, reinforcing the necessity of integrating EV manufacturer data into the ITS, a requirement highlighted in Section 1.1.5.

The study indicates that energy consumption predictions are more accurate at higher than lower speeds. This is intuitively understandable as the variability in energy usage at lower velocities is often due to auxiliary systems like lighting, air conditioning, and the radio rather than the power used for driving itself. Additionally, evidence suggests that about 32% of drivers consciously alter their driving routes to optimise regenerative braking, particularly in stop-and-go traffic conditions. This behavioural adaptation underscores the importance of the objectives of this thesis, mentioned in Section 1.3 of providing EV drivers with relevant information that affects range.

The effect of road gradient on vehicle efficiency is also commented on, particularly in relation to speed. It states that for the speed range of 50–70 km/h, there is a more significant variation in efficiency concerning the gradient. In contrast, there is comparatively slight variation in efficiency concerning gradients at low speeds. So, at lower speeds, especially in stop-and-go traffic, EVs can benefit from regenerative braking, which can partially recharge the battery during braking events. This could offset some of the energy loss due to climbing a slope, making the net impact on efficiency less significant at lower speeds making altitude change less important in inner-city areas where lower speeds are expected. However, at higher speeds, the gradient becomes a more substantial factor affecting efficiency (Page 6). This study underscores the importance of including altitude change as a critical variable in the ITS's calculations, as discussed in Section 1.5.1.

The paper also emphasises that a detailed understanding of these factors is essential for ITS applications. Accurate range prediction can alleviate range anxiety, one of the main barriers to EV adoption, confirming the challenges of EV adoption, from Section 1.1.2. The study suggests that such information can be integrated into ITS applications to determine the best route available, thereby minimising energy consumption and extending the range (Page 10).

The findings of this paper are highly pertinent to the objectives of this thesis, mentioned in Section 1.3, which aims to develop a comprehensive ITS for EVs focusing on short urban routes, but easily adaptable and scalable for longer routes. Understanding the factors that affect SoC and range can significantly enhance the system's ability to provide accurate and personalised recommendations for CSs. For instance, by considering the topography and current traffic conditions, the system can more precisely estimate the SoC upon arrival at a CS, enhancing the user's overall experience.

Moreover, the paper's methodology and findings offer a robust foundation for developing the ITS. The use of linear models to determine energy consumption based on diverse data can be adapted to suit the specific needs of the ITS being developed in this thesis.

While speed and slope are significant factors, external conditions like temperature can drastically impact EV performance, affecting both SoC and range.

The paper [21] titled "Effect of extreme temperatures on battery charging and performance of electric vehicles" by Juuso Lindgren and Peter D. Lund offers a comprehensive investigation into the impact of temperature on EV performance. Utilising a hybrid artificial neural network-empirical Li-ion battery model combined with a lumped capacitance EV thermal model, the paper provides valuable insights into how temperature variations can significantly affect EV efficiency and charging capabilities.

This paper is particularly relevant to the objectives of this thesis, which aims to develop a comprehensive system for EVs that efficiently identifies nearby CSs and estimates the SoC. One of the most striking findings of the paper is the substantial decrease in median efficiency at extreme temperatures. Specifically, the article notes that at low temperatures, around 10°C, the median efficiency decreases by 16% compared to the reference case. At high temperatures, around 40°C, the efficiency reduces by over 25%. These numbers underscore the importance of considering temperature as a variable in any system aiming to provide accurate SoC and range estimations, confirming the selection of temperature data for the ITS from Section 1.1.5. Given that the thesis focuses on mostly short to medium length urban routes where quick and precise SoC estimations are crucial, understanding and mitigating the impact of temperature on EV performance becomes essential to the system's reliability and effectiveness.

2.3 State-of-the-Art of Technology

2.3.1 Azure Maps

Azure Maps provides a comprehensive geospatial service APIs, including functionalities specifically designed for EV routing. One of its standout features is calculating a vehicle's reachable range based on its current SoC. This feature is handy for drivers who must understand how far they can travel with their remaining battery life. Additionally, Azure Maps can identify CSs within this calculated range, enhancing the driver's ability to plan their journey effectively. Furthermore, Azure Maps offers the potential for integrating real-time traffic and weather data into the routing algorithm, which could provide more accurate and timely routing recommendations in future iterations of a routing system [22].

The platform can locate stations within the calculated reachable range regarding CS identification. The service sources its CS data independently and in collaboration with Open Charge Map, an open-source project that maintains a global database of EV CSs [23]. This dual sourcing potentially broadens the scope of available CS data, although it is worth noting that Azure Maps does not provide real-time information on station availability or pricing.

However, despite its robust features, Azure Maps has several limitations that could impact its utility in a comprehensive EV routing system. The service does not provide real-time data on the availability or pricing of CSs. Additionally, its range calculation is a general estimate that does not consider the specific model of the vehicle, nor does it consider discharge rates. This could affect the accuracy of the range prediction, especially for users who require more precise information for short urban routes.

2.3.2 Mapbox

Mapbox Directions API with EV routing offers a unique set of features tailored to the needs of EV drivers. One of its most compelling features is calculating the SoC at the destination. This is particularly useful for long-distance trips, as it gives drivers a clearer understanding of their battery status upon arrival.

Additionally, the service automatically adds optimal CS stops along the route to ensure the vehicle can reach its final destination. The CS data is sourced from OpenStreetMap, providing users a wide range of options. An option to input charger type for charger compatibility features is provided [24].

Another noteworthy feature is the ability to input the energy consumption curve for a specific EV manually. This allows for more personalised and accurate routing, although most users may not have access to this information. The service supports various routing profiles, including traffic, driving, walking, and cycling, and allows users to specify optional parameters like maximum AC charging power and auxiliary consumption in watts.

However, Mapbox's service has its limitations. It is primarily designed for long-distance routing, meaning it does not allow users to query CSs around a specific location for shorter trips. This could be a significant drawback for users interested in urban or neighbourhood settings where short-distance travel is more common. Additionally, the service does not allow users to input their EV model, which could affect the accuracy of the SoC and range predictions. There is also no indication that the service considers external factors like road gradient or temperature, which can significantly impact an EV's battery performance.

2.4 Gaps

While existing systems offer some level of routing optimisation, they often fall short in providing real-time adaptability based on many factors such as traffic, weather, and road conditions. Most systems are designed for long-distance travel and do not adequately address the unique challenges of short urban routes, where frequent stops and varying speed limits can significantly affect an EV's SoC

Another significant gap is the lack of comprehensive charger compatibility features. Existing systems may provide CS locations, but they often do not offer detailed information on the types of chargers available, their status, or compatibility with specific EV models. This lack of knowledge can lead to inefficient routing, as drivers may arrive at a CS only to find that it does not meet their vehicle's charging requirements.

Also, one of the biggest limitations is the absence of standardised communication protocols between EVs and CI. This can make it challenging for Energy Management Systems (EMSs) to communicate with CSs and optimise the charging process, leading to longer charging times and lower energy efficiency. The limited availability of CI is also a big challenge, making it difficult to develop personalised routing strategies.

Current systems generally offer a rudimentary calculation of SoC, often failing to consider factors like driving behaviour, vehicle load, and environmental conditions. This lack of precision can result in inaccurate range predictions, exacerbating the issue of range anxiety among EV drivers.

The existing literature, while extensive, often focuses on isolated aspects that are key for EV routing systems. There is a lack of comprehensive studies that combine routing optimisation, SoC management, and charger compatibility into a unified framework. Moreover, most studies do not delve into the real-world applicability of their findings, leaving a gap between theoretical models and practical solutions. This literature review has explored the various dimensions of ITSs for EVs, from routing optimisation and SoC management to the impact of driving behaviour and environmental variables. These collective insights set the foundation for developing a more robust and comprehensive ITS for EVs.

3. Methodology and Approach

3.1 Introduction

Building on the insights uncovered in Chapter 2's comprehensive literature review, Chapter 3 delves into the methodology and approach for developing the ITS. This chapter aims to bridge the gap between theoretical concepts and practical implementation, drawing upon the research findings and identified gaps in existing systems to construct an efficient system tailored to the unique needs of EV drivers.

3.2 ITS Concept Architecture

3.2.1 Service Goal

This thesis focuses on developing a comprehensive system for EVs that is equipped to locate compatible CSs quickly and efficiently for drivers in short urban settings, as well as for those undertaking longer journeys, by identifying charging facilities en route or nearby their anticipated destination. The service's goal is to also calculate the distance and travel time to each chosen CS, to then estimate the vehicle's SoC upon arrival. This estimation will consider a range of factors potentially affecting battery life, thereby offering a comprehensive solution to EV drivers for their charging needs.

3.2.2 Required Inputs

The research necessitates multiple data categories to create an ITS for EV routing services. These categories include:

- Geographical coordinates for both the user's location and CSs within the user's feasible range.
- EV specifications from manufacturers' datasheets, including weight, battery capacity and discharge rates.
- Temperature data, at each origin and CSs locations, to calculate a mean temperature for the route. This is critical since temperature affect the SoC value influencing range and further CS accessibility.[1]
- Information on CSs, such as locations, the types and number of available chargers and their pricing structures.

[1] After a thorough literature review and careful consideration, it was determined that adapting the ITS to account for multiple weather scenarios, would add a whole new layer of complexity which was beyond the project's current scope.

- Routing information, encompassing distances and estimated travel times for paths connecting the user to the CSs.
- Elevation details from origin and each CS, significant for incorporating route steepness, which have been proved essential by the literature review.

To facilitate user interaction with the system, a series of inputs are required from the user to tailor the search for optimal CSs:

- Starting Point & Destination: The user must enter their journey's starting point and intended destination.
- Radius for CS Search: The user should specify a maximum radius in kilometres around the destination for locating CSs. This flexibility caters to users willing to deviate from their direct destination or route for better charging facilities.
- EV Model: Inputting the EV model is essential for estimating the SoC upon arrival at the CS and determining charger compatibility.
- Initial SoC: The user must input their vehicle's initial SoC.
- Fast Charging Preference: In line with the ITS's user-centric design, the system will prompt users to indicate if they prioritise fast charging. This information is crucial to narrow down the search to CSs with compatible fast charging connectors.

3.2.3 Target Platform

Following the primary objective, the system will be developed using Python [25], a high-level programming language known for its versatility and ease of use. Python was chosen for several reasons, including its extensive libraries and frameworks, which facilitate rapid development and prototyping. Its rich ecosystem allows easy integration with various APIs, making it ideal for gathering routes, CSs, and weather data. Additionally, Python's widespread adoption in academia and industry ensures robust community support, which is invaluable for troubleshooting and optimising the system.

The ITS relies on several external platforms to assemble a comprehensive set of information crucial for route planning and energy management. It gathers diverse route data including road types, elevation changes, distances, estimated travel times, and walking times for pedestrians. Additionally, it sources detailed CS information, providing the availability of charging ports and associated pricing. The system also incorporates temperature data, which is essential for precise energy management and route optimisation in varying weather conditions.

To estimate the vehicle's battery's SoC when reaching a CS, the system considers the battery's capacity and discharge rates, as provided in the manufacturers' data sheets. Relying solely on external services to access vehicle characteristics data in the real-time, makes the system vulnerable to network issues, possibly becoming almost non-functional in cases such as system failures, server maintenance, bandwidth problems, and cyber-attacks. Therefore, the objective is to develop a local database that can store comprehensive information about EVs. It is proposed to store this information locally to mitigate this risk and enhance system resilience. Manufacturers typically provide this data as data sheets, making it feasible to compile and store it in a database for offline access. By performing simple calculations based on this information, the system can provide an estimated SoC, offering EV drivers an indication of the remaining charge upon their arrival at a CS.

The developed ITS prototype will be named “Navigator” and calculates the nearest CSs within a user-specified radius, compatible with the user’s vehicle in the proposed architecture. It also provides essential details such as the availability of chargers, the time required to reach each station, the associated pricing, and the final SoC.

In the context of this thesis, the Navigator has been designed as a microservice to function as part of a larger ecosystem called Electric Vehicle Routing in Charging Hotspots (evrich) developed by the ACS Institute at RWTH Aachen University, implemented with the SOGNO architecture [26].

3.2.4 Target Workflow

The first step in the Navigator workflow should be identifying CSs within the specified range from the user's current location. The ITS should employ geolocation data and an external API to identify nearby CSs within the vehicle's driving range. This helps narrow the options and ensures the user can reach a CS without excessively depleting their battery.

Once potential CSs are identified, the ITS should consider compatibility with the user's vehicle, by checking in the database the specific EV charging connector types or power levels. Therefore, the system should filter out CSs that do not support the necessary charger for the user's vehicle. This ensures that the selected stations are compatible with the user's EV, preventing charging compatibility issues.

The process should also consider the SoC of the vehicle's battery. By considering the weight of the EV, current SoC, distance, temperature, journey type and elevation change to the selected CSs, the ITS should calculate the estimated SoC when the user is expected to reach each station. This calculation helps determine the practicality and necessity of charging at each station based on the available battery capacity. The station becomes a viable charging option if the estimated SoC falls within an acceptable range.

3.3 Planned Deployments

Following the upcoming implementation of the Navigator microservice, which is intended to follow the previously outlined workflow, the microservice's value and capabilities will be demonstrated in this thesis through two distinct models.

The first deployment, in Chapter 6, will aim to offer an initial insight into the Navigator's standalone functionality, before the integration to the broader evrich service.

The second deployment, in Chapter 7, will showcase how the Navigator can be seamlessly integrated into the evrich service. In this configuration, the Navigator collaborates with other microservices to achieve a common goal while contributing its unique capabilities to complete the overall system architecture.

These two models will serve to highlight both the independent utility and the collaborative potential of the Navigator within a larger, interconnected environment.

Simulations will be conducted for both deployments using test scenarios that reflect user profiles, charging demands, and environmental conditions. Synthetic data may also be generated to simulate different CS distributions and traffic patterns. By leveraging simulation tools, the system's behaviour and performance can be evaluated under diverse circumstances, and potential improvements can be identified.

4. Pre-implementation: Analysis of External Data Platforms and Resources

4.1.1 Introduction

In Section 1.5.2, the research question on how the necessary data types could be integrated via external services was analysed. In this section, laying the groundwork for the implementation phase, a detailed comparison of various external services is conducted, assessing them against multiple criteria defined within the project's goal. Different database management systems will also be analysed, with a focus on identifying the optimal solution that aligns with the Navigator's requirements for data integrity, scalability, and efficient data handling.

4.1.2 External Services and Libraries

4.1.2.1 Operational Data Requirements & Service Selection Criteria

To begin, the Navigator requires a geocoding service to convert destination inputs from users into precise geographical coordinates. These coordinates are crucial for interacting with the CS external services, which provide valuable information such as CS locations, pricing, availability of chargers, and other operational details. Once CSs are identified, the Navigator engages an external mapping service, to compute the optimal route from the starting point to the CSs, also providing the ETA and the distance involved. Afterwards, the Navigator leverages geospatial elevation and weather services to incorporate changes in altitude and temperature, which are critical for accurately predicting the final SoC at each CS.

The aim is to select services that not only meet the specific needs of this research but also stand out in terms of comprehensiveness, practicality, and usability. Key factors for evaluation include the accuracy of the data provided, the extent of coverage, availability of real-time updates, and the detail of the information offered. The evaluation does also consider the long-term viability and expandability of the services to ensure they can scale with the growth of the Navigator microservice. The ITS will access these services via APIs and libraries, enabling the exchange of data and commands.

It is important to note that a thorough analysis of the cost of these services is beyond the thesis's scope due to variable pricing models that may change over time. The initial priority is to find a service that is effective and freely accessible. However, as the system scales up and request volumes increase, it will be necessary to examine the pricing models in detail to maintain the sustainability of the solution.

4.1.2.2 Geocoding Service

In developing the ITS, geocoding is essential for converting user-provided addresses in string format into latitude and longitude coordinates. Several services offer this functionality, including Google Maps, Bing Maps and Geopy library's Nominatim geocoding.

Geopy's service is selected due to the availability of a Python library, which significantly expedites the development process. This feature allows for seamless integration into the project's existing Python-based codebase. The library simplifies many of the complexities associated with raw API calls, thereby reducing the likelihood of errors, and saving valuable development time. Therefore, the Geopy library's Nominatim is not only chosen for its cost-effectiveness and straightforwardness but also for the rapid development it enables through its Python library [27].

4.1.2.3 Weather Services

In the search for a suitable Weather service for the ITS, three major contenders were considered: OpenWeatherMap, Weatherbit, and Open-Meteo. Each service offers unique features and capabilities that could serve the project's needs.

OpenWeatherMap is a comprehensive service that provides weather-related data, including current conditions, historical data, and forecasts. It offers different pricing tiers, with the free tier allowing up to 60 calls per minute [28].

Weatherbit is another robust option known for its high level of accuracy and diverse range of weather-related data. The service is fed by multiple sources like live weather stations, trusted meteorological models, and machine learning algorithms, ensuring real-time accuracy and reliability. However, the trial plan allows for just 50 API calls per day [29].

Open-Meteo service does not offer as comprehensive a set of features as OpenWeatherMap or Weatherbit. However, the platform does provide a free, open-source API, which is particularly attractive for trial periods when an unrestricted volume of queries is desirable. At this stage, the primary focus is to obtain real-time temperature data to adjust the final SoC for the EV. So, Open-Meteo was found to be sufficient for this purpose, making it a cost-effective choice for the trial phase of the project [30].

4.1.2.4 CS Services

Various platforms offer services with APIs that provide location, pricing, charger types and operational data on CSs, including Open Charge Map (OCM), PlugShare, and ChargePoint.

OCM boasts global data coverage, with information on 370,000 CSs [31] spanning multiple countries and regions, however, availability data of CS is not provided. The OCM service collects data from various sources, including CS manufacturers, network operators and community users. This multi-source approach lends a certain degree of reliability to the data. The google+ community-driven platform allows users to add and update CS data, enhancing its comprehensiveness [23].

While this approach can lead to highly updated data, it also introduces the risk of outdated or incorrect information if not adequately moderated. Regarding the ease of extracting specific details about CSs, OCM's free API is well-documented and user-friendly, making it relatively easy to extract specific information.

PlugShare also offers extensive global data coverage, providing detailed information on over 750,000 CSs [32], including user reviews and photos, which adds qualitative aspects to the data [33]. As OCM, availability information on CSs is not provided. PlugShare provides updates through user-generated content, with users reporting the status of CSs, which also relies on active community participation for the currency of data. However, PlugShare's API is primarily for commercial use and is not available for personal or non-commercial licences. Due to the high volume of inquiries and limited developer support resources, they do not currently provide personal or non-commercial licences for data access.

ChargePoint offers a robust API but is geared towards developers with a good understanding of their network's specifics. ChargePoint, being a proprietary network, has the advantage of offering highly accurate real-time updates, with data directly sourced from their hardware, ensuring reliability. ChargePoint primarily operates its CS network, with data coverage relating to its proprietary infrastructure. It holds a significant position in the charging network landscape in both the United States and Europe, making it a key player in the market [34]. Accessing this API likely incurs costs and involves contacting their sales department, making it more suitable for larger companies or specialized applications.

After a thorough analysis, OCM emerges as a highly versatile option for the testing phase of the ITS due to its free and open-sourced nature. While PlugShare offers a larger amount of CS data, and ChargePoint provides valuable features, such as real-time availability updates and extensive CI metrics, OCM's open and cost-free approach makes it ideal for testing and experimentation in this thesis.

4.1.2.5 Mapping Services

In developing the ITS, the choice of mapping services plays a crucial role. In the aim of the thesis, the service should provide route information such as driving, walking time, distance and information on street types, which is essential for dividing each journey into highway or city modes. This categorisation is critical for calculating a more accurate final SoC for the EV.

Google Maps Directions is a widely used service known for its accuracy and extensive data coverage. It offers features like real-time traffic updates, multiple modes of transportation, and waypoints. However, it lacks the granularity to provide street-type information, a crucial requirement for this project. The absence of this feature would make it challenging to categorise the journey into different road types, affecting the accuracy of the final SoC calculation. Google Maps API offers comprehensive and well-organised documentation that makes it relatively straightforward for developers to integrate its services into various applications [35].

Mapbox Directions, on the other hand, is known for its customisation capabilities and is generally more affordable than Google Maps. It offers turn-by-turn navigation, real-time traffic updates, and dynamic routing. However, like Google Maps, it does not provide detailed information on street types. While it offers a rich set of features, the lack of this specific data point makes it less suitable for the precise SoC calculations needed in this project. Mapbox API comes with extensive documentation and community support, although some developers find that its wide range of customization options can present a steeper learning curve compared to other mapping services [36].

Bing Maps Routes, on the other hand, stands out for its ability to provide detailed street-type information, a feature crucial for this project's SoC calculations. Unlike Google Maps and Mapbox, Bing Maps categorises routes based on road type—whether it is a highway, arterial road, or local street—allowing for more granular and accurate SoC estimates. This feature can be particularly useful in differentiating between road conditions that have significantly different impacts on EV battery consumption, such as highway driving at higher speeds compared to stop-and-go city driving. Furthermore, Bing Maps also offers real-time traffic updates and robust navigation features, ensuring that it does not fall short in other essential areas. The platform is also known for its user-friendly documentation, making it relatively easy to integrate into the project [37].

In summary, while Google Maps Directions and Mapbox offer comprehensive functionalities and extensive data coverage, they fail to provide the detail needed for accurate SoC calculations in this project. Despite coverage and data accuracy may not be as extensive as Google Maps in some regions, Bing Maps delivers the critical feature of street-type information. All three platforms offer free tiers for trial versions, making the final decision less dependent on pricing. Therefore, the decision to employ Bing Maps aligns well with the project's requirements, contributing to enhancing the reliability and efficiency of EV travel.

4.1.2.6 Geospatial Elevation Services

The choice of a service that computes the route gradient or altitude change is another pivotal aspect in developing an EV navigation system, particularly for accurately calculating the SoC. Elevation data is crucial for understanding the energy requirements of different terrains, which directly impact the vehicle's battery consumption. Two significant services were considered in this context: Bing Maps Elevation and Google Elevation.

Bing Maps Elevation offers the advantage of being part of the same ecosystem as the Bing Maps Routing service, which was chosen for its detailed street-type information. Utilising the same API for routing and elevation data streamlines the integration process, reduces the complexity of handling multiple APIs, and simplifies scalability considerations. This is particularly beneficial when considering future scaling, as it would involve dealing with a single payment and documentation system [38].

Google Elevation, on the other hand, is renowned for its accuracy and extensive global coverage. It provides elevation data relative to the local mean sea level and allows for the input of multiple coordinates. Google's service yielded highly accurate and reliable results during the preliminary testing phase. Therefore, it is a strong alternative and could serve as a backup solution in scenarios where Bing Maps API might fail or be unavailable [39].

Bing Maps is chosen for its compatibility with the already selected routing API. This decision aligns with the project's aim to minimise the number of different APIs used, simplifying both the development process and future scalability considerations. However, the high reliability and accuracy of Google Elevation service make it a worthy candidate for future enhancements, potentially serving as a backup to augment the system's robustness.

4.1.3 Database Management Systems

4.1.3.1 Database Selection Criteria & Requirements

In the development of the Navigator, selecting an appropriate database is a critical factor that can significantly impact the system's performance and scalability. The database stores a wide array of information for EV models, from weight to battery capacities and charger types. Given the structured nature of this data, a relational database is a natural fit. However, choosing which relational database to use is not straightforward and warrants a detailed comparison of the available options.

4.1.3.2 MySQL System

MySQL is a widely used open-source relational database that employs Structured Query Language (SQL) for data definition and manipulation [40]. One of its key strengths lies in its Atomicity, Consistency, Isolation and Durability (ACID) compliance, which ensures that all transactions are processed reliably. This is particularly important for maintaining the integrity of the EV data. MySQL is also highly scalable and designed to meet the demands of web-based applications that may experience varying levels of user traffic.

Security is another strong suit of MySQL, offering robust data protection features such as SSL support and SSH tunnelling. SSL is a standard protocol for establishing encrypted links between a web server and a browser in online communications. The usage of SSL ensures that all data transmitted between the web server and browser remains encrypted and therefore secure. It is widely used for secure communications over websites, such as online banking, emails, and online shopping carts [41]. SSH is a protocol that provides a secure channel over an unsecured network in a client-server architecture. It allows for secure logins, file transfers, and provides strong encryption for other network services. In the context of MySQL, SSH can be used to securely connect to the database server over a network, ensuring that the data transmitted between the client and the server is encrypted and secure from a third-party interception or monitoring [41]. Moreover, MySQL enjoys extensive community support, which is beneficial for troubleshooting and performance optimisation [42].

4.1.3.3 PostgreSQL System

On the other hand, PostgreSQL is an object-relational database that extends the SQL standard to include advanced features like table inheritance and function overloading. Table inheritance in PostgreSQL allows a table to inherit characteristics from another table, promoting organised and structured data models by reusing existing database structures. Function overloading allows multiple functions with the same name but different parameters (number or type), enabling more flexible and organised code by reusing function names for similar operations [42] [43].

Like MySQL, PostgreSQL is ACID-compliant and ensures data integrity. It also offers native support for JavaScript Object Notation (JSON) and JSONB data types, which could be advantageous for storing semi-structured data. JSON is a text format used for data interchange, easily readable by both humans and machines. In contrast, JSONB is a binary format of JSON-like data in PostgreSQL, offering more efficient storage and quicker querying capabilities. While JSON preserves the original formatting and duplicate keys, JSONB does not, but it supports indexing, which can significantly improve query performance. Indexing in databases speeds up query performance by creating a smaller, easily searchable data structure, the index, that points to the location of the data in the database PostgreSQL's concurrency control is another feature that sets it apart, allowing multiple record versions to exist simultaneously, which can benefit real-time applications [43] [42].

4.1.3.4 MongoDB System

MongoDB presents a different approach as a document-oriented database. It stores data in BSON format, a binary representation of JSON documents, allowing for a more flexible schema. This format permits a more flexible schema compared to the rigid schema in relational databases. Unlike JSON, BSON can store more data types and is designed to be efficient in both encoding and decoding. This flexibility can be particularly beneficial for applications necessitating diverse and unstructured data storage MongoDB is designed with a scale-out architecture, capable of handling large volumes of data and traffic. However, it lacks the standard ACID compliance in relational databases like MySQL and PostgreSQL, which could concern applications requiring high data integrity [44].

4.1.3.5 Database System Selection

After a thorough evaluation, MySQL is the most suitable choice for this project for several reasons. Firstly, the structured nature of the EV data aligns well with MySQL's relational database model. Its ACID compliance ensures reliable transaction processing, crucial for maintaining data integrity. While PostgreSQL has robust support for complex queries and can handle write-heavy operations more efficiently, the nature of this project suggests that read operations will be more frequent, making MySQL's optimisation more relevant. MySQL also has a slight edge regarding ease of use and community support. Its user-friendly interface and extensive documentation make setting up, managing, and troubleshooting easier, which is particularly beneficial for a project requiring quick iterations and adjustments.

5. Navigator: Microservice Implementation

5.1 Introduction

Following the previous chapter's introduction to the external data platforms and resources, this chapter presents the detailed implementation of the Navigator, an ITS that constitutes the core of this thesis. The discussion encompasses the selected dependencies, libraries, external services, and database integral to the Navigator's functionality. Additionally, the chapter explores the main functions and error handling features of the microservice. This chapter aims to provide an in-depth understanding of the Navigator's architecture that ensures the seamless integration of various external services and a database.

5.2 Selected External Data Platforms and Resources

5.2.1 Dependencies and Libraries

The code employs several Python libraries and APIs to accomplish its objectives. The geopy library [27] is utilised for geolocation services, specifically for converting addresses into latitude and longitude coordinates. The requests library [45] is instrumental in making HTTP requests to APIs, while the JSON library is used for parsing JSON responses. The MySQL Connector library [46] is essential for database interactions, allowing the code to fetch EV information stored in a MySQL database. Each library was chosen for its reliability, ease of use, and community support, ensuring the Navigator microservice's robustness.

5.2.2 External Services

5.2.2.1 Open Charge Map Service

The OCM service allows for the input of various parameters to customise queries, catering to the specific needs of the Navigator microservice. These parameters include geographic filters like bounding boxes or specific latitude and longitude, enabling precise location-based searches. Users can also specify other details such as charge point IDs, country codes, connection types, and operator IDs. Additionally, there are options to control the output format, and include user comments and media items. For more information, refer to the GitHub repository [47] and API documentation [48].

In response, the OCM API delivers an extensive set of data about EV CSs. This includes unique identifiers, user comments, media items, verification status, and comprehensive details about each CS such as operator information, usage type, operational status, and connection types. The API also provides core reference data, encompassing charger types, country details, current types, operators, and status types.

Designed as a public utility, the OCM service is generally liberal regarding the number of API requests it allows. Nonetheless, users are encouraged to utilise the API judiciously to prevent system overloads. The API administrators reserve the right to impose bans on users who engage in excessive or indiscriminate usage [48]. In addition, since the API relies on information provided by third parties, there may be limitations regarding the geographical coverage of CSs and the accuracy of the data. When designing a system that relies on the OCM service, it is prudent to consider these potential limits and plan error and retry handling strategies.

5.2.2.2 Bing Maps Routing Service

The Bing Maps Routing service allows users to obtain route data between different locations using HTTP requests. Key inputs include start and end points, waypoints for multi-stop routes, and desired route attributes like distance, duration, and traffic conditions. Users can specify route constraints such as avoidances (tolls, highways, ferries), route optimisation (shortest, fastest), and desired travel mode (driving, walking, transit) [37].

The API returns turn-by-turn navigation instructions, travel time, distance, and other details for various modes of transportation, including driving, walking, and estimated traffic conditions based on accumulated data. The API also features optimised itineraries based on travel time or distance and can calculate routes from major roads in all directions.

For the Navigator in its testing phase, the Bing Maps API's limitation of 125,000 transactions per year under a free account presents a viable framework. This cap is generally adequate for developmental and testing activities, where the volume of API requests is typically moderate. Adhering to the query rate of 5 queries per second is also crucial to ensure smooth functioning. While this transaction limit supports the iterative process of testing and refinement well, it's important to be conscious about API usage, particularly as development intensifies.

5.2.2.3 Bing Maps Elevation Service

The Bing Maps Elevation API enables users to input specific data to obtain elevation information for various geographical locations. Key input parameters include points (latitude and longitude coordinates), paths (a set of coordinates to define a path), and area (defining a bounding box or a grid). This flexibility allows for a wide range of elevation-related queries, such as determining the elevation of a particular point, the elevation profile along a route, or the elevation across a defined area [38].

In response to these queries, the Bing Maps Elevation API provides detailed elevation data. For point queries, it returns the elevation of the specific coordinates provided. For path queries, the API delivers elevation profiles, which are essentially a series of elevations along the specified path. For area queries, it returns a grid of elevation data covering the specified area.

The primary limitation of the Bing Maps Elevation API is the maximum number of elevations returned in a single request, which is capped at 1024. This limit applies to the number of points for point queries, the number of samples along a polyline path, and the combined total of rows and columns for area queries. Additionally, the API does not support latitude coordinates outside the range of -85 to 85 degrees. These limitations are to be considered, particularly in the testing phase, where the number of API calls might be moderate but would need careful management as the system scales up.

5.2.2.4 Open-Meteo Service

In the Open-Meteo service, the primary user input is the geographic location, specified by latitude and longitude coordinates. This input is essential for retrieving localised weather data. Users can specify the type of weather information they need, such as current conditions, hourly forecasts, or historical weather data.

The service, designed for non-commercial use, offers various weather-related data inputs without requiring an API key. Key features include hourly weather forecasts for up to 16 days, access to global weather models with resolutions between 1.5 km and 11 km and updates every hour for Europe and North America [30].

One key limitation of the Open-Meteo API is its request cap for non-commercial applications, set at 10,000 requests per day. Exceeding this limit may lead to service restrictions, and users are advised to contact the service providers if their application's usage exceeds this threshold. It is also crucial to note that while the API is freely available for non-commercial purposes, it can also be utilised for commercial purposes, provided that proper attribution is given, as per the Attribution 4.0 International (CC BY 4.0) licence.

5.2.3 Database

The database built using MySQL system serves as a foundational element in the architecture of the Navigator microservice, providing a robust and reliable repository for the necessary EV attributes. This data is crucial for the microservice to offer accurate and personalised navigation and CS recommendations. The initial set of EVs parameters included in the database has been manually sourced from ev-database.org [49], a comprehensive platform that provides up-to-date information on various EV models. This source has been instrumental in the initial population of our database.

The database consists of a table, designed to store various attributes of EVs. The schema includes fields that can be observed in Table 1.

Table 1: Example of the built-in EV database.

EV Model	Useable Capacity	Charge Port	Fast Charge Port	Charge Power	Charge Speed	City Cold Rate	Highway Cold Rate	City Mild Rate	Highway Mild Rate	Weight
BMW-iX-M60	105.2 kWh	Type 2	CCS	11 kW AC	43 km/h	219 Wh/km	301 Wh/km	154 Wh/km	236 Wh/km	3160 kg
Renault-Twingo-Electric	21.3 kWh	Type 2	NULL	22 kW AC	110km/h	158 Wh/km	237 Wh/km	104 Wh/km	178 Wh/km	1186 kg
Tesla_model_S Plaid	95.0 kWh	Type 2	CCS	11 kW AC	55 km/h	181 Wh/km	232 Wh/km	120 Wh/km	178 Wh/km	2629 kg
Audi-SQ8-e-tron-Sportback	106.0 kWh	Type 2	CCS	11 kW AC	42 km/h	221 Wh/km	303 Wh/km	155 Wh/km	238 Wh/km	3290 kg
Nissan-Leaf	39.0 kWh	Type 2	CHAdeMO	3.6 kW AC	18 km/h	166 Wh/km	236 Wh/km	110 Wh/km	181 Wh/km	1995 kg

The fields shown explained are:

- EV Model: The name of the EV model.
- Useable capacity: The total capacity of the battery in kilowatt-hours (kWh).
- Charge Port & Fast Charge Port: The type of charging connector for normal and fast charging.
- Charge power: The power at which the vehicle can be charged in kilowatts (kW).
- Charge speed: The speed at which the vehicle can be charged in kilometres per hour (km/h).
- Discharge rates: Various efficiency ratings in watt-hours per kilometre (Wh/km) at different speeds (highway or city) and operating temperatures (cold or mild).
- Weight: The weight of the vehicle in kilograms (kg).

It is important to note that certain variables such as charging speeds and combined mode rates are not currently utilised in the system's calculations. However, the architecture includes these variables to facilitate future enhancements.

The design of the database offers several advantages. It is modular, allowing for the easy addition of new fields as the requirements evolve. The database's modularity is exemplified in its ability to adapt without overhauling the entire structure. This is achieved through the SQL design that allows for the easy addition or modification of tables and fields. For example, new attributes for EVs can be added as new columns to the table without disrupting existing data.

The design also avoids data redundancy, ensuring the database is easy to manage and update. This is facilitated by the structured format of SQL tables and the use of primary keys, such as EV model, which prevent duplicate records for the same model and maintains data integrity.

Regarding scalability, the current database includes only a subset of available EVs, serving as a trial run. However, the architecture is designed to be scalable in several ways. New data can be batch-inserted into the database, allowing for efficient updates. MySQL has been chosen as the database engine for its scalability features, which include partitioning and replication. Partitioning in MySQL divides large tables into smaller parts, improving scalability by distributing data and enhancing query performance. Replication duplicates databases across multiple servers, improving scalability by allowing multiple copies to handle read requests, reducing the primary server's load, and increasing the system's capacity. The database can also be easily migrated to a cloud-based solution to handle increased data and user loads.

Additionally, future implementations could explore forming partnerships with platforms like EV Database to ensure the database remains up to date.

5.3 System Implementation Details

5.3.1 Overview

In this section, the detailed system implementation of the Navigator microservice is showcased. This microservice is equipped with a range of specific functions, each designed for different operations, ensuring smooth performance and features customised to user preferences. For a clear visual representation of the workflow, refer to Figure 4.

This section offers a comprehensive overview of the core functions that drive the ITS. Each function is broken down to explain its purpose, mechanics, and how it integrates with the microservice's broader functionality. The level of detail is intended to provide clarity on the inner workings of the system. Therefore, while the detailed descriptions are available for those interested in the technical aspects, they are not essential for a general understanding of the system’s operation.

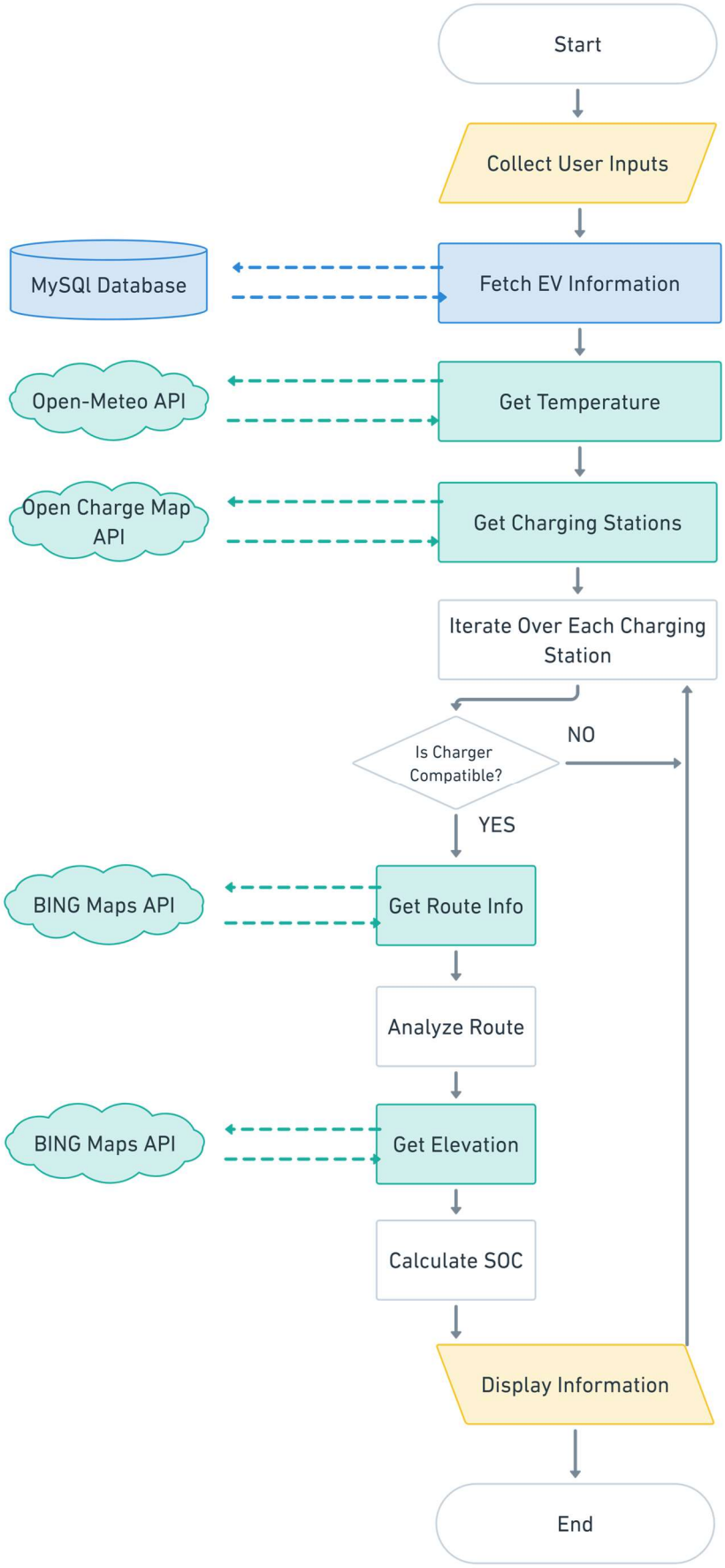

Figure 4: Navigator microservice core logic.

5.3.2 Main Functions

5.3.2.1 Collecting User Inputs

The **collect_user_inputs()** function currently acts as an initial method for user interactions, using Python's standard input method to get key details for the EV journey planning. This method effectively sets the stage, serving as a prototype for the subsequent API integration in Chapter 6 aimed at enhancing both data collection and user interaction. The user is prompted to provide the following essential details:

- Origin Location: The starting point address of the journey.
- Destination Location: The endpoint address of the journey.
- Maximum Radius: The maximum distance in metres around the destination where the user is willing to find CSs.
- EV Model: The specific make and model of the EV.
- Initial SoC: The starting battery level of the EV, expressed as a percentage.
- Fast Charging Priority: A boolean variable that indicates the user prioritises DC fast charging capabilities.

In conjunction, this function invokes the **get_coordinates** method to translate the user-provided addresses into geographical coordinates (latitude and longitude) for both the origin and destination. These coordinates are then returned to the main program for further processing. This data is subsequently relayed back to the primary function for continued computations.

The function **get_coordinates(geolocator, location)** employs the Nominatim geolocator from the geopy library to convert user-inputted addresses into geographical coordinates. It is designed to prompt the user until a valid address is entered, ensuring the data's reliability.

5.3.2.2 Fetching EV Information

The **get_ev_information(ev_model)** function is instrumental in retrieving comprehensive data about the EV specified by the user.

Upon invocation, the function establishes a connection to the MySQL database hosted locally. It uses exception handling to manage connectivity issues gracefully, ensuring the system's robustness. Once the connection is established, a cursor object executes SQL queries.

The SQL query defined within the function aims to fetch many attributes related to the EV, stored in the database, see Table 1 in Section 5.2.3. These attributes are crucial for subsequent calculations and decision-making processes within the system:

- Model name.
- Battery capacity (kWh).
- Connector types.
- Charge power (kW).
- Charging speed (km/h).
- Discharge rates (Wh/km) for different driving modes (city and highway) and operating temperatures (cold and mild).
- Weight (kg).

The function then iterates through the query result rows to find a match with the user-specified EV model. If a match is found, the relevant data is extracted and returned for further processing. If the specified model is not found in the database, the user can re-enter the model's name or abort the operation.

After the data retrieval, the cursor and the database connection are closed to free up resources. The function then returns a dictionary containing the fetched data, which includes the specifications for the specific EV model. This data is vital for accurately estimating the SoC upon reaching a CS and making informed decisions on selecting CSs.

5.3.2.3 Getting Temperature Data

The function **get_temperature(latitude, longitude)** interacts with the Open-Meteo API to fetch real-time temperature data based on latitude and longitude. It constructs an HTTP GET request using the 'requests' library and parses the JSON response to extract the current temperature. The function also includes error-handling mechanisms to manage potential API failures. Temperature is calculated for the origin and destination coordinates to estimate a mean value for the route.

5.3.2.4 Getting CS Data

The **fetch_charging_station_data(API_key, latitude, longitude, max_radius)** function is the point of contact with the Open Charge Map API. It takes in four parameters: the API key for authentication, the latitude and longitude coordinates for the location around which to search, and the maximum radius in metres within which to find CSs.

With the support of the 'requests' library, the function constructs an API request with these parameters, setting a custom parameter of maximum results to be received and sending it to the OCM endpoint. The API's response, which is in JSON format, contains details about the CSs within the specified radius. If the API request fails, the function prints an error message and returns None. It is crucial to query as many results as possible as some EVs may have specific chargers that not every CS has.

After sending the API request and receiving the JSON response, this function iterates through the list of CSs. For each station, it creates a dictionary containing key details such as location (latitude and longitude), name, operator, usage type, and usage cost. It also fetches information about the connections available at each station, including the connection type and pricing model, and appends this to the station's dictionary under the key 'connections'.

The function then returns a list of these dictionaries, each representing a CS with its associated details. This list serves as the primary data structure for subsequent operations, such as filtering based on user preferences or calculating distances to these stations.

5.3.2.5 Addressing Charger Compatibility

The function **is_compatible_charger(ev_charge_port, ev_fast_charge_port, station_connections, fast_charging_priority)** is a crucial component in the system's iterative CS evaluation. It takes four parameters: the standard charge port of the user's EV, the fast charge port if available, the types of connections available at the CS, and the user's preference (True/False) for fast charging.

The function iterates through each connection type available at the CS and checks for compatibility with the user's EV. If the user has indicated a preference for fast charging, the function looks for a match between the EV's fast charge port and the station's available connections. Conversely, if the user has not specified a need for fast charging, the function searches for compatibility with the standard charge port.

A noteworthy aspect of this function is its string-matching approach to determine compatibility. Instead of looking for an exact match, the process checks if the EV's charge port string is within the station's connection type string. This is particularly important because CS operators often provide more detailed or variant descriptions for the same port type. For example, instead of simply listing "Type 2," an operator might specify "Type 2 (socket only)." Using a string-contains method, the function can flexibly match such variations, ensuring a more comprehensive and accurate compatibility assessment.

5.3.2.6 Getting Route Information

The **get_route_info(origin, destination, API_key)** function serves to procure comprehensive route details, bridging the user's starting point (origin) with their chosen CS (destination). By leveraging the Bing Maps API and employing the ‘requests’ library in Python, the function sends a request through that includes parameters such as the origin, destination, and an option to minimise toll roads.

Upon receiving a successful response, the function parses the route JSON data to extract valuable route details. Among the numerous details provided, the most pivotal for analysis are the 'roadType', which indicates the nature of the road, and the 'travelDistance', denoting the driving distance for each road segment. These are then returned to the main code for further analysis and presentation to the user. This function is critical in the system's ability to provide accurate and valuable information for the user's journey, including travel time and distance.

5.3.2.7 Analysing Route Information

The **analyze_route(route_data)** function takes the route data dictionary from the Bing Maps API and performs a detailed analysis. It initialises a dictionary, route information, to store total distance and duration metrics. Additionally, it categorises the distance travelled on different types of roads, such as Highways, Major Roads, and Streets.

To achieve this, the function iterates through the itinerary_items in the route data, calling the **get_road_type** function for each item to determine its road type. It then updates the corresponding distance for that road type in the route information dictionary, see example in Listing 1 below.

```
Route_info = {
    'distance': 21.15,
    'duration': 1602,
    'traffic_congestion': 'None',
    'Highway': {
        'distance': 0
    },
    'MajorRoad': {
        'distance': 12.12
    },
    'Arterial': {
        'distance': 7.18
    },
    'LocalRoad': {
        'distance': 0
    },
    'Street': {
        'distance': 1.29
    },
    'Ramp': {
        'distance': 0.56
    },
    'LimitedAccessHighway': {
        'distance': 0
    }
}
```

Listing 1: Example of route information dictionary.

5.3.2.8 Calculating Walking Routes

The **get_walking_route API_key, station_coordinates, destination_coordinates)** function is designed to obtain walking route information between an origin and a destination using the Bing Maps API. The procedure constructs a URL that includes the origin and destination coordinates, optimisation parameters (in this case, distance), and the API key. By using the 'requests' library, a GET request is then sent to the API to fetch the route data.

Upon receiving a successful response, the function checks for the 'resourceSets' field in the returned JSON data. If this field is present and populated, the process extracts the travel distance and duration for the walking route. The travel duration is converted from seconds to minutes for more straightforward interpretation.

If the API response lacks the required route information or an error occurs during the request, the function returns "Unknown" for distance and duration and logs an error message. This function is essential for users who may need to walk from a CS to their destination, providing an estimate of the time and distance involved.

5.3.2.9 Getting Elevation Changes

The **get_elevation_change(origin_coordinates, destination_coordinates)** function calculates the elevation change between the route's origin and destination points. This is necessary for calculating the final SoC with potential energy altitude adjustment, which is done in the next step. It utilises the Bing Maps API to fetch elevation data for the given coordinates. The function constructs a URL with the origin and destination coordinates, and the API key then sends a GET request to the API.

Upon receiving the response, the function checks for 'resourceSets' in the returned JSON data. If found, it extracts the elevation values for both the origin and destination. The elevation change is then calculated by subtracting the origin's elevation from the destination's elevation.

If the API response does not contain elevation data or an error occurs during the request, the function returns "Unknown" and prints an appropriate message to the console. This function is crucial for understanding the topographical variations along the route, which can significantly impact the energy consumption of EVs.

5.3.2.10 Calculating Final SoC

The **calculate_soc** function is a cornerstone in the system's architecture, meticulously estimating the SoC of the EV upon arrival at a CS. This function is designed to be comprehensive, considering many parameters that include altitude changes, the vehicle's usable battery capacity, weight, initial SOC, ambient temperature, and specific route information.

The function categorises the total travel distance into two main segments, highway, and city kilometres, based on the **route_info** dictionary (see Listing 1 in Section 5.3.2.7) generated by **analyze_route**. This dictionary provides the distance covered for each type of road on the journey. The system simplifies the road types into city or highway modes to provide a conservative estimate of the final SOC.[2]

[2] While the system currently uses only city and highway discharge rates, future improvements could incorporate combined-mode rates, already incorporated in the EV database, for a more nuanced SOC calculation, especially after real-world testing with EVs.

Specifically, any road type other than 'Street' or 'LocalRoad' is classified under highway mode, where speed limits generally exceed 50 kilometres per hour. This approach is considered safer, especially since the system does not yet account for battery degradation. The function then consults the rates dictionary to select the appropriate discharge rates for these two driving conditions. These rates are contingent on the ambient temperature; colder temperatures generally result in higher discharge rates. If the temperature is below 10 degrees Celsius, the function selects the discharge rates suitable for cold weather; otherwise, it opts for mild weather rates. Energy loss is calculated by multiplying the distance driven for highway or city mode by the discharge rate specific to this mode, see equation (1).

In scenarios where discharge rates are unavailable, the function is designed to return None and log a message indicating insufficient data for SOC calculation.

It is imperative to factor in altitude, particularly under adverse conditions when the net elevation gain is positive. Such circumstances are to be considered to prevent overestimation of the final SOC. Therefore, the function calculates the approximation of the total potential energy by considering the weight of the EV in conjunction with the altitude change and gravitational acceleration. If the vehicle encounters an uphill trajectory, indicated by a positive altitude change, the total energy consumption is increased for the extra energy needed to overcome gravitational forces. Conversely, if the altitude change is negative, meaning a downhill trajectory, the energy consumption remains unadjusted to provide a conservative estimate to the user.

The energy consumption for altitude change is then added to the total energy loss due to the discharge rates. Both energy consumptions are subtracted from the original EV usable capacity to obtain an accurate estimate of the final SOC, as can be observed in (1).

The total calculation for the EV energy at each CS, measured in kilowatt-hours (kWh) can be observed below:

$$\begin{aligned} &Energy\ at\ Destination\ (kWh) \\ &\quad = EV\ Useable\ Capacity - (Discharge\ City\ Rate\ _{mild/cold} \times City\ Distance) \\ &\quad - (Discharge\ Highway\ Rate\ _{mild/cold} \times Highway\ Distance) \\ &\quad - (EV\ Mass \times g \times Altitude\ Change) \end{aligned} \tag{1}$$

The significance of each variable in the equation is detailed below to show how they collectively influence the EV's SoC upon arrival at the destination:

- EV Usable Capacity (kWh): This represents the total usable energy capacity of the EV's battery, measured in kilowatt-hours (kWh).
- Discharge City Rate (Wh/km): This is the rate at which the EV's battery discharges energy while driving in city conditions. It's measured in watt-hours per kilometre (Wh/km). There are two conditions considered: mild and cold, which refer to different battery discharge efficiencies under varying temperature conditions.

- Discharge Highway Rate (Wh/km): Like the discharge city rate, this is the discharge rate while driving on highways. Also measured in watt-hours per kilometre (Wh/km), with considerations for mild and cold conditions. Highway driving generally involves higher speeds and fewer stops, leading to a higher battery discharge rate compared to city driving.
- City Distance (m): This is the portion of the total distance travelled by the EV within city limits or urban areas, measured in metres (m).
- Highway Distance (in metres): This represents the part of the journey made on highways or freeways, also measured in metres (m).
- EV Mass (kg): This is the mass of the EV, measured in kilograms (kg). It's used to calculate the energy consumed due to changes in altitude.
- g (m/s^2): This is the acceleration due to gravity, which is assumed to be a constant value of 9.81 meters per second squared (m/s^2).
- Altitude Change (m): This is the change in total altitude over the course of the journey, measured in metres (m). It is used along with the EV's mass and the gravitational constant to calculate the energy consumed due to positive road gradient.

5.3.2.11 Iterating over CSs

After invoking the **get_charging_stations** function and successfully retrieving a set number of results (a modifiable parameter within the function), the system transitions into an iterative evaluation phase. Each CS is examined during this stage to determine its compatibility with the user's specific charging requirements.

The evaluation involves multiple layers of analysis. First, the system checks if the CS offers the type of charger compatible with the user's EV model by calling **is_compatible_charger**. This is particularly crucial if the user has indicated a preference for fast charging.

Once a station meets the charger compatibility criteria, the system calculates the temperature at each CS location. Then it averages it with the temperature at the origin location, which has been previously determined. This average temperature serves as an approximate measure for the entire journey. This renders the Navigator adaptive for extended routes, where temperature variations occur from the starting point to the endpoint. This is, moreover, particularly important because temperature can significantly impact the discharge rates of the EV's battery. Specifically, if the temperature is below 10 degrees, the **calculate_soc** function will use cold rates in the calculation, which imply higher discharge rates, affecting the vehicle's range and SoC.

Afterwards, the system uses **get_route_info** and **analyse_ route** functions to calculate essential route information. This includes the time and distance required to reach the CS from the current location and the walking time from the station to the final destination.

Finally, the system estimates the SoC the EV will have upon arrival at the CS using the **calculate_soc** function. This is calculated based on factors already explained such as the initial SoC, the type of roads on the route, and the EV's energy consumption rates.

This iterative evaluation serves as a comprehensive filter, ensuring that only the most suitable CSs are presented to the user. It also provides a holistic view of what the user can expect regarding travel time and EV battery status, enabling more informed decision-making. This meticulous approach aligns well with the system's overarching goal of maximising convenience and efficiency for the EV driver.

5.4 Error Handling

Effective error handling is crucial for maintaining the reliability and usability of software systems. This section outlines the key strategies our system implements to manage errors gracefully.

The system incorporates timeouts in API calls, ensuring the application does not become unresponsive during prolonged external service delays. If a request fails due to network-related issues or service unavailability, the system reports the error to the user for manual retry, preserving the user interface's responsiveness.

Upon receiving data from external services, the system checks the validity of the data format, particularly when parsing JSON responses. This validation process safeguards against processing and operational errors that could arise from malformed data.

Throughout the system, a structured exception handling strategy is implemented. The code includes try-except blocks that capture exceptions arising from network requests and database operations. While general exceptions are captured, the focus is on providing informative error messages to facilitate user understanding and corrective action.

The system ensures the robustness of user input through validation loops. Users are prompted to re-enter data should it fall outside of expected ranges or formats, enhancing the system's resilience to incorrect inputs.

6. Navigator: Standalone Deployment

6.1 Introduction

Chapter 6 explores the standalone deployment of the Navigator microservice, highlighting the integration of an API, the modular division into Docker containers, and the system's operational workflow. This chapter sets the foundation for understanding how the Navigator, in an independent deployment, can meet its objectives. The chapter emphasises the system's architecture and functionality, leading into its testing and validation in user scenarios.

6.2 System Design and Implementation

This section explores the design of the standalone deployment model of the Navigator microservice. In this architecture the integration of an API built with FastAPI [50] serves as the gateway for user interactions. The API is designed to receive POST requests containing the user parameters already commented (origin, destination, maximum radius for CSs, the EV model, fast charging priority and its initial SoC). Upon receiving these parameters, the API invokes the Navigator logic, which interacts with the portable database to fetch relevant data and perform computations. The results are then returned to the user via the API, completing the interaction loop.

This system is modularly divided into three primary components: the API, the Navigator logic and the EV specifications database, as it can be observed in Figure 5. To ensure that each component can be developed, tested, and deployed in isolation, they are each encapsulated within their own Docker container.

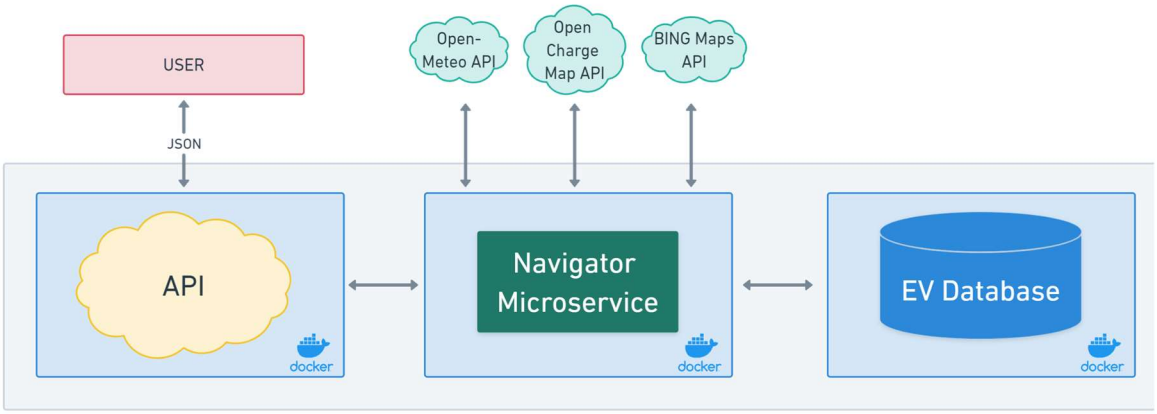

Figure 5: Navigator standalone model.

6.2.1 API Integration and Framework

The API is built using FastAPI, a modern, fast web framework for building APIs with Python [50]. FastAPI provides automatic interactive API docs, validation, serialisation, and many other features beneficial for this project. The API intermediates the user's client application and the Navigator logic. It receives POST requests containing the necessary parameters for the Navigator to process, such as the origin and destination locations, maximum radius for CSs, EV model, and initial SoC.

The API is configured to handle Cross-Origin Resource Sharing (CORS) using FastAPI's CORSMiddleware [51]. CORS, fundamentally, is a web browser security specification designed to control and manage requests originating from distinct domains. CORS dictates and filters which domains are granted permission to interact with the host system. This ensures that only authenticated and designated origins can access and exchange data, thereby fortifying the system against potential vulnerabilities or unauthorised intrusions. This is essential for enabling the client application to interact with the API securely. The origins are set to localhost for development purposes but can be updated to include the deployed client's URL.

The API defines a model to validate the incoming request data using the ‘pydantic’ python library [52]. This model ensures that the data conforms to the expected form, as it can be observed in Listing 2, such as format types and constraints, such as the SOC being between 0 and 100. Once the data is validated, the API calls the Navigator logic, passing in the necessary parameters. This level of validation is crucial for ensuring that the Navigator logic receives clean, usable data, thereby reducing the likelihood of runtime errors.

```
{
  "vehicle_model": "Nissan Leaf",
  "start_SOC": 0.8,
  "start_location": "Maastricht",
  "sojourn_location_center": "Aachen",
  "sojourn_location_radius": 2000.0,
}
```

Listing 2: JSON structure for incoming API request data in Navigator standalone deployment.

The term 'sojourn location center' refers to the central point or primary location within which the driver is seeking available charging options. Adjacently, the 'sojourn location radius' parameter, often referred to as the 'maximum radius', defines the permissible range in metres from this central point.

Selecting FastAPI as the tool to build the API comes with multiple advantages. At its core, FastAPI uses Starlette, which can be thought of as its foundational layer, efficiently managing web traffic and ensuring timely responses [53]. This is particularly important when there is a surge in data exchanges.

Furthermore, FastAPI is designed to create RESTful APIs efficiently. (Representational State Transfer) REST is an architectural style introduced by Doctor Roy Fielding in [54] which follows a set of rules and conventions that applications follow to communicate using standard web protocols. A significant aspect of these APIs is performing (Creating, Reading, Updating and Deleting) CRUD operations. This ensures that data interactions remain consistent and standardised.

One of FastAPI's standout features is its automatic generation of OpenAPI [55] and JSON Schema [56] documentation. OpenAPI, previously known as Swagger, provides a standardised format for describing RESTful APIs. It offers a clear way to define API endpoints, request-response structures, and authentication methods. This helps in creating comprehensive and user-friendly documentation that allows developers, even those unfamiliar with the specifics of the API, to understand its functionalities and capabilities quickly.

On the other hand, JSON Schema describes the structure and validation rules of JSON data. It is a blueprint for the data that the API expects to receive or send. With JSON Schema, developers can ensure the data consistency and integrity by validating incoming requests against predefined standards.

The API also incorporates robust error handling, implemented with try-except blocks in the code. This ensures that any exceptions raised during the execution are caught and logged, providing valuable debugging information while informing the client application of the issue in a user-friendly manner.

6.2.2 Containerisation and Benefits

Docker [57] is a platform that employs containerization technology to ensure that applications run seamlessly in any environment. Unlike traditional virtual machines that simulate entire operating systems, containers package up an application with all its dependencies and libraries, ensuring that it has everything it needs to run, regardless of where it is deployed. This encapsulation ensures a consistent runtime environment, mitigating issues that arise from discrepancies between development and production settings. Moreover, Docker containers are lightweight, making it possible to run multiple containers on a single machine without incurring the overhead typical of virtual machines. As a result, developers and system administrators can ensure both the portability and scalability of their applications, streamlining development and deployment processes.

Using Docker containers offers numerous benefits crucial for a seamless development lifecycle. Achieving a consistent environment through development, testing, and deployment phases is one of the key advantages [58], eliminating discrepancies and ensuring uniform application behaviour. Docker also simplifies dependency management, allowing developers to manage all dependencies within the container, reducing conflicts and improving system reliability.

Deployment and scaling are streamlined with Docker [59], facilitating quick updates and optimal resource utilisation, crucial for adapting to the system's dynamic needs and maintaining high performance and availability. Docker containers also enable a scalable and modular architecture, allowing each component to be developed, tested, and deployed independently. This is vital for continuous integration and continuous deployment practices, leading to faster development cycles and regular releases of high-quality software. Docker enhances security [59] by isolating applications and their dependencies, minimising vulnerability risks, and is backed up by secure image verification, ensuring the integrity of container images.

In this implementation, a triple-container approach has been adopted. The first container is dedicated to housing the API, where user requests are received and processed. The second container holds the Navigator microservice, serving as the operational core of the system. This container is intricately designed to manage the logic and functionalities of the ITS The third container is exclusively allocated for the database, ensuring a secure and isolated environment for data storage and management. This setup not only organizes the system components efficiently but also optimizes the operational flow, allowing each container to focus on its specialised tasks.

The communication between these containers is facilitated through well-defined interfaces and protocols [60]. Specifically, Docker utilizes networking drivers to manage how containers communicate:

- **bridge**: The default driver that creates a private internal network on the host system, allowing containers to communicate while isolated from external networks.
- **host**: This driver removes network isolation between the container and the Docker host, and uses the host's networking directly, essentially treating the container as if it is running directly on the host's network.
- **overlay**: Designed for multi-host communication, allowing containers across different hosts to communicate as if they are on the same local network.
- **macvlan**: This driver assigns a unique MAC address to each container, making them appear as physical network interfaces on a network, allowing direct communication with external network devices without a bridge.

Alongside these drivers, the Docker API [61] serves as a management interface for containers, enabling operations like starting, stopping, or inspecting containers programmatically. Additionally, in Docker's "swarm mode" [62] — a clustering and orchestration feature — service discovery mechanisms automatically recognize and connect containers based on their service names, simplifying multi-container communication.

To enable communication between containers, they are typically configured to be on the same Docker network, acting like a private internal network that allows containers to interact. This arrangement functions analogously to a private internal network, permitting containers to interact as if they were independent systems connected within a local area network. Each container can expose specific ports to interact with other containers or the external environment.

This structured and secure communication approach enables the containers to work in tandem, contributing to the overall functionality and performance of the system while maintaining its integrity and reliability. It ensures cohesive and unified operational flow, enhancing the system's responsiveness and adaptability to user needs, all while maintaining the sanctity of the data within.

Docker offers several key advantages for this framework. The segregation of the Navigator, API, and Database into distinct containers promotes a clear division of responsibilities, ensuring that each system can be developed, maintained, and scaled independently, thereby minimising downtime, ensuring optimal performance and resource usage. This separation also simplifies debugging processes, as issues can be pinpointed to a single container without ambiguity. This design choice also supports the SOGNO microservice architecture, further explained in Section 7.2.1.

On a more granular level, containerization affords specific advantages to each component:

- Database: housing it in its own container isolates sensitive information, providing an added layer of security.

- API: this isolation allows for more efficient management of incoming and outgoing network traffic, ensuring that the API can handle high volumes of requests without impacting the performance of other components.

- Navigator: individual containerisation is strategically advantageous when considering its integration into the larger evrich service, the Navigator container can be seamlessly integrated into diverse architectures with minimal modifications.

Moreover, future deployments in the evrich service could also see the benefit of containerisation. To handle high demand scenarios, multiple Navigator containers could be created from the same Navigator image. This capability could ensure that the service can handle a surge in user queries efficiently, as each Navigator container can process individual requests independently, ensuring swift response times and maintaining the overall system reliability.

6.2.3 Operational Workflow

The operational workflow commences when a user submits a request. Typically, this request originates from a client application made in the form of POST requests, which carry essential parameters shown in Listing 2, in Section 6.2.1. These parameters include the user's origin and destination locations, the EV's current SoC, the maximum radius for searching CSs, the specific EV model, and whether the user prioritises fast charging.

Upon receiving a POST request, the containerised API assumes the role of gatekeeper and intermediary. The FastAPI framework undertakes the task of data validation, employing the 'pydantic' library to confirm the received data's conformity to expected types and constraints. If the data encounters any discrepancies, the API immediately communicates this to the user, requesting rectification. Once validated, the API translates and relays the user's parameters to the Navigator logic, initialising the next phase in the operational flow.

The Navigator logic, thoroughly explained in Chapter 5 and which can be observed in Figure 6 below, processes the user data to furnish optimal navigation recommendations. The Navigator container communicates with the database container to extract the necessary information of the selected EV model specifications. It then collects the list of CSs that fall inside the specified radius and, after checking charger compatibility with the user’s EV, calculates the fastest route to each of the CSs. The Navigator then calculates the walking distance from the stations to the user’s specified destination and, finally, the final SoC at each CS.

Completing the operational loop, once the Navigator logic completes the calculations for each CS, this information is transmitted in a dictionary back to the user through the API, see Figure 6 below. The FastAPI framework manages the delivery of these results, encapsulating them in a structured format for easy comprehension by the user's client application. The user is then equipped with a detailed navigation plan, complete with suggested CS stops, estimated SoC after arriving to each, and an overall efficient distance and time to reach the suggested stations, plus the walking time from the stations to the final destination.

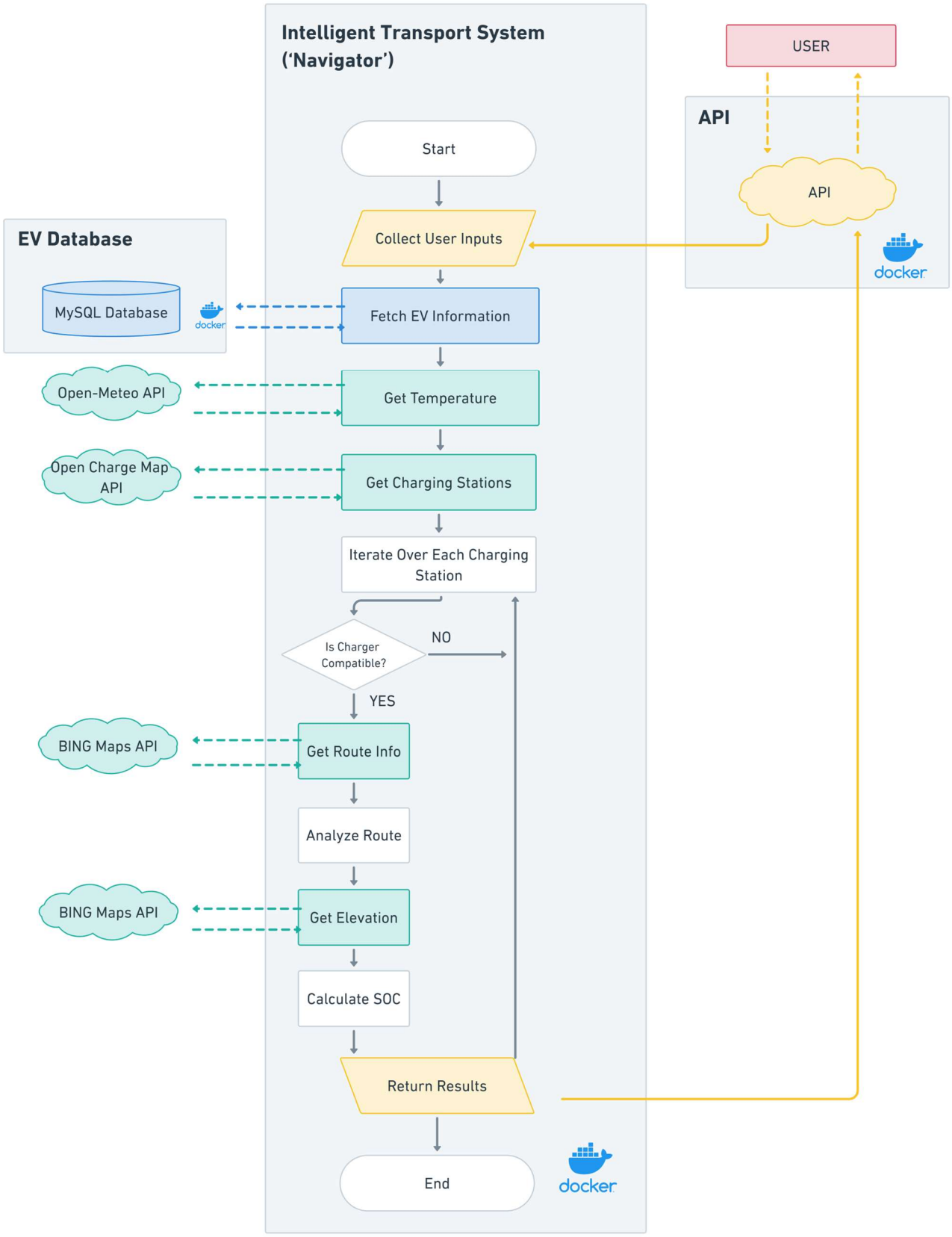

Figure 6: Expanded view of Navigator Standalone Deployment highlighting core microservice logic.

6.3 Practical Application and Validation

This section focuses on four user scenarios that demonstrate the Navigator microservice's functionality in scenarios reflective of actual use cases. Each scenario is designed to illustrate the system's flexibility in accommodating diverse requirements, including varying EV models, charging preferences, and environmental influences such as temperature and geography. The results from these scenarios provide tangible evidence of how the Navigator processes inputs to generate estimated SoC calculations and CS suggestions, showcasing its practical value and adaptability for EV drivers' needs. From everyday commutes to more challenging routes affected by weather and altitude, these scenarios will test the Navigator's capacity to optimise routes for EVs, emphasising its accuracy and dependability.

6.3.1 Scenario 1: Daily Common Trips

On a summer day in Valencia, a student living in Godella - a town on the city's outskirts - must attend morning lectures at Universidad Politécnica de Valencia (UPV) University in Valencia and leave their Nissan Leaf to charge for a few hours. The vehicle's battery is currently at 50% SoC. The student wants to park the car close to the destination to minimise walking. The student has sufficient battery charge and does not prioritise fast charging.

To find a suitable charging spot, the student runs Navigator with the following input parameters shown in Table 2 below.

Table 2: Input parameters for the Godella to UPV University route.

Start Location	**Sojourn Location Center**	**Sojourn Location Radius**	**Vehicle Model**	**Starting SoC**	**Is Fast Charging Priority?**
Godella	UPV University	1000m	Nissan Leaf	50%	no

The Navigator finds two CSs, shown in Figure 7, within the one-kilometre range with compatible chargers for Nissan Leaf, meeting the student's needs.

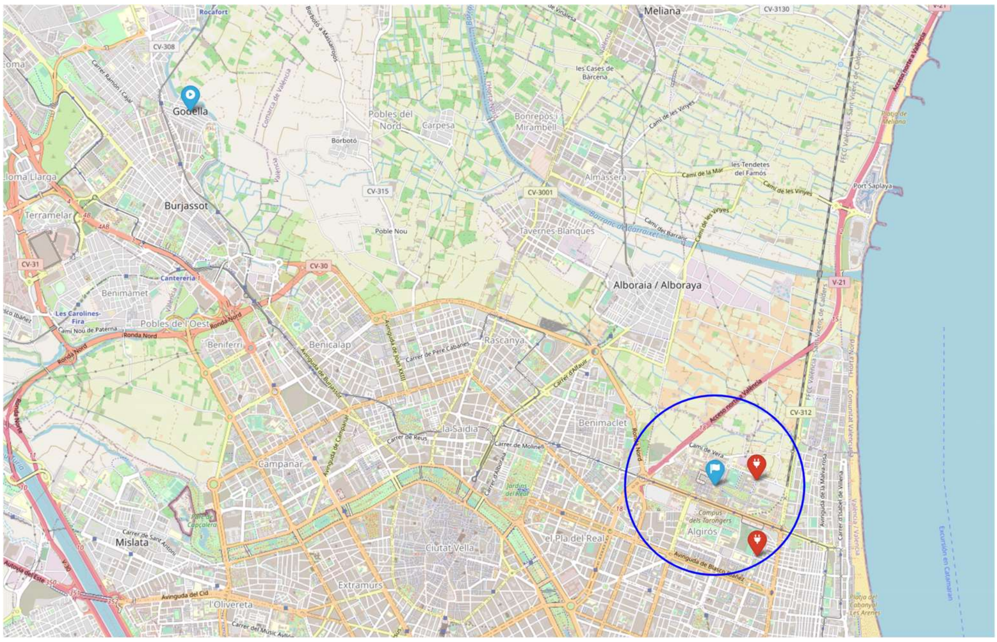

Figure 7: Mapping of the CSs within the search radius from Godella to UPV route.

The Navigator first, extracts the EV parameters from the database and calculates the mean temperature for the route, which can be observed in Table 3.

Table 3: Extracted EV parameters and route temperature for the Godella to UPV route.

EV Model	Weight	Useable Capacity	Charge Port	Fast charge Port	Route Temperature
Nissan-Leaf	1995 kg	39.0 kWh	Type 2	CHAdeMO	29.5°C

The Navigator generates a response listing two CSs, seen in Figure 8:

```
Charging Station 1: CEDAT UPV
Time: 20 mins
Distance: 10.30 km

Available chargers:
- Charger Type: Type 2 (Socket Only)
Operator: (Unknown Operator)
Price (if available):
Walking time: 8 mins
Elevation: -115.15 m
Pre-Elevation Adjustment SoC:
45.63 %
Final SoC: 45.63 %
```

```
Charging Station 2:
Repsol Camí del Cabanyal
Time: 19 mins
Distance: 10.00 km

Available chargers:
- Charger Type: CCS (Type 2)
- Charger Type: CHAdeMO
- Charger Type: Type 2
Operator: Repsol - Ibil (ES)
Price (if available): Unknown
Walking time: 15 mins
Elevation: -114.47 m
Pre-Elevation Adjustment SoC:
46.06 %
Final SoC: 46.06 %
```

Figure 8: CS information returned by the Navigator for the Godella to UPV route.

The Navigator shows that the distance between the origin and CSs is around 10 kilometres, which is a short distance that will not generate range anxiety. Both CSs show a similar final SOC of around 46%. However, no information is available about the pricing for either station, making it difficult for the user to choose. The time to get to both stations is only one minute apart, so the only significant difference is the walking distance, which is 8 minutes for CS 1 and 15 minutes for CS 2. It can be expected that the user decides to choose CS 1, in this case.

In this scenario, the Navigator effectively demonstrates its capability to provide precise and relevant information, enhancing the user's charging experience by allowing the user to make informed decisions. The user's specific requirements, such as location, vehicle model, and charging preferences, are accurately processed, and suitable CSs are identified within the specified radius. The detailed response, including distances, estimated SoC, and walking times, illustrates the ITS's data relevance. Although pricing information was unavailable in this instance, the Navigator still facilitated a user-friendly experience by narrowing down the options and highlighting the differences, enabling the user to select the most convenient CS.

6.3.2 Scenario 2: EV Range: Critical SoC and Temperature Influence

6.3.2.1 Standard Conditions Journey

The user, living in Valencia, owning an EV for less than a year is attending important job meetings in Barcelona that occur once a month. Typically, the user is used to completing this journey with at least 80% initial SoC, influenced over range anxiety issues. The user prompts the input parameters seen in Table 4.

Table 4: Input parameters for the Valencia to Barcelona route.

Start Location	Sojourn Location Center	Sojourn Location Radius	Vehicle Model	Starting SoC	Is Fast Charging Priority?
Valencia	Sagrada Familia, Barcelona	2000 m	Tesla Model S Plaid	80%	no

The Navigator then retrieves EV parameters from the database and calculates the average temperature along the route (Table 5).

Table 5: Extracted EV parameters and route temperature for the Valencia to Barcelona route.

EV Model	Weight	Useable Capacity	Charge Port	Fast charge Port	Route Temperature
Tesla-Model-S-Plaid	2629 kg	95.0 kWh	Type 2	CCS	25°C

During this scenario, the Navigator identifies ten compatible CSs with compatible chargers for the EV, within a 2000m radius of the Sagrada Familia, Barcelona (Figure 9).

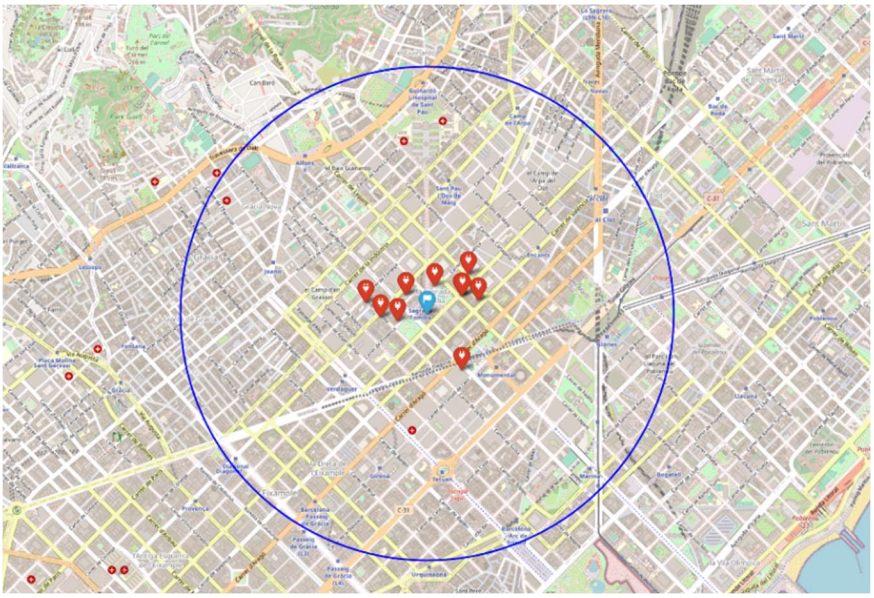

Figure 9: Mapping of the CSs within the search radius for the Valencia to Barcelona route.

The Navigator generates a response listing 10 CSs complete with relevant details for each. To ensure clarity and prevent visual overload, only the information for the first four CSs out of the ten is presented in Figure 10 below:

Charging Station 1: Hotel Ayre Roselló Time: 213.3 mins Distance: 365.72 km Available chargers: - Charger Type: Type 2 (Socket Only) Operator: Endesa Price (if available): 0,39€/kWh Walking time: 4 mins Elevation: 26 m Pre-Elevation Adjustment SoC: 11.5% Final SoC: 11.4%	**Charging Station 2: B:SM Av Gaudí - Sagrada familia (TRIO) FC009** Time: 212.9 mins Distance: 374.15 km Available chargers: - Charger Type: Type 2 - Charger Type: CHAdeMO - Charger Type: CCS (Type 2) Operator: B:SM (ES) Price (if available): Unknown Walking time: 4 mins Elevation: 22 m Pre-Elevation Adjustment SoC: 10.0% Final SoC: 9.8%
Charging Station 3: Parking Sagrada Familia Time: 213.8 mins Distance: 365.83 km Available chargers: - Charger Type: Type 2 (Socket Only) - Charger Type: Type 2 (Socket Only) Operator: Iwatatt (Es) Price (if available): 0,41€/kWh Walking time: 3 mins Elevation: 24 m Pre-Elevation Adjustment SoC: 11.5% Final SoC: 11.3%	**Charging Station 4: Parking Bond - Sagrada Familia** Time: 211.6 mins Distance: 373.94 km Available chargers: - Charger Type: Type 2 (Socket Only) - Charger Type: Type 2 (Socket Only) Operator: Iwatatt (Es) Price (if available): 0,41€/kWh Walking time: 3 mins Elevation: 20 m Pre-Elevation Adjustment SoC: 10.0% Final SoC: 9.8%

Figure 10: CS information returned by the Navigator for the Valencia to Barcelona route.

6.3.2.2 Reduced SoC Challenge

In the following month, as the user prepares for their routine trip to Barcelona, they inadvertently overlook the initial SoC of their EV. Assuming it to be sufficiently high, they later discover it to be at 77%. This unexpected realisation leads them to prompt the Navigator with this updated condition for the Valencia to Barcelona route, see Table 6. It is important to note that the temperature remains consistent with the conditions of the optimal journey (Section 6.3.2.1), see Table 5.

Table 6: Input parameters for the Valencia to Barcelona route (77% initial SoC).

Start Location	Sojourn Location Center	Sojourn Location Radius	Vehicle Model	Starting SoC	Is Fast Charging Priority?
Valencia	Sagrada Familia, Barcelona	2000 m	Tesla Model S Plaid	77%	no

The Navigator identifies the same 10 CSs as in the previous instance, see Figure 9, however, the final SoC for these CSs falls below 10%.

Figure 11 displays only two representative CS details returned by the Navigator, for clarity in demonstration. The first CS illustrates a typical response for a CS where the final SoC is above 7.5%, showcasing standard recommendation criteria.

In contrast, the second CS highlights the Navigator's more cautious approach for CSs where the final SoC dips below the 7.5% threshold. This indicates to the user that the SoC at arrival in these stations is deemed too low for the Navigator to recommend them as reliable stops confidently.

```
Charging Station 1: Hotel Ayre Roselló  Charging Station 2: B:SM Av Gaudí
Time: 213.3 mins                        - Sagrada familia (TRIO) FC009
Distance: 365.72 km
                                        Predicted SoC at this charging station
Available chargers:                     is too low (6.8%).
- Charger Type: Type 2 (Socket Only)    Please choose a closer destination.
Operator: Endesa
Price (if available): 0,39€/kWh
Walking time: 4 mins
Elevation: 26 m
Pre-Elevation Adjustment SoC:
8.5%
Final SoC: 8.4%
```

Figure 11: CS information returned by the Navigator for the Valencia to Barcelona route (reduced SoC).

The service adopts a conservative approach in estimating SoCs to ensure reliability and prevent range anxiety. It avoids recommending CSs if the predicted arrival SoC is below 7.5%. This threshold accounts for the variability in battery conditions and degradation rates among EVs, which may not align with standard manufacturer specifications.

6.3.2.3 Temperature Influence Challenge

In this scenario, the same user embarks on their routine journey during the winter season, faced with average temperatures below 10 degrees Celsius. This change presents an opportunity to analyse the impact of colder weather on the final SoC of the EV. The journey user parameters remain consistent with the standard journey conditions and initial 80% SoC (Section 6.3.2.1), as outlined in Table 4.

Upon receiving these inputs, the Navigator retrieves the EV's specifications and calculates the route's mean temperature, as shown in Table 7.

Table 7: Extracted EV parameters and route temperature for the Valencia-Barcelona route in cold temperature circumstances.

EV Model	Weight	Useable Capacity	Charge Port	Fastcharge Port	Route Temperature
Tesla-Model-S-Plaid	2629 kg	95.0 kWh	Type 2	CCS	8°C

During the assessment, the Navigator identifies the same CSs as in previous journeys (Section 6.3.2.1 and 6.3.2.2). However, it computes an estimated SoC at the first CS that is below zero, indicating the factoring and significant impact of the cold weather on the vehicle's range calculations by the ITS. Recognizing that other nearby CSs would likely yield similar results, the system advises against continuing with these options, as illustrated in Figure 12.

```
Your range is not enough to reach the destination.
Please choose a closer destination.
```

Figure 12: Navigator's alert for insufficient vehicle range to reach the destination.

Considering these findings, the user is recommended to select an alternative destination within the EV’s range, preferably a location a few kilometres before the initially intended destination, to ensure secure charging without exhausting the EVs range.

This user example illustrates the significant impact of temperature on EV SoC calculations and the adaptability of the Navigator system. It shows how the Navigator adjusts its estimates in response to temperature changes, emphasising the substantial difference in EV range below ten degrees Celsius compared to warmer conditions. While the system offers precise charging options and SoC estimations during milder months, it quickly adapts in colder conditions, acknowledging range limitations and advising users accordingly. This approach demonstrates the Navigator's commitment to providing conservative, reliable estimations, enhancing user safety and satisfaction, and reducing range anxiety.

6.3.3 Scenario 3: Prioritising Fast Charging in Route Planning

In this scenario, a user intends to journey from Aachen to the city centre of Dusseldorf. Given that the user has limited time to spend in Dusseldorf and their EV has an initial SoC of 65%, locating a CS compatible with their EV's fast-charging port becomes crucial. Initially, the user interacts with the Navigator service, inputting these specific conditions (see Table 8) to identify a CS within a reasonable walking distance from their intended destination.

Table 8: Input parameters for the Aachen to Düsseldorf route with fast-charging priority.

Start Location	Sojourn Location Center	Sojourn Location Radius	Vehicle Model	Starting SoC	Is Fast Charging Priority?
Aachen	Düsseldorf	1000 m	Nissan Leaf	65%	yes

The Navigator first, extracts the EV parameters from the database, storing the fast-charging port that will use for comparison, and calculates the mean temperature for the route. All can be observed in Table 9 below.

Table 9: Extracted EV parameters and route temperature for the Aachen to Düsseldorf route.

EV Model	Weight	Useable Capacity	Charge Port	Fastcharge Port	Route Temperature
Nissan-Leaf	1995 kg	39.0 kWh	Type 2	CHAdeMO	20.3°C

The Navigator finds 6 CSs within the 1000 metres range but none with chargers compatible with Nissan Leaf's fast charging port, therefore it prompts an alert to the user (Figure 13).

```
No charging stations are compatible with a fast-charging port in your EV.
                Start again without fast-charging priority.
```

Figure 13: Navigator's alert for no CSs with compatible fast-charging ports found.

Upon receiving the initial response from the Navigator, the user grows concerned about the potential unavailability of CSs compatible with the CHAdeMO port specific to the Nissan Leaf.

To mitigate this concern, the user opts to query the Navigator once more, this time expanding the search radius to 3000 metres, see Table 10 below.

Table 10: Input parameters for the Aachen to Düsseldorf route after increasing maximum radius.

Start Location	Sojourn Location Center	Sojourn Location Radius	Vehicle Model	Starting SoC	Is Fast Charging Priority?
Aachen	Düsseldorf	3000 m	Nissan Leaf	65%	yes

Fortunately, the Navigator finds two CSs inside the new maximum radius that offer fast charging compatible with CHAdeMO, see Figure 14.

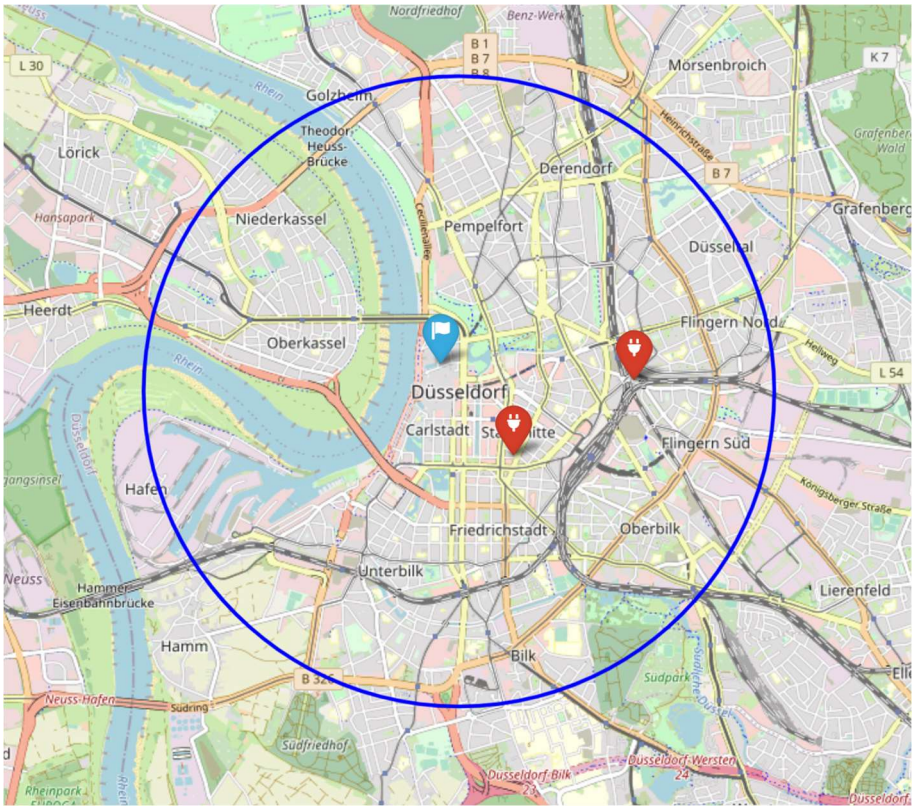

Figure 14: Mapping of the CSs within the search radius from Aachen to Düsseldorf route.

The Navigator sends the response back, see Figure 15. Both stations can be reached with an estimated remaining SoC of approximately 12%, which is encouraging news. This allows the user to proceed directly to Düsseldorf and charge their vehicle while spending a few hours in the city. Given that neither station provides pricing indicators, the user may opt for a station closer in walking time to their ultimate destination.

Charging Station 1: Hauptbahnhof	**Charging Station 2: Gustav Heinemann Str. 9**
Time: 55.5 mins	Time: 57.0 mins
Distance: 80.55 km	Distance: 80.93 km
Available chargers:	Available chargers:
- Charger Type: Type 2 (Tethered Connector)	- Charger Type: Type 2 (Tethered Connector)
- Charger Type: CHAdeMO	- Charger Type: CCS (Type 2)
- Charger Type: CCS (Type 2)	- Charger Type: CHAdeMO
Operator: (Business Owner at Location)	Operator: EnBW (D)
Price (if available): Unknown	Price (if available): Unknown
Walking time: 18 minutes	Walking time: 22 minutes
Elevation: -135 m	Elevation: -130 m
Pre-Elevation Adjustment SoC: 12.7%	Pre-Elevation Adjustment SoC: 12.5%
Final SoC: 12.7%	Final SoC: 12.5%

Figure 15: CS information returned by the Navigator for the Aachen to Düsseldorf route.

In this instance, the Navigator adeptly demonstrates its capability to refine search parameters in real-time, based on user preferences and requirements. Initially, the user's specific need for a fast-CS within a limited radius yields no results, reflecting the system's accuracy in filtering and presenting only compatible options. However, when the user broadens the search radius, the Navigator promptly identifies suitable stations, showcasing its responsiveness and flexibility in accommodating user modifications to search criteria.

This scenario underscores the Navigator's aim to adapt to user’s needs, such as fast charging, ensuring they are presented with viable options. It highlights the system’s utility in facilitating informed decision-making for users, allowing them to optimise their journeys efficiently, even when faced with constraints like time and charging speed preferences.

6.3.4 Scenario 4: Elevation Influences on Route Planning

This example highlights the route's gradient critical role in the Navigator's SoC calculations. As previously elaborated, changes in altitude are factored into the calculations to account for potential energy shifts, which are then reflected in the final SoC estimates.

In the first scenario, the user wants to complete a journey of approximately 200 km from Barcelona to the Principality of Andorra in the Pyrenees mountains, user inputs are shown in Table 11.

Table 11: User inputs for Barcelona-Andorra route.

Start Location	Sojourn Location Center	Sojourn Location Radius	Vehicle Model	Starting SoC	Is Fast Charging Priority?
Barcelona	Andorra	3000 m	Mercedes EQE 350 plus	100%	no

In the second example, another user wants to complete a slightly longer route of 230 km from Amsterdam to Aachen with the same EV and initial SoC as the route before, see Table 12.

Table 12: User inputs for Amsterdam-Aachen route.

Start Location	Sojourn Location Center	Sojourn Location Radius	Vehicle Model	Starting SoC	Is Fast Charging Priority?
Amsterdam	Aachen	3000 m	Mercedes EQE 350 plus	100%	no

The query results in the Navigator providing a final SoC of 57.9%, starting from a fully charged EV for the Barcelona to Andorra route. However, the SoC is revised to 47.5% when altitude adjustments are incorporated, shown in Figure 22. Notably, an ascent of 1300 metres can lead to a 10% reduction in SoC, attributable to the energy required to overcome gravitational potential. This adjustment provides the user with a more accurate SoC estimate, mitigating the risk of overconfidence.

The second trip of Aachen to Amsterdam, being of longer length, results in a final SoC of 53.5%. After accounting for an altitude change of just 162 metres, the SoC is minimally adjusted to 52.2%, see Figure 16 below. Interestingly, this demonstrates that a greater travel distance does not necessarily correlate with a lower SoC, provided all other conditions remain constant except for altitude. This underscores the significance of considering altitude as a key variable in SoC calculations.

It is worth noting that the full responses from both Navigator queries, which include a list of all available CSs, have been omitted for this scenario. This focuses on the key aspects relevant to the case at hand.

Barcelona – Andorra

Charging Station 1:
Encamp La Palaqueta
Time: 181.8 mins
Distance: 208.65 km

Available chargers:
- Charger Type: Type 2 (Socket Only)
- Charger Type: Europlug 2-Pin (CEE 7/16)
Operator: (Unknown Operator)
Price (if available): Unknown
Walking time: 8 minutes
Elevation: 1322 m
Pre-Elevation Adjustment SoC: 57.9%
Final SoC: 47.5%

Amsterdam – Aachen

Charging Station 1:
Mostardstraße 5
Time: 132.5 mins
Distance: 230.4 km

Available chargers:
- Charger Type: CEE 7/4 – Schuko Type F
- Charger Type: Type 2 (Socket Only)
- Charger Type: CEE 7/4 – Schuko Type F
- Charger Type: Type 2 (Socket Only)
Operator: Unknown
Price (if available): Unknown
Walking time: 2 minutes
Elevation: 162 m
Pre-Elevation Adjustment SoC: 53.5%
Final SoC: 52.2%

Figure 16: CS information returned by the Navigator for both the Barcelona to Andorra steep route and the Aachen to Amsterdam flat route.

This example underscores the importance of considering the altitude variable to add value to the accuracy and reliability of the Navigator's recommendations and estimations. However, it would be highly beneficial for future developments to include real-life validation processes, incorporating direct feedback from users. Such empirical testing, particularly focusing on the altitude factor, is essential to assess the precision and reliability of the Navigator's route estimations for EV drivers and could offer an opportunity to fine-tune the Navigator based on user experiences, thereby enhancing its effectiveness and user satisfaction in practical scenarios.

7. evrich: Integrated Deployment

7.1 Introduction

Following the development of the standalone Navigator deployment detailed in Chapter 6, this chapter demonstrates how the Navigator operates within the broader context outlined in Chapter 3, showcasing its functional integration into the evrich –EV routing in charging hubs- service. This chapter will start by introducing the SOGNO architecture and providing insights into evrich's actual model and design, leading into its testing and validation in a user scenario.

7.2 Evrich: Service Overview

7.2.1 SOGNO Architecture

Service-based Open-source Grid automation platform for Network Operation of the future (SOGNO) is, in essence, a cloud platform with open-source grid automation software, catering for the needs of grid operators. It leverages a microservice-based architecture to extend functionalities in a flexible and scalable way, integrating Distributed Management Systems (DMSs) to ensure a sturdy and adaptable grid infrastructure. A DMS is a collection of applications designed to monitor and control an entire electric distribution network efficiently and reliably, see Figure 17. This innovative approach efficiently manages evolving assets that might function as loads, resources, or provide flexibility to the grid. Initially designed with the upcoming 5G mobile network in mind, SOGNO aims to harness 5G's low latency and high availability for more reliable grid operations.

SOGNO has been tested in real distribution system sites, showing its ability to facilitate smooth and cost-effective network operations. The open-source nature of SOGNO is a key feature, enabling a collaborative approach towards improving grid automation. It offers a set of automation functions like state-estimation, load prediction, and voltage control, packaged for easy deployment within cloud environments. Moreover, SOGNO supports the integration of real-time simulation, allowing thorough testing of new automation functions against a virtual real-time representation of the power system before they are deployed. This feature is particularly useful for ensuring the reliability and effectiveness of new automation functions before they are integrated into actual grid operations.

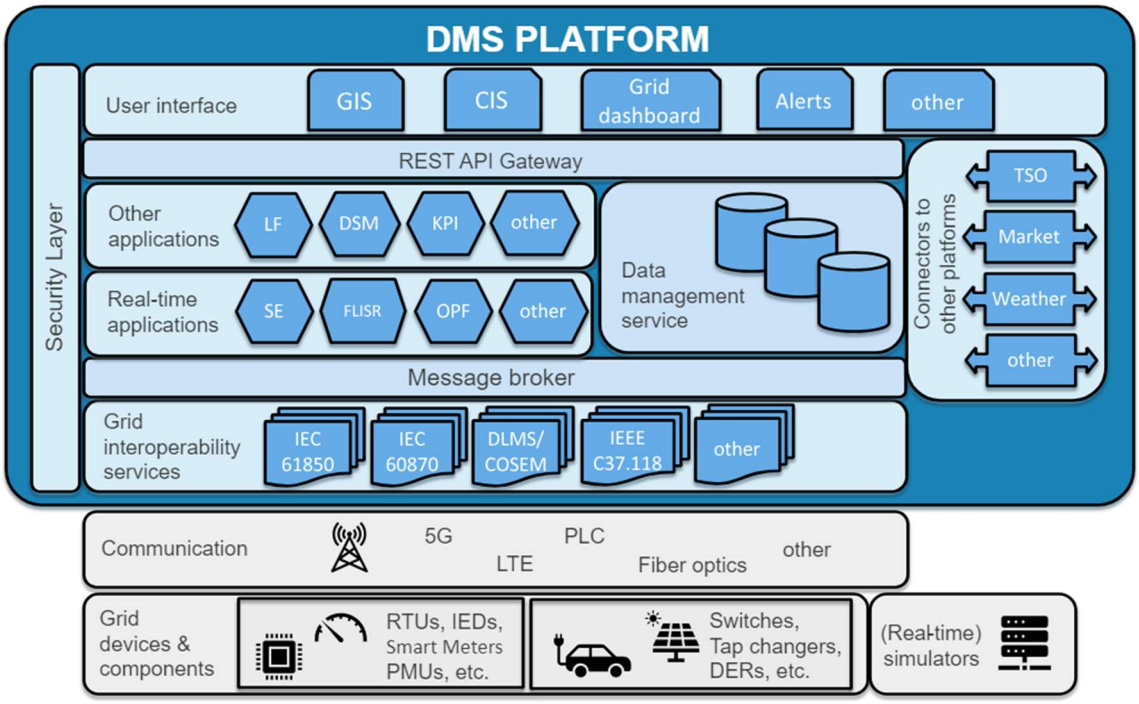

Figure 17: Distributed Management System Platform with SOGNO architecture [26].

7.2.2 Initial Proposed evrich Service Model

evrich is an intelligent cloud-based routing service specifically designed for EVs in urban environments with dense CI, referred to as urban charging hotspots. Open-source code for evrich is available in the Linux Foundation Energy repository in [63].

The platform focuses on helping EVs find the most optimal charging offers from various CSs that act as energy aggregators. These aggregators (CSs operators) provide grid-to-vehicle (G2V) and vehicle-to-grid (V2G) charging services. The purpose of evrich is to assist EV drivers in finding charging offers that best meet their preferences and requirements while also helping aggregators fulfil their G2V and V2G service contracts. Additionally, it aims to enhance the distribution system operator's capabilities in monitoring and predicting the state of the distribution grid that supplies energy to the CSs.

The evrich service follows a service-oriented architecture and is integrated as a function of the DMS within the distribution system operator's technical setup. It leverages microservices, each performing a specific function, such as querying charger availability and optimising charger selection. The microservice-based architecture is chosen because it is highly scalable and flexible, allowing for easy upgrades and integration of new functionalities. Each microservice is independent and communicates with others through input/output interfaces via lightweight communication mechanisms.

These microservices are crucial to the system's overall functioning and communicate using the lightweight, publish-subscribe network protocol (Message Queuing Telemetry Transport) MQTT [64] protocol.

An essential component of the service is its containerised API. EV drivers or clients can post their charging preferences, which include various parameters such as vehicle ID, model, battery specifications, starting location, desired SoC, and willingness to allow discharging of their batteries.

Once the client request is received, a series of microservices within evrich are executed to find the optimal charging offer. The overall workflow involves four main types of microservices: the Coordinator, the Connector, the Optimiser, and the ITS, see Figure 18:

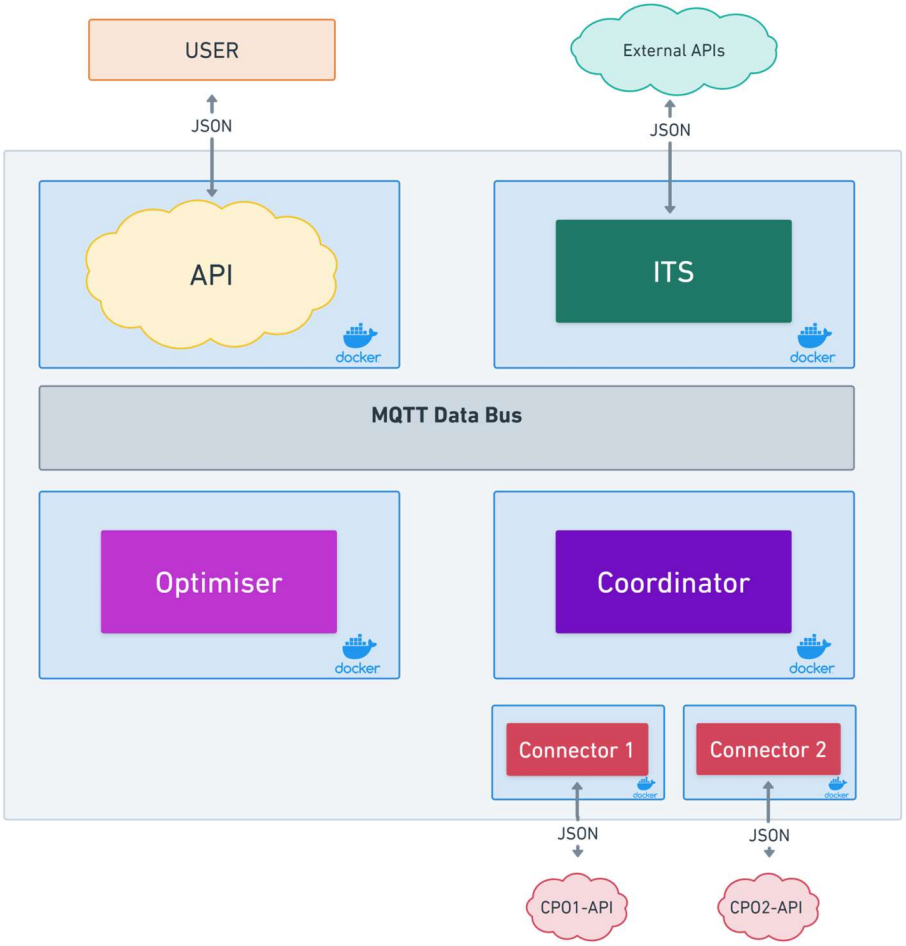

Figure 18: evrich initial proposed model.

The API publishes a message to a topic on the MQTT broker for the ITS in response to a client's request. The ITS´s key responsibilities in evrich include:

1. Identifying CSs within the maximum distance from the destination provided by the EV driver.
2. Determining the best driving routes to these CSs.
3. Calculating the estimated travel time, distance to destination and SoC for the EV at these CSs.

The Coordinator microservice manages the whole workflow, communicating with all other containers via MQTT messages that extend to both the API and microservices. The Coordinator manages the ITS's response on different CSs and sends availability requests to the Connectors that will communicate to each external aggregator. It also receives the final decision from the Optimiser to inform the user via the API.

The Connectors are designated for gathering real-time data concerning the chargers overseen by aggregators. Post receiving availability queries from the Coordinator, a Connector communicates with an aggregator's EMS to check availability, pricing, among other details, before relaying this data back to the coordinator.

In this setup, aggregators employ specialised software tools named EMSs to oversee and manage distributed energy resources. These EMSs are equipped with information and communication technologies facilitating real-time data interaction with utility companies. SOGNO service operates on the assumption that these EMSs of the aggregators possess APIs capable of responding to queries about charger availability. Every Connector is associated with an aggregator's EMS (e.g., EMS-AG-1). When a Connector sends an availability inquiry to an aggregator's EMS, the query contains details like the estimated parking duration for the EV, the energy amount needed by the EV, and the desired accuracy for the dynamic pricing signal. The aggregator processes this data using decision-making algorithms, responding with:

1. The identification information of the available charger and its charging/discharging limits.
2. The amount of energy the aggregator is willing to supply to the EV within the given parking duration.
3. The dynamic pricing signal indicates the price per kWh for charging at different intervals.

The Optimiser microservice executes the routing algorithm to determine where the incoming EV should be charged within multiple CSs. It leverages the data obtained by the Coordinator and Connectors via optimisation models to make this decision.

All microservices are deployed in Docker containers [57], which enables them to be bundled along with their dependencies into a standardised unit for software development. This containerisation approach ensures the system is easily deployable and maintainable, enhancing its portability across different computing environments.

7.2.3 Integrated evrich Service Model

The refined evrich service model preserves all functionalities from the initial proposal from the ACS institute, but now integrates the Navigator microservice reflecting the intended design, with the parallel database container and selected external services—Open-Meteo, Bing Maps, and OCM—detailed in Section 5. Figure 19 illustrates this integration.

The confirmation of the Navigator's core functionality and compatibility with the database and external services through standalone deployment, as detailed in Section 6, facilitates its subsequent integration within the broader evrich service architecture.

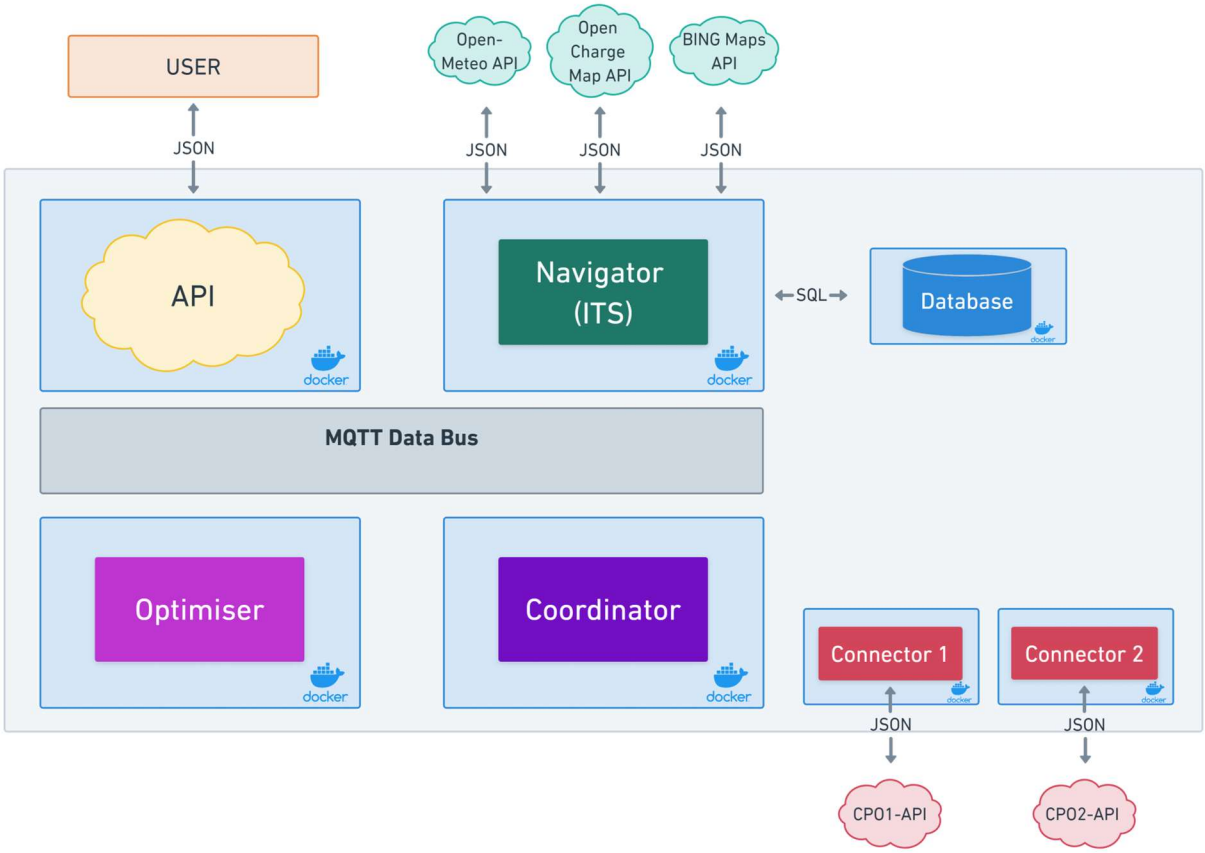

Figure 19: evrich service model after Navigator integration.

7.3 System Design Details

7.3.1 Communication Infrastructure

The evrich service comprises an API and MQTT message broker that serve as foundational components that facilitate seamless communication between various microservices and end-users.

MQTT is a lightweight messaging protocol designed for low-bandwidth, high-latency or unreliable networks, commonly used in IoT applications. It follows a publish-subscribe model rather than the client-server model used in HTTP communication.

The API, implemented using FastAPI, acts as the entry point for client requests. It exposes a RESTful interface that allows clients to send HTTP POST requests containing essential parameters for routing and charging. These parameters are shown below in Listing 3:

```
{
   "vehicle_id": "1",
   "vehicle_model": "Nissan Leaf",
   "start_SOC": 0.8,
   "start_time": null,
   "start_location": "Maastricht",
   "sojourn_location_center": "Aachen",
   "sojourn_location_radius": 2000.0,
   "sojourn_period": 3600.0,
   "demand_target_SOC": 1.0,
   "demand_v2g_allowance": 0.0
}
```

Listing 3: JSON structure for incoming API request data in evrich deployment.

The 'sojourn period' parameters indicative of the duration an EV driver plans to park their vehicle. It offers the system a preliminary insight into the vehicle's availability window for charging. The ‘Demand Target SoC’ encapsulates the desired SoC level the EV driver aims their vehicle to reach post the charging session. Lastly, the concept of V2G (Vehicle-to-Grid), denotes if the user is willing to provide stored energy from the EV back to the power grid.

Upon receiving a POST request at the endpoint, the API initiates a series of actions. It first validates the incoming data against the predefined schema of the class observed in Figure 10. After validation, the API publishes this data to an MQTT topic, using the Paho MQTT client library [65]. The API also subscribes to the response topic to receive the final processed data from the Coordinator.

The MQTT data broker, implemented using the Mosquitto [66] MQTT broker, plays a pivotal role in this architecture. It serves as a message broker that decouples the API from the various microservices, such as the Coordinator, Navigator and Connector. The broker enables asynchronous communication by acting as an intermediary, allowing each component to operate independently. This decoupling is particularly beneficial for system scalability and fault tolerance. For instance, if a microservice goes down or is undergoing maintenance, the broker can queue messages until the service is back online.

The utility of having an MQTT data broker becomes evident when considering the real-time nature of the system. The broker efficiently handles the publish-subscribe messaging pattern, ensuring that all subscribed microservices receive the messages intended for them. This is crucial for a system that requires real-time decision-making based on CS availability, and dynamic pricing.

7.3.2 Coordinator

In the evrich system, the Coordinator microservice is a central hub that manages various tasks to provide an optimised routing and charging plan. It starts by listening to incoming client requests through the MQTT protocol. These messages contain all the parameters shown before on Listing 3 that have been sent from the API to the MQTT data broker once the user posts an HTTP request.

Once a message from the client is received, the Navigator, also subscribed to that topic receives the user query. Its function is very similar to the one in the standalone deployment, to gather CS data, including estimated travel times and distances at potential CSs and the expected SoC upon arrival.

After obtaining the CS data, the Coordinator issues queries to Connectors, which link the evrich service to the external EMSs of the aggregators, to check the availability of CSs. These Connectors are responsible for specific CSs and can provide details like available power and charging slots. This information is essential for receiving information about suitable charging options based on the vehicle's needs and the current state of the CI.

With all the necessary data, the Coordinator sends these parameters to the Optimiser. This service solves the optimised routing algorithm introduced in [67] to find the best possible charging location and scheduling, considering time, energy efficiency, and cost.

Finally, the Coordinator receives the optimised plan and translates it into a final routing decision, which is then sent to the user.

7.3.3 Navigator and Database

Although extensively discussed in previous sections of this thesis, the Navigator microservice merits a brief mention of its role in the evrich architecture. The Navigator publishes a list of viable CSs after receiving the user query through the MQTT broker. This list is complete with details such as the EV's final SoC, walking distance, and time required to reach the station.

The database follows the same architectural pattern as described in earlier sections. The database is containerised in a single Docker container for portability and ease of data management. This approach simplifies deployment and makes it easier to update or add more information related to EVs. Importantly, the connection between the Navigator and the database is facilitated through a Docker network, ensuring a seamless flow of information, and maintaining the system's overall coherence and efficiency.

The primary distinction between the Navigator microservice used in standalone deployment and the evrich service lies in the handling of charger compatibility checks for a user's EV and the lack of fast charging priority consideration. In the evrich service setup, the responsibility for compatibility verification is shifted to the Coordinator. It collaborates with the Connectors to manage these checks during availability queries.[3]

7.3.4 Connector

The Connector microservice is a critical intermediary that bridges the gap between the Coordinator and the external platforms. Its primary role is to facilitate data exchange with the aggregators, ensuring that the system remains informed and can make decisions based on the most recent and relevant information.

Upon initialisation, the Connector retrieves essential environment variables, such as its unique identifier, the aggregator's URL, and the specific MQTT topics it should subscribe to and publish on. These environment variables are crucial for the Connector to know which aggregator it corresponds to and which topics it should listen to for incoming requests. Once a connection to the MQTT broker is established, the Connector subscribes to its designated request topic, awaiting incoming messages.

[3] Currently, the evrich service's testing phase does not include live charger compatibility checks or fast charging priority considerations due to the use of mock-up aggregators. In the future, as real aggregators provide the type of charger, the system will enable the Coordinator, with the support of Connectors, to perform these verifications.

A message from the Coordinator informs of estimated arrival and departure times and energy demand. The Connector then formulates a request to the external aggregator service using the provided URL, receiving real-time data on charger availability, pricing, and other parameters. Upon receiving a response from the aggregator, the Connector processes the data and publishes it to the designated response topic on the MQTT broker. This streamlined process ensures the system remains updated with the latest availability data from various aggregators, allowing for more informed and efficient routing decisions. It also enables scalability as new connectors can be added to expand the system's reach.

7.3.5 Optimiser

The Optimiser microservice receives the parameters from the Coordinator and solves a Mixed-Integer Linear Programming (MILP) problem to find the optimal charging schedule and cluster for the EV. The results are then published back to an MQTT topic to which the Coordinator is subscribed. The Optimiser is designed for efficiency and customizability, utilising advanced optimisation algorithms to find the most cost-effective and energy-efficient charging schedule.

7.3.6 Containerisation

The evrich platform has been designed and implemented to be a robust and efficient system, to facilitate routing EVs to parking lots with CSs, following the guidelines of SOGNO architecture. A pivotal aspect of the design is the utilisation of containerisation, which has been achieved through Docker containers. Each functional component of the platform, such as the API, coordinator, connectors, and optimiser, is housed in its own Docker container. This design choice offers a variety of benefits, including isolation, scalability, and ease of deployment.

The use of Docker containers for individual components, as elaborated in Section 6.2.2 of Chapter 6, provides significant benefits in terms of component isolation and scalability. This isolation implies that a failure in one component does not necessarily result in a cascade effect, bringing down the entire system. Moreover, each component can be scaled independently. For instance, if the platform experiences more queries, the API or Navigator component can be scaled up independently without affecting other components.

7.4 Operational Workflow

In a scenario where a user wishes to optimise their charging experience, it is presumed that an application has been developed for user interface connection and input retrieval.

The driver inputs their specific requirements, such as charging preferences and location, into a potential future mobile application. This application then sends a POST in JSON format to the API, which in turn, publishes this data to an MQTT topic. The Coordinator, subscribed to this topic, acts as the central orchestrator of the system.

The Navigator also receives the message published by the API on the subscribed topic. The microservice identifies potential CSs within a specific radius around the driver's location and calculates the most efficient route to these stations, providing details on parameters such as distance, time, and estimated SoC upon arrival.

After identifying the potential CSs, the Navigator returns this information to the Coordinator. The Coordinator then engages the Connector microservices to inquire about the real-time availability and pricing of these stations. The Connector contacts each station and gathers essential data, such as current availability, pricing, and other conditions affecting the driver's charging experience.

Once the detailed information from the Connector is received, the Coordinator forwards it to the Optimiser. The Optimiser considers all the gathered data, including the driver's preferences and the real-time conditions provided by the Connector, to calculate the most efficient charging schedule.

Finally, the optimised schedule is sent back to the Coordinator, who then relays both the optimised route from the Navigator and the efficient charging schedule from the Optimiser to the API. This ensures the driver receives a seamless, efficient, and fully informed charging experience. The entire workflow can be better understood by looking at Figure 20 below.

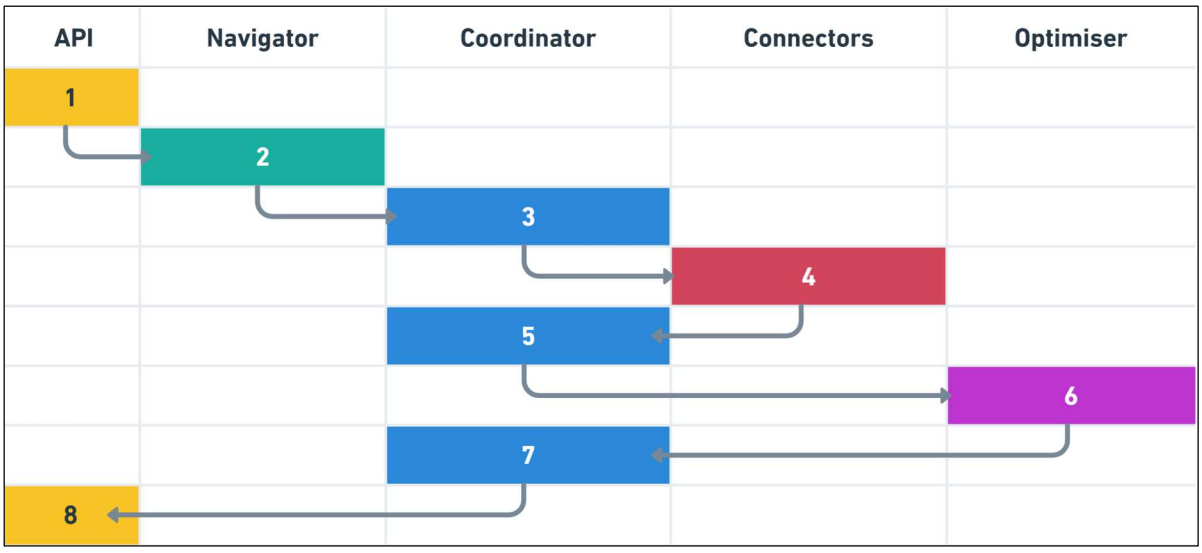

Figure 20: evrich operational workflow diagram.

7.5 Practical Application and Validation

7.5.1 Scenario 1: Evrich Integrated Service

In this scenario, the whole evrich service is showcased working as a unit with the coordination of each microservice performing with the SOGNO architecture.

In this case the user wants to travel from Maastricht to Aachen, so the user prompts the inputs shown in Table 13 to the API.

Table 13: Input parameters for the Maastricht to Aachen route

Vehicle ID	Vehicle Model	Starting SoC	Starting Time	Start Location	Sojourn Location Center	Sojourn Location Radius	Sojourn Period	Demand Target SoC	Demand V2G Allowance
1	Nissan Leaf	80%	14:00	Maastricht	Aachen	5km	1 hour	100%	0

The Navigator finds eight CSs, shown in Figure 21, within the three-kilometre range with compatible chargers for Nissan Leaf, meeting the user's needs.

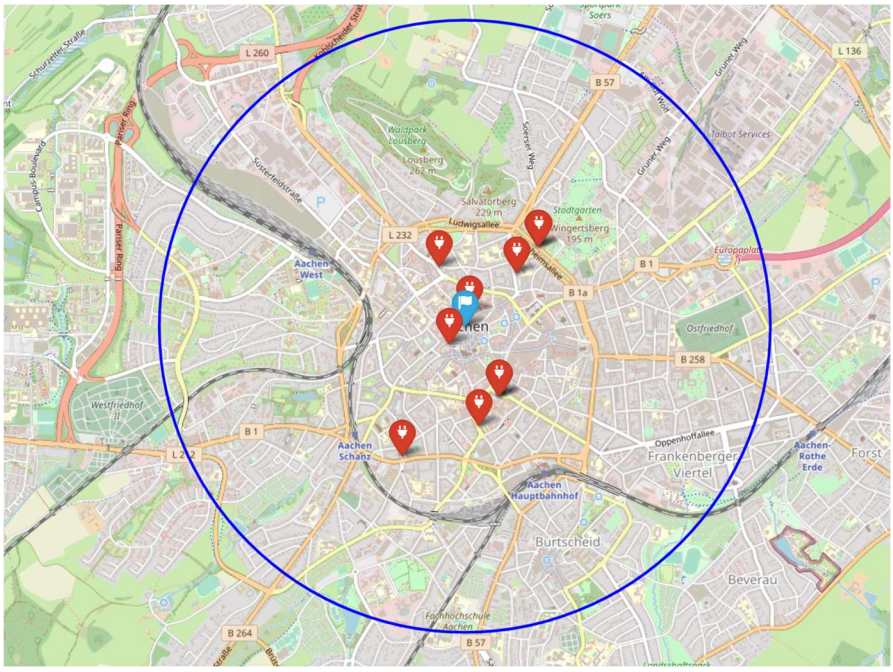

Figure 21: Mapping of the CSs within the search radius from Maastricht to Aachen route.

The Navigator this time, apart from extracting the usual parameters from the EV database for the SoC calculations, also extracts the EV's charge power (kW) and useable capacity (kWh) and publishes the CS list (see Figure 22) with both parameters via MQTT for the Coordinator, which receives the information.

Charging Station 1: Mostardstraße 5
Route Information:
Duration: 31.0 minutes
Walking time: 2.2 minutes
Altitude adjusted SOC: 62.39%

Charging Station 2: Klosterplatz
Route Information:
Duration: 36.4 minutes
Walking time: 2.3 minutes
Altitude adjusted SOC: 60.43%

Charging Station 3: Pontdriesch 4
Route Information:
Duration: 28.9 minutes
Walking time: 7.0 minutes
Altitude adjusted SOC: 62.43%

Charging Station 4: Innside Aachen
Route Information:
Duration: 28.9 minutes
Walking time: 7.0 minutes
Altitude adjusted SOC: 62.43%

Charging Station 5: Theaterplatz 6
Route Information:
Duration: 28.9 minutes
Walking time: 7.0 minutes
Altitude adjusted SOC: 62.43%

Charging Station 6: Franzstraße
Route Information:
Duration: 28.9 minutes
Walking time: 7.0 minutes
Altitude adjusted SOC: 62.43%

Charging Station 7: Quellenhof
Route Information:
Duration: 28.9 minutes
Walking time: 7.0 minutes
Altitude adjusted SOC: 62.43%

Charging Station 8: Monheimsallee 52
Route Information:
Duration: 28.9 minutes
Walking time: 7.0 minutes
Altitude adjusted SOC: 62.43%

Figure 22: CS list from Navigator posted for Coordinator.

The Coordinator serves the primary function of associating CSs with their respective Connectors. During the current testing phase, three connectors have been established, each corresponding to one of the initial three CSs detected by the Navigator in this example, linked to three different mock-up external aggregators. Each aggregator possesses a distinct URL and API, facilitating the conveyance of availability data to the connector microservices.

For illustrative purposes, a dictionary[4] has been created to function as a prototype for a future database that could be developed. This dictionary serves as an example of how CS names can be linked with aggregator IDs. See Listing 4 on the subsequent page for the example dictionary structure:

[4] Its structure is designed to inform and facilitate the development of a robust and detailed database system intended for future implementation.

```
connector_dict = {
    'Mostardstrasse 5': {
        'agg_id': 'aggregator1',
    },
    'Klosterplatz': {
        'agg_id': 'aggregator2',
    },
    'Pontdriesch 4': {
        'agg_id': 'aggregator3',
    }
}
```

Listing 4: Example of the connectors dictionary.

Through this dictionary, the Coordinator determines if the CSs identified by the Navigator correspond with any Connectors. When a match is detected, the Coordinator calculates the necessary parameters to dispatch to the Connectors for availability checks:

- The ETA at the CS, which is calculated by adding the anticipated travel duration to the journey's initiation time.
- The departure time, which is computed by adding the sojourn duration to this arrival time.
- Energy demand, which is calculated by finding the minimum between two values:

  1. The energy demanded by the user: calculated by taking the difference between the demand's target SoC and the arrival SoC, multiplied by the usable capacity of the EV, and then converted to joules.

  2. The demand constrained by the charging power limit which is the multiplication of the charging power with the sojourn period.[5]

Upon querying the availability of aggregators through the Connectors, the Coordinator receives structured responses from the aggregators. The structured response typically contains vital information regarding CS availability and other parameters.

[5] It is noteworthy that the time spent at a CS might sometimes be insufficient to achieve the desired SoC, due to the charge power limit.

A typical aggregator response is composed by:

- The identifier of the charger.
- Maximum charging and discharging power, respectively.
- Maximum amount of energy that can be supplied.
- Dynamic price signals for grid-to-vehicle (G2V) and vehicle-to-grid (V2G) respectively. Each entry in these dictionaries contains a timestamp and a corresponding price value.

From these responses, the Coordinator extracts key details to determine the optimal charging parameters. These details include understanding the capacity of each charger, how much power they can deliver or draw, the total energy they can provide, and the cost implications based on dynamic charging prices. With these insights, the Coordinator is equipped to establish a set of parameters to undergo the optimisation process.

Subsequently, the parameters are transmitted via MQTT to the Optimiser, which addresses the complex task of determining the most favourable charging schedule. It considers the dynamic pricing signals from each aggregator alongside their current availability. Upon reaching a decision, the Optimiser communicates the chosen charging plan back through MQTT, publishing it to the designated topic monitored by the Coordinator. The Coordinator, upon receiving this information, forwards it to the API, culminating in the delivery of the finalized charging details to the user. An example of the output that the user would receive is provided in Listing 5 below.

```
1 {"Charger": "cu_01_01", "Aggregator": "aggregator1"}
```

Listing 5: evrich service example response to user via API.

The scenario exemplifies the evrich service functioning within the SOGNO architecture, providing a clear practical application and validation. It demonstrates how the service determines the best CS for EV users by seamlessly integrating various microservices. The interaction begins with user input and culminates in a decision-making process, which not only suggests potential CSs but also actively selects the most suitable one based on dynamic pricing and availability, incorporating real-time data.

The scenario also effectively shows that the Navigator can be smoothly integrated into the broader evrich service, fulfilling the specific needs of this more complex service. Through careful testing, it has been proven that the Navigator's contributions are aligned with evrich's operations. The Navigator has demonstrated flexibility, adjusting features like fast charging preferences to complement evrich’s detailed charging strategies. This adaptability confirms that the Navigator is not just an independent microservice but a complementary component of a larger, interconnected service.

7.6 Integration Details

The integration of the Navigator microservice into the evrich service has required substantial updates to communication protocols. It has been designed to post and subscribe to an MQTT broker, aligning with the main communication platform used across evrich. Such updates necessitated a comprehensive re-evaluation of the data flow and inter-service communication strategies to ensure seamless integration.

Within the evrich ecosystem, certain functionalities of the Navigator, such as the fast-charging priority and charger compatibility checks, have been reassigned to Connectors, diminishing the Navigator's direct role in these areas. Since the evrich aggregators now provide the most current data on CS pricing and operators, these parameters are no longer the Navigator’s primary concern. Although walking time is still factored in by the Navigator, its utilisation in the current optimisation calculations within evrich remains provisional, with potential integration anticipated in future enhancements.

Another implementation for the integration is the linking system, mapping CS names to aggregator IDs through a dictionary. This is a short-term arrangement until a more robust database is developed within the evrich service, which will dynamically manage the connections between CS names and aggregator IDs, facilitating straightforward access to the aggregators' APIs.

Moreover, the API, already built in the evrich service has been expanded to support multiple request types, allowing for both the Navigator isolated feature testing and the existing holistic assessment within the evrich environment. New classes have been created to cater to the specific input requirements of the Navigator, ensuring compatibility with the evrich service’s data format specifications.

Finally, significant revisions have been made to the Coordinator's main code within evrich to aptly process the Navigator’s responses. This included implementing a new function to round estimated arrival and departure times to the nearest five-minute mark, a critical feature to match the time resolution used by the optimizers for dynamic pricing. This attention to temporal precision is key for adapting the Navigator's output with the varying pricing intervals managed by the different aggregators within evrich.

In conclusion, the scope of this thesis has encompassed a thorough understanding of the entire evrich system. Prior to the creation of the Navigator, it was imperative to fully comprehend the existing system to ensure a seamless integration. The analysis and adjustments to the system architecture and communication methods within evrich were integral to this process. The adaptation made to the Coordinator and API are indicative of the thesis's commitment to not only effectively integrate a new ITS component but also to enhance the overall system structure. Consequently, the effort invested has not only facilitated the successful inclusion of the Navigator but has also refined the system's framework, providing a solid foundation for ongoing innovation in evrich.

8. Navigator: Computational Analysis and Prospects

8.1 Introduction

This chapter is dedicated to the Navigator standalone deployment to facilitate a clearer visualisation and demonstration of the system's capabilities, as simulating the entire system would entail validating components not developed within this thesis. The ACS institute will be responsible for the showcasing of the additional microservices and the whole evrich service. This approach maintains the focus on validating the components developed in this work, offering insights into their functionality and reliability within the broader system.

This chapter conducts a performance evaluation first at the individual container level, examining the performance of the Navigator logic, Database, and API containers independently; followed by an assessment of the Navigator as a standalone deployment to understand its overall functionality and efficiency.

8.2 Performance Metrics

Before delving into the evaluation and presentation of system performance, it is essential to establish a clear understanding of the metrics used and the methodology adopted for measuring these metrics. By presenting these metrics upfront, we ensure that the reader is equipped with the necessary context to interpret the subsequent results.

8.2.1 CPU Usage (%)

The Central Processing Unit (CPU) usage percentage is a critical metric that reveals the extent to which the system's computational resources are being utilised. Specifically, it denotes the proportion of time the CPU spends executing tasks as opposed to remaining idle over a designated period. High CPU usage could indicate a system under stress, which may require optimisation or resource augmentation.

8.2.2 Memory Usage (megabytes)

Memory, commonly referred to as Random Access Memory (RAM), serves as the temporary storage that a computer uses to hold data that is actively being processed. Memory Usage quantifies the volume of RAM currently being used by the system or container. Monitoring this metric is vital, as excessive memory usage can lead to system slowdowns or crashes if the system runs out of usable memory space.

8.2.3 API Response Time (seconds)

API response time measures the duration taken for an API to revert with a response following a request initiation. This metric is instrumental in measuring the efficiency and performance of an API. Longer response times might point towards bottlenecks or inefficiencies that require addressing.

8.2.4 API Response Size (kilobytes)

When an API responds to a request, it returns data. The volume of this data, quantified in kilobytes, is referred to as the API response size. This metric can typically be sourced directly from response headers or network logs. Larger response sizes might lead to increased download times and could affect user experience, especially on slower network connections.

8.2.5 API Download Time (seconds)

The download time specifically measures the duration from the moment the API server completes generating or fetching the response from the Navigator until the data is fully received by the client. It is a crucial factor in assessing the efficiency of the data transfer process, elongated download times might signal network issues or server inefficiencies.

8.2.6 Number of CS Queried

This metric represents the cumulative count of CSs that the ITS processes for calculation during a designated request. Tracking this number helps in understanding the scope and scale of data the system is managing in relation to CI.

8.3 Container Performance

To ascertain the container's performance under stress, a series of 5 tests were conducted using identical user input information. These tests simulated real-world scenarios where a user navigated from Godella to Valencia with specific parameters: seeking CSs within a 3000 metre radius for a Nissan Leaf EV, starting at a 100% initial SoC, with no priority for fast charging. Each test retrieved a maximum of 10 CSs, thus ensuring that the container's performance was evaluated at its highest data retrieval capacity. Testing of all containers was conducted on a personal computing system equipped with an Intel Core i7 processor operating at 2.8 Gigahertz (GHz).

8.3.1 Navigator Container

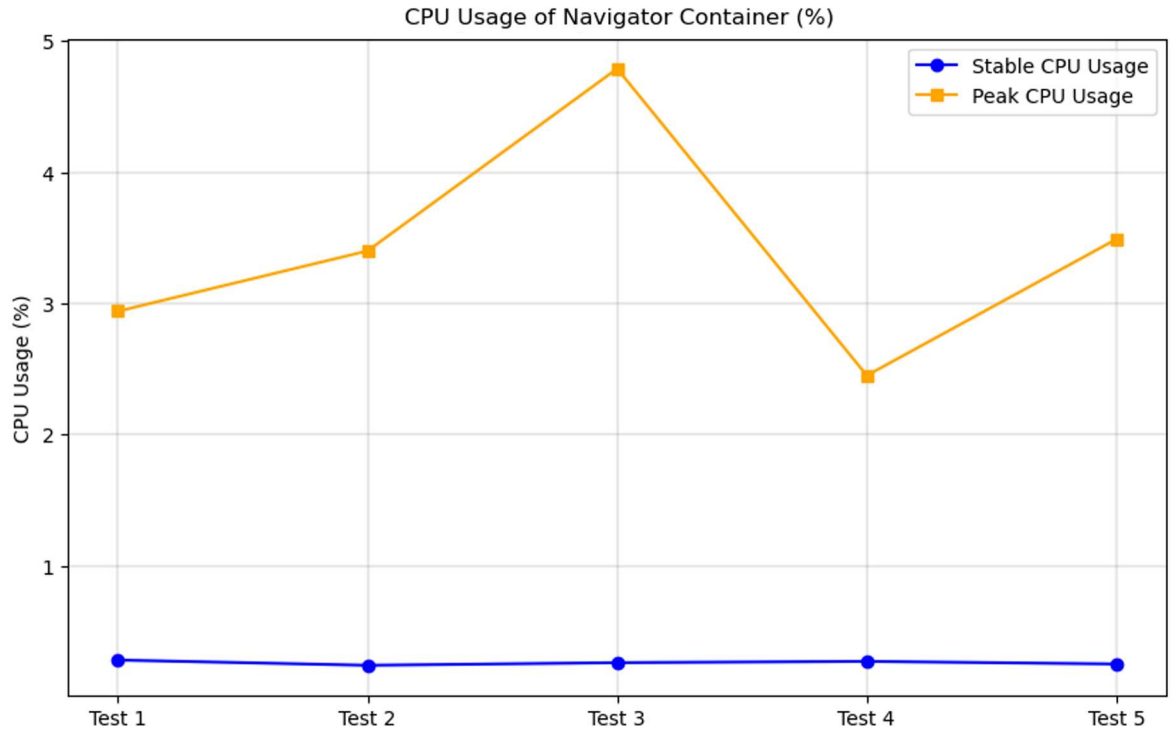

Figure 23: Graph of CPU usage (%) of Navigator container.

The CPU usage of the Navigator container shown in Figure 23 was closely monitored using the Docker Desktop application. Unlike static measurements that capture a singular point in time, the metrics observed were dynamic, reflecting real-time CPU consumption of the Navigator container. Over a set of observations, a series of 10 CPU usage data points were recorded per request to derive both idle and peak CPU utilizations. In the graphs presented, the label 'stable peak usage' refers to periods when the CPU usage is consistently idle, indicating times of low or no processing activity.

Under idle conditions, the CPU usage displayed a consistent pattern, fluctuating between 0.24% and 0.28%. This consistent low consumption suggests that the container is highly efficient when not actively processing requests. The stability in CPU consumption during idle states provides an assurance of its optimized resource management.

During these tests, the CPU usage demonstrated a significant increase, reaching up to 4.79%, with an average around 3.41%. Although this marked a considerable elevation from the idle state, the increase remained within a comfortable range. This suggests that even when processing requests at its upper limit, the Navigator container can manage the demand without excessively compromising the CPU.

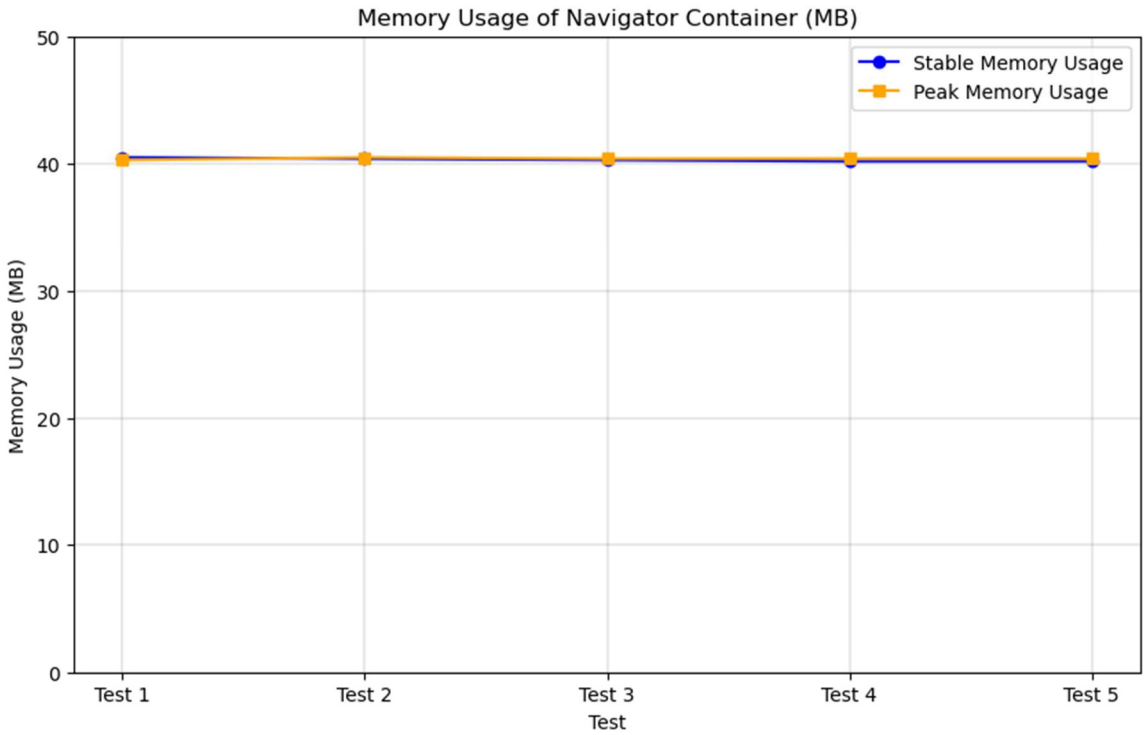

Figure 24: Graph of memory usage (MB) of Navigator container.

Figure 24 shows memory usage remains relatively stable, hovering around 40.3 MB during idle conditions and slightly increasing to around 40.5 MB during peak load. The minimal fluctuation in memory usage is a good indicator of efficient memory management within the container. There are no signs of memory leaks or excessive memory allocation, even during periods of high activity. This stability in memory usage suggests that the container is well-configured and that the application running within it efficiently utilises memory resources.

In summary, the CPU and Memory usage metrics for the Navigator container indicate a well-optimized and efficiently running service. The container demonstrates good resource management under varying conditions, making it a reliable component in your microservices architecture.

8.3.2 API Container

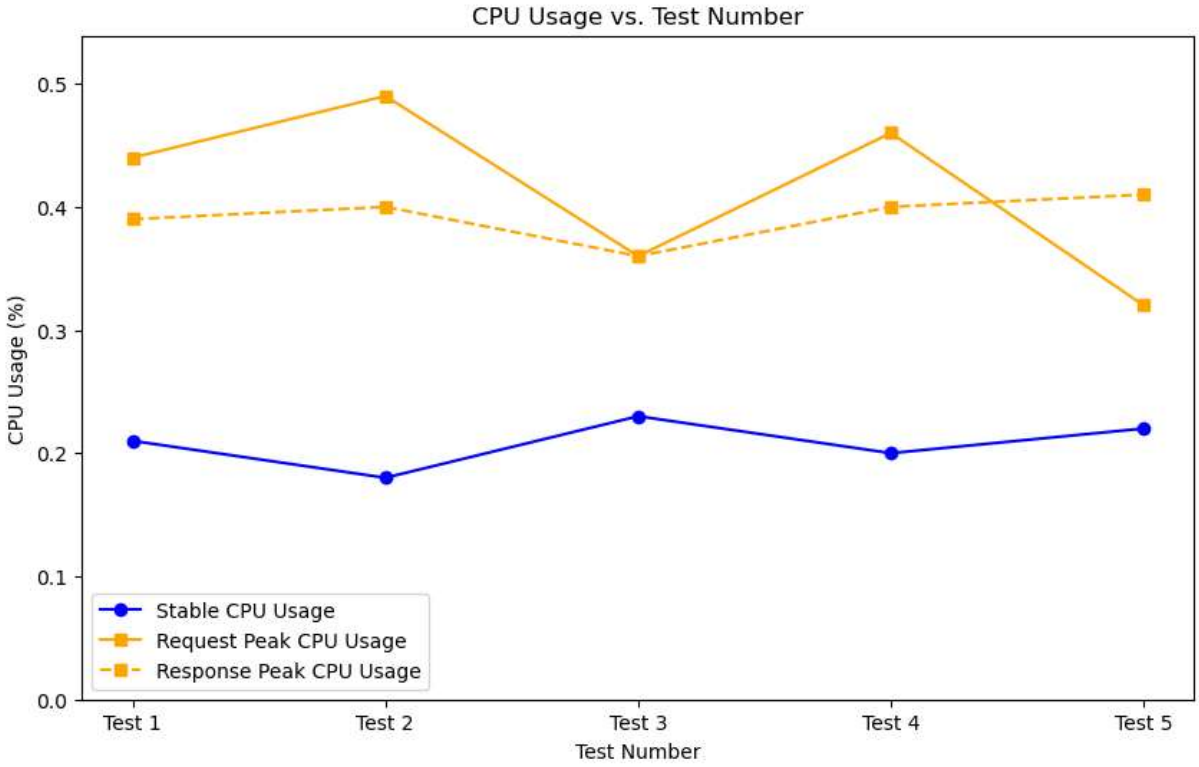

Figure 25: Graph of CPU usage (%) of API container.

The CPU usage graph for the API container (Figure 25) exhibits a low and consistent level of resource utilisation during idle conditions, with usage percentages finely hovering between 0.18% and 0.23%. This low variation in idle CPU usage implies that the container maintains a stable and efficient baseline when not processing active requests. Such stability is indicative of a well-configured system that does not exert unnecessary computational effort while in a waiting state.

When observing peak CPU usage, the graph delineates two distinct types of peaks: those that occur during request handling and those during response generation. The 'Request Peak CPU Usage' shows a slightly higher demand on CPU resources, with peaks reaching between 0.32% and 0.49%. Meanwhile, the 'Response Peak CPU Usage' presents a similar but slightly more consistent pattern, with peaks ranging from 0.36% to 0.41%. Notably, neither the request nor response peak CPU usages show drastic spikes, instead portraying a controlled and moderate rise in CPU demands.

The profile of CPU usage during idle and peak conditions is well within an acceptable range, suggesting that the API container is effectively managing its computational load. This balanced CPU usage is crucial for maintaining system responsiveness and reliability, particularly important in containerized environments where resource efficiency is key.

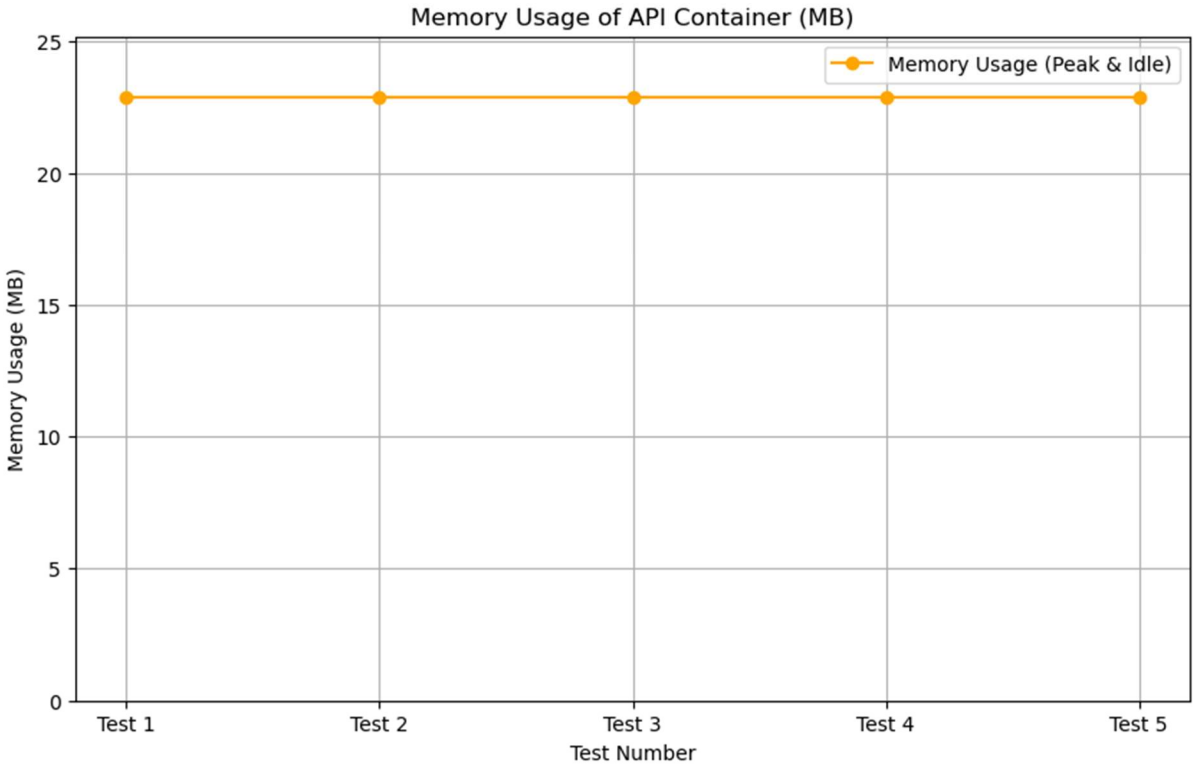

Figure 26: Graph of memory usage (MB) of API container.

In Figure 26 the memory usage graph for the API container within the Docker environment displays a flat line, indicating a perfectly consistent utilisation across all tests. The memory usage remains constant at 22.9 MB during both idle and peak conditions. This uniformity suggests that the memory allocation for the container is sufficient and stable, unaffected by the load variations that typically accompany different states of operation.

Such consistency in memory usage can be advantageous, pointing to a predictable and reliable memory requirement. For container management and orchestration, this predictability allows for more effective memory resource allocation and scaling strategies. Given that memory usage is one of the key factors in container performance, especially in environments with limited resources, the graph signifies that the API container is well-optimised in terms of memory consumption.

Moreover, the static nature of the graph implies that there is no memory leakage or unexpected spikes in allocation that could lead to performance degradation over time. This aspect is particularly vital in long-running containers, ensuring they maintain their performance without the need for frequent restarts or intervention.

8.3.3 Database Container

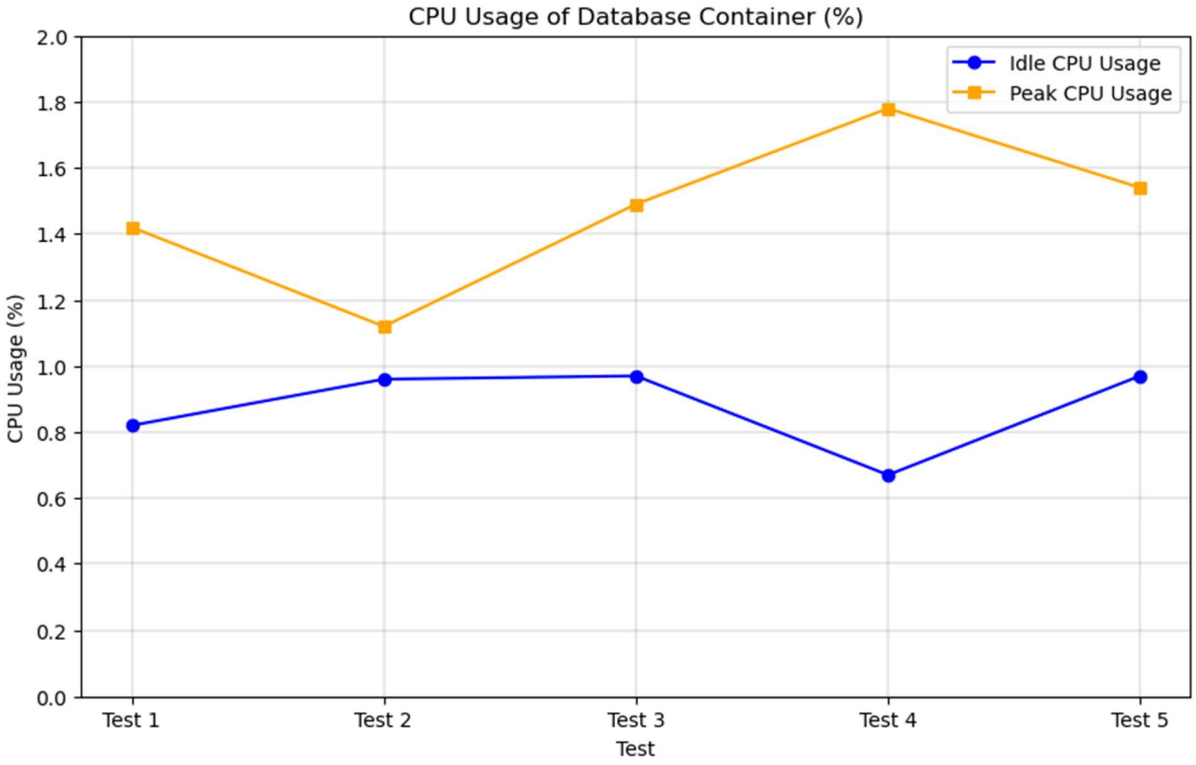

Figure 27: Graph of CPU usage (%) of Database container.

Figure 27 shows that the CPU usage for the Database container is relatively low during idle conditions, ranging between 0.67% and 0.97%. This suggests that the database is not consuming excessive computational resources when it is not under active use, which is a good indication of an optimised system. During peak load conditions, the CPU usage experiences a moderate increase, peaking at 1.78% with an average of around 1.47%. This increase is expected and indicates that the database can handle a higher load without significantly straining the CPU. The moderate range of CPU usage at idle and peak states suggests that the database is efficiently handling its computational tasks.

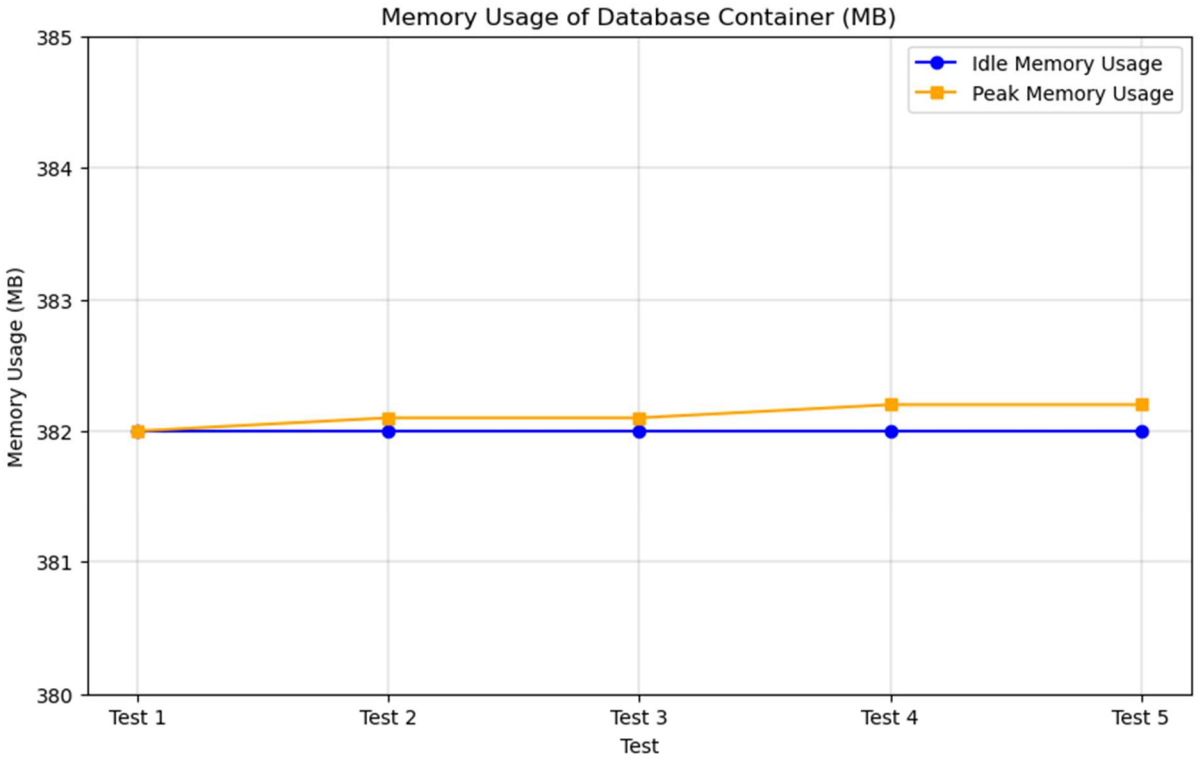

Figure 28: Graph of memory usage (MB) of Database container.

In Figure 28, the memory usage for the Database container remains remarkably stable, staying around 382 MB in both idle and peak conditions. Consistency in memory usage is a strong indicator of effective memory management. There are no signs of memory leaks or sudden spikes in memory allocation, even during peak load conditions. This suggests that the database is well-configured and efficiently uses its allocated memory.

8.4 System Performance

This section presents a comprehensive analysis of the Navigator's system performance, examining various parameters such as response times and download times to analyse the deployment's data processing efficiency. The focus is on how different scenarios and configurations impact the overall system responsiveness. A series of targeted experiments and detailed data evaluations are conducted to gain insights into the ITS's behaviour under various operational conditions.

8.4.1 Influence of Fast-Charging Priority and Usual API Response Metrics

In the initial exploration, 10 random requests were conducted, targeting different routes and EV models to assess the influence of the fast-charging priority setting on API response times and sizes. Each route was subjected to two tests: one with the fast-charging set to 'true' and another set to 'false'. This preliminary investigation aimed to lay the groundwork by providing insights into the typical response behaviours when varying the fast-charging priorities. See Figure 29 below.

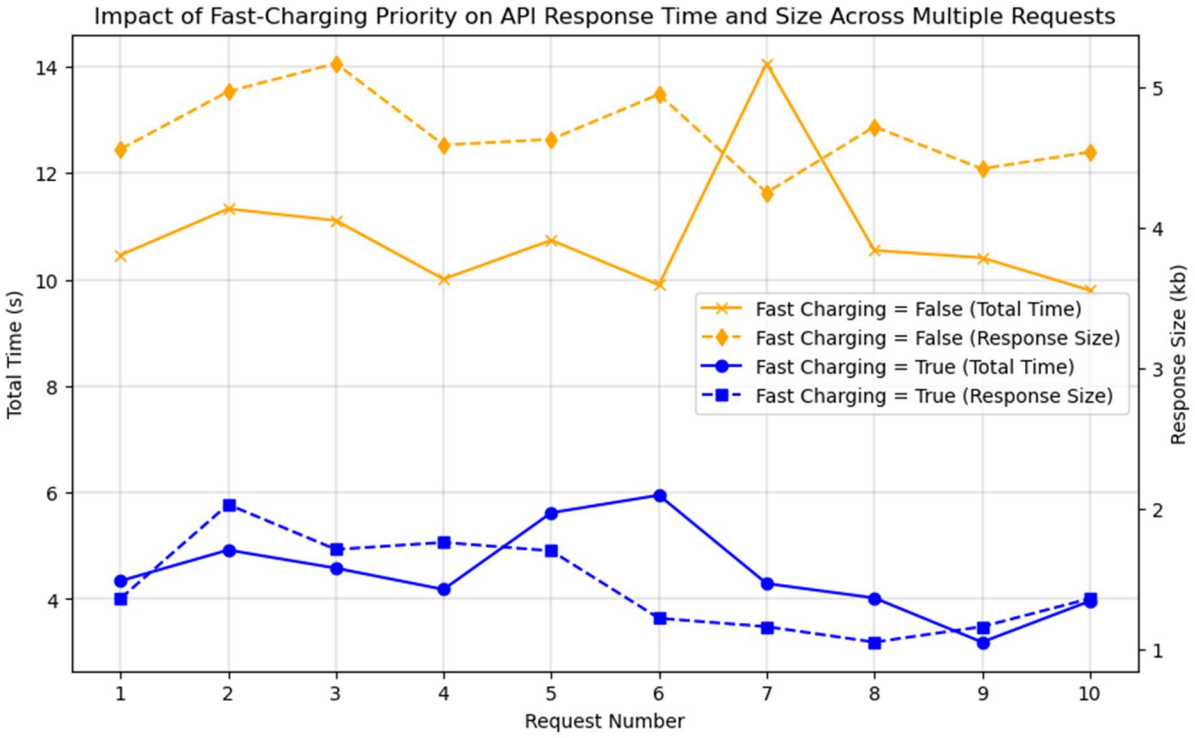

Figure 29: Graph of usual API response times of the Navigator standalone system.

The graph "Impact of Fast-Charging Priority on API Response Time and Size Across Multiple Requests," offers an insightful depiction of the average response size and time for varying settings. When the priority is set to 'true', the average response time is approximately 5 seconds with a size of 1.5 kb. In contrast, setting it to 'false' results in an average time of 11 seconds and a size of 4 kb.

In addition, it also shows how the fast_charging_priority parameter affects the response size and total time taken for API responses. The graph plots the time taken in seconds and the response size in JSON format in (kb) for each of the 10 random requests, comparing scenarios where fast charging is prioritised against those without. It is easy to observe that when fast charging is prioritised, the total time and the response size for API responses are consistently lower across all requests. This trend suggests that prioritising fast charging leads to quicker API responses due to smaller-sized responses.

The underlying reason for this observed behaviour can be attributed to the nature of the calculations performed by the main code. When fast_charging_priority is set to True, the algorithm matches the EV's fast charger with CSs offering fast-charging capabilities. Fast chargers are generally less common than standard Type 2 or Type 1 AC chargers. As a result, the pool of potential CSs to consider is significantly reduced, leading to fewer calculations and, consequently, faster response times.

On the other hand, when fast_charging_priority is set to false, the total time experiences a noticeable increase, with values ranging from 9.8s to 14.06 and response sizes ranging from 4.25 to 5.17 kb. This indicates the service taking more time to loop through a larger set of CSs, offering a more comprehensive but time-consuming analysis. While this setting provides a broader range of options, it comes at the cost of increased computational time, making it more suitable for users looking for exhaustive options rather than quick results.

However, some anomalies in the data deviate from this trend. For instance, a total time of 5.95s corresponds to a relatively smaller response size of 1.22 kb, and a total time of 14.06s corresponds to a response size of 4.25 kb, which is, in fact, the smallest size of all fast-charging set to false queries received.

These anomalies could be attributed to network latency, server load, or other unpredictable variables affecting performance metrics. They serve as a reminder that while there is a general trend linking total time and response size, the relationship is not strictly linear and can be influenced by other factors. These outliers offer valuable insights for further investigation, as understanding their underlying causes could lead to more robust and efficient system performance.

Considering these anomalies, a more detailed examination focused on download times plotted against total times in Figure 30. Interestingly, the download times remained relatively consistent across all queries, irrespective of the total time taken for each request. This consistency suggests a stable and reliable network connection between the client and the server, indicating that the fluctuations in total time are likely not due to network issues but are more attributable to server-side processing or other backend operations.

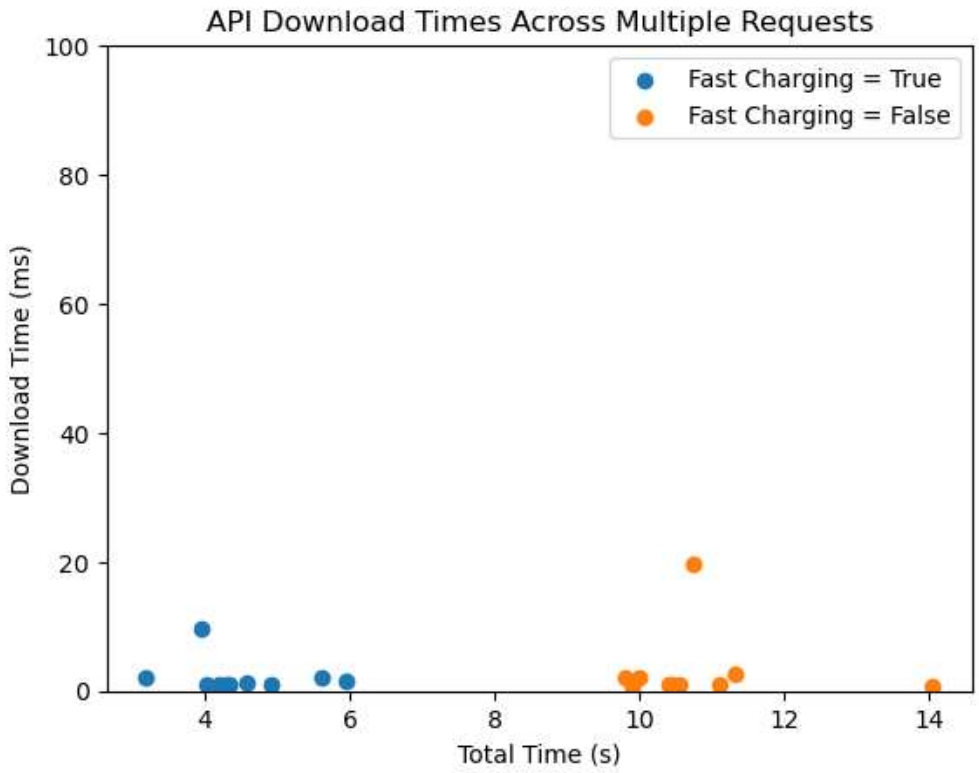

Figure 30: Graph of API download times across multiple requests.

The uniformity in download times in milliseconds (ms) shifts the focus of performance optimisation towards server-side logic, database queries, and other backend operations. This is a significant insight for resource allocation, as it suggests that investments in optimising core operations could yield more substantial improvements in performance than focusing on network infrastructure. This is beneficial from a user experience standpoint; it implies that users are less likely to experience delays due to data transfer, which is often more variable and harder to control than server-side processing time.

Furthermore, the consistent download time is a baseline for future anomaly detection. Any significant deviations from this baseline could serve as a red flag, warranting immediate investigation. This level of detail not only helps in the current analysis but also sets the stage for more targeted and effective performance improvements in the future.

8.4.2 CS Processing Performance: Big City vs. Small Town

Following an analysis of 10 random requests, which encompassed varied routes and EVs, the observed trends were the differential response times and sizes based on whether the fast-charging priority was set to true or false.

To further validate the hypotheses – that prioritizing fast charging results in reduced response times and sizes, and that the greater the number of CSs processed, the longer the response times – a more focused study was undertaken. This involved processing two specific requests with consistent parameters. Each of these requests was repeated 10 times for both fast charging set to true and set to false: one targeting a major city and the other aimed at a smaller town. This approach aimed to ascertain the consistency of the trends observed in the initial graph.

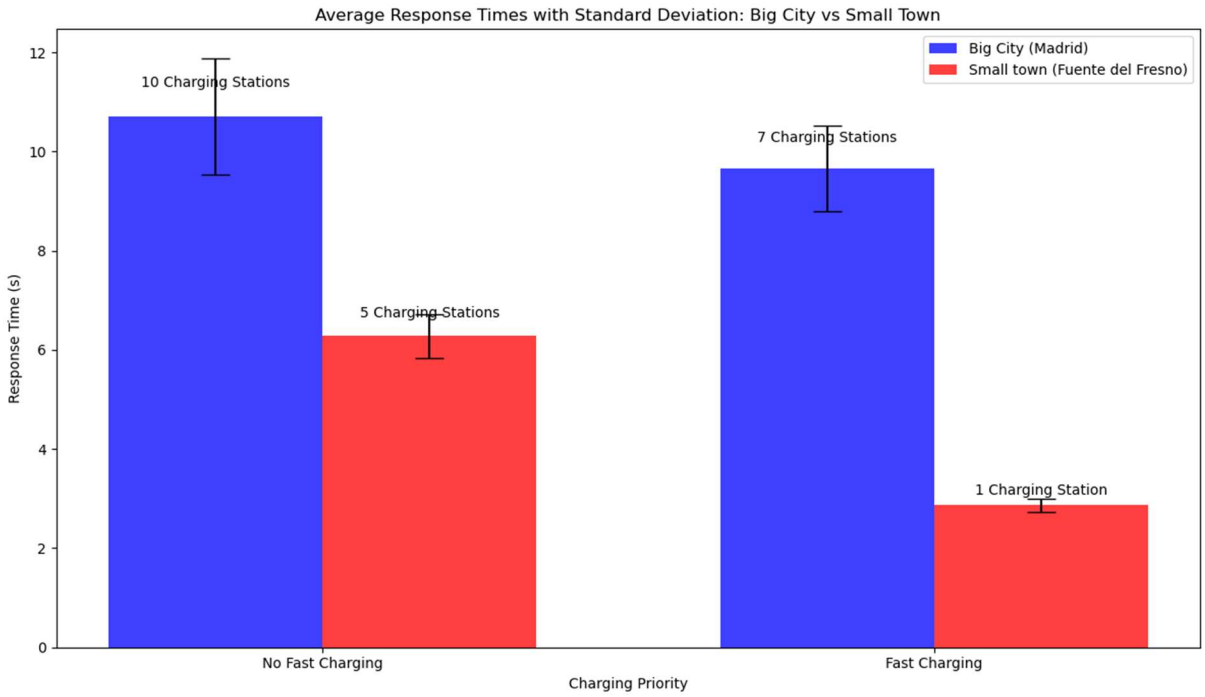

Figure 31: Graph of API average response times for an example of a big city and a small town.

The subsequent graph in Figure 31, offers a comparative analysis of the API mean response times and deviation for two requests with the same origin and two distinct destinations to find the CSs: a metropolis (Madrid) and a smaller town (Fuente del Fresno). The number of CS shown is the number of CS found for that request.

The response times showcased on the graph range from a minimum of 3 seconds to a maximum of 11 seconds. This relatively narrow span represents a consistent and reliable margin for the Navigator. Despite the variability introduced by the number of CSs and location specifics, the system consistently returns results within this range.

It is evident that the response times for queries in Madrid are generally higher than those in Fuente del Fresno. This is a result of the greater availability of CSs (ten for standard charging and seven for fast charging connectors) in a larger city, leading to more extensive data processing and consequently longer response times. However, when fast charging is prioritised, the response times in both locations tend to decrease, reinforcing the earlier observation that prioritizing fast charging streamlines the search process.

Interestingly, while Madrid has a broader range of response times, Fuente del Fresno showcases more consistency, especially when fast charging is prioritised. This is attributed to the limited number of CSs in the smaller town (five for standard charging and one for fast charging connectors), leading to fewer variations in the data processing time.

Another noteworthy observation is the difference in response times between the two locations when fast charging is not prioritised. Madrid's response times are more spread out, with some instances taking significantly longer. In contrast, Fuente del Fresno's response times remain relatively clustered, indicating a more predictable and consistent performance in smaller towns.

A crucial takeaway from the graph is the relationship between the response size, time, and the number of CSs queried. The response time appears to follow a linear approach relative to the number of CSs queried. This proves the hypothesis from Section 8.4.1: that as the number of stations increases, so does the computational effort required by the navigator to calculate the final SoC and so does the response times. This linear relationship underscores the direct impact of the number of stations on the system's performance, emphasising the need for efficient algorithms and optimisations, especially in areas with a high density of CSs.

8.4.3 Response Times with Increasing number of CS Processed

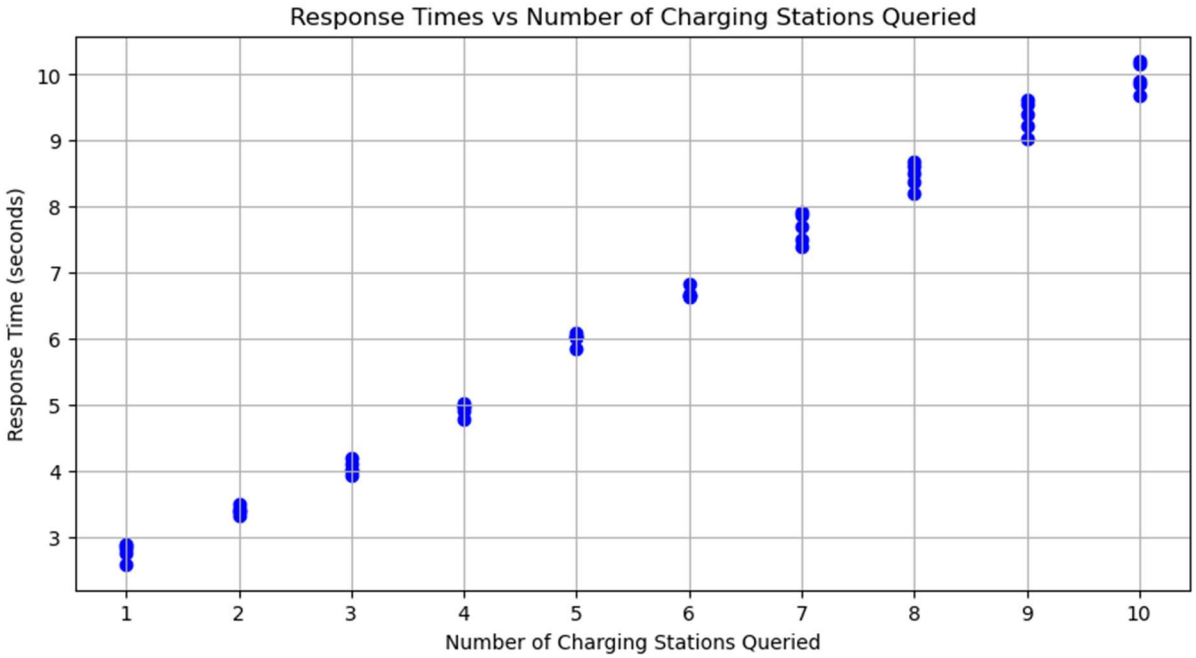

Figure 32: Graph of API response times with increasing number of CS processed by the Navigator.

After the hypothesis closed to be confirmed in Section 8.4.2, the graph titled "Response Times vs Number of CSs Queried"in Figure 32 showcases ten different response times for a constant route query just increasing the maximum radius to find CSs. It clearly confirms the expected relationship between the number of CSs computed and the response times. A clear linear trend is observed, suggesting a direct proportionality between the number of CSs queried and the response time. As the number of stations increases, the Navigator requires more time to process the query, likely due to the increased computational effort involved.

As the number of CSs computed increases, especially at the higher end, there is a noticeable trend of increasing standard deviation. This suggests a slight rise in variability in response times, which might originate from the added complexity of processing a more extensive set of CSs. It is essential to note, however, that this increasing standard deviation, is within a maximum range of just one second in API response times. Given the context of real-world applications and user expectations, a variability of one second is generally considered negligible and should not pose any significant usability or functionality concerns.

In the lower range of CSs queried (from 1 to about 6), the response times and their deviations remain notably consistent, further supporting the observed linear relationship. This consistent behaviour provides a foundational understanding of the system's performance scalability relative to the number of CSs queried. Nevertheless, while the variability in higher ranges remains minor, studying this trend further could help in refining the system and ensuring its efficiency in all scenarios.

8.4.4 API Traffic Management Testing for Multiple Requests

It is critical to understand the behaviour and performance characteristics of APIs, especially when subjected to high loads. The aim of this study is to delve deep into the performance aspects of the API container built with FastAPI.

A testing request is created, choosing a big city (Madrid) for the destination to look for CSs. This request leads to the Navigator responding with 10 CS, which ensures that the response time depends on the processing of the Navigator and not network issues.

The primary metric of interest is the response time, which gives a clear indication of the system's latency. The first phase of testing involves sending single requests to set a benchmark response time, after ten response times are taken into consideration for the mean value.

Once this benchmark is established, the focus shifts to how the system performs under concurrent requests, a vital aspect since APIs typically serve multiple users simultaneously. The concurrency testing is structured in stages: starting with two simultaneous requests, progressing to three, and then to four. Each level is tested ten times, maintaining a two-second gap between each request. This approach aims to detect any potential queuing or blocking and analyses how an initial request might influence the processing of subsequent ones.

In the observed tests, the API consistently and accurately responds to each request in the sequence they were received, ensuring that responses are directed to the appropriate originating user, even in concurrent request scenarios. The collected data is then visualised in a graph that displays the response times for each test level (single, two requests, three requests, and four requests), see Figure 33 below.

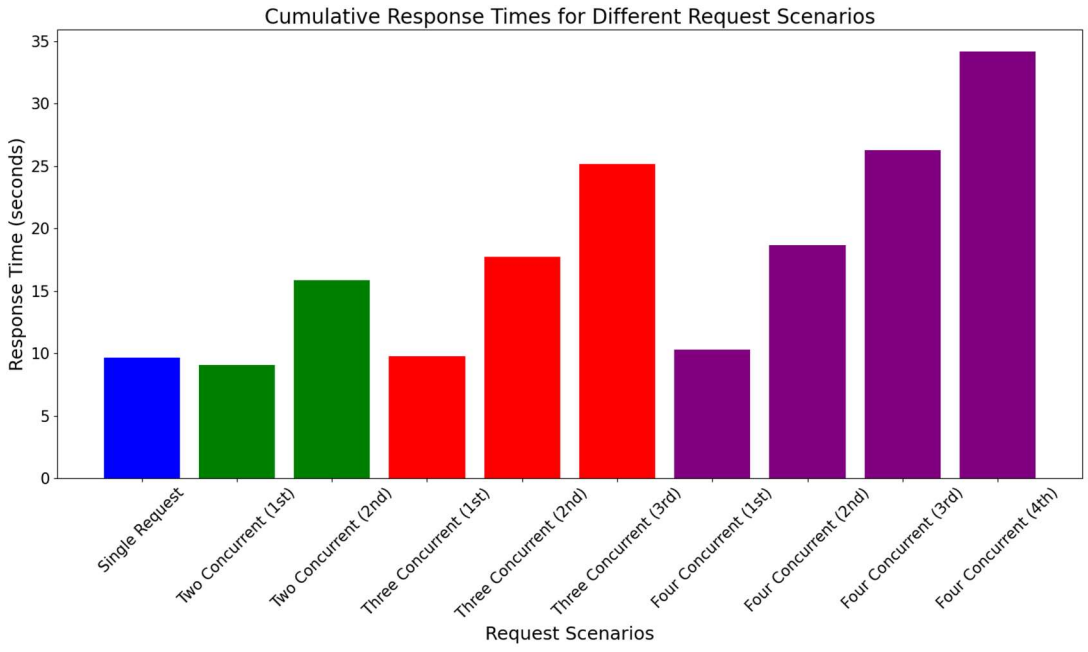

Figure 33: Graph of Cumulative Response Times for Different Request Scenarios.

The graph provides a clear view of the growing response times with increasing concurrency. This increase is due to the API's behaviour: as it receives new requests, they are placed in a queue. Subsequent requests must wait for their turn, which means the second request starts processing only after the first one finishes. This pattern reflects the synchronous nature of the API. Although it is designed to handle multiple requests, managing, and processing them concurrently introduces some overhead.

A second graph is then constructed, considering the waiting times for the 2nd, 3rd, and 4th requests, see Figure 34 below.

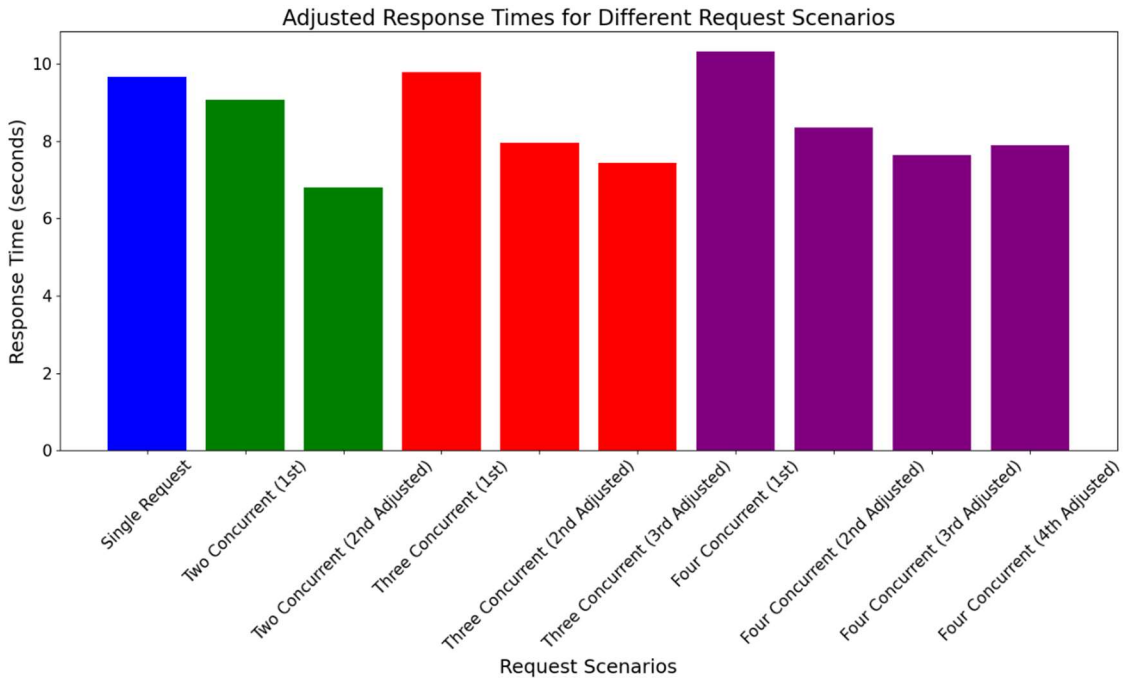

Figure 34: Graph of Adjusted Response Times for Different Request Scenarios.

This graph effectively captures the "net" processing time for each request, excluding the waiting duration caused by previous requests. An interesting observation from this graph is that often the first request, which does not wait for any predecessor, takes the longest. This might initially seem counterintuitive, but there are logical explanations behind this behaviour.

When the initial request is made, the API container needs to communicate with the Navigator container, which in turn must establish a connection with the database container to fetch EV data. This multi-step communication process might not be fully optimised during the very first request due to various factors.

Firstly, the phenomenon of caching might be more pronounced in this scenario. After the Navigator container processes the initial request and communicates with the database container, certain operations or data can be cached within the Navigator or even in the API layer. This ensures that subsequent requests can access this cached data or operations more swiftly, bypassing the need to re-fetch or recompute them in the database container.

Furthermore, the communication overhead between these containers can be another influential factor. For the first request, connections between these containers might be established or certain routing paths might be determined, adding to the time taken. However, for subsequent requests, these connections or paths could be reused, streamlining the communication process, and thus reducing the response time.

Moreover, the "warm-up" phase, a concept prevalent in many modern frameworks and databases, might also play a role here. The first request could be responsible for initialising certain components or resources across these containers. Once these components are "warmed up", subsequent requests can be processed with reduced overhead, even if they must wait in a queue.

In conclusion, this thorough testing regimen provides a clear picture of the API's performance characteristics under varying loads. It emphasises the importance of understanding not just how an API performs under isolated conditions, but also how it scales and manages concurrent demand. The insights from this study can guide future optimizations and architectural decisions to ensure robust and efficient service delivery, while also enhancing the asynchronicity of the system.

8.5 Interpretation and Scalability

The Navigator's system performance metrics provide insights into its scalability. It is evident that the system delivers solid response times across different settings, from high CS density cities to smaller towns with less availability. Such versatility indicates the microservice's preparedness to accommodate more demands as they emerge. Furthermore, with response times consistently falling between 3 to 11 seconds, it underscores the ITS's reliability. This range suggests that despite varying factors, like the number of CSs, fast charging prioritised or specific locations, the Navigator maintains a dependable performance, positioning it well for future expansions.

Despite observing a variability in response times of one to two seconds when handling 10 CSs, such minimum fluctuations are relatively moderate within a broader perspective. Such marginal variability further accentuates the system's reliability, making it a predictable solution for end-users.

Performance under concurrent requests further elucidates the Navigator's scalability. Testing revealed a notable trend in response times. While an individual request offers a benchmark for system performance, the introduction of concurrent requests sees a discernible escalation in response times.

A more thorough analysis reveals that the initial request often takes the longest to process, but subsequent requests, even accounting for their inherent waiting times, often clock in with shorter net processing times. Several factors can be attributed to this observation, including potential caching post initial data retrieval, streamlined communication pathways across the API, Navigator, and database containers, and the initialization processes that occur during the initial request's warm-up phase. It is important to note that while the system is skilled at managing multiple concurrent requests, there is a sequential delay within each request – a detail that is especially relevant in scenarios where many requests may be sent to the API that works with just one Navigator container.

However, while the Navigator showcases impressive scalability attributes, there are specific trends to be addressed. A linear relationship is observed between the increase in response times and the number of CSs queried. This predictable pattern is beneficial for scalability, allowing for strategic planning. Understanding this relationship can determine the number of Navigator containers needed to achieve desired response times for multiple queries. Future modifications in the evrich service could involve the inclusion of multiple Navigator containers. Such a multi-container approach diffuses computational demands, ensuring that even with an influx of queries, response times remain optimal.

8.6 Additional Considerations

While it would have been highly beneficial to measure the accuracy of the SoC calculations, the constraints of this evaluation prevent such an in-depth analysis. Ideally, the focus would extend to comparing final SoCs against real-world measurements. Such a comparison would require access to multiple EVs to test various routes under different conditions, which is beyond the scope of this study.

However, envisioning the potential experiment, it could be implemented using a select group of EVs from the initial database. This approach would involve conducting a series of trials with these EVs under varying conditions to gather empirical data. The variables for these trials would include different temperatures, road gradients, distances, and initial SoCs.

For each trial, an EV would follow a route as directed by the microservice, with the actual SoC values at each CS being recorded. These real-time measurements would then be compared against the SoC estimates provided by the microservice. Such comparative analysis would offer invaluable insights into the algorithm's effectiveness and how closely the theoretical discharge rates, derived from manufacturers' data sheets, align with actual performance.

During this experiment, additional parameters could be evaluated to determine their relevance for the Navigator's computations. These include the impact of extreme high temperatures on battery performance and the effects of downhill gradients on routes, particularly in relation to regenerative braking. Regenerative braking, which recovers energy during braking and channels it back to the battery, could significantly enhance the accuracy of the Navigator's estimations.

8.7 Possible Improvements and Limitations

8.7.1 Integrated Directions Feature

While the Navigator provides valuable information to users about the best CSs, including distance and estimated travel time, it currently does not display the actual route to reach the selected CS. This limitation requires users to manually choose a CS and then input the destination into a separate routing application to obtain directions. In future enhancements, there is a compelling opportunity to improve user convenience and efficiency by incorporating route directions directly into a built-in interface. This could involve integrating the Navigator with a routing app, allowing users to seamlessly transition from selecting a CS to receiving turn-by-turn directions within the same interface. This enhancement would not only streamline the user experience but also promote greater user engagement with the Navigator, further enhancing its utility for EV owners.

8.7.2 Incorporating Real-Time Traffic Data

The current system's effectiveness could be significantly improved by incorporating real-time traffic data into the SoC and route calculations. Although the Bing maps service does provide an estimation for general traffic conditions for the total route, see Figure 10 in Section 6.3.2, it does not account for the dynamic nature of road conditions, which can change rapidly due to traffic jams, accidents, or road closures. These factors can substantially impact the vehicle's energy consumption and, consequently, the SoC at the destination.

One of the most significant advantages theoretically possible with an ITS for EVs would be its ability to optimise routes using real-time traffic information dynamically. By analysing traffic congestion and road conditions data, such systems could intelligently adjust routes to bypass heavily congested areas, reducing travel time and improving overall efficiency. This benefits individual EV drivers by providing smoother and faster journeys, alleviates traffic congestion, enhances air quality in urban areas, reduces electricity consumption and lowers charging costs for EV drivers. Potential features could include users becoming part of the data ecosystem, as applications such as Waze already do, leading to a highly accurate calculation of ETA [68].

Fortunately, several real-time traffic data providers offer API subscriptions with associated fees. Integrating such services into the system would improve the accuracy of the SoC calculations and provide more reliable route options. This would be particularly beneficial during peak hours or in congested urban areas, where traffic conditions significantly affect travel time and energy consumption.

The primary challenge in incorporating real-time traffic data would be scaling up the system to handle the increased computational complexity and ensuring that the API subscriptions are cost-effective. However, the benefits in terms of improved accuracy and user experience would likely outweigh the costs, making it a valuable avenue for future development.

8.7.3 Accuracy in CS Data

Relying solely on a single API for CS data introduces a point of failure and potential inaccuracies in the system. OCM is a reputable source, but its update frequency and data accuracy are not guaranteed. A backup API could be integrated to serve as a fallback when OCM is unavailable or returns incomplete data to mitigate this. This would enhance the system's reliability and robustness. Additionally, a data validation layer could be introduced to cross-reference information from multiple sources, ensuring that the most up-to-date and accurate CS data is used. This is particularly important for EV users who may rely on real-time data for charging options, especially when the nearest station is crucial.

8.7.4 Accurate Road Gradient Impact on SoC

The current system calculates the total altitude change to estimate its impact on the SoC. While this is a reasonable approximation, it does not account for the nuances of the journey, such as steep inclines or declines that could significantly affect energy consumption. Future versions could incorporate more granular elevation data along the route to provide a more accurate SoC estimate. Additionally, the system could consider regenerative braking systems in some EVs, which can recover energy during descents, partially offsetting the energy used in ascents.

8.7.5 Battery Aging Factor in SoC Estimation

Battery ageing is a critical factor that the system currently does not consider. As batteries age, their capacity to hold a charge diminishes, affecting the vehicle's range and SoC. Future iterations could include an option for users to input the age of their EV battery. This data could then be used with manufacturer-provided battery degradation models to offer a more accurate SoC estimate. This feature would make the system more adaptable and accurate over the long term, as EV batteries inevitably degrade with use [69].

8.7.6 Extensive Weather Data for Enhanced Accuracy

The current system uses a simplified approach to weather's impact on the SoC by calculating the mean temperature between the origin and destination. Then, as explained, it applies two types of discharge rates depending on whether they are above or below 10 degrees Celsius. While this provides a basic understanding of how cold weather might affect battery performance, it leaves out other critical weather factors that could significantly impact SoC and overall trip planning.

For instance, extreme weather conditions like storms, heavy rain, or snow could be integrated into the SoC calculations. These conditions affect not only the vehicle's energy consumption but also the road conditions and, consequently, the time required to reach each CS. While predicting these conditions for precise SoC calculation might be ambitious, the system could at least warn the user about upcoming severe weather conditions, allowing them to make informed decisions, or even introduce a negative factor in the ETA when adverse conditions are detected.

Moreover, the system currently does not account for the impact of extreme heat on battery performance. High temperatures can also harm the battery's SoC, significantly reducing the vehicle's range. According to studies, extreme heat can reduce the EV's range by up to 31% [70]. Future system iterations could incorporate heat-related discharge rates to provide a more comprehensive and accurate SoC estimate.

8.7.7 Including Eco-conscious Driving Mode

In the context of the findings from Chapter 2: Literature Review, the route preferences for EV drivers are fundamentally different from those driving ICE vehicles. Specifically, EV drivers often opt for routes that facilitate slower speeds to conserve the battery's SoC and mitigate the well-known issue of 'range anxiety.' This distinct behaviour pattern suggests a valuable avenue for future enhancements to the microservice: introducing an 'eco-conscious driving mode.'

This eco-conscious driving mode would be an optional feature that users could enable when they are more concerned with energy efficiency than reaching their destination as quickly as possible. The underlying algorithm for this mode would focus on selecting routes that allow for slower driving speeds, which are generally more energy efficient.

Integrating advanced analytics which takes the eco-conscious driving mode into account is technically feasible improvement as services such as Google Maps' Routes already offer an ecological routing option optimised for fuel or energy efficiency. When the vehicle type is set to electric, the API uses traffic data, road conditions, and other factors to provide a route that minimises energy consumption. This is particularly beneficial for EVs, which often achieve greater efficiency in stop-and-go urban environments and on inclines where regenerative braking can be utilised [71].

9. Discussion and Conclusion

9.1 Introduction

This chapter offers a detailed discussion and conclusion of the thesis, critically examining the Navigator's effectiveness against its objectives and exploring its broader implications for sustainable mobility and EVs. It also assesses future directions for the evrich service in the SOGNO framework, highlighting its role in the field of digitalisation and automation.

9.2 Reflecting on Research Questions and Objectives

9.2.1 What Data Types Should be Considered for the ITS?

The initial concept for the ITS suggested a variety of factors, such as routing and mapping data, weather conditions, traffic dynamics, CI information, user preferences, and pricing, might be crucial for optimizing routes. Reflecting on the outcomes and data gathered through the implementation of the Navigator and its subsequent integration into the evrich service, some initial assumptions have been largely validated while others have provided a more complex and varied understanding of their actual impact and utilization.

The use of mapping data in the Navigator has been confirmed crucial. This data facilitates accurate estimations of the ETA and distance to each CS, enhancing route efficiency. Moreover, it enables segmenting the route into two different segments (city and highway), allowing for a more precise calculation of the SoC by accounting for the different discharge rates provided by the vehicle manufacturers for these two driving modes. Additionally, it has facilitated the estimation of pedestrian walking times from the CS to the actual destination, a beneficial feature for enhancing user comprehension.

The integration of real-time traffic data, particularly from sources like Bing Maps, has not been fully realised in the Navigator's test phase. While the mapping service offers estimations of traffic flow, it lacks real-time data analysis, relying instead on longer-term traffic estimations. Though not utilised in the test stage, incorporating real-time traffic data into the Navigator, and subsequently into the evrich service, could significantly enhance the system's effectiveness. However, this would likely incur additional costs.

In the Navigator's implementation, weather data, specifically temperature, has been utilised primarily for SoC estimation, acknowledging the impact of temperature on battery performance. Manufacturers often provide different discharge rates for temperatures below or above ten degrees Celsius. However, other weather elements like rain or snow, which might affect travel speed and SoC calculations, have not been incorporated, which leaves a potential enhancement opportunity.

CI data has proven to be essential. The infrastructure determines the location, accessibility, and feasibility of CSs on the routes. An incompatible CS or an error in locating them precisely can lead to significant inconvenience, potentially leaving the user stranded or compelling them to deviate considerably from their intended path. Pricing data integration also allows users to make informed decisions, ensuring the viability of their route and optimising their expenses.

The way the two deployments handle pricing information can vary. While the Navigator in the standalone model presents potential pricing options to users, provided by the OCM service, the evrich service goes a step further. It delves into real-time availability and dynamic pricing, ensuring that users have a comprehensive understanding of their potential expenses, thereby refining the user experience and eliminating potential inconveniences.

Additionally, incorporated personalised inputs have proved to be invaluable for a potential user. These include the user's origin and destination points and the specific EV model, which is crucial for accurate SoC calculations, and compatibility checks with CSs. Additionally, the Navigator as a standalone microservice considers user preferences for fast charging, allowing for a more tailored user experience. Evrich service, on the other hand, allows users to input their desired target SoC and V2G allowance. This enables the service to provide the best charging option considering dynamic pricing and the estimated charging time frame.

Road gradient or elevation changes, initially not included in the data types to be considered in Chapter 1, were incorporated following the analysis in Chapter 2: Literature Review, to enhance the accuracy of SoC estimations. It was found that journeys with significant positive gradients result in considerably reduced predicted ranges compared to flat routes. This insight has been integrated into the Navigator, leading to more conservative SoC calculations by incorporating the gradient factor. Future work, following the completion of this thesis, will involve real-life testing to verify the accuracy of this altitude factoring.

9.2.2 How Can These Parameters Be Integrated?

Upon reflecting on the integration of various parameters in the ITS, the research has underscored the effectiveness of relying on external services. This approach has been instrumental in sourcing data for EV characteristics, mapping, temperature, and CI, as well as in the implementation of these elements within the system. The analysis, selection of external services and resources, and subsequent deployment have proven key to the system's success.

9.2.3 Is Real-Time Data Essential?

The Navigator has demonstrated its ability to effectively integrate all necessary parameters, despite the lack of real-time data. It has successfully estimated SoC calculations, which still would need to be validated in real-life scenarios and performed checks for CI compatibility and fast charging priorities. This performance establishes a solid foundation for the system, indicating the potential for further enhancements, including the integration of real-time data. This successful performance demonstrates that real-time data is not a prerequisite for the testing phase, and it underscores that the cost factor associated with real-time data integration can outweigh its benefits during this initial stage of deployment.

With financial resources available for a real-use case deployment, due to the dynamic environment of EVs, factors like traffic, CS availability, and electricity pricing show that, ideally, the optimal frequency for updating most would be in real-time. This is particularly true for critical functions such as SoC calculations and CS selection. The evrich service exemplifies this with its real-time analysis of dynamic pricing for optimal CS selection, showcasing a feature that would be beneficial to replicate across other parameters of the ITS. Implementing similar real-time updates for these factors could significantly enhance the system's efficiency and decision-making accuracy.

9.2.4 Objectives

9.2.4.1 Enhancing EV User Experience and Alleviating Range Anxiety

The Navigator microservice has proved to enable smart decision-making and mitigate range anxiety. By providing accurate manufacturer based estimated SoC calculations and indicating reachable CSs, showcased in the Practical Application and Validation (Sections 6.3 and 7.4) for both deployments, it directly addresses concerns about an EV's travel capabilities. In the standalone model, this information empowers users to select their preferred CSs based on their immediate needs and preferences. Meanwhile, when paired with the evrich service, the service points the users to the most optimal CSs, considering real-time factors like pricing and availability.

This dual functionality, whether giving users the autonomy to choose or directing them to the best available option, significantly improves the user experience. Having these tangible, reliable recommendations assure users about their journey's feasibility and their vehicle's capabilities.

Furthermore, user experience has been significantly enhanced and validated in the Practical Application and Validation for both deployments, through the development of features that ensure charging compatibility with the CS identified following the CS external service query. Allowing users to input their preference for fast charging also showcases the user-centric approach of the Navigator.

9.2.4.2 Optimisation of Current CI

The Navigator microservice has proven to be able to play an important role in improving the current CI. In its standalone mode, the Navigator microservice ensures EV users find compatible chargers, by providing users with information about compatible CSs around their destination, it is ensuring that drivers are directed to stations they can use. This helps prevent congestion at popular stations and spreads the demand more evenly across available infrastructure.

Moreover, the Navigator contributes significantly to the improvement of the CI by giving visibility to all CSs within the user-specified radius, including those at less known locations, thus ensuring a broader awareness of available charging options.

Beyond that, working on the evrich service, the Navigator helps the Coordinator to prepare availability queries for the charging operators and handle demand smoothly. By notifying them about when an EV is expected to arrive and how much charge it will need, operators can manage and allocate their resources more effectively. This system, powered by data, means operators can respond to actual demand quickly, ensuring stations can handle peak times without overloading.

Moreover, the inclusion of V2G functionalities in evrich is a significant step forward. This feature allows EVs to not only draw power from the grid but also supply it back when necessary. Such bidirectional energy flow not only maximises the use of available energy but also supports a more sustainable and balanced power grid.

9.3 Analysis of Research Findings

The research explores the complexities of EV routing. This subject has become increasingly relevant as the world shifts towards sustainable modes of transportation. Central to the study is the development of an ITS, which serves as a multi-dimensional solution to the challenges faced by EV drivers.

One of the most significant achievements of this research is the system's ability to detect a set of CSs compatible with the user’s preferences and compute the EV’s SoC estimate based on proven and tested manufacturers discharge rates, also considering parameters such as weather conditions, road gradient, and mapping data. This is a critical factor for EV drivers, as inaccurate SoC estimates can lead to range anxiety—a significant barrier to adopting EVs.

The adaptability of the Navigator was tested through various user scenarios, each with its challenges and requirements. These ranged from daily commutes in fluctuating weather conditions to long-distance journeys that involved significant altitude changes. The system proved its robustness through these tests and demonstrated its ability to adapt to various conditions. For instance, a student in Valencia could easily find the closest CS to their university. Similarly, a user who frequently travels to Barcelona on a medium-length route could rely on the system to adjust for seasonal weather changes that affect battery performance. Another user planning a stop for a long route through Germany in Dusseldorf found the system valuable for locating a fast-CS when their EV's battery was running low.

Across these diverse scenarios, the ITS emerged as a reliable assistant capable of adapting to various user needs, EV models and environmental conditions.

From a technical standpoint, performance metrics were crucial to this research. The system was evaluated regarding CPU and memory usage under different operational states, including idle and peak conditions. The results were promising; the system demonstrated excellent resource management capabilities, maintaining stable CPU and memory usage across different operational states. This is a vital attribute for any real-world application, as it suggests that the system can scale effectively, accommodating a growing number of users without compromising performance or reliability.

Throughout the systematic performance evaluation of the built-in API in the standalone model, critical insights into the system's responsiveness and handling of concurrent requests have been gained. Initial tests, which involved considering the priority of fast charging for users, revealed a significant impact on both the API response time and size, underscoring the computational efficiency gained when the system's focus is narrowed to fast-CSs. The response time was halved, and the response size significantly reduced when the priority was set to 'true', illustrating the direct correlation between the system's search scope and its performance metrics.

As the examination progressed to assess traffic management under multiple concurrent requests, the API demonstrated robustness by consistently delivering accurate responses in sequence. However, as anticipated with increasing concurrent requests, a growth in response times was observed, attributable to the queuing system inherent to FastAPI's operation. Notably, the 'warm-up' effect observed in the initial request hinted at potential optimisations, as subsequent requests benefited from cached data and established connections, thereby enhancing processing times. These outcomes illustrate the microservice current performance under various load conditions but also highlight areas for optimisation.

Moreover, the microservice architecture of the ITS has proven to be highly flexible and interoperable, seamlessly integrating with other entities or microsystems within evrich. This allows for easy updates and enhancements without disrupting the entire system, ensuring the ITS can adapt to new technologies and standards as they emerge. The interoperability with SOGNO opens avenues for collaborative data sharing and analytics, enhancing the system's capabilities and making it more robust and versatile.

In addition, the Navigator, by leveraging Docker containerisation has enhanced its scalability and deployment efficiency further. This containerisation simplifies the complexities of deploying and managing the system across diverse computing environments, from local servers to cloud-based infrastructures. It also facilitates easier version control and rollback capabilities, allowing for rapid iterations and updates without affecting the system's overall stability. Docker has added another layer of flexibility and robustness to the ITS, making it well-suited for large-scale, real-world applications.

In summary, the research has successfully tackled some of the most pressing challenges in EV routing by developing reliable, adaptable, and scalable ITS. The Navigator's ability to integrate multiple data sources into its routing algorithm makes it a promising solution in ITSs for EVs.

9.4 Implications

The introduction of the Navigator microservice represents an innovative approach to addressing the practical needs of EV drivers, specifically targeting the enhancement of their on-road experience. Its capability to present a list of accessible CSs within a given radius and estimate the SoC necessary to reach them has direct implications for the day-to-day operation of EVs.

The most immediate implication for users is the reduction of range anxiety. With the Navigator's provision of potential charging locations and SoC estimations based on EV manufacturers actual discharge rates, drivers can embark on journeys with a clearer understanding of where they can charge and how much energy they will have upon arrival. This level of informed planning is critical to user confidence and could be easily validated with real user interactions, providing a feedback loop for further refinement of the microservice.

The Navigator enhances users' ability to efficiently plan their journeys and make informed decisions by ensuring that:

- Users can select CSs that align with their route and schedule, minimising detours and unnecessary stops. This not only saves time but also ensures that charging aligns with the natural breaks in their journey.
- Drivers can manage their vehicle's energy use more effectively by understanding the impact of driving habits, speed, and route topography on their vehicle's range, leading to energy-conserving behaviours.
- By providing information on CS availability and cost, the Navigator allows users to plan cost-effective routes, avoiding expensive CSs or peak charging times when possible.
- Knowing the vehicle's range and the locations of CSs can allow drivers to focus more on enjoying their trip, whether it be the scenery, company, or driving itself, rather than on concerns about whether they have enough charge to reach their destination.
- In unforeseen circumstances, such as road closures or unexpected low Soc, drivers can quickly reassess their route via the Navigator and reach alternative CSs if needed.

The design of the Navigator ensures that it is not only a theoretical model but also ready for practical deployment. Its implementation into a live environment promises to deliver tangible benefits to users and could serve as a case study for the application of ITSs in sustainable transportation.

9.5 Contributions and Impact

The research of this thesis could impact various domains in the electro-mobility sector in the short to medium term. This chapter outlines these contributions and their broader implications.

9.5.1 Promoting EVs

The Navigator, designed to provide a range of SoC estimations that consider a multitude of factors including weather, CI information and routing, seeks to support EV use. It aims to reduce range anxiety by offering a tool that helps users plan their routes with CS locations in mind, thereby potentially increasing the adoption of EVs.

The intended service could help in encouraging the broader adoption of EV by enhancing the user experience and promoting energy efficiency. The convenience, cost savings, and reduced environmental impact associated with EVs become more apparent and appealing to a broader range of consumers.

9.5.2 Cost-Efficient EV Charging

The Navigator microservice estimates costs for potential CSs aiding users in focusing their search, particularly when cost is a primary concern. This feature enables the selection of more economical CSs. Concurrently, the evrich service utilises dynamic pricing signals and optimisation algorithms, empowering EV drivers to pinpoint the nearest and most cost-effective CSs efficiently.

9.5.3 Enhancing Sustainability

By optimising routing and charging strategies, the Navigator[6] can contribute to the overall efficiency of the transportation network and reduce energy consumption, leading to minimising environmental impacts.

Moreover, the increased adoption of EVs (mentioned in Section 9.5.1), in turn, leads to a more significant reduction in ICE vehicles, dependence on fossil fuels and greenhouse gas emissions, further contributing to a sustainable and clean transportation future.

[6] Within the evrich service, the future enhanced Navigator could play a crucial role in sustainability and facilitating the integration of renewable energy sources into the power grid. It has the potential to offer intelligent charging solutions by guiding EV users to CSs that predominantly use renewable energy, especially during times of peak renewable generation.

9.5.4 Driving CI Development

The impact of an increase in EV adoption would definitely drive the need for an expanded and robust CI to accommodate the growing user base.

The Navigator, as part of the evrich service, can substantially influence the development of new CI. By accurately mapping out EV usage patterns and charging demands, the Navigator could provide critical data that can identify gaps in current CI. These data-driven insights could encourage investment and growth in the construction and management of new CSs.

9.5.5 Impact on Education and Future Development

While the research project itself is proprietary, the methodologies and algorithms developed have the potential to be shared openly. Open-sourcing these elements could lead to further innovation in the field, allowing other researchers and developers to build upon this foundational work. This collaborative approach could speed up advancements in sustainable transportation technology, making it more accessible and effective for a broader audience.

9.6 Future Avenues for Research and Innovation in evrich

The overarching aim of the evrich service and the Navigator microservice developed in this thesis is to leverage e-mobility by creating a comprehensive, intelligent, and user-centric routing service. As part of this broader vision, there are several exciting avenues for future research and development.

9.6.1 Advanced Machine Learning for Personalized SoC Estimation

An intriguing area for future research is using advanced machine learning algorithms to provide more personalised SoC estimates. By collecting and analysing data on individual EV and driving behaviours, weather conditions, and traffic patterns, the system could adapt its SoC algorithms to each user's specific circumstances. This would be particularly beneficial for long-term users, as the system would become increasingly accurate in battery degradation prediction and SoC and route estimations the more it is used.

In the long term, the evrich service could offer increasingly personalised and accurate route and charging recommendations by leveraging these advanced machine learning algorithms. This would make the platform more adaptive and user-centric, setting a new standard for what consumers expect from an e-mobility service.

9.6.2 Dynamic Demand Management

One of the most promising aspects of evrich long-term impact lies in its potential for supporting dynamic demand management. As the service gains more users, it could coordinate with CS operators to manage increases in demand and price variations effectively. This would prevent queuing at popular CSs and direct drivers to less frequented, yet more cost-effective, charging options that may be slightly farther away. Such a feature could create a win-win situation: drivers get to charge their vehicles without waiting, and underutilised CSs see an increase in demand.

Furthermore, evrich could integrate predictive analytics to forecast demand surges, enabling CS operators to adjust pricing or availability in real time. This level of coordination could improve consumption and distribution in urban settings, making the entire system more efficient and sustainable.

9.6.3 Integration with Smart Grids for Sustainable Urban Ecosystems

In the context of future research, an area that warrants focused attention is the synergy between the evrich service in SOGNO and smart grids. With evrich's existing V2G capabilities, the platform is aligned with the smart grid architecture, providing a two-way flow of electricity and information. However, there are layers of additional complexities and opportunities that remain unexplored. For example, a critical aspect could be the development of 'Grid-Aware Routing Algorithms.' Unlike traditional algorithms, these would not only consider distance, time, and traffic but also consider real-time data on grid load and the availability of renewable energy sources. This way, the evrich service could actively contribute to the optimal usage of renewable resources, thereby lowering the carbon footprint of each trip.

With highly adaptable and responsive intelligent grids, their real-time data can be leveraged to enable 'Dynamic Load Balancing' across CSs. This goes beyond merely managing the energy supply during peak and off-peak hours. Instead, it can involve highly sophisticated coordination where the grid, in conjunction with evrich in the SOGNO framework, predicts demand surges and allocates energy supply, accordingly, even redirecting vehicles to less busy or more energy efficient CSs.

9.7 Personal Reflection

In conclusion, this thesis has served as a profound learning experience, advancing my understanding of programming, Docker utilisation, microservice architecture, and the emerging field of e-mobility. The practical work of developing the Navigator microservice has provided valuable insight into the complexities and potentials of IoT systems in transportation and mobility, highlighting the vital role such innovations play in the digitalization of this sector.

Throughout this research, the process of integrating the Navigator into the evrich system necessitated a holistic comprehension of the existing digital infrastructure. By delving into the system's architecture and refining communication protocols, this work has underscored the importance of seamless interaction within evrich, even into a larger DMSs aligning with broader objectives like electric grid optimisation and the anticipation of renewable energy needs.

The process of adaptation, particularly within the microservices ecosystem, has been both challenging and enlightening. It became clear that the development of microservices often involves trade-offs, where individual feature sets may be sacrificed for the greater good of system integrity and operational fluidity. This effort has reinforced the principle that the integration and functionality of the larger system take precedence over the capabilities of any single service.

Beyond the technical and academic growth, this journey has reaffirmed a personal belief in the power of motivation and purpose. It has taught me that the path to learning is inexhaustible, and progress often lies in perseverance and a steadfast commitment to the direction chosen, even when immediate results are not apparent.

On a personal note, the experience has developed my passion for how digitalisation and data analytics can profoundly impact the physical world—shaping systems that are not only technologically advanced but also beneficial to society. Particularly in the realm of mobility, where the stakes include carbon neutrality and sustainability, the drive to contribute meaningful work is immensely motivating.

This thesis represents a modest contribution to the expansive field of ITSs and electric mobility. It reflects the commitment to ongoing learning and improvement. The Navigator microservice, while a small part of a larger routing system, illustrates how thoughtful technology use can lead us toward a future that is both more efficient and more sustainable.

9.8 Concluding Remarks

I would like to express my heartfelt gratitude to Erdem Gümrükcü for serving as my thesis supervisor. His invaluable guidance, wealth of knowledge, and insightful advice have been crucial in shaping my understanding of the evrich service and the fundamentals of the SOGNO architecture and microservices. I would also like to extend my sincere appreciation to Prof. Ferdinanda Ponci for her role in reviewing this thesis.

I extend my thanks to the ACS Institute for affording me the opportunity to contribute to this ambitious project centred around the SOGNO architecture for the Linux foundation. Additionally, I am immensely grateful to RWTH Aachen for granting me a one-year exchange opportunity, which has been a transformative experience that has enriched my personal and professional growth.

I would also like to acknowledge ev-database.org for providing a comprehensive and constantly updated source of information on various EV models. Their data played an important role in the initial population of the EV Database.

Bibliography

[1] H. Bhasin, “Marketing91 - The Importance of Transportation Explained,” Marketing91, 20 January 2020. [Online]. Available: https://www.marketing91.com/importance-of-transportation/.

[2] E E. A. Team, „European Environment Agency,“ European Environment Agency, 04 April 2023. [Online]. Available: https://www.eea.europa.eu/en/topics/in-depth/transport-and-mobility.

[3] E. C. Team, „European Commission - Climate Action | 2050 Long-term Strategy,“ European Commission, 2023. [Online]. Available: https://climate.ec.europa.eu/eu-action/climate-strategies-targets/2050-long-term-strategy_en.

[4] LRP Energy Team, „LRP Energy - MOVES III Plan: What the subsidy is for and who can benefit from it?,“ LRP ENergy, 2023. [Online]. Available: https://lrpenergy.com/moves-iii-plan-what-the-subsidy-is-for-and-who-can-benefit-from-it-/.

[5] Wallbox. Team, „Wallbox - Everything You Need to Know About EV Incentives in Spain,“ Wallbox, 2023. [Online]. Available: https://blog.wallbox.com/spain-ev-incentives/.

[6] J. A. Soria-Lara, J. Gómez, J. M. Vassallo, and J. Tarriño-Ortiz, "Public Acceptability of Low Emission Zones: The Case of 'Madrid Central'," MDPI Sustainability, vol. 13, no. 6, p. 3251, 2021.

[7] R. Kakkar, R. Gupta, S. Agrawal and S. Tanwar, „A Review on Standardizing Electric Vehicles Community,“ *MDPI,* 2022.

[8] ClearWatt Team, „ClearWatt,“ ClearWatt, 2023. [Online]. Available: https://www.clearwatt.co.uk/charging/connectors-tech-specs.

[9] Best Chargers Team, „Best Chargers - Different types of EV Charging Connectors,“ Best Chargers EU, 2023. [Online]. Available: https://bestchargers.eu/blog/different-types-of-ev-charging-connectors/.

[10] EV Charging Summit Team, „EV Industry Blog - 10 Biggest Challenges Facing the EV Industry Today,“ EV Charging Summit, 2023. [Online]. Available: https://evchargingsummit.com/blog/challenges-facing-the-ev-industry-today/.

[11] M. Choudhary, „Geospatial World,“ Geospatial Media and Communications, 15 January 2019. [Online]. Available: https://www.geospatialworld.net/blogs/what-is-intelligent-transport-system-and-how-it-works/.

[12] ACECC TC-16, "ITS Introduction Guide: ITS-based Solutions for Urban Traffic Problems in Asia," *Japan Society of Civil Engineers (JSCE)*, 2023.

[13] „European Commission - Mobility and Transport,“ European Commission, 19 October 2023. [Online]. Available: https://transport.ec.europa.eu/transport-themes/intelligent-transport-systems_en.

[14] L. F. Luque-Vega, D. A. Michel-Torres, E. Lopez-Neri, M. A. Carlos-Mancilla, and L. E. González-Jiménez, "IoT Smart Parking System Based on the Visual-Aided Smart Vehicle Presence Sensor: SPIN-V," *Sensors*, vol. 20, no. 5, p. 1476, March 2020, doi: 10.3390/s20051476.

[15] Cognizant Team, „Cognizant - Get past EV ‘range anxiety’ with intelligent trip planning,“ Cognizant, 10 February 2023. [Online]. Available: https://www.cognizant.com/us/en/insights/insights-blog/get-past-ev-range-anxiety-with-intelligent-trip-planning-wf1410961.

[16] J. Neate, “Oracle Blogs - OHIP Streaming API: Understanding our strategy,” 13 December 2021. [Online]. Available: https://blogs.oracle.com/hospitality/post/ohip-streaming-api-understanding-our-strategy

[17] S. Agrawal, H. Zheng, S. Peeta, and A. Kumar, "Routing Aspects of Electric Vehicle Drivers and Their Effects on Network Performance," *Transportation Research Part D: Transport and Environment*, vol. 46, pp. 246-266, April 2016, doi: 10.1016/j.trd.2016.04.002.

[18] M. Cuchý, M. Štolba, and M. Jakob, „Benefits of Multi-Destination Travel Planning for Electric Vehicles,“ *2018 21st International Conference on Intelligent Transportation Systems (ITSC),* pp. 327-332, 2018.

[19] M.M. de Weerdt, S. Stein, E. Gerding, V. Robu, and N. Jennings, "Intention-Aware Routing of Electric Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 5, pp. 1472-1482, 2015. doi: 10.1109/TITS.2015.2506900.

[20] M. Neaimeh, G. Hill, Y. Hübner, P. T. Blythe, "Routing systems to extend the driving range of electric vehicles," *IET Intelligent Transport Systems*, vol. 7, no. 3, pp. 327-336, September 2013. doi: 10.1049/iet-its.2013.0122.

[21] J. Lindgren and P. D. Lund, "Effect of extreme temperatures on battery charging and performance of electric vehicles," *Journal of Power Sources*, vol. 328, pp. 37-45, 2016. doi: 10.1016/j.jpowsour.2016.07.038.

[22] A. M. Team, „Microsoft Learn - Azure Maps Documentation,“ Microsoft, 22 March 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-maps/.

[23] „Open Charge Map,“ Open Charge Map, [Online]. Available: https://openchargemap.org/site.

[24] Mapbox Team, „Mapbox Documentation,“ Mapbox, 2023. [Online]. Available: https://docs.mapbox.com/api/navigation/directions/#electric-vehicle-routing.

[25] Python Team, „Python,“ Python.org, 2023. [Online]. Available: https://www.python.org/.

[26] The Linux Foundation Team,„LF Energy - SOGNO,“ LF Energy, 2023. [Online]. Available: https://lfenergy.org/projects/sogno/.

[27] Geopy Contributors, „Geopy 2.3.0 documentation,“ Geopy, 2023. [Online]. Available: https://geopy.readthedocs.io/en/stable/.

[28] OpenWeatherMap Team, „Weather API - OpenWeatherMap,“ OpenWeather Ltd., 19 October 2023. [Online]. Available: https://openweathermap.org/api.

[29] Weatherbit Team, „Weatherbit - API Documentation,“ Weatherbit, 19 October 2023. [Online]. Available: https://www.weatherbit.io/api.

[30] Open-Meteo Team, „Open-Meteo,“ Open-Meteo, 2023. [Online]. Available: https://open-meteo.com/.

[31] Open Charge Map Team, „Open Charge Map - Statistics,“ Open Charge Map, 2023. [Online]. Available: https://openchargemap.org/site/stats.

[32] EVgo Team, „EVgo - PlugShare Achieves Milestone of More Than 6.5 Million Check-Ins as Global EV Adoption Grows,“ EVgo, 1 June 2023. [Online]. Available: https://investors.evgo.com/news/news-details/2023/PlugShare-Achieves-Milestone-of-More-Than-6.5-Million-Check-Ins-as-Global-EV-Adoption-Grows/.

[33] PlugShare Team, „PlugShare for Business,“ PlugShare, 19 October 2023. [Online]. Available: https://company.plugshare.com/business.html.

[34] ChargePoint Team, „ChargePoint,“ ChargePoint, 19 October 2023. [Online]. Available: https://www.chargepoint.com/en-gb.

[35] Google Maps Team, „Google Maps Platform Documentation | Routes API | Google for Developers,“ Google, 8 March 2023. [Online]. Available: https://developers.google.com/maps/documentation/routes.

[36] Mapbox Team, „Mapbox Navigation API Documentation,“ Mapbox, 8 March 2023. [Online]. Available: https://docs.mapbox.com/api/navigation/.

[37] Bing Maps Team, „Elevations API - Bing Maps,“ Microsoft, 21 July 2022. [Online]. Available: https://learn.microsoft.com/en-us/bingmaps/rest-services/elevations/.

[39] Google Maps Team, „Google Maps Platform Documentation | Elevation API | Google for Developers,“ Google, 18 October2023. [Online]. Available: https://developers.google.com/maps/documentation/elevation/start.

[40] MySQL Team, „MySQL,“Oracle Corporation, 2023. [Online]. Available:https://www.mysql.com/.

[41] Kinsta Team, „SSH vs SSL: What’s the Difference?,“ Kinsta Ltd., 28 November 2022. [Online]. Available: https://kinsta.com/knowledgebase/ssh-vs-ssl/.

[42] A. Rehman, „Cloudways - PostgreSQL vs MySQL: Which To Choose?,“ 2023, 15 June 2023. [Online]. Available: https://www.cloudways.com/blog/postgresql-vs-mysql/.

[43] PostgreSQL Global Development Group, „PostgreSQL Official Documentation,“P.G.D. Group, 2023. [Online]. Available: https://www.postgresql.org/docs/16/index.html.

[44] MongoDB Team, „MongoDB - Features & Key Characteristics,“ MongoDB, 2023. [Online]. Available: https://www.mongodb.com/features.

[45] K. Reitz, „PyPI (Python Package Index) - requests,“ Python Software Foundation, 2023. [Online]. Available: https://pypi.org/project/requests/.

[46] MySQL Team, „MySQL - MySQL Connectors,“ Oracle Corporation, 2023. [Online]. Available: https://www.mysql.com/products/connector/.

[47] Open Charge Map, 'Open Charge Map GitHub Repository,' [Online]. Available: https://github.com/openchargemap.

[48] Open Charge Map, „Open Charge Map - API,“ Open Charge Map, 2023. [Online]. Available: https://openchargemap.org/site/develop/api#/.

[49] EV Database Team, „EV Database,“ EV Database , 2023. [Online]. Available: https://ev-database.org/.

[50] FastAPI Team, „FastAPI,“ FastAPI, 2023. [Online]. Available: https://fastapi.tiangolo.com/.

[51] Fast API Team, „Fast API - CORS,“ Fast API, [Online]. Available: https://fastapi.tiangolo.com/tutorial/cors/.

[52] Pydantic Team, „Pydantic,“ Pydantic, 2023. [Online]. Available:https://docs.pydantic.dev/latest/.

[53] Starlette Team, „Starlette,“ Starlette, 2023. [Online]. Available: https://www.starlette.io/.

[54] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures (Doctoral dissertation),“ *University of California,* 2000.

[55] OpenAPI Team, „OpenAPI Initiative,“ Linux Foundation, 2023. [Online]. Available: https://www.openapis.org/.

[56] JSON Schema Team, „JSON Schema,“ JSON Schema, 2023. [Online]. Available: https://json-schema.org/.

[57] Docker Team, „Docker,“ Docker,Inc., 2023. [Online]. Available: https://www.docker.com/resources/what-container/.

[58] Amazon Web Service Team, „AWS Whitepapers - Container Benefits,“ Amazon Web Services, Inc., 2023. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/docker-on-aws/container-benefits.html.

[59] KnowledgeHut Team, „KnowledgeHut - Why Use Docker?,“ KnowledgeHut, 2023. [Online]. Available: https://www.knowledgehut.com/blog/devops/why-use-docker.

[60] Docker Team, „Docker Documentation - Networking,“ Docker, 2023. [Online]. Available: https://docs.docker.com/network/.

[61] Docker Team, „Docker Docs - Engine API,“ Docker, 2023. [Online]. Available: https://docs.docker.com/engine/api/.

[62] Docker Team, „Docker Docs - Swarm Mode,“ Docker, 2023. [Online]. Available: https://docs.docker.com/engine/swarm/.

[63] E. Gumrükcü, „Github - sogno-platform/evrich,“ Github, Inc., 7 November 2023. [Online]. Available: https://github.com/sogno-platform/evrich.

[64] MQTT Team, „MQTT,“ MQTT.org, 2023. [Online]. Available: https://mqtt.org/.

[65] R. Light, „ Python Package Index (PyPI) - paho-mqtt,“ Eclipse Foundation, 2021. [Online]. Available: https://pypi.org/project/paho-mqtt/.

[66] Eclipse Mosquitto Team, „Eclipse Mosquitto - MQTT Broker,“ Eclipse Foundation, 2023. [Online]. Available: https://mosquitto.org/.

[67] E. Gümrükcü, J.R.A. Klemets, J.A.W. Suul, F. Ponci, A. Monti, "Decentralized Energy Management Concept for Urban Charging Hubs with Multiple V2G Aggregators," *IEEE Transactions on Transportation Electrification*, vol. 9, no. 2, pp. 2367-2381, 2022. doi: 10.1109/tte.2022.3208627.

[68] Vanessa Page, „What is Waze? - Investopedia,“ 2022. [Online]. Available: https://www.investopedia.com/articles/investing/060415/pros-cons-waze.asp.

[69] B. Xu, A. Oudalov, A. Ulbig, D.S. Kirschen, "Modeling of Lithium-Ion Battery Degradation for Cell Life Assessment," *IEEE Transactions on Smart Grid*, vol. 99, no. 2, pp. 1-1, June 2016. doi: 10.1109/TSG.2016.2578950.

[70] I. American Automobile Association, „AAA ELECTRIC VEHICLE RANGE TESTING," American Automobile Association, Inc. , Heathrow, Florida., 2019.

[71] Google Maps Team, „Google for Developers," Google, 31 August 2023. [Online]. Available: https://developers.google.com/maps/documentation/routes/eco-routes.