Unlocking Sensor-Based Utilization Analysis in Smart Buildings

A Case Study at AAU Innovate

Lars Jan J Depuydt

Computer Engineering, CE10-NDS, group 1049E, 2025

Master's thesis

This report is typeset using the Typst system.

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
Aalborg University
https://www.es.aau.dk/

Title:
Unlocking Sensor-Based Utilization Analysis in Smart Buildings

Theme:
A Case Study at AAU Innovate

Project Period:
Spring Semester 2025

Project Group:
CE10-NDS, group 1049E

Participants:
Lars Jan J Depuydt

Supervisor:
Jimmy Jessen Nielsen

Copies: 1

Number of Pages: 112

Date of Completion:
03/06-2025

Abstract:

Modern smart buildings collect large volumes of data through Internet of Things systems, yet much of this data remains unused, especially for understanding how spaces are actually used. This project explores whether existing systems in AAU Innovate, a multi-functional building, can be used to generate useful insights and support informed decision-making. Five different data sources were accessed, analyzed, and integrated into a unified and hostable data graph. These sources included facility management, booking calendars, building management systems, coffee machines, and entry counters. To test the value of this integration, five practical questions were developed together with AAU Innovate's management and solved by an expert using the combined data. All questions led to actionable insights. The project also tested the use of a large language model as an interface for nontechnical users. While the model showed promise in writing Python scripts for simple tasks, it struggled with more complex reasoning. Still, the results show the potential of combining data infrastructure with artificial intelligence.

Preface

This project was carried out in collaboration with Aalborg University (AAU) Innovate. My name is Lars Depuydt, and this is my master's thesis. Since November 2024, I have been employed as a student helper at AAU Innovate, where I was partly responsible for data collection. After a few months, I realized the potential of the project, as well as the sheer scale of the task, which exceeded what could be accomplished within the scope of a student job alone. My direct supervisor and project collaborator, Christoffer Mørch, supported the idea of turning this effort into a master's thesis and has provided guidance throughout the process, including access to data, project discussions, and valuable feedback. While my student job and this thesis partially overlap, all work performed as part of my student employment will be explicitly marked as such in the report.

One of the biggest challenges of the project turned out to be getting access to the necessary data. I want to thank all the helpful people I met along the way, many of whom went out of their way to search for solutions or provide access, sometimes successfully. I would also like to thank my academic supervisor, Jimmy Nielsen, for his support and advice during the thesis work. This project has been both technically challenging and personally rewarding, giving me valuable insights into data infrastructure, applied research, and collaborative development.

AI tools such as ChatGPT, Gemini, and DeepSeek were used during the writing process for tasks such as spelling correction, sentence restructuring, and phrasing. They were also used to a limited extent in writing implementation code. Importantly, these tools were never used to handle or process any data from AAU. All diagrams in the report were created using Draw.io. The computationally intensive parts of the project were executed on the UCloud interactive HPC system, managed by the eScience Center at the University of Southern Denmark.

Contents

Introduction

The Fourth Industrial Revolution is transforming our interaction with technology, driving increased automation, digital connectivity, and intelligent decision-making across various sectors, including the built environment [1]. As a result, residential and commercial buildings are undergoing a significant shift, with the Internet of Things (IoT) playing a central role in developing smart, efficient, and secure infrastructures. Researchers have leveraged IoT technologies to enhance real-time monitoring, automation, and control of building systems, thereby improving occupant comfort, energy efficiency, and security.

In recent years, the development of smart building (SB) control systems has surged. These systems connect monitored environmental variables—such as temperature, humidity, luminosity, and air quality—with building management systems like heating, ventilation, and air-conditioning (HVAC) and lighting systems to optimize indoor environmental conditions [2], [3]. The focus of most studies has been on optimizing energy efficiency, as buildings consume approximately 40% of global energy to maintain healthy and comfortable indoor environments for occupants, who spend over 90% of their time indoors [4]. Research demonstrates that occupancy detection can lead to energy savings of 39.4% for HVAC systems [5] and up to 75% for lighting systems [6]. Given these significant potential savings, an increasing number of buildings are being equipped with IoT sensors.

The global adoption of smart building technologies is expanding rapidly. By 2026, the number of buildings deploying these technologies is expected to reach 115 million, up from 45 million in 2022, representing over 150% growth [7]. This increase is driven by rising energy costs and the growing demand for energy efficiency among businesses and residents. Non-residential smart buildings will account for 90% of global smart building expenditures in 2026, maintaining a similar share to 2022. Consequently, it is reasonable to assume that most new buildings will be equipped with IoT sensors. However, while smart technology can optimize energy consumption, an often-overlooked issue is the waste of resources due to unused or underutilized spaces.

In the U.S., office buildings account for approximately 4 billion square feet of real estate [8]. At an average electricity cost of $2.10 per square foot, the total annual utility expense amounts to $8.4 billion. Research indicates that nearly 40% of corporate office space is paid for but remains vacant—unused conference rooms, empty desks, decorative lounge areas, and even entire unoccupied floors. This inefficiency results in an estimated $3.36 billion wasted on electricity to power, heat, and cool unoccupied spaces. This corresponds to approximately 32 billion kilowatt-hours of wasted energy and more than 22 million metric tons of CO_2 emissions annually.

In summary, buildings consume vast amounts of energy, making efficiency improvements crucial. While IoT devices enhance energy management by optimizing various systems, underutilized spaces remain a significant source of waste. Moreover, improving the utilization of existing resources can prevent unnecessary new developments, leading to long-term savings. Therefore, decisions regarding space utilization should be based on data rather than intuition to ensure efficient use of resources.

To optimize resource usage, decisions should be based on data rather than intuition. While intuition can provide valuable insights, relying solely on gut feelings can lead to inefficient use of resources. For instance, studies indicate that many decision-makers prioritize intuition even when presented with contradicting evidence [9]. However, data-driven approaches offer a more reliable foundation for optimization. A survey of over 1,000 senior executives conducted by PwC found that organizations leveraging data effectively are three times more likely to report significant improvements in decision-making compared to those relying less on data [10]. This underscores the necessity of using data to analyze space utilization rather than assumptions, ensuring that underutilized spaces are identified and addressed efficiently.

Occupancy detection systems have been employed to address this issue, with commercial solutions capable of monitoring and displaying space or desk utilization in real time. These systems are highly effective for buildings with clear functional purposes—such as offices, where people work, or meeting rooms, where discussions take place. However, in more complex, multi-functional buildings, room usage is not always straightforwardly linked to function.

AAU Innovate, established at Aalborg University in 2022 to foster innovation, collaboration, and entrepreneurship, provides a dynamic, multi-purpose environment. This study, undertaken in collaboration with AAU Innovate, explores resource usage within this unique setting. A key challenge in distinguishing it from traditional office spaces is that the actual activities taking place often cannot be determined simply by looking at a room's intended function.

The challenge of effectively managing resources in dynamic environments like AAU Innovate, where simple occupancy metrics fall short, highlights the need for advanced solutions such as digital twins. Applying this technology, however, brings forth fundamental questions critical to its successful implementation. Central among these is how a digital twin can move beyond mere presence detection to accurately map and analyze resource utilization based on specific activities and user interactions, which in turn requires clarity on the necessary data sources, analytical techniques, and data handling infrastructure. Equally important is understanding what specific capabilities define truly actionable insights in this context, ensuring the model provides value beyond basic reporting for operational decisions. Furthermore, significant considerations involve how such models can be designed for potential replication or adaptation across other campuses and similar facilities to address broader applicability and scalability needs. This links closely to the challenge of ensuring these potentially complex systems remain accessible and user-friendly, ultimately serving as viable platforms for future research endeavours and practical, long-term facility management.

The report is organized as follows: it begins with a detailed description of the project background in Chapter 1, including an overview of the current state of the art. This is followed by a clear definition of the project's motivation, scope, and final research question in Chapter 2. To address these research questions, IoT data from multiple systems are examined and analyzed in Chapter 3. The integration of these diverse systems, involving various strategies, is explored in Chapter 4, which establishes the technical foundation for the application. The design and development of the user interface are then presented in Chapter 5. To validate this interface, a test environment is established as described in Chapter 6. The report concludes with a comprehensive evaluation and reflection on the full system in Chapter 7.

Chapter 1

Background

The background of this study comprises two main parts. First, the state of the art in smart building technologies is presented in Section 1.1, focusing on research closely related to the topic at hand. Second, a detailed description of the AAU Innovate building and its structural characteristics is provided in Section 1.2. This understanding of the building's layout and usage patterns informs the assumptions made later in this project.

1.1 State of the Art

Recent advancements in smart building technologies have explored various approaches to energy optimization, occupancy detection, and digital twin implementations. These studies highlight the increasing significance of IoT, sensor networks, and machine learning in improving building management and efficiency. This section delves into three key research areas: energy optimization, occupancy detection systems, and WiFi-based indoor positioning. While each represents a broad and active field of inquiry, the discussion below focuses on the research most directly relevant to the scope of this work.

One notable contribution comes from K. Bäcklund, P. Lundqvist, and M. Molinari [11], which integrates multiple aspects of the questions asked in the introduction. It presents a novel approach to digital twin technology for higher educational buildings, emphasizing a human-centric design that caters to both building occupants and operators. The study showcases a digital twin developed for a Swedish university campus, integrating 3D scanning, geospatial data, and IoT sensors to create a real-like navigational indoor environment. Among the various components of the digital twin, the building analysis module is used to optimize space utilization, which optimizes space utilization by analyzing occupancy data and room booking systems to identify underutilized areas and "no-show" rooms. This module provides insights into how, when, and where spaces are used, enabling building managers to improve room allocation and reduce energy inefficiencies associated with underutilized spaces. Additionally, the digital twin includes features for real-time navigation, room booking, and building operations management. It addresses challenges in indoor environmental quality and operational efficiency, demonstrating the potential of digital twins to enhance building management and sustainability.

1.1.1 Digital Twins and IoT for Energy Optimization

The integration of digital twins with IoT systems has been widely studied to improve energy efficiency in buildings. Z. Ni, C. Zhang, M. Karlsson, and S. Gong [12] introduced the Deep Energy Twin, which combines ontology-based digital twins with deep learning to forecast energy consumption. Their case study in a historic Swedish building demonstrated improved prediction accuracy, with the Temporal Convolutional Network (TCN) model excelling in point forecasting.

Cost-effective approaches have also been developed, particularly for buildings without pre-existing sensor networks, which, as discussed in the introduction, can be the case in older buildings. S. Anik, X. Gao, N. Meng, P. Agee, and A. McCoy [13] proposed BDL, a system based on Raspberry Pi and modular sensors for indoor environmental monitoring at approximately $73 per building zone. This approach offers an affordable and scalable alternative to traditional sensor installations, making IoT-based energy optimization accessible to a wider range of buildings.

Furthermore, data silos in the built environment sector remain a significant barrier to effective energy analytics and building management. D. Hugo *et al.* [14] emphasize that data from BMS, IoT sensors, and metering systems are often stored in isolated, proprietary formats, creating interoperability challenges and limiting the potential for data re-use. To address this, the author introduced the Data Clearing House—a semantic platform designed to integrate heterogeneous data from BMS and IoT sensors. This platform facilitates seamless data exchange and analysis, enabling more efficient energy management strategies. However, challenges such as data health issues (e.g., equipment faults, incorrect timestamps, and mapping discrepancies) and the need for ongoing semantic model maintenance highlight the complexity of achieving scalable, interoperable solutions in this domain.

1.1.2 Occupancy Detection Systems

Accurate occupancy detection is a crucial component of energy optimization and space management. Various studies have explored sensor-based and machine learning-driven methods to achieve high detection accuracy.

For instance, A. Floris, S. Porcu, R. Girau, and L. Atzori [15] developed an IoT-based system utilizing low-cost hardware, such as Raspberry Pi devices with cameras and environmental sensors. Their lightweight neural network achieved 99.5% classification accuracy for room occupancy detection and accurately estimated Total Volatile Organic Compounds (TVOC) levels via regression analysis.

A comprehensive review by Z. Chen, C. Jiang, and L. Xie [16] analyzed different sensor technologies—including Passive Infra-Red (PIR), CO_2, cameras, and Wi-Fi—and highlighted the effectiveness of sensor fusion and machine learning techniques, such as decision trees and neural networks, in enhancing real-time detection accuracy.

Another study, S. S. Kumar, R. Chandra, and S. Agarwal [17], introduced a rule-based system that employs decision tree algorithms with predefined thresholds for parameters like light intensity and CO_2 levels, achieving precise occupancy classification. Additionally, K. H. Andersen *et al.* [18] utilized supervised learning with indoor environmental quality sensors and Extreme Gradient Boosting (XGBoost) models, demonstrating that both global and room-specific models can effectively capture occupancy patterns, particularly in residential settings.

Recent research goes even further by attempting to estimate human activities in smart homes. Addressing challenges like data variability and sparsity, studies propose the use of digital twins, such as the VirtualSmartHome simulator, to generate realistic synthetic datasets. These datasets model daily activities, enabling human activity recognition (HAR) algorithms to achieve robust performance, with an average F1 score of 80% on real-world data. This approach demonstrates significant potential for improving HAR in smart home environments.

1.1.3 Wi-fi-based indoor Positioning and Occupancy Mapping

Wi-Fi-based systems offer a scalable and cost-effective solution for occupancy mapping and indoor positioning. I. P. Mohottige, H. H. Gharakheili, T. Moors, and V. Sivaraman [19] used Wi-Fi session logs and machine learning to estimate classroom occupancy, achieving a 13.10% symmetric mean absolute percentage error (sMAPE) accuracy. Meanwhile, V. Bellavista-Parent, J. Torres-Sospedra, and A. Perez-Navarro [20] reviewed Wi-Fi-based indoor positioning systems, noting that artificial neural networks (ANNs) dominate the field, with Deep Neural Networks (DNNs) achieving a mean error of 0.11 meters using mmWave signals. Challenges like signal interference persist, but advancements in ML and signal parameters like Channel State Information (CSI) and Signal-to-Noise Ratio (SNR) show promise.

Additionally, Y. Wang and L. Shao [21] analyzed Wi-Fi-based positioning in a university library, identifying four occupancy patterns over 30 days. The study found most users were short-term visitors, suggesting adjustments to opening hours could improve efficiency. This highlights Wi-Fi-based systems' ability to provide granular occupancy data, aiding in optimizing space utilization and energy efficiency.

1.2 Organizational Structure of AAU Innovate

AAU Innovate serves as a central hub for fostering innovation, entrepreneurship, and the commercialization of research at the university. The building accommodates two primary entities:

- **AAU Innovation** The core of the facility, dedicated to cultivating innovation and entrepreneurial activities.
- **Institute for Advanced Study in PBL (IAS PBL)** An independent institute primarily requiring office space, currently occupying a significant portion of the second floor.

Within AAU Innovation, three key teams drive its operations:

1. **Team Event & Support (E&S)** Responsible for the daily management and operations of the building, ensuring smooth functionality and an optimal working environment.
2. **Team Student Entrepreneurship (SE)** Focuses on student-driven innovation and business creation. Their flagship initiative, the Startup Program, provides AAU students with support and resources to launch their own businesses.
3.

Technology Transfer Office Plays a critical role in securing research funding and commercializing university research across various departments.

Beyond these core teams, AAU Innovate fosters a vibrant ecosystem by hosting a community of resident startups. These residents, typically students who have progressed through the Startup Program and successfully applied for dedicated "garage" spaces (individual offices within AAU Innovate), are deeply integrated into the building's innovation culture despite not being formal employees. Their consistent presence and engagement contribute significantly to the overall dynamic.

Further contextual information, including an exterior view of the building and detailed floor plans, can be found in Appendix A.

On a typical semester week, AAU Innovate experiences a significant flow of visitors, ranging from 2000 to 3000 individuals. These visits predominantly occur during the standard work week and daytime hours, although activity can extend beyond conventional closing times (e.g., 4:00 PM) due to student engagements and scheduled events.

The building's utility is multifaceted, encompassing a wide array of spaces and functions. It features seven large meeting or event spaces, which regularly host diverse events such as conferences, celebrations, PhD defenses, keynote speeches, community meetups, social gatherings, and numerous internal workshops. These events vary significantly in scale, accommodating anywhere from ten to over 300 attendees. Additionally, the building includes several smaller meeting rooms primarily used for internal collaborations and interdepartmental meetings within AAU.

Furthermore, AAU Innovate houses six specialized laboratories: a design lab, wood lab, metal lab, play lab, audio-video lab, and an IT lab, alongside a dedicated 3D printing room. The building also incorporates various open collaborative spaces, notably the AAU startup flex zone, a shared workspace specifically designed for early-stage startups participating in the Startup Program. Beyond departmental and employee offices, dedicated workspace is also provided to Entrepreneurs in Residence and other specific programs or startups, such as Code Lab, which currently hosts three software startups. The building also features a canteen, which attracts a significant number of people from neighbouring buildings for lunch daily.

Chapter 2

Project Scope and Motivation

In Section 1.1, the state of the art is subdivided into areas such as energy savings, occupancy detection, and Wi-Fi-based indoor positioning. As mentioned in the introduction, empty or underused spaces in commercial buildings represent a significant waste of resources, affecting energy consumption, construction investments, and personnel allocation. To optimize resource utilization, building management must make decisions that are tailored to the building's needs and are based on data rather than intuition—a practice that research has shown leads to markedly better outcomes [10].

This project has been developed in collaboration with AAU Innovate. With the building now just over two years in operation, the first comprehensive reports and insights regarding its usage after the initial startup phase have begun to emerge. It is, therefore, an opportune moment to evaluate the current usage and steer it towards the intended goals.

A major challenge in this context is the lack of objective insights into how spaces are used. Take, for example, the usage of garages at AAU Innovate. As noted in Section 1.2, startup teams have the opportunity to utilize garages within the building. However, there are only eight garages available, in contrast to the 147 startups enrolled in the startup program in 2024 [22]. This disparity makes it crucial to ensure that the garages are allocated to those startups that truly need them and that their usage is effectively monitored. At present, the evaluation of usage relies on what will be referred to as **anecdotical evidence** — essentially, individual observations of activity. Such an approach runs the risk of misrepresenting actual usage; for instance, employees may primarily work during standard weekday hours (8:00 to 16:00), while startups, constrained by academic schedules, might only gather in the garages after 16:00, leading to the false assumption that these spaces are unused.

2.1 Conceptual Model for Space Utilization

To overcome the shortcomings of subjective observations, there is a clear need for an objective *observer* that can supply accurate data to decision-makers. In cooperation with several members of team E&S, the following conceptual model was developed (see Figure 2.1). This model is designed to answer the question: **"What did AAU Innovate facilitate?"**

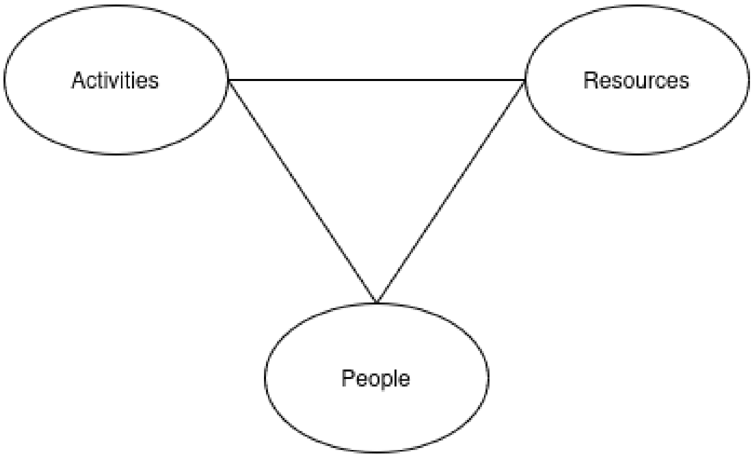

Figure 2.1: Conceptual model for space utilization

The conceptual model is structured around three primary components: activities, resources, and people. These elements form the basis for analyzing space utilization within AAU Innovate. A more detailed discussion of each component is presented in sections 2.1.1, 2.1.2, and 2.1.3.

2.1.1 Activities

Activities refer to the actions or tasks carried out by individuals within the building. These activities have been categorized as follows:

- **Events**, which include:
  - Conferences
  - Celebrations
  - PhD defenses
  - Keynotes
  - Community meetups
  - Social events
- **Workshops**
- **Desk work**
- **Dining**

2.1.2 Resources

Resources encompass the physical and non-physical assets that individuals utilize to conduct their activities efficiently. These resources are classified into the following categories:

- **Rooms**, which include:
  - Meeting rooms
  - Event rooms (e.g., auditoriums, pitch rooms, conference rooms)
  - Labs

    - Offices
    - Garages
    - Shared spaces
- **Equipment**
- **Competences**, referring to human expertise and time availability

A distinction is made between meeting rooms and event rooms based on their intended use and the facilities available. While meeting rooms are typically designed for smaller gatherings (e.g., 2–4 people or 10-person meetings), event rooms cater to larger audiences and are equipped accordingly. Although equipment and competencies could be further detailed, this level of granularity is not necessary for the current scope of this report.

2.1.3 People

Given General Data Protection Regulation (GDPR) regulations and broader privacy concerns, individuals are not represented as unique entities within the model. Instead, people are categorized based on their affiliations, ensuring privacy while still enabling meaningful data-driven insights. The classification is as follows:

- **Building residents**
    - Staff (subdivided by team/department)
    - Entrepreneurs (subdivided by their startup)
- **AAU community**
    - Staff (subdivided by department)
    - Students (subdivided by department)
- **External visitors**
    - Categorized based on their company or domain of affiliation

This approach ensures that decision-making processes focus on broader usage patterns rather than individual behaviours, thereby minimizing privacy risks while still providing valuable insights into space utilization.

2.2 Problem Statement

Modern IoT devices continuously gather data without significant bias or downtime, providing a solid foundation for making data-driven decisions at AAU Innovate. Given that the building is equipped with a wide array of sensors to control various systems, a critical question arises:

> **Can already deployed systems and sensors in smart buildings be utilized to create an accurate and detailed picture of building usage?**

This project explores this question as a case study in collaboration with AAU Innovate, seeking to determine whether the current sensor infrastructure can deliver the precise and comprehensive insights necessary for effective building management.

Chapter 3

Overview of Data Systems

This chapter presents a detailed examination of the various data-generating systems operating within the AAU Innovate building. A thorough understanding of these systems, their data outputs, and associated access methods is fundamental for constructing a comprehensive digital twin and enabling nuanced analyses of building occupancy and utilization. The section is structured to provide a clear and consistent overview of each relevant data source.

Each system will be described using a uniform structure to facilitate comparison and ensure clarity. This structure begins with a General Overview outlining the system's primary purpose and the types of data it collects. Following this, the Potential Insights and Assumptions subsection explores how the data from this system can be interpreted within the context of AAU Innovate. This part will involve creating context through the use of dummy assumptions regarding the system's role and data characteristics, followed by a critical reflection on these assumptions to better understand the practical value and limitations of the data for analyzing building usage. Finally, the Data Collection and Processing subsection details how the data can be potentially integrated and accessed, including methods for retrieval and potential data flows.

The following subsections will explore the key data sources that are either currently available or hold potential for future integration. These include the FMS, which manages the spatial layout of all buildings; the Room Calendar System (Section 3.2), responsible for handling reservations of bookable spaces; and the Building Management System (Section 3.3), which oversees environmental controls and sensor networks. Additional insights may be obtained from coffee machine usage logs (Section 3.4), which can offer indirect indicators of occupant activity patterns, and the Door Counter System (Section 3.5), which tracks pedestrian flow at main entrances. The Door Access System (Section 3.6) records entries through secured doors, while the Lighting System (Section 3.7) manages facility-wide illumination —though detailed data for this system is currently unavailable. Lastly, the WiFi System (Section 3.8) provides wireless connectivity and may also serve as a source of location-based data.

3.1 Facility Management System

An Facility Management System (FMS) is a system used to store the layout of all buildings within a given organization. It is typically organized hierarchically, starting from areas, which are divided into buildings, further subdivided into floors, and ultimately into individual rooms. The system contains detailed information for each room, and it is common for it to include floor plans for each building and floor.

AAU has such a system in place. Each campus is divided into buildings, which contain multiple floors, each with individual rooms. For each room, data such as room number, group, area, and perimeter are provided. The *group* field reflects the intended use or function of the room.

3.1.1 Potential Insights and Data Assumptions

The data within such a system is generally static or changes only infrequently, as modifications to physical building structures are typically time-consuming and resource-intensive. However, the usefulness of the data increases significantly if the following assumptions hold true:

1. Changes made to buildings are also reflected in the FMS system.
2. The room group, or function, aligns well with the actual use of the room.

These assumptions were verified, leading to the following discussions and conclusions about the data value:

- **Assumption 1:** This assumption appears valid. Building modifications are generally large-scale and well-planned operations. As a result, such changes are systematically updated in the FMS, and floor plans are revised when needed. However, it is worth noting that the associated Revit model is not updated during these changes.
- **Assumption 2:** For the Innovate building, this assumption holds in almost all cases. Only a single instance was found where the actual use of a room differed from its designated function—it was used as a meeting room rather than an office. Information about other, particularly older, buildings is not directly available but is assumed to be relatively accurate. Room functions tend not to change frequently, and when they do, modifications often require structural changes, which would typically trigger updates by the same team responsible for maintaining the FMS database.

3.1.2 Data Collection and Processing

AAU is currently in the process of transitioning to a new system, which will allow for API integration. This feature is not yet available in the current system.

For the Innovate building, the author received data via email. This included an Excel spreadsheet containing detailed room data, PDF floor plans for each floor, and a 3D Revit model. These sources collectively provide a relatively static dataset. However, caution should be exercised, as updates made in the live system are not automatically reflected in this data export. While such updates are expected to be infrequent, due to the slow nature of building changes, it is recommended to periodically validate the data to ensure continued accuracy.

3.2 Room Calendar System

Room calendar systems are increasingly vital in commercial buildings featuring bookable meeting rooms, a trend noted by [23]. AAU Innovate utilizes such a system.

Microsoft 365 facilitates this through *room mailboxes*, which function similarly to standard Outlook accounts but are designated as resources [24]. This setup allows meeting rooms to be invited as resources within calendar events, automatically logging the booking in the respective room's calendar.

This system empowers users to check room availability and reserve spaces efficiently. Furthermore, door-mounted screens outside meeting rooms often display upcoming reservations and permit *Ad Hoc* bookings—instant reservations made directly on the system without needing to specify participants beforehand. Specifically, AAU Innovate employs LiveConnect from MediaConnect for functions including room booking and visualizations on these screens. They also utilize AskCody, another meeting room visualization tool. Regardless of the front-end tool, all underlying booking data resides within the AAU Exchange system. While this describes the setup at AAU Innovate, similar resource calendars and visual display systems are standard practice in shared office environments and conference centers.

At AAU Innovate, room calendars are configured for various spaces, such as meeting rooms, event areas, offices, and garages. However, this study disregards the office and garage calendars, as they are currently unused and serve no practical function for analyzing room utilization.

3.2.1 Booking Procedures

The method for booking rooms at AAU Innovate varies by room size. Smaller and medium-sized rooms are typically booked using the standard Outlook *Room Finder* function, where the room's email address is included as a resource invitee in the meeting request.

However, AAU Innovate employs two dedicated staff members to manage bookings for larger meeting rooms (those with a capacity exceeding 12 people). To reserve these spaces, users must submit a detailed request via a Microsoft Form. This form submission triggers a Power Automate flow, which sends an email containing the request details to the designated staff. These employees then manually create the booking in the room calendar, pasting the information from the form into the body of the calendar event.

3.2.2 Potential Insights and Data Assumptions

Room calendar data, stored within Microsoft Outlook, offers valuable insights into room usage patterns, the purpose of meetings, and attendee information. This data aligns with the key aspects of the theoretical model (Figure 2.1), provided the following assumptions hold true:

1. Every use of a bookable space results in a corresponding calendar booking.
2. All individuals attending the meeting (whether physically present or joining remotely) are formally invited via the calendar event.
3. Meeting start and end times recorded in the calendar are accurate and promptly updated to reflect any changes, such as delays, extensions, or cancellations.

These assumptions were verified, leading to the following discussions and conclusions about the data value:

- **Assumption 1:** Observations and interviews indicate that most meetings are indeed booked. Instances where individuals briefly use an empty room without booking it ('just jumping in') are rare. More commonly, users create an Ad Hoc booking directly at the room screen, primarily to avoid interruptions if someone else checks the calendar and sees the room as available. The prevalence of Ad Hoc meetings is also reflected in the booking data.

- **Assumption 2:** Analysis of the data shows numerous bookings with only a single participant listed. While single-person use is possible, it may not align with the intended purpose of a meeting room. This pattern likely occurs when individuals book rooms on behalf of others or invite attendees through alternative channels (e.g., separate emails or different calendar systems) instead of directly via the room booking event. It may also happen for meetings involving external partners not within the AAU system.

- **Assumption 3:** Booked meetings are generally reliable when it comes to reflecting cancellations. However, the scheduled start and end times are less precise. Users sometimes reserve longer durations than strictly necessary to allow for setup or guest arrival buffers. Additionally, meeting overruns are rarely updated in the calendar, with participants typically checking the room's calendar directly to see if it's available for the additional time needed.

3.2.3 Data Collection and Processing

The room booking data is accessible via the Microsoft Graph REST API, specifically using the *events* endpoint [25]. This API provides structured access to room booking details, facilitating seamless integration into the digital twin model. Using this API, it's possible to retrieve events from all relevant room calendars. Since calendar events list invited participants, the application accessing the API can determine the number of attendees and their affiliations, leveraging the personnel information available within the AAU Outlook directory (e.g., department data).

For the large rooms booked via the form process, the detailed information submitted through the form can be accessed within the body text of the corresponding calendar event. This is necessary because inviting potentially hundreds of participants (e.g., for a large event) directly to the calendar entry is impractical; attendees typically receive notifications through other means.

This relatively straightforward data access setup is visualized in Figure 3.1.

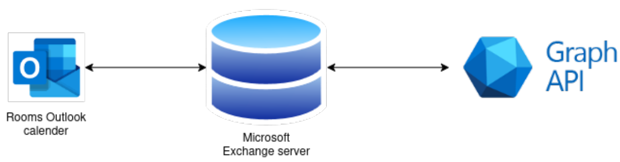

Figure 3.1: A model representing the Outlook room calendar data access

3.3 Building Management System

At Aalborg University, all modern buildings — including AAU Innovate — are managed using the Schneider Electric EcoStruxure Building Operation system for comprehensive building management. This integrated Building Management System

(BMS) platform serves as the central hub for monitoring, controlling, and optimizing key building systems such as heating, ventilation, and air-conditioning (HVAC), lighting, power, and security [26]. By unifying these systems under a single interface, building managers can improve energy efficiency, enhance occupant comfort, and streamline facility operations for smarter, more sustainable buildings. It should be noted that lighting at AAU is managed through a separate system, as detailed in Section 3.7.

The EcoStruxure Building Operation system provides an intuitive interface, typically starting with a campus overview where users can select specific buildings equipped with the system. This report focuses on the AAU Innovate buildings. After selecting a building, users are presented with an overview of various systems, including ventilation, heating, hot water, outside lighting, and sun covers. Each system can be explored individually, or users can switch to a floorplan view to navigate the building spatially. Within the floorplan, individual rooms can be selected to view near real-time data on system and sensor statuses. This detailed room view displays information such as window status (open/closed), heating output percentage, ventilation openness, CO_2 levels, current temperature, acceptable temperature ranges, and the temperature of incoming air. Each room is also equipped with a Passive Infra-Red (PIR) sensor to detect occupancy. A visual example of this room overview interface is shown in Figure 3.2.

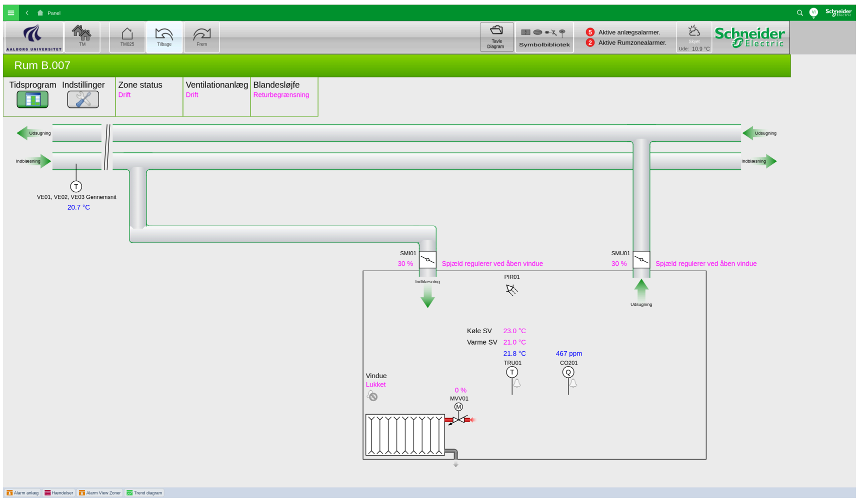

Figure 3.2: A screenshot of the room overview interface of the Schneider EcoStruxure Building Operation software.

In each room, a wall-mounted physical device houses the various sensors and provides a local user interface for adjusting the desired temperature. Open areas within the buildings are similarly equipped with sensors but typically lack the user interface screen.

3.3.1 Potential Insights and Data Assumptions

The data collected by this system holds significant potential for understanding room occupancy and estimating the number of people present. By analyzing the data from the PIR sensor in conjunction with CO2 and temperature measurements and the ventilation state, valuable insights into room usage patterns can be derived. However, the accuracy and reliability of these insights depend on the following key assumptions:

1. The PIR sensor is optimally positioned to detect all movement within the room.
2. Individuals present in a room exhibit sufficient movement to be detected by the PIR sensor.
3. Doors between rooms remain closed unless someone is actively passing through.

These assumptions were verified, leading to the following discussions and conclusions about the data value:

- **Assumption 1:** During testing, the PIR sensor consistently detected movement within its field of view. Typically, this sensor is positioned on the wall next to a door, approximately 1.5 meters above the floor. While this placement generally proves effective in simple, rectangular rooms, the performance of the PIR sensor can be affected by both the layout of the room and the sensor's specific placement. For instance, office *C.108* on the first floor presents a potentially problematic room configuration (refer to Appendix A for the detailed building layout). Consequently, these factors concerning room layout and sensor placement should be taken into account when analyzing the data collected by the PIR sensor.
- **Assumption 2:** A known limitation of PIR sensors is their inability to detect stationary individuals. To mitigate this, a sufficiently long data aggregation interval can be used. For instance, if the interval is set such that even individuals who move every 10 to 15 minutes are counted as 'continuously present', this issue can be partially addressed. However, this approach introduces a trade-off between data granularity and accuracy.
- **Assumption 3:** At AAU Innovate, it is generally observed that doors remain closed, except for a few specific rooms. Furthermore, the doors are equipped with auto-closing mechanisms. To ensure doors remain open when needed, a common practice is to use a physical object to prop them open. This practice aligns with the requirements of the access control system, further reinforcing the likelihood of doors being closed.

3.3.2 Data Collection and Processing

As explained in S. P. Melgaard *et al.* [27], a separate BMS database was established to enable convenient access to system data. Data from the various sensors deployed throughout the building is first collected by local terminals (Automation Servers) within the BMS. From there, the data is forwarded in chunks, rather than at fixed intervals, to a centralized PostgreSQL database, which uses TimescaleDB, a time-series optimized extension for PostgreSQL.

TimescaleDB is an open-source time-series database built on top of PostgreSQL, designed to efficiently store and query large volumes of time-stamped data. It enhances PostgreSQL with features such as automatic data partitioning, continuous aggregation, and optimized time-series queries, making it particularly well-suited for handling sensor data from building management systems.

The transfer of sensor data from the Automation Servers to the AAU BMS database is event-driven, triggered when a certain amount of data has been collected rather than on a strict schedule. Due to the large number of sensors and the volume of data, it typically takes between 10–15 minutes for data from all logged sensors to become available in the database.

In addition to timeseries data, metadata — including sensor locations, types, and other descriptive information — is also managed through the Schneider BMS platform. Whenever a new sensor or data point is added to the Schneider system, the corresponding metadata is automatically transferred into a dedicated METADATA table within the AAU BMS database.

The system provides access via two separate API endpoints:

- Metadata Endpoint: Returns relevant information about each sensor, including the externalID, source path, and unit. This endpoint does not require any input arguments.
- Trend Data Endpoint: Accepts an externalID, a start time, and an end time as input, and returns a list containing the externalID, timestamp, and recorded value for the specified sensor and period.

An overview of the system setup is shown in Figure 3.3.

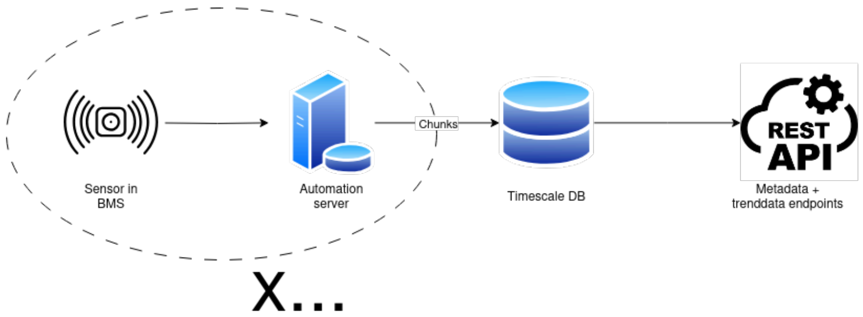

Figure 3.3: An overview of the setup of the BMS data capture system.

3.4 Coffee Data

AAU Innovate, like many other workspaces, provides coffee machines for its residents. Specifically, there are four Wittenborg 9100 ES 2FB machines located throughout the building: one in the kitchen, one in the first-floor hallway, and one each in the hallways on the second and third floors. These machines are capable of exporting usage data either via a USB key or over a serial connection [28]. This data logs every beverage dispensed, effectively maintaining a usage record.

It is important to note that non-residents cannot access the first floor and above, meaning only residents have access to these machines, as mentioned in Section 1.2. Consequently, the data gathered provides, at least in theory, insights into resident behaviour.

3.4.1 Potential Insights and Assumptions

While coffee machine usage data may not seem significant at first glance, it has potential applications. For example, it could offer insights into floor usage patterns or assist in automating maintenance and cleaning schedules. The information would provide the most valuable insights if the following assumptions hold:

1. Individuals obtain hot beverages from the machine closest to their current workspace.
2. Each person consumes a consistent number of beverages daily.
3. Beverage consumption is distributed evenly throughout the working day.
4. Only residents use the coffee machines.

If these assumptions hold, the data can provide a broad and generalized representation of resident movement within the building, making it a valuable dataset. However, these assumptions were verified through discussions with residents and direct observation.

- **Assumption 1:** This assumption generally holds. An exception was observed where individuals working on the ground floor sometimes move up a floor to get their drinks.
- **Assumption 2:** This assumption does not hold well. Significant individual variations, daily fluctuations, and potential seasonal effects influence the data.
- **Assumption 3:** This assumption is inaccurate. Consumption patterns indicate that some periods of the day, particularly in the afternoon, may not be well represented by this data.
- **Assumption 4:** This assumption holds fairly well. Only employees and students in the startup program have access to floors 1 and above. While guests can also use the machines, their contribution is minimal compared to residents.

Overall, while coffee machine data can reflect facility usage by residents and capture usage variations, interpreting it requires caution. Due to the impracticality of frequent data collection and the strong potential for variation, a one-month collection period was chosen to balance out individual fluctuations and provide a general activity overview. Despite this, data interpretation should be done carefully, considering the limitations highlighted by assumption validation.

3.4.2 Data Collection and Processing

The coffee machines store usage data locally, and the most practical method for extracting it is through a USB key. Unlike an automated system, this manual approach limits data collection to weekly or monthly intervals, as more frequent extraction is impractical.

To streamline processing, a computer program has been developed. Once the data is transferred from the USB key, the program automatically uploads it to a database and generates a CSV file for backup. The CSV format ensures easy access for non-technical personnel. This workflow is illustrated in Figure 3.4.

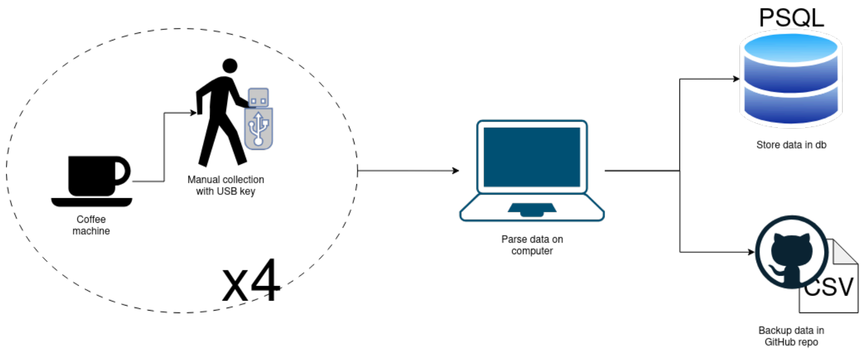

Figure 3.4: A model representing the coffee data extraction process

It is essential to highlight that the automated data extraction, utilizing the USB key and processing software, was developed as part of my student job. Therefore, it should not be considered an accomplishment of this project but rather a pre-existing system.

3.5 Door Counter System

To monitor building occupancy, AAU Innovate employs a people-counting system installed at the primary entrances. Specifically, counters are located at entrances A, B, and C, whose positions are detailed in the ground floor plan (Figure A.3).

The system utilizes the MultiTeknik People Counter. While this counter is generally marketed towards retail businesses for tracking customer traffic, typically uploading data via USB to a local computer Multiteknik [29], the implementation at AAU Innovate deviates from this standard. Instead of a USB connection, the AAU Innovate setup transmits data over an unspecified protocol to a Thingsboard server managed by Multiteknik. ThingsBoard itself is an open-source IoT platform designed for collecting, processing, visualizing, and managing data from connected devices, supporting the development of scalable IoT applications. A conceptual diagram of the basic people counter hardware setup, adapted from Multiteknik [29], is shown in Figure 3.5.

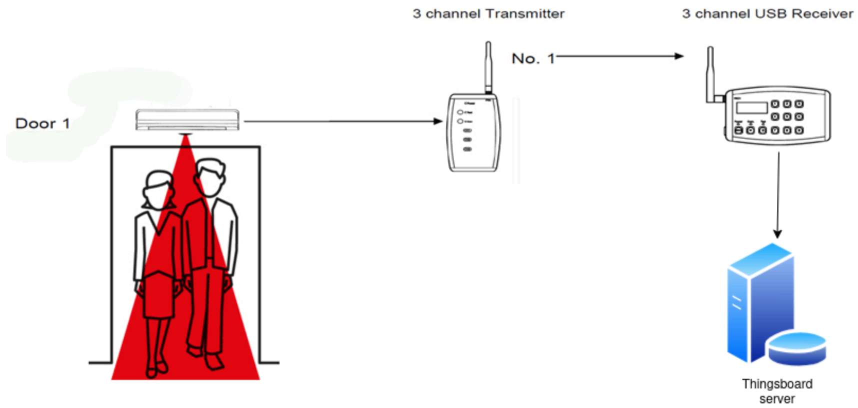

Figure 3.5: A model representing the people counter setup, inspired by Multiteknik [29]

Data captured by the sensors is stored and visualized on the Multiteknik-managed Thingsboard server. The underlying mechanism involves an infrared (IR) beam sensor positioned approximately one meter high just inside each main door, with a corresponding reflector placed opposite. Each interruption of this IR beam registers as a single "passage". Periodically (specified as hourly), each of the three sensor units (transmitters) sends a value representing the total number of detected passages divided by two to a central receiver. This receiver assigns a unique label (c1, c2, or c3) corresponding to each sensor and forwards this labelled data to the Thingsboard server.

3.5.1 Potential Insights and Data Assumptions

The passage data collected by the sensor system offers valuable insights into the building's usage patterns, particularly the number of people entering and exiting each day. However, the validity of these insights depends on several key assumptions:

1. Visitors' entry and exit occur within the same hourly measurement window.
2. Individuals pass through the sensor detection zone one at a time.
3. All entries and exits occur solely through the three monitored main doors (A, B, C).

These assumptions were verified, leading to the following discussions and conclusions about the data value:

- **Assumption 1:** This assumption is often inaccurate. Visitors frequently stay for multiple hours, and their movements rarely align with hourly reporting intervals. The sensors also lack directionality and cannot distinguish between entries and exits. As a result, a reported value (e.g., 40 passages) may reflect any number of entry/exit combinations. The system attempts to estimate this by halving the count (passages/2), but this method introduces ambiguity and limits its accuracy on an hourly basis. A more reliable approach is to analyze data on a daily level, assuming most individuals leave before the following morning (e.g., using a 04:00–03:59 'day' definition).
- **Assumption 2:** This assumption generally holds. Narrow doorways and social norms around personal space usually ensure that people pass through one at a time. Occasional inaccuracies may arise when individuals pass closely together, but these occurrences are considered infrequent and not a significant source of error.
- **Assumption 3:** This assumption is known to be incomplete. Several alternative routes exist, including a fourth door to a terrace and emergency exits connected to four staircases (Appendix A). The terrace door is often used during working hours (e.g., 08:00–15:30), and emergency exits sometimes serve as shortcuts to parking areas. Movements through these unmonitored routes are not captured by the system and introduce a measurable error in estimating total building occupancy. While the main doors account for the majority of traffic, these alternative paths should be considered when interpreting sensor data.

3.5.2 Data Collection and Processing

The data stored on the Thingsboard server can be accessed programmatically via its built-in REST API. The standard procedure involves:

1. Authenticating using valid credentials (username and password) to obtain an access token.

2. Using the obtained token in subsequent API requests to fetch the desired telemetry data (i.e., the hourly passage counts labelled c1, c2, and c3).

This data retrieval process, forming part of the overall system implementation at AAU Innovate, is depicted conceptually in Figure 3.6.

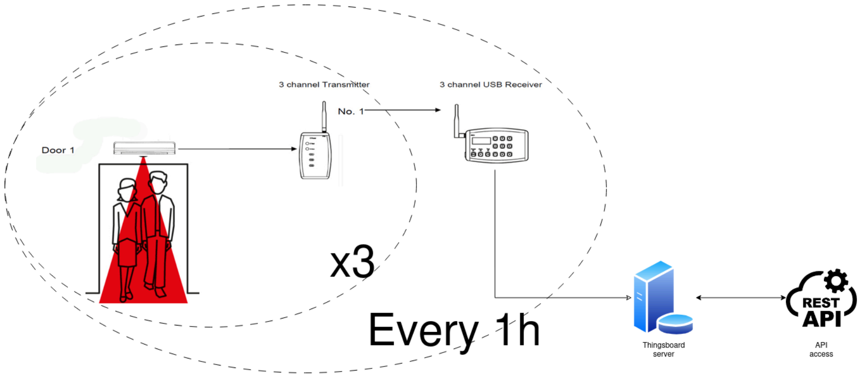

Figure 3.6: A model representing the data collection setup for the people counter system at AAU Innovate, utilizing the Thingsboard API.

3.6 Door Access System

Aalborg University utilizes a digital door access system. Scanners are installed at designated doors. Furthermore, every student, employee, and guest (via guest cards) possesses a personal card equipped with an NFC chip. This chip contains the card number. To gain entry, an individual scans their card by holding it in close proximity to the scanner. The scanner subsequently communicates with a central server, signalled by a flickering orange indicator light, which verifies access permissions defined for each door or door group. If access is granted, the indicator light turns green. Conversely, if entry is denied, the light turns red. These visual indicators are accompanied by corresponding audible signals. Successful authentication potentially results in the door unlocking.

3.6.1 Potential Insights and Assumptions

Given that each card is personal, the logged data could offer valuable insights into the 'people' aspect (see Section 2.1) concerning individuals within the building. The value of this information is contingent upon the validity of the following assumptions:

1. All doors consistently require a card scan prior to opening.

2. Every individual always scans their card before entry, including when entering immediately after another person.

3. Comprehensive logs are maintained for every card-scanner interaction.

4. Each scanner possesses a unique ID and an associated position description, ensuring clarity regarding its placement within the building.

These assumptions were verified, leading to the following discussions and conclusions about the data value:

- **Assumption 1:** This assumption does not hold universally. Main entry doors are typically open by default during normal office hours (8:00–15:30). Certain doors can also be set to a 'kept open' state, where scanning a card unlocks the door and a second scan relocks it. This functionality is inactive outside of normal office hours.
- **Assumption 2:** This assumption does not always hold. Individuals may hold doors open for others or allow entry without an additional scan. The main access gates to the first floor remain open for approximately 5–10 seconds, making it relatively common for people to enter immediately after someone else. Circumventing the gate by jumping over it is reportedly rare.
- **Assumption 3:** This assumption holds. When equipment theft occurs at AAU, access logs dating back up to six months can be requested for investigation.
- **Assumption 4:** This assumption likely holds. Each scanner must possess a unique ID to enable permission control for specific doors or rooms. It is presumed that scanners are organized into logical groups. Whether detailed location descriptions are associated with scanner IDs is unknown to the author, but it is probable to support maintenance and repair jobs.

3.6.2 Data Collection and Processing

This access data is managed by AAU Campus Security; currently, the author has been unable to obtain access. Access would likely first require establishing an official collaboration agreement with AAU Innovate and other relevant parties. Such an agreement could potentially secure the necessary resources for this project. Pending data access, a hypothetical schematic representing the author's assumptions about the system's operation is provided in Figure 3.7. The specific methods for potential (automated) log file access remain speculative and are likely system-specific.

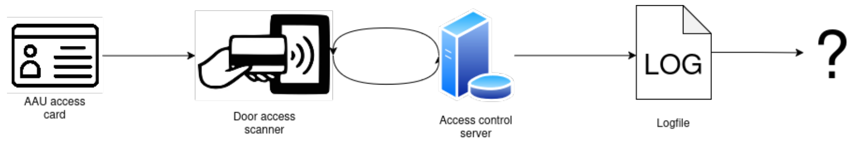

Figure 3.7: A model representing the believed setup of the card reader system.

3.7 Lighting System

The lighting system at AAU Innovate is based on the *LITECOM* platform from Zumtobel. As one of Zumtobel's European test buildings, Innovate often runs beta versions of the system software. Currently, the system interface lacks user management and is protected only through IP restrictions. The full installation comprises 16 individual controllers, each with its own interface, as the size of the building exceeds the capacity of a single controller. All controllers and components communicate using the Digital Addressable Lighting Interface (DALI) protocol (see Section 3.7.1).

While Zumtobel has started integrating DALI-2 devices, the sensors installed at Innovate use the original DALI version 1. This decision was reportedly made because version 1 sensors were more stable and reliable at the time of deployment.

The controllers used are compatible with both DALI and DALI-2, ensuring system flexibility.

3.7.1 Digital Addressable Lighting Interface

DALI, which stands for Digital Addressable Lighting Interface, is a standardized digital communication protocol used for controlling lighting systems [30]. It enables two-way communication between control devices (like sensors and switches) and lighting components (like LED drivers), allowing precise and flexible control of lighting installations.

Originally defined in IEC60929 and later expanded into DALI-2 (IEC62386), the protocol supports addressability, meaning each device on the network has its own unique address. This allows advanced features such as individual control, grouping, scene-setting, and feedback. Unlike analog systems like 1–10V, DALI provides consistent and stable dimming and ensures interoperability between devices from different manufacturers, especially under the DALI-2 standard, which requires certification.

DALI uses a simple two-wire bus that supports various wiring topologies (line, tree, star) and does not require shielding. Devices with terminals marked 'DA' should be connected to the DALI bus using these terminals. While DALI-2 introduces many improvements, both DALI and DALI-2 devices can typically coexist in the same installation, provided compatibility is managed.

An example of all the devices in a typical DALI system can be seen in Figure 3.8.

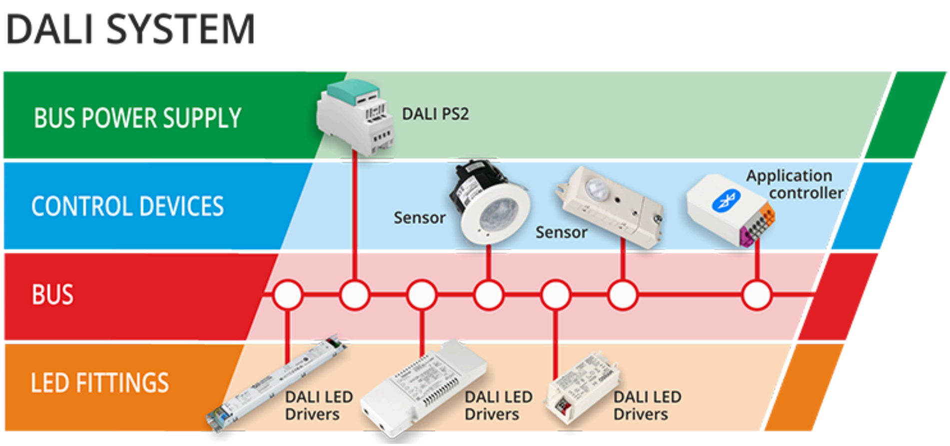

Figure 3.8: An overview of a DALI system [30]

3.7.2 Potential Insights and Assumptions

The data collected by the lighting system at AAU Innovate offers significant potential for deriving insights into room occupancy and estimating the number of people present. By analyzing information from the PIR sensor, usage patterns can be identified and modeled. However, the accuracy and reliability of these insights rely on two key assumptions:

1. The PIR sensor is optimally positioned to detect all relevant movement within the room.
2. Occupants move frequently enough for the PIR sensor to register their presence.

These assumptions were verified, leading to the following discussions and conclusions about the data value:

- **Assumption 1:** This assumption is considered highly reliable. The PIR sensors in the building are used to control the lighting. When someone enters a room, the light typically turns on. During the early days after opening, the building experienced issues with light instability, which were later traced back to a software bug. Since then, the lights have consistently activated upon entry, indicating that the sensors are well-placed and capable of detecting entry reliably.
- **Assumption 2:** It is well known that PIR sensors have difficulty detecting stationary individuals. To mitigate this, the lighting system is configured to reset a 30-minute countdown timer every time movement is detected. If the timer reaches zero, the lights turn off. According to the author's knowledge, no occupants have experienced the lights turning off while they were still in the room after this configuration change. This supports the claim that the system can reliably confirm occupancy with a time resolution of at least 30 minutes.

3.7.3 Data Collection and Processing

Although not investigated in detail for this project, the *LITECOM* system provides both a REST API and an MQTT interface, as documented in [31]. These interfaces allow polling and subscribing to data from the lighting system, enabling advanced integrations and data-driven control.

The REST API offers access to a range of data and functions. Users can poll information about all effective ranges (such as rooms or groups), retrieve details for specific zones, and access device-level information. In addition, the API allows execution of control functions such as recalling scenes, adjusting lighting parameters (intensity, color temperature, and color), and controlling connected motors like blinds, windows, and screens.

The MQTT interface complements the REST API by allowing subscriptions to various events. Thanks to a universal data model, REST resources and MQTT topics follow a mirrored structure. For example, general zone data can be accessed via `/zones` in REST or `/zones/#` in MQTT. Similarly, detailed device-level data can be obtained using hierarchical paths down to specific services or global devices such as sky scanners.

Each controller in the system has its own IP address, and access to the APIs requires whitelisting through the firewall. In the case of this project, the author was granted access for testing purposes. Meetings have been held to explore the integration of this data into a central AAU database, as there is broader interest across other projects in utilizing the available data.

To test the system, the author interacted with the REST API using the built-in documentation hosted at `https://[IP address]/docs/rest`. However, since the controllers support DALI-2 while the installed sensors operate on the earlier DALI 1 protocol, most requests returned no (meaningful) data. In particular, attempts to retrieve information such as PIR sensor data were unsuccessful.

Although the documentation also describes how to subscribe to data using the MQTT interface, this was not tested due to concerns about potential performance impacts on the system. Discussions were held with Zumtobel's advisors, and while expert responses were eventually received in late May, the decision was made, due to the tight deadline of this report in early June, not to further pursue data integration at this time.

3.8 WiFi System

Aalborg University maintains a comprehensive WiFi infrastructure, managed by IT support (ITs), which extends to AAU Innovate. This infrastructure comprises three primary networks designed for different user groups.

- First, the `AAU 1-DAY` network serves guests without university affiliations. Access is granted via a daily password, retrievable through the dedicated `AAU 1-DAY` application, which displays credentials for the current day and the subsequent three days. This design facilitates temporary access for visitors.
- Second, the `Eduroam` network is the primary access point for students and staff. Users configure unique credentials for each device they wish to connect. As these credentials are generated while logged into an AAU account, devices connected to `Eduroam` can, in principle, be associated with their respective owners.
- Third, the `AAU` network is primarily intended for employees. Reportedly, only AAU-certified computers with registered MAC addresses are permitted to connect. As a student, I have no direct experience with this network.

This segmented network architecture presents an opportunity for user categorization. Devices connected to `AAU 1-DAY` can generally be assumed to belong to guests, while `Eduroam` connections signify students or staff. A potential challenge in occupancy estimation arises from users carrying multiple devices (e.g., a smartphone and a laptop). This issue is more pronounced on the guest network, as the device-specific credentialing of `Eduroam` mitigates double-counting for affiliated users to some extent.

Leveraging this WiFi infrastructure for occupancy estimation and indoor positioning offers significant potential, building upon techniques discussed in the state of the art (Section 1.1.3). For instance, even devices not actively connected can contribute to occupancy counts. Research by M. Vega-Barbas, M. Álvarez-Campana, D. Rivera, M. Sanz, and J. Berrocal [32] demonstrates methods using passive WiFi sensing to analyze “probe request” messages broadcast by smartphones. By developing unique fingerprints for devices, this technique overcomes MAC address randomization challenges and estimates crowd sizes with high accuracy (reported near 95%), making it applicable for estimating the total number of individuals present, regardless of network connection status.

Furthermore, data from connected devices can be analyzed. Work by S. Liu, Y. Zhao, F. Xue, B. Chen, and X. Chen [33] introduced DeepCount, a system utilizing deep learning (specifically Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs)) to analyze Channel State Information (CSI) from commercial WiFi devices. This approach infers crowd counts in multi-person indoor environments, achieving notable accuracy (up to 90% with refinement mechanisms) without relying solely on connection logs. Such techniques could potentially be applied to the AAU Innovate environment.

Accurate indoor positioning is another valuable application derivable from the WiFi infrastructure. As surveyed by J. Dai, M. Wang, B. Wu, J. Shen, and X. Wang [34], numerous WiFi-assisted positioning schemes exist, leveraging signal characteristics to determine device location within buildings. Integrating such positioning capabilities with occupancy data, potentially supplemented by alternative methods like mobile connection analysis [35], and knowledge of the AAU Innovate floor plan, could provide a highly detailed spatiotemporal understanding of building usage.

3.8.1 Potential Insights and Assumptions

Effective utilization of the WiFi system data could yield granular, near-real-time estimates of occupant counts within specific rooms or zones. This enables the mapping of movement patterns and density heatmaps throughout the building. A significant advantage of the AAU setup is the potential to categorize occupants based on their network affiliation (`Eduroam` vs. `AAU 1-DAY`). By correlating `Eduroam` usage with AAU account types (student, staff, faculty), it may be possible to approximate the distribution of different user groups defined in Section 2.1.3 across the building.

This detailed occupancy information holds considerable value for optimizing internal processes, managing resources (e.g., HVAC, lighting), evaluating space utilization, and informing strategic decisions. For large events, accurate visitor counts derived from WiFi data could provide crucial feedback. However, the collection and analysis of such data, particularly in real-time and at an individual level, must carefully address data privacy regulations (e.g., GDPR) and ethical considerations, likely requiring data aggregation and anonymization.

3.8.2 Data Collection and Processing

Given that the precise technical specifications and data logging policies of the AAU WiFi system are not currently known to the author, a detailed description of the data collection and processing pipeline cannot be provided at this stage. Accessing and utilizing this data would require collaboration with ITs and adherence to university data governance protocols.

Chapter 4

Data Integration

As mentioned by D. Hugo *et al.* [14] and discussed in Section 1.1.1, data silos remain a significant barrier to research in the built environment. Isolated systems and proprietary data formats make integration difficult, especially when data lacks spatial context. For example, sensors might be identifiable by ID, but determining their physical location, such as which room they are in, is a major challenge when using these systems independently.

This leads to the first challenge of this project: to meaningfully assemble and integrate all deployed systems and sensors. Achieving this not only enhances future research opportunities but also increases the contextual value of each individual data source.

This chapter begins with a discussion of the choice of technologies used throughout the data integration part of the project, outlining the reasoning behind each major technical decision in Section 4.1. It then proceeds to describe the implementation of each individual subgraph, corresponding to the systems outlined in Chapter 3 and described in detail in Section 4.2. For each, the setup and its corresponding Graph Query Language (GraphQL) schema are explained. Following that, Section 4.3 describes the supergraph, showcasing the complete graph model and detailing how Apollo Federation is applied in practice to integrate all subgraphs into a unified API.

4.1 Choice of Technologies

This chapter outlines the core technologies chosen for the implementation of the system. Each technology was selected to support the goals of modularity, scalability, maintainability, and ease of development. The chapter is divided into subsections, each covering a key component: GraphQL (Section 4.1.1), Apollo Federation (Section 4.1.2), GoLang with gqlgen (Section 4.1.3), and containerization via Docker and Docker Compose (Section 4.1.4).

4.1.1 GraphQL

GraphQL, initially developed by Facebook and later open-sourced, is a query language for APIs combined with a runtime for executing those queries against existing data sources [36]. It allows clients to specify exactly what data they need, shifting control over data fetching from the server to the client.

At the core of GraphQL is its Schema Definition Language (SDL), which establishes a strongly typed contract describing all available objects, fields, and relationships. This means that clients query a single flexible endpoint, typically `_/`

`graphql_`, with queries that match exactly the data they require. Modifications to the data are handled through *mutations*, and GraphQL also supports real-time updates via *subscriptions*.

Compared to REST [37], [38], which uses multiple endpoints with fixed structures, GraphQL reduces issues of over-fetching and under-fetching by letting the client tailor the query to its needs. This is especially beneficial for complex or bandwidth-constrained applications. Although it introduces challenges such as more complex server-side architecture and caching, its flexibility and client-centric approach make it well-suited for this project.

GraphQL was chosen as the primary API technology due to its strength in handling complex data structures and its ability to unify access to disparate systems through a single interface. Since the system must avoid requiring users to manually query multiple independent APIs, GraphQL offers a natural and efficient solution. Additionally, GraphQL schemas are self-documenting, which means the structure of available data can be explored directly via tools like the GraphQL sandbox. This removes the need for separate documentation and allows both developers and researchers to easily understand and interact with the API.

After selecting GraphQL, the author's confidence in this choice was further reinforced by insights from a recent article on the intersection of MCP and GraphQL, M. DeBergalis [39]. The article highlighted several architectural advantages of GraphQL in the context of AI integration, which are outlined below: GraphQL is uniquely suited to serve as the foundation for MCP, making it an ideal choice for enabling AI systems to interact intelligently with organizational APIs. At the heart of MCP is the need for a clean, declarative interface that abstracts the complexity of underlying services. GraphQL delivers this by offering a structured, schema-based language that defines exactly what data is needed—no more, no less. This precision is essential in AI applications where excessive or irrelevant data can mislead models and bloat token usage. Because GraphQL queries can seamlessly traverse multiple backend systems while enforcing access control and data policies, it enables deterministic, efficient execution, key for creating reliable, consistent AI experiences.

Moreover, GraphQL aligns perfectly with the agility and scalability required in modern AI development. Unlike REST or individual MCP servers for each API, GraphQL allows organizations to compose and evolve capabilities rapidly through a single semantic layer. This self-documenting and introspectable nature makes it easy for LLMs to understand and reason about available capabilities. Tools like Apollo MCP Server extend this paradigm by making GraphQL-native MCP tooling accessible, allowing teams to orchestrate complex logic through a single, cohesive interface.

4.1.2 Apollo Federation

Apollo Federation is an architecture designed to scale GraphQL across teams and microservices [40]. Instead of maintaining a monolithic schema, it enables developers to split the graph into *subgraphs*—modular schemas that can be owned and maintained independently by different teams.

These subgraphs are composed into a *supergraph* using an Apollo gateway, which serves as the central entry point for client queries. This composition allows subgraphs to extend types from other subgraphs, enabling rich inter-service relationships while maintaining service independence. The gateway handles query planning and routing, ensuring each part of a query is dispatched to the correct service.

Apollo Federation was selected to support the system's modularity and scalability. As each subsystem (see Chapter 3) is implemented as an individual GraphQL service, this approach allows for incremental development and maintenance while preserving a unified client-facing API. Adding a new data source requires only the creation of a new subgraph, which can then be integrated into the overall supergraph, offering adaptability and ease of extension.

4.1.3 GoLang with gqlgen

In the GoLang ecosystem, *gqlgen* is a popular tool for developing GraphQL servers [41]. It generates Go types and resolver interfaces from a GraphQL schema, ensuring strong type safety and alignment between backend logic and schema definition.

The decision to use Go was influenced by prior experience with the language, which enabled rapid and confident development. Go's statically typed nature contributes to highly readable and maintainable code, with many potential bugs being caught at compile time. Its type system aligns naturally with GraphQL, which is also based on strict typing through its SDL.

Additionally, Go features a robust standard library, reducing the need for external dependencies and thereby minimizing long-term maintenance efforts. The combination of static typing, good tooling, and language familiarity made Go a natural fit for this project.

Gqlgen in particular improves developer productivity by generating boilerplate code and guiding developers through the resolver implementation process. It also supports Apollo Federation, making it suitable for developing federated subgraphs that integrate cleanly into the larger architecture.

4.1.4 Containerization with Docker and Docker Compose

Docker is a platform that packages applications and their dependencies into lightweight, portable containers [42]. These containers ensure consistent behavior across different environments and eliminate issues related to environmental configuration mismatches.

Docker Compose extends Docker's capabilities by allowing developers to define multi-container applications in a single YAML configuration file. This makes it easy to run and manage related services—such as databases, APIs, and web servers—with a single command like `'docker compose up'`.

Containerization was chosen to streamline the deployment and development process. It ensures environment consistency across deployment targets, facilitates reproducible builds, and simplifies infrastructure management. Docker Compose further enhances productivity by allowing developers to launch the entire development stack with minimal effort.

Together, Docker and Docker Compose make the system easier to develop, test, and deploy, supporting the overall goals of modularity and maintainability.

4.2 Individual Data Systems

As discussed in Section 4.1, scalability and simplicity are key goals of the system. This is reflected in the selected technologies. Apollo Federation was chosen as the mechanism for implementing the data API. To better understand the system architecture, the structure of Apollo Federation is illustrated in Figure 4.1. Each data system must first be implemented as an individual GraphQL subgraph before it can be integrated into the federation.

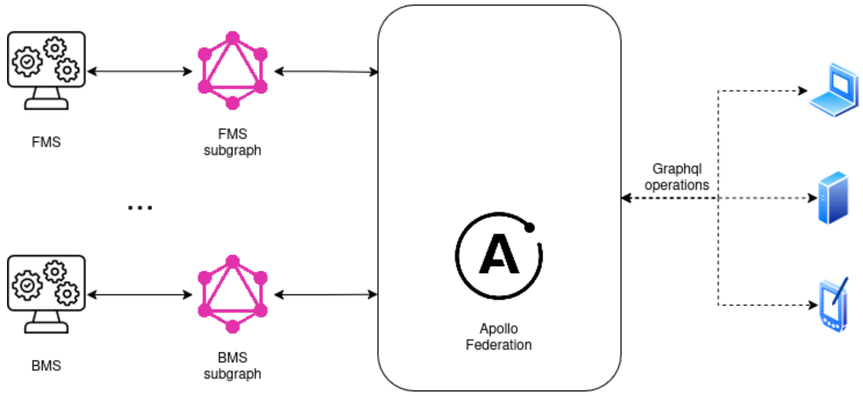

Figure 4.1: An explanatory drawing of the Apollo Federation system.

Each individual data system is implemented as a standalone Go server using *gqlgen*, as described in Section 4.1.3. The *gqlgen* tool generates a `server.go` file, which can be executed to launch the server. This exposes both the GraphQL Playground at the root domain and the GraphQL query endpoint at the */query* path.

The GraphQL schema is defined in `schema.graphqls`, from which *gqlgen* generates the corresponding Go types. Documentation comments can be added directly above each argument and type definition, making them an inherent part of the schema. This ensures that relevant information is visible when exploring the system, either through developer tools or programmatically—for instance, by an AI. In the following subsections, these comments are omitted for clarity. The logic for handling queries and mutations is implemented in `schema.resolvers.go`, maintaining a clean separation between schema definition and business logic.

Additional configuration is specified in the `gqlgen.yml` file. Federation support can be enabled here, which results in the generation of additional files, including `entity.resolvers.go`, where entity resolvers are implemented to support cross-service references. Specific implementations and schema overviews for each system are provided in the following subsections.

4.2.1 Facility Management System

As described in Section 3.1, the current FMS system lacks an API implementation. As a temporary solution to serve this essential information, the system operates using *static* data. This data is periodically received via Excel sheets sent by email and exported from the FMS system.

To simplify server-side processing, the Excel data is converted into a CSV file and placed in the server's file system. On startup, the server reads the CSV file and loads its contents into memory. Since the dataset is relatively small, this approach poses no memory concerns. However, if the dataset were to grow significantly, alternative approaches should be considered, such as reading the CSV file on each request or, preferably, importing the data into a self-hosted database for efficient querying. If performance becomes an issue, caching strategies can be employed to optimize data retrieval.

This setup results in the GraphQL schema shown in Listing 4.1. The schema defines three core types: `Building`, `Floor`, and `Room`. Since this is the first data source in the supergraph and acts as the central connector for other systems, no additional federation types are needed. Only a key definition is required for each of the current types to support entity resolution within the federated architecture.

```
type Room @key(fields: "id") {
    id: ID!
    roomNumber: String!
    type: String!
    area: Float!
    circumference: Float!
}

type Floor @key(fields: "id") {
    id: ID!
    name: String!
    rooms: [Room!]!
    floorplanUrl: String!
}

type Building @key(fields: "id") {
    id: ID!
    address: String!
    city: String!
    property: String!
    floors: [Floor!]!
}

type Query {
    buildings(ids: [ID!]): [Building!]!
    floors(ids: [ID!]): [Floor!]!
    rooms(ids: [ID!]): [Room!]!
}
```

Listing 4.1: The GraphQL schema for the Facility Management System.

4.2.2 Room Calendar System

Meeting room usage data at AAU Innovate is stored in Outlook. Although the storage backend has changed several times, shifting between on-premise solutions and cloud-based services, it is currently hosted on Azure servers. Accessing the calendar data associated with room email accounts requires registering an application in Azure Active Directory. This step must be performed by a system administrator. The original setup can be seen in Figure 3.1.

At the time of writing, this registration has not yet been completed. Despite several meetings and a prolonged email exchange, it was agreed that accessing the calendar data should be both technically and legally feasible through a digitalization partner. However, no single individual responsible for facilitating this process could be identified. Several staff members initially thought to be in charge were eventually found to have no affiliation with the Innovate building.

Due to this administrative bottleneck, alternative methods were explored. A precedent existed: a colleague at AAU Innovation had previously developed a script to extract room calendar data using their personal Outlook account. This was viable because both their account and the author's AAU account have the necessary access to shared room calendars.

Adopting a similar approach, a new system was developed, comprising two main components: a data scraper and a database. This new setup can be seen in Figure 4.2.

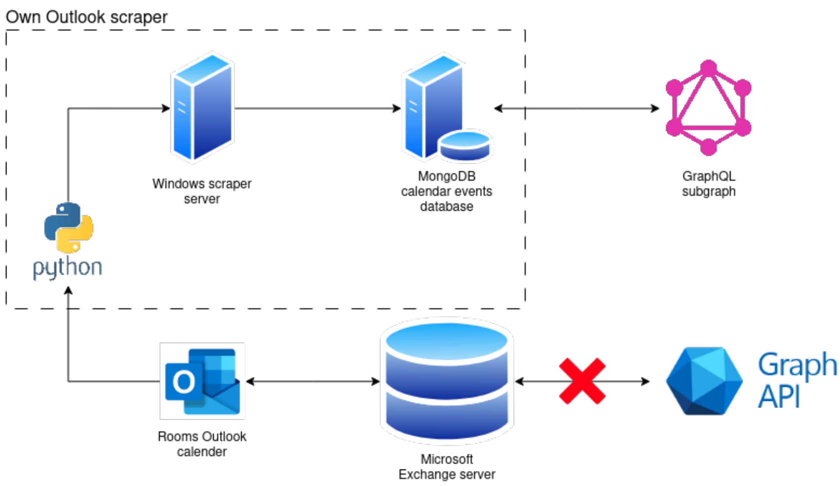

Figure 4.2: An overview of the custom-made Outlook scraper.

The scraper is implemented in Python using the `pywin32` library, which provides access to Windows COM APIs [43], [44], [45]. The key interface used is `win32com.client.Dispatch("Outlook.Application")`, which allows Outlook to be accessed as a COM object. Through this interface, the scraper connects to the Outlook Messaging Application Programming Interface (MAPI) namespace, opens the shared calendar folders associated with a list of room email addresses, and extracts the relevant event data. Since this capability is only available on Windows, an unused machine at AAU Innovate was repurposed as a dedicated scraping host. The scraper runs every 15 minutes, covering events from the previous day up to one year into the future. Updates are applied only when changes are detected. This frequent interval ensures timely and accurate data for biannual reports and supports potential future applications, such as automated promotion of upcoming events and analysis of meeting room bottlenecks.

An example of this code is seen in Listing 4.2.

```python
import win32com.client

outlook = win32com.client.Dispatch("Outlook.Application")
namespace = outlook.GetNamespace("MAPI")
recipient = namespace.CreateRecipient("room@aau.dk")
recipient.Resolve()

calendar = namespace.GetSharedDefaultFolder(recipient, 9)  # 9 = calendar
items = calendar.Items
items.Sort("[Start]")
items.IncludeRecurrences = True
events = items.Restrict("[Start] >= '01/01/2024'")
```

Listing 4.2: Python code to extract shared Outlook calendar data using `win32com`.

In addition to basic event details (subject, time, etc.), the scraper collects participant information to support anonymized reporting. Personal data is never stored. Instead, only the number of attendees from each department is retained. A caching mechanism helps improve performance by storing previously resolved mappings from email addresses to departments.

For larger events, the participant count reported by Outlook often underrepresents actual attendance, especially when events are booked via a form that lists only one or two organizers. To mitigate this, the scraper also parses the body text of calendar entries to extract the actual number of participants and the organizing department. These adjusted values are stored in separate database fields and exposed via the final data interface.

All scraped data is stored in a MongoDB database hosted on the Strato platform. Given the naturally JSON-like structure of calendar data and the need for schema flexibility in future iterations, a NoSQL database was preferred. MongoDB was chosen based on the author's prior experience and its widespread use. Access is restricted via credentials and limited to the AAU network for security.

The data is made available through a GraphQL API. Specific resolvers connect to the MongoDB backend, enabling efficient querying of room calendar data. The following schema illustrates the structure used to represent events and participation:

```
type DepartmentParticipation {
    department: String!
    attendeeCount: Int!
    isExternal: Boolean!
}

type Event {
    eventID: String!
    subject: String!
    start: Time!
    end: Time!
    durationMinutes: Int!
    departmentBreakdown: [DepartmentParticipation!]!
    formParticipants: Int
    formDepartment: String
}

type Query {
    rooms(emails: [String!]): [Room!]!
}
```

Listing 4.3: The GraphQL schema for the room calendar system.

To integrate this system into the existing infrastructure, the `Room` type from the FMS system was extended to support federation. The extension adds event data to each room object:

```
 extend type Room @key(fields: "id") {
   id: ID! @external
   name: String!
   email: String!
   events(startTime: Time!, endTime: Time): [Event!]!
}
```

Listing 4.4: The extra code in the room calendar GraphQL schema to integrate it into the Federation.

4.2.3 Building Management System

As described in Section 3.3.2, the BMS system exports its data into a database consisting of two main tables: *metadata* and *timeseries data*. This process is illustrated in Figure 3.3. The Timescale DB can be accessed via a REST API that exposes two endpoints—one for *metadata* and one for *timeseries data*.

Authentication is handled through HTTP Basic Authentication, with the username and password included in the request header. Requests are made to *https://bms-api.build.aau.dk/api/v1/* followed by the relevant endpoint. It is important to note that this server is only accessible from within the AAU network or through a Virtual Private Network (VPN) connection.

When the server starts, it sends a request to the *metadata* endpoint and caches the response, as this data rarely changes. This improves performance, since subsequent incoming requests only need to query the *trend data* endpoint and not repeatedly request metadata. The data structure is represented in the GraphQL seen in Listing 4.5.

```
type Value {
    timestamp: Time!
    value: Float!
}

type Sensor @key(fields: "externalID") {
    externalID: String!
    sourcePath: String!
    unit: String!
    values(startTime: Time!, endTime: Time): [Value!]!
}

type Query {
    sensors(ids: [String!]): [Sensor!]!
}
```

Listing 4.5: The GraphQL schema for the Building Management System.

Although the BMS system already includes spatial metadata such as area, building, floor, and room, this information is not currently exported into the database. Therefore, each sensor is manually connected to a corresponding `Room` in the FMS schema, which is defined in Listing 4.1.

This integration is implemented by extending the schema from Listing 4.5 with additional code, shown below in Listing 4.6.

```
extend type Room @key(fields: "id") {
    id: ID! @external
    roomNumber: String! @external
    sensors: [Sensor!]! @requires(fields: "roomNumber")
}
```

Listing 4.6: The extra code in the BMS GraphQL schema to integrate it into the Federation.

This code declares the `Room` type as external and identifies it using the `id` field as a key. It then extends the `Room` type by adding a new field, `sensors`, which lists all associated sensors. The resolver for `sensors` requires the `name` field (also marked as external) to function correctly.

4.2.4 Coffee Data

At AAU Innovate, beverage consumption is also monitored through a semi-automated data collection process, as described in Section 3.4. As shown in Figure 3.4, data is still manually exported to a USB key, after which a Python script processes the data and stores it in both CSV files and a database. The CSV export functionality was developed during my student job at AAU Innovate, while the integration of a PostgreSQL database was implemented specifically for this project.

Since data is collected monthly, the dataset grows over time. Relying solely on CSV files and reading them into memory becomes inefficient and problematic as the volume increases. Additionally, using only CSV files would require manually updating the server with new files each month and either restarting the server or implementing logic to detect changes. This approach was considered inadequate, which led to the implementation of a PostgreSQL database.

The database schema is defined using SQL, as shown in Listing 4.7. It consists of a `machines` table containing general information about each coffee machine. There are also two tables for beverage data: `beverage_counts` and `beverage_details`. The `beverage_counts` table stores total beverage counts, while `beverage_details` records the number of each specific beverage consumed.

The `beverage_counts` table was introduced because aggregated data is queried much more frequently than detailed beverage data. Without it, calculating totals would require repeated aggregation of the `beverage_details` table. Although it would be theoretically possible to use separate tables for each drink type, this would complicate the schema, slow down queries due to the need for frequent joins, and offer only marginal storage benefits, as each entry would still require a joinable identifier.

```sql
CREATE TABLE machines (
    machine_id VARCHAR(20) PRIMARY KEY,
    machine_name VARCHAR(100) NOT NULL,
    floor_id VARCHAR(30) NOT NULL
);

CREATE TABLE beverage_counts (
    id SERIAL PRIMARY KEY,
    machine_id VARCHAR(20) REFERENCES machines(machine_id),
    total_beverages INTEGER NOT NULL,
    timestamp TIMESTAMP NOT NULL,
    UNIQUE(machine_id, timestamp)
);

CREATE TABLE beverage_details (
    id SERIAL PRIMARY KEY,
    machine_id VARCHAR(20) REFERENCES machines(machine_id),
    beverage_name VARCHAR(50) NOT NULL,
    count INTEGER NOT NULL,
    timestamp TIMESTAMP NOT NULL
);
```

Listing 4.7: The SQL setup for the coffee data database

Based on this structure, a corresponding GraphQL schema was created for the coffee system, shown in Listing 4.8. The schema closely mirrors the database structure. Data can be queried through the `machines` query, which accepts an optional `machineIDs` argument—an array of IDs used to filter the results. If this argument is omitted or left empty, data for all machines is returned.

For each machine, beverage data can be requested via the `beverageCounts` and `beverageDetails` fields, both of which require a `startTime` and accept an optional `endTime`. If `endTime` is not provided, it defaults to the current time.

```
type Machine @key(fields: "id") {
    id: ID!
    name: String!
    beverageCounts(startTime: Time, endTime: Time): [BeverageCount!]!
    beverageDetails(startTime: Time, endTime: Time): [BeverageDetail!]!
}

type BeverageCount {
    id: ID!
    totalBeverages: Int!
    timestamp: Time!
}

type BeverageDetail {
    id: ID!
    beverageName: String!
    count: Int!
    timestamp: Time!
}

type Query {
    machines(machineIDs: [ID!]): [Machine!]!
}
```

Listing 4.8: The GraphQL schema for the coffee data system.

Finally, the coffee data was integrated into the federated system, as shown in Listing 4.9. Since coffee machines are typically located in hallways rather than specific rooms, they are associated with a `Floor` entity rather than a `Room`. The `machines` field is defined as an extension on the external `Floor` type from the FMS system graph.

```
extend type Floor @key(fields: "id") {
    id: ID! @external
    machines: [Machine!]!
}
```

Listing 4.9: The GraphQL code to implement the coffee data into the federation.

4.2.5 Door Counter System

The door counter system is initially described in Section 3.5. It was set up by an external company, and their configuration is illustrated in Figure 3.6. The collected data is stored in ThingsBoard, an open-source IoT platform [46], which is hosted by Multiteknik, the company behind the system, at *http://iot.multiteknik.dk:8080* .

ThingsBoard offers a REST API that can be leveraged to access the data. To authenticate, a `POST` request must be sent to the */api/auth/login* endpoint with a payload containing the `username` and `password`.[1] Upon successful login, an access token is returned, along with a renewal token. The system authenticates once on server start and uses the renewal token to refresh the access token before it expires.

The telemetry data itself is accessed via a `GET` request to the */api/plugins/telemetry/DEVICE/[DEVICE_ID]/values/timeseries* endpoint. The request must include query parameters such as `keys`, `startTs`, `endTs`, and `limit`. The API returns a JSON object structured as shown in Listing 4.10. Notably, the `ts` field represents the timestamp in Unix milliseconds.

```
{
  "c1": [
    {
      "ts": number,
      "value": "string"
    },
    {
      "ts": number,
      "value": "string"
    }
    // ... more objects
  ],
  // ... doors "c2" and "c3"
}
```

Listing 4.10: The structure of the response of the door counter system REST API.

This data access is encapsulated in the GraphQL schema shown in Listing 4.11. The schema defines two types: `Entrance` and `TelemetryData`. Data can be requested using the `getEntrances` query, optionally filtered by specific entrance IDs. Each `Entrance` can return telemetry data for a given time range using the `telemetryData` field.

```
type Entrance @key(fields: "id") {
    id: ID!
    name: String!
    telemetryData(startTime: Time!, endTime: Time): [TelemetryData!]!
}

type TelemetryData {
    timestamp: String!
    value: Int
}

type Query {
    getEntrances(ids: [ID!]): [Entrance!]!
}
```

Listing 4.11: The GraphQL schema of the door counter system.

To integrate the door counter system into the federation, the `Building` type is extended with an `entrances` field, as shown in Listing 4.12. While entrances could

[1]The author is aware that this request is transmitted over HTTP, which exposes the request payload to interception during network transit. A request was made to Multiteknik to upgrade to HTTPS while outlining the potential security risks; however, this request was denied. Until a new system is in place, the credentials used are both unique and rotated regularly. In the future, AAU Innovate plans to replace this system with one that offers more detailed data, such as direction of movement, which could allow real-time estimations of building occupancy instead of only daily counts.

also be associated with a specific floor, this would often default to the ground floor, leaving other floors with empty entrance data. This could create confusion, for example, interpreting staircases or elevators as entrances. Therefore, associating entrances at the building level provides a simpler and more intuitive structure.

```
extend type Building @key(fields: "id") {
    id: ID! @external
    entrances: [Entrance!]!
}
```

Listing 4.12: The GraphQL code to integrate the door counter system into the federation.

4.3 The Supergraph

All the individual data sources described in Section 4.2 are combined using Apollo Federation (see Section 4.1.2). This results in the formation of a unified supergraph, which aggregates all previously defined subgraphs. A schematic overview of this supergraph is shown in Figure 4.3. From the diagram, it is evident that the FMS system serves as the backbone of the entire architecture. This makes logical sense, as it anchors each system through its spatial positioning in the building. Consequently, the system forms a truly integrated API, now enriched with spatial context.

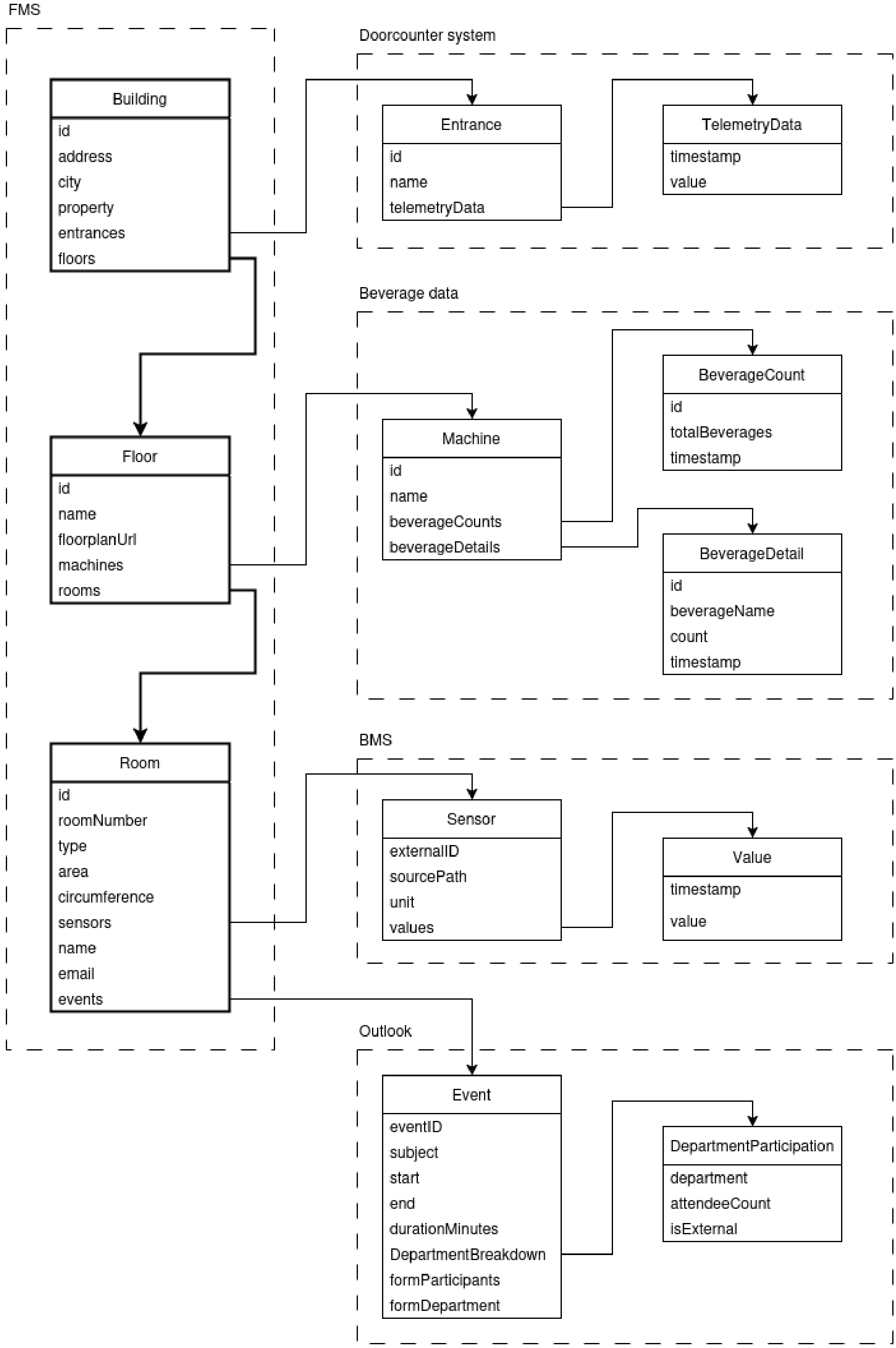

Figure 4.3: The integrated API schema of the GraphQL Federation.

To implement this in practice, a NodeJS gateway server is created using the `apollo-server` and `@apollo/gateway` packages. These packages are not available in GoLang, which is why NodeJS is used for this component.

The source code required to create this server is straightforward and is shown in full in Listing 4.13. Each subgraph is defined with a name and a URL inside the `IntrospectAndCompose` configuration of the `ApolloGateway` instance. The federated server is then instantiated using the `ApolloServer` class.

```
const { ApolloServer } = require("apollo-server");
const { ApolloGateway, IntrospectAndCompose } = require("@apollo/gateway");

const gateway = new ApolloGateway({
  supergraphSdl: new IntrospectAndCompose({
    subgraphs: [
      { name: "doorcounters", url: "http://doorcounters:4001/query" },
      { name: "bms", url: "http://bms:4002/query" },
      { name: "fms", url: "http://fms:4003/query" },
      { name: "outlook", url: "http://outlook:4004/query" },
      { name: "coffee", url: "http://coffee:4005/query" },
    ],
  }),
});

const server = new ApolloServer({
  gateway,
  subscriptions: false,
});

server.listen().then(({ url }) => {
  console.log(`🚀 Server ready at ${url}`);
});
```

Listing 4.13: The source code to create the gateway for the Apollo Federation.

To deploy the entire system efficiently, Docker and Docker Compose are employed. The benefits and setup of this approach are discussed in detail in Section 4.1.4. Each subgraph service runs in its own container. Since the GoLang-based subgraphs follow a similar hosting structure, a single reusable Dockerfile is used across them. The gateway uses its own Dockerfile tailored to NodeJS. Docker Compose is used to orchestrate all these containers. A full overview of this deployment setup is provided in Appendix B.

Chapter 5

Creating a User Interface

As described in Chapter 4, the project's data model results in a unified *supergraph*, enabling access to all underlying data. While this is a crucial step, solving the research question outlined in Section 2.2 requires more than just data access. That section also emphasizes the role of data in supporting objective, data-driven decisions, proven to enhance decision quality. However, at AAU Innovate, decision-makers typically lack the technical skills to query a GraphQL endpoint or analyse the results. While tailored scripts or interfaces could be developed for specific cases, these are time-consuming and only address known needs.

A deeper challenge lies in the fact that AAU Innovate has never had access to such extensive data before, making it unclear what questions should be asked or what insights would be most valuable. This uncertainty makes it difficult to design static interfaces. Instead, there is a strong case for using adaptive tools like Large Language Models (LLMs). A well-integrated language model could dynamically query the data, perform relevant analysis, and deliver insights—all without requiring technical knowledge from the user. This approach eliminates the need for predefined interfaces, offering a flexible and future-proof solution that can adapt to new data sources and evolving user needs.

The chapter begins by introducing data integration into LLMs in Section 5.1. Subsequently, the LLM used in the project is selected, establishing the user interface of the final application in section Section 5.2.

5.1 Large Language Model Data Integration

As mentioned earlier, the data must be made available to the LLM in a meaningful and accurate way. As T. Sassmannshausen [47] describes, there are essentially two main methods for integrating company-specific data into a Large Language Model: **fine-tuning** and **Retrieval-Augmented Generation (RAG)**.

Fine-tuning refers to adapting an existing language model by training it further on organization-specific content such as marketing material, internal reports, or domain-specific language. While this can result in a deeply specialized model, it is also resource-intensive and costly. Estimates range from \$10,000 to over \$1,000,000, depending on scope and data preparation. Moreover, fine-tuning carries risks related to overfitting and data balance. For example, a model trained on 90% code and 10% support tickets will reflect this imbalance in its performance.

RAG, on the other hand, dynamically injects relevant data into prompts at runtime. Instead of storing facts inside the model, RAG retrieves them on demand using methods such as vector search and knowledge graphs. This makes RAG more adaptable, cost-effective, and easier to update. However, the quality of the output depends heavily on the retrieval system's design. Combining vector search with metadata filters or structured knowledge bases can significantly enhance RAG's performance.

The general consensus is that RAG should be the default approach, while fine-tuning is best reserved for highly specialized or offline scenarios. Even in domains such as finance and medicine, fine-tuned models have frequently been outperformed by general-purpose LLMs models combined with well-designed RAG pipelines. This trend underscores the strength of generalist models when paired with robust retrieval strategies. For our use case—where data is dynamic and questions are constantly evolving—RAG offers a more scalable, maintainable, and future-proof solution.

> "The smartest generalist frontier models beat specialized models in specialized topics."
>
> — T. Sassmannshausen, 2024, [47]

This highlights a key concern: training and maintaining a specialized model is not only costly, but also risky, given the rapid pace of AI advancements. Even a newly fine-tuned model may quickly be surpassed by the next generation of general-purpose models.

5.1.1 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) is a method for improving the performance of LLMs by integrating external knowledge at inference time [48]. Instead of relying solely on the static information stored in a model's parameters, RAG retrieves relevant documents from an external corpus and conditions the generation on this information. This approach enables the model to provide more accurate, factual, and context-aware responses, especially in knowledge-intensive tasks.

Traditional language models are limited by the knowledge encoded during pre-training, which becomes outdated over time and may not include domain-specific or proprietary information. RAG addresses this limitation by combining two components: a retrieval mechanism and a generation model. The retrieval component provides real-time access to external information, while the generation component uses this retrieved information to produce informed and relevant outputs.

The RAG process typically consists of the following steps:

1. **Query Formulation:** The user provides a prompt or question. This prompt is either directly used as a query or transformed into one using an optional query rewriting step, especially in multi-turn conversations.

2. **Document Retrieval:** The query is passed to a retrieval system, such as a dense vector index or traditional keyword-based search engine. This system searches an external corpus and returns the top-k relevant documents based on similarity to the query.

3. **Context Construction:** The retrieved documents are combined with the original prompt to form an augmented input. This may include concatenating document text, titles, metadata, or summaries alongside the query.

4. **Generation:** The augmented input is passed to a language model, which generates a response conditioned on both the original prompt and the retrieved context. This allows the model to incorporate information it was not trained on.

5. **Post-processing (optional):** In some applications, the output may be filtered, ranked, or further validated to improve accuracy, coherence, or format compliance.

5.1.2 Functions calling with LLMs

RAG is good when specific information does not change often. However, it fails to provide a good solution for ever-changing information, such as "What is the time now in Brazil?" or "What is the current stock price of Apple?". This type of information is often available through APIs, which cannot be retrieved effectively using static documents alone. To solve this, function calling, also referred to as tool usage, can be used.

Function calling allows LLMs to interact with external tools or APIs in a structured and reliable manner [49]. Instead of generating a final answer directly, the model detects when a function should be invoked and produces a JSON object containing the necessary arguments. These arguments are then used to call the external function, after which the result can be incorporated back into the final response.

For instance, when a user asks "What is the weather like in London?", the model may output a structured call like `get_current_weather(location: "London", unit: "celsius")`. The function is executed outside the model, and the returned weather data is passed back to the model to generate a natural-language summary for the user.

Function calling can be used to build conversational agents that access up-to-date weather, financial, or calendar data, as well as to convert natural language into valid API requests or database queries. This mechanism enables a new generation of LLM-powered applications that can:

- Access live or dynamic data sources through APIs
- Extract and structure information from user prompts
- Automate tasks that require real-time information
- Support knowledge retrieval from structured databases rather than free text
- Solve complex problems by delegating parts of the reasoning to external logic

By combining RAG and function calling, systems can benefit from both document-based grounding and dynamic tool use. RAG excels in knowledge-intensive tasks where up-to-date information is not critical, while function calling empowers LLMs to act as orchestrators that query live systems and react to external context.

5.1.3 Model Context Protocol

The growing demand for LLM-based applications has led to a proliferation of ad hoc integrations between models and external tools or data sources. These implementations are often tightly coupled, fragile, and difficult to maintain, especially in a rapidly evolving ecosystem where new models and capabilities are frequently released.

The Model Context Protocol (MCP), developed by Anthropic and announced on 25 November 2024, tackles this challenge by introducing a standardized interface for communication between models and tools [50], [51]. Rather than requiring bespoke integrations for each tool-model pair, MCP offers a scalable, interoperable solution where models can dynamically discover and invoke external capabilities using a universal protocol.

Although introduced in late 2024, the protocol only began gaining significant traction in early 2025 and has since emerged as one of the most prominent trends in the AI field. Its adoption is particularly timely, given the pace of model innovation and shifting APIs. By embracing MCP, organizations can decouple model selection from integration logic—future-proofing their systems and substantially reducing development overhead.

For a detailed discussion of the protocol's architecture, communication mechanisms, and benefits, see Appendix C.

5.2 Large Language Model Selection

Next, an LLM must be selected. There are many different models available, each with varying capabilities and training methods. For our purposes, models are evaluated based on the following criteria:

- As discussed in Section 5.1, the model must support **tool calling** to retrieve additional context. For instance, the `deepseek-r1` model lacks this capability. In Ollama, this can be filtered directly.
- The model must run **locally** with no data being shared externally. Cloud-based models like ChatGPT, Gemini, or Claude cannot fulfill this requirement.
- The model must be **free** to use.
- It must be able to run on the available hardware. While this often depends on model size and quantization, access to the UCloud environment removes most hardware constraints.

The following subsections highlight a curated set of well-known models that satisfy the previously established criteria. Only high-performing models have been included, with smaller variants omitted when clearly outperformed by their larger or more capable counterparts. Each model is briefly described in terms of its size, performance characteristics, and typical use cases.

5.2.1 Qwen3

Qwen3 is the latest generation in the Qwen series, featuring both dense and mixture-of-experts (MoE) variants [52]. It shows significant improvement in reasoning, outperforming previous models such as Qwen2.5 and QwQ across benchmarks in mathematics, code generation, and commonsense reasoning.

The flagship model, `Qwen3-235B-A22B`, performs competitively against DeepSeek-R1, o1, o3-mini, Grok-3, and Gemini-2.5-Pro. Benchmark results are illustrated in Figure 5.1.

	Qwen3-235B-A22B MoE	Qwen3-32B Dense	OpenAI-o1 2024-12-17	Deepseek-R1	Grok 3 Beta Think	Gemini2.5-Pro	OpenAI-o3-mini Medium
ArenaHard	95.6	93.8	92.1	93.2	-	96.4	89.0
AIME'24	85.7	81.4	74.3	79.8	83.9	92.0	79.6
AIME'25	81.5	72.9	79.2	70.0	77.3	86.7	74.8
LiveCodeBench v5, 2024.10-2025.02	70.7	65.7	63.9	64.3	70.6	70.4	66.3
CodeForces Elo Rating	2056	1977	1891	2029	-	2001	2036
Aider Pass@2	61.8	50.2	61.7	56.9	53.3	72.9	53.8
LiveBench 2024-11-25	77.1	74.9	75.7	71.6	-	82.4	70.0
BFCL v3	70.8	70.3	67.8	56.9	-	62.9	64.6
MultiIF 8 Languages	71.9	73.0	48.8	67.7	-	77.8	48.4

1. AIME 24/25: We sample 64 times for each query and report the average of the accuracy. AIME'25 consists of Part I and Part II, with a total of 30 questions.
2. Aider: We didn't activate the think mode of Qwen3 to balance efficiency and effectiveness.
3. BFCL: The Qwen3 models are evaluated using the FC format, while the baseline models are assessed using the highest scores obtained from either the FC or prompt formats.

Figure 5.1: Performance of Qwen3 compared to other leading models [52].

5.2.2 LLaMA

Meta's LLaMA series includes two main families: the recently released `LLaMA 4` and the older `LLaMA 3`, the latest model of which is `llama3.3` [53].

LLaMA 4 includes three models:

- `Behemoth`: 288B × 16 experts (2T total parameters)
- `Maverick`: 17B × 128 experts (400B total)
- `Scout`: 17B × 16 experts (109B total)

At the time of writing, `Behemoth` is still undergoing training and is therefore not yet available for use. Performance comparisons are illustrated in Figure 5.2. As shown in the figure, even though the largest model has not been released, the fourth-generation models already outperform all models from the third generation.

Category	Benchmark	# Shots	Metric	Llama 3.3 70B	Llama 3.1 405B	Llama 4 Scout	Llama 4 Maverick
Image Reasoning	MMMU	0	accuracy	No multimodal support		69.4	73.4
	MMMU Pro^	0	accuracy			52.2	59.6
	MathVista	0	accuracy			70.7	73.7
Image Understanding	ChartQA	0	relaxed_accuracy			88.8	90.0
	DocVQA (test)	0	anls			94.4	94.4
Code	LiveCodeBench (10/01/2024-02/01/2025)	0	pass@1	33.3	27.7	32.8	43.4
Reasoning & Knowledge	MMLU Pro	0	macro_avg/acc	68.9	73.4	74.3	80.5
	GPQA Diamond	0	accuracy	50.5	49.0	57.2	69.8
Multilingual	MGSM	0	average/em	91.1	91.6	90.6	92.3
Long Context	MTOB (half book) eng->kgv/kgv->eng	-	chrF	Context window is 128K		42.2 / 36.6	54.0 / 46.4
	MTOB (full book) eng->kgv/kgv->eng	-	chrF			39.7 / 36.3	50.8 / 46.7

*reported numbers for MMMU Pro is the average of Standard and Vision tasks

Figure 5.2: Performance of LLaMA 4 compared to other LLaMA models [53].

5.2.3 Mixtral

Mixtral is a family of pretrained sparse MoE models [54]. The largest variant is Mixtral 8x22B (80GB). Other models in the Mistral family, such as Mistral Nemo, focus more on efficiency than raw performance, and are not considered here.

5.2.4 Mistral Large

Mistral Large 2 is the latest flagship model from Mistral, offering strong capabilities in code generation, mathematics, and reasoning. It supports a 128k context window and multilingual output [55].

5.2.5 Hermes 3

Hermes 3 is fine-tuned from LLaMA 3.1 (8B, 70B, and 405B variants) and trained on a synthetic dataset [56]. It demonstrates strong long-context handling, roleplay, and precise instruction-following, making it particularly suited to structured tasks such as strict GraphQL schema adherence.

5.2.6 Command R+

Command R+ is Cohere's open model optimized for real-world enterprise use cases [57]. It balances efficiency and accuracy, with a strong emphasis on production-readiness.

5.2.7 DBRX

DBRX is a decoder-only transformer with a fine-grained MoE architecture (132B total parameters, 36B active per input) [58]. It excels in programming tasks, outperforming CodeLLaMA-70B, and achieves 70.1% on HumanEval and 73.7% on MMLU [59].

5.2.8 Watt-tool

Watt-tool-70B is a tool-optimized model based on LLaMA-3.3-70B-Instruct [60]. It ranks highly on the Berkeley Function Calling Leaderboard (BFCL), which evaluates models on structured function-calling tasks such as correct function selection, multi-function execution, and relevance detection. Watt-tool stands out for its enhanced tool usage and strong performance in multi-turn conversations. More specifically, the `fireshihtzu/watt-tool-70B-formatted` model is selected, as it supports tool usage.

5.2.9 Final Selection

Selecting a model based purely on benchmarks may not always yield the best real-world results. However, due to time and resource constraints, evaluating all models through hands-on testing is infeasible. Instead, benchmark results will be used to narrow down the strongest candidates.

According to [59], DBRX Instruct stands out as a top-performing open model across a wide range of benchmarks, excelling in programming, mathematics, and general knowledge. It outperforms models like Mixtral Instruct and Grok-1—even though Grok-1 has more than twice the parameters. DBRX achieves 70.1% on HumanEval (surpassing CodeLLaMA-70B Instruct) and 73.7% on MMLU, outperforming GPT-3.5. With strong results in long-context tasks and retrieval-augmented generation, DBRX offers robust general-purpose performance that rivals both open and proprietary models. Despite these strengths, DBRX is ultimately excluded. While it clearly outperforms Mixtral (which itself is often compared to LLaMA 2 70B and GPT-3.5 by [61]), newer models such as LLaMA 4 have advanced significantly. As shown in Figure 5.2, DBRX delivers similar performance to models that are now considered outdated. With DBRX being nearly a year old and largely benchmarked against older models, its continued inclusion is no longer justified.

Command R+ shows comparable tool usage performance to the original Mistral Large model [62]. However, since the release of Mistral Large 2—which significantly improves upon its predecessor—Command R+ has been outclassed [63]. As such, it is excluded from further consideration.

The Hermes 3 model, evaluated in [64], delivers results on par with LLaMA 3.1 Instruct 405B. However, as seen in Figure 5.2, LLaMA 4 Maverick surpasses these models on all fronts. Given this, Hermes 3 is also excluded.

After applying these filters, four models remain:

- `Watt-tool`: A function-calling specialist with strong tool integration and multi-turn performance.
- `Mistral Large`: A high-performance general-purpose model with an excellent cost-to-performance ratio.

- `LLaMA 4 Maverick`: A state-of-the-art general-purpose model with strong reasoning and tool usage.
- `Qwen3`: A versatile model with competitive benchmark results, comparable to proprietary models like Gemini 2.5 Pro (Figure 5.1).

Since those models were selected based on theoretical criteria, the final choice will be made based on their actual performance in the implementation found in Chapter 7.

Chapter 6

The Test Setup

This section describes the evaluation framework used to compare expert-driven and AI-assisted data analysis in a smart building context. Specifically, we pose five data-centric questions that can be answered using the current GraphQL-based platform.

Each question is answered twice:

1. Once by a technical expert who writes custom scripts and queries the data manually.
2. Then by the LLM, which uses the same data — exposed via MCP — to produce its response autonomously.

The goal of this dual-analysis approach is to assess how effectively the LLM can interpret and act on complex, real-world sensor data using structured access mechanisms. The following sections detail the preparation of both the expert and LLM environments, including server integration, AI model selection, and network configuration.

The chapter begins with the question selection process, described in Section 6.1. Subsequently, the test environment for both the expert and the LLM is detailed in Section 6.2. Finally, a brief overview of the expert test results is presented in Section 6.3, which serves as a basis for comparison with the LLM test outcomes.

6.1 Question Selection

All currently available data sources are discussed in Chapter 3. After brainstorming together with Innovate, we identified key areas of interest to guide our data evaluation.

One of Innovate's primary goals is to attract as many people as possible to the building. To support this, they regularly organize events aimed at bringing in new visitors. Understanding which days and events have been the most successful is therefore crucial. This led to the formulation of the following question:

1. **Which have been the busiest days of the year and overall? Can we identify which events were organized on those days?**

Another commonly mentioned issue is that meeting rooms are often hard to find. To determine whether this is a real problem, the following question was created:

2. **What can we say about meeting room usage? What is the average usage between 8:00 and 16:00 (in %), both overall and per room?**

What was the peak usage and when did it occur? Have we reached 100% utilization at any point?

One of Innovate's goals is to support external collaboration. This often leads to meetings, or at least the booking of meeting rooms. To assess how well Innovate is achieving this goal, we pose the following question:

3. **How many meetings have we hosted where an external partner was present?**

Innovate has been in use for a little over two years. During that time, rooms were assigned specific functions, although not everything was set in stone. As teams and processes evolve, it's important to reevaluate how space is being used. There have been rumors that some rooms on the second floor are underutilized. Can we confirm this through the data? Or are there other, even less-used rooms?

4. **Which rooms are underused or show the least activity year to date? And which rooms show the most activity?**

Finally, data should not only be used to analyze the current situation or identify where change is needed, but it can also be a powerful tool to evaluate the impact of those changes.

For example, Innovate realized that its IT lab was underused. Despite having exciting equipment, people rarely visited the room. As a result, part of the space was transformed into a prototype lab, now hosting AI computers and some 3D printers. Initial impressions suggest increased activity, but is that truly the case?

5. **Innovate changed the function of the IT lab to attract more users. Can a measurable increase in activity be seen, and if so, how much more is it being used?**

6.2 Preparing the Test Environment

To enable the expert to run the tests, several preparatory steps must be undertaken. First, the expert test setup needs to be created by leveraging the expert's responses to the previously defined questions, as outlined in Section 6.2.1. Next, in preparation for the LLM tests, the GraphQL schema must be made available over the MCP interface, which is described in detail in Section 6.2.2. Following this, an AI model runner must be selected that is capable of executing the model chosen in Section 5.2; this selection process is discussed in Section 6.2.3. Subsequently, the actual setup of the LLM test machine can be performed, according to the procedures described in Section 6.2.4. Afterward, connectivity between the various systems must be configured to enable proper communication, as explained in Section 6.2.5. Finally, the tool enabling the deployment of an MCP host, which integrates all components of the system, should be set up as detailed in Section 6.2.6.

6.2.1 The expert test setup

To test the system, the following setup was used, shown visually in Figure 6.1. The hosting of the Apollo Federation, etc, is described in Appendix B.

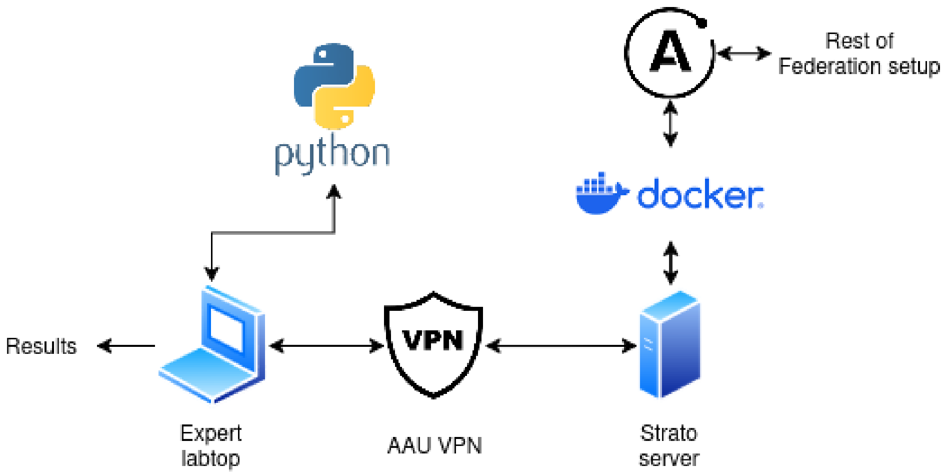

Figure 6.1: The test setup used by the expert.

The tests are conducted on the expert's own computer. First, Apollo Federation is run on a server at AAU. Note that Figure 6.1 is not fully complete, as the actual components involved are already described in Figure 4.1. The current schema could be extended with that figure and further with the schemas of each individual subsystem to provide the full picture.

To ensure the data remains inaccessible to unauthorized users, the server only runs on the AAU network. This means that a VPN connection is required to access it when working off-campus. This is an added security precaution.

6.2.2 MCP Server Integration

The GraphQL schema described in Section 4.3 can easily be exposed over MCP using the `mcp-graphql` tool [65]. This tool takes a running GraphQL server as input and exposes it as an MCP server with two main tools available:

- **introspect-schema:** Retrieves the GraphQL schema. This tool should be used first if the schema is not available as a local resource. It supports either a local schema file or an introspection query against the server.
- **query-graphql:** Executes GraphQL queries against the endpoint. By default, mutations are disabled unless the environment variable `ALLOW_MUTATIONS` is explicitly set to `true`.

This approach allows the schema to be used by any MCP host, enabling both introspection and querying of data without the need to write a custom server. Since the GraphQL structure includes self-contained documentation within the schema itself, the LLM has access to all necessary context to understand the available data and its structure.

6.2.3 AI Model Runner Selection

To enable a usable AI system, the underlying language model must be executed in an environment that supports external tool calling and the integration of additional context. As explained in Section 5.1, this capability is essential for enabling the model to perform dynamic lookups, access private datasets, and support user-specific workflows.

However, much of the data used in this project (see Chapter 3) is provided by AAU and may contain AAU specific data. Based on discussions with the case partner AAU Innovate, it was concluded that the AI system must not rely on external cloud-based services unless full control over data flow can be guaranteed. In practice, this means the language model must be hosted and run locally, ensuring that all processing remains behind closed doors and under our governance.

Among the available options for running LLMs locally, *Ollama* stands out as the most robust and accessible solution. Ollama is a local LLM runner that supports a wide range of state-of-the-art models—including Mistral, Meta's Llama family, and Google's Gemma, through a simple command-line interface and an HTTP API [66].

One of the reasons Ollama is particularly suited for this use case is its use of *quantization*, which allows large models to run efficiently on consumer-grade hardware by reducing their computational and memory requirements. This enables real-time, offline AI interaction without the need for expensive GPU clusters or constant internet connectivity. In addition, Ollama ensures that all model inputs and outputs remain on the host device, aligning well with privacy constraints.

Another strength of Ollama lies in its support for lightweight customization through *LoRA (Low-Rank Adaptation)* adapters. Using a simple text-based configuration called a `Model file`, developers can define base models, default prompts, and task-specific LoRA adapters—all without retraining the model from scratch. These configurations can be reused, shared, and swapped to adapt the same model to different use cases efficiently.

Due to prior positive experience with Ollama, its excellent documentation, active open-source community, and full support for offline, privacy-conscious model execution, Ollama was selected as the model runner for this project.

As Ollama does not come with out-of-the-box support for MCP, the tool `mcphost` was used. MCPHost is a CLI host application that enables LLMs to interact with external tools through the MCP [67].

6.2.4 Creating the LLM Test Machine

To run the large LLMs model selected in Section 5.2, a machine with substantial GPU capacity is required. AAU Innovate provides access to two AI machines equipped with NVIDIA RTX 4090 GPUs, each featuring 16,384 CUDA cores and 24 GB of vRAM [68]. However, this hardware is insufficient for hosting most of the selected LLM models, which have substantial memory requirements:

- Watt Tool: 128k context window, 43 GB [60]
- Mistral Large: 128k context window, 73 GB [55]
- Qwen3: 40k context window, 142 GB [52]
- Llama4: 1M context window, 245 GB [53]

To meet these demands, other High Performance Computing (HPC) options offered by AAU were evaluated: Strato, UCloud, AI Cloud, and AI-Lab [69]. AI-Cloud is excluded from this evaluation, as it is not accessible to students. AI-Lab provides a system with 8× NVIDIA L4 GPUs, each with 24 GB of vRAM [70]. Strato allows student access to a single NVIDIA T4 instance with 2,560 CUDA cores and 16 GB of vRAM [71].

UCloud, in contrast, offers access to three clusters: SDU/K8, AAU/K8, and AAU/VM [72]. The SDU/K8 cluster includes high-end GPUs but restricts their use to researchers. On the AAU/VM cluster, the most powerful configuration available to students is 3× NVIDIA A10 GPUs, each with 48 GB of vRAM, 696 GB/s of memory bandwidth, and a bidirectional interconnect of 112.5 GB/s [73]. The AAU/K8 cluster supports configurations with 1 to 8 NVIDIA L40 GPUs, each offering 48 GB of vRAM and 864 GB/s of memory bandwidth [74]. This enables configurations of up to 144 GB of vRAM (6 GPUs) or 192 GB (8 GPUs).

Given the superior memory capacity and performance of the AAU/K8 cluster, this platform was selected for experimentation. However, due to long queue times —exceeding 48 hours—for the 8× L40 GPU configuration, the decision was made to proceed with the 6× GPU configuration. This setup is easier to access and still supports running the largest LLaMA4 model, with partial offloading to the CPU when required.

6.2.5 Network Access to GraphQL Federation

The next challenge was enabling data access on the UCloud machine, which, like the expert setup shown in Figure 6.1, must communicate with the GraphQL Federation. VPN access was not viable, and the UCloud instance is not part of the AAU internal network. Furthermore, the GraphQL endpoint is not authenticated and cannot be exposed to the public internet.

To resolve this, the data server was migrated to a public IP space on Strato. Access was restricted by opening only specific ports for designated IP addresses, relying on IP-based filtering as a security measure. While effective for external systems, this approach encountered a limitation: due to AAU's internal security policies, machines within the AAU network, even those with public IPs, cannot directly communicate with each other.

After discussions with the AAU CLAAUDIA team and testing various configurations, a viable solution was established, as illustrated in Figure 6.2.

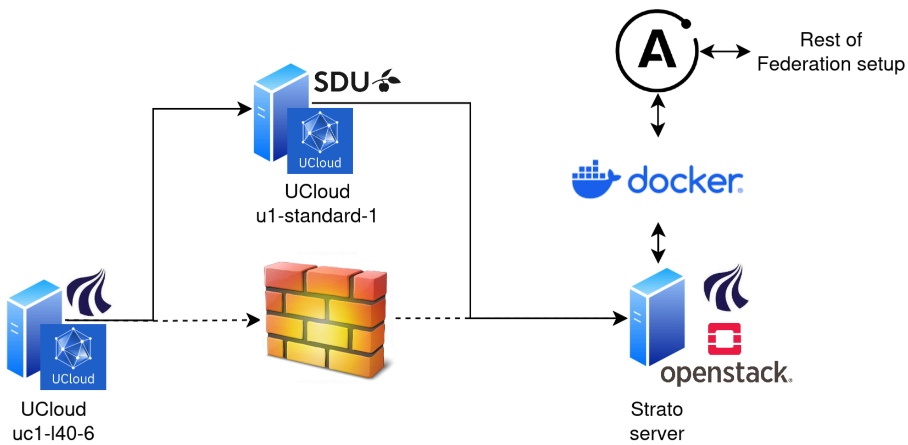

Figure 6.2: The LLM test setup used.

A proxy server was deployed on the SDU/K8 cluster. Its sole purpose is to hold the correct private SSH key to access the Strato resource. The SDU UCloud machine was configured for SSH access, and the key was uploaded to the AAU/K8 machine. Using SSH tunneling, the required port (`4000`) was forwarded to `localhost` on the

AAU/K8 machine, thereby granting access to the data server. This is accomplished using the command shown in Listing 6.1.

```
ssh -J ucloud@ssh.cloud.sdu.dk:[ucloud-port] \
ubuntu@[strato-ip] -L 4000:localhost:4000
```

Listing 6.1: The command needed to forward port 4000 on Strato to the UCloud machine, giving it access to the GraphQL Federation.

6.2.6 Running the Model with `mcphost`

With networking in place, the model was run on the UCloud server using `Ollama` and `mcphost`, as described in Section 6.2.3. `mcphost` connects to the MCP server, created using the `mcp-graphql` tool detailed in Section 6.2.2. The tool configuration for `mcphost` is shown in Listing 6.2.

```
{
  mcpServers: {
    time-server: {
      command: "uvx",
      args: [
        "mcp-server-time",
        "--local-timezone=Europe/Copenhagen"
      ]
    },
    mcp-graphql-aau-innovate-data-server: {
      command: "npx",
      args: ["mcp-graphql"],
      env: {
        ENDPOINT: "http://localhost:4000/graphql",
        NAME: "mcp-graphql-aau-innovate-data-server"
      }
    }
  }
}
```

Listing 6.2: The config file for `mcphost` used to perform the experiments.

During testing, a bug was discovered in `mcphost` versions 2.0.1 to 2.0.3. When the tool `introspect-schema` was invoked with the argument `undefined` instead of `{}`, it would break the MCP protocol.[2]

Additionally, the configuration includes a time server. This provides temporal context to the LLM, enabling it to answer time-sensitive questions such as those involving “Year to Date” metrics more accurately.

6.3 The Expert Test Results

The expert (the author of this report), who has extensive knowledge of the systems and data structure, attempted to solve the five selected questions defined in collaboration with AAU Innovate (see Section 6.1). Each question was answered by writing custom Python scripts and manually issuing queries to the data platform. The steps taken and complete output are documented in Appendix D. Below, a brief summary

[2]Issue reported to developer. A workaround was implemented by forcing a dummy argument to ensure valid JSON transmission, as no fix is yet available.

of each task and outcome is presented, along with an estimate of the time spent, later to be compared with the LLM.

6.3.1 Question 1: Identifying the Busiest Days and Their Causes

To determine the busiest days at AAU Innovate and understand what caused the spikes in activity, entrance telemetry data was queried and aggregated by day. Total foot traffic was summed across all entrances, and calendar data from the same days was analyzed to identify large events and their expected attendance. The following steps were taken to complete the analysis:

1. Retrieve the unique identifier for the relevant building, which is `TMV25`.
2. Fetch entrance telemetry data for building `TMV25` starting from 2025-01-01.
3. Use a Python script to sum hourly entrance data into daily totals across all entrances.
4. Generate an overview of the busiest days based on total daily traffic.
5. For each of these dates, issue a query to the events endpoint to retrieve scheduled activities.
6. Write a Python script to process the event data and list events by estimated attendance.

This process can be repeated to produce results for all available data, not just Year to Date (YTD).

The analysis led to the following key results:

- The busiest day in 2025 was **2025-04-29** (1057 entries), primarily due to **Energy Research Day 2025** with more than 1000 expected attendees.
- **2025-04-03** (1027 entries) featured several small to medium-sized meetings, with no single large event.
- **2025-04-01** (930 entries) was driven by a combination of the **ALF meeting** and the **Game Tech Academy Partner Meeting**.
- The all-time peak occurred on **2024-10-02** (1109 entries), largely due to **Science Day**, supported by several smaller events.

The full details, including GraphQL queries, raw output, and event breakdowns, are available in Section D.1. **Estimated time spent:** 1 hour.

6.3.2 Question 2: Meeting Rooms Usage Analysis

Meeting rooms are a key resource at AAU Innovate. To evaluate their usage, Outlook calendar data from 2022 onward was queried and processed, excluding PIR data due to its unreliability. A typical workday was defined as 8 hours, allowing daily room usage to be expressed as a percentage. The following steps were taken:

1. Query all calendar events for meeting rooms from 2022 onward, including `durationMinutes`.
2. Use Python to calculate daily usage percentages per room.
3. Identify peak usage days where rooms were booked well over 100%.

4. Compute long-term average usage per room from 2022 to 2025.
5. Segment the data into half-year intervals to detect usage trends over time.
6. Analyze occupancy per hour to find periods with the highest simultaneous usage.

This led to several key results:

- Some rooms showed extremely high booking rates on specific days (e.g., **TMV25-Stue-C.004** at 2593.8%).
- Usage peaked in late 2023 (32.32%) and has slightly declined since.
- Peak hourly demand never exceeded 14 out of 26 rooms booked simultaneously, disproving the notion of meeting room overuse.

Listing D.13 and related charts confirm that while some rooms are highly used, the building has adequate capacity for meetings. The full details, including GraphQL queries, raw output, and event breakdowns, are available in Section D.2.

Estimated time spent: 1.5 hours.

6.3.3 Question 3: Meetings with External Partners

Collaboration with external partners is central to AAU Innovate's mission. While meetings alone do not provide the full picture, they offer a strong proxy. To estimate how often external partners are involved, the following steps were taken:

1. Query all meetings from 2022 onward, including the `start` field to group by year and the `isExternal` flag within `departmentBreakdown` to identify external attendees.
2. Write a Python script to filter and count meetings per year that include at least one external participant.

This yielded the following results: External collaboration peaked in **2023** with **1146** meetings involving external partners, followed by a slight decline in **2024** (**823** meetings), reflecting a similar trend seen in room usage (Section 6.3.2). Notably, large events not tied to a specific meeting room are excluded, so the actual numbers are likely higher. The full details, including GraphQL queries, are available in Section D.3.

Estimated time spent: 30 minutes.

6.3.4 Question 4: Analyzing Room Usage

Understanding which rooms are most and least used is key to optimizing building layout and function. Due to the lack of access to lighting data, BMS PIR sensors were used as a proxy for motion-based activity. The following steps were taken:

1. Query all sensors in the building to identify potential PIR devices.
2. Use Python to filter sensors with `PIR` and `TM025` in their `sourcePath`.
3. Collect sensor data month-by-month from `2022-09-01` onward using the filtered `externalIDs`.

4. Calculate active hours by measuring the time between `1.0` (motion start) and the next `0.0` (motion end).

5. Correlate sensor activity with room names and total hours per room.

6. Summarize usage both since 2022 and for 2025 year-to-date.

Querying all rooms at once failed due to response size limits, and because the BMS system lacks metadata for identifying sensor types, data had to be fetched in smaller batches. A start date of `2022-09-01` was selected based on availability, and data collection proceeded monthly per sensor using an iterative script.

The results reveal:

- **Most used rooms**: A.115, B.106b, A.314, B.008, A.215

- **Least used rooms**: A.106, A.001b, B.111, A.310, A.011

Rooms like A.314 and A.115 consistently show high activity, while others like A.310 and B.111 appear largely unused. These insights are valuable for space planning and reassignment, though they should not be taken literally. PIR sensors may be imperfectly placed, and other contextual factors can affect the data. The full details, including GraphQL queries and raw output, are available in Section D.4.

Estimated time spent: 3–4 hours.

6.3.5 Question 5: IT Lab Change Detection

To evaluate whether the IT Lab at AAU Innovate experienced increased usage following a functional change in January 2025, PIR sensor data was analyzed. The dataset from Section 6.3.4 could be reused for this purpose. While anecdotal evidence suggested a rise in activity, this analysis aimed to quantify that change.

1. Identify rooms A.101a and A.101b as the IT Lab using Innovate's floor plan.

2. Use the same method from Section 6.3.4 to collect monthly PIR data for these rooms.

3. Write a Python script to compute monthly active hours and month-over-month percentage changes.

4. Perform a year-over-year comparison to correct for seasonal variation.

The results show a marked increase starting in **January 2025**, with substantial growth in February and March, as shown in Listing 6.3. Compared to the same months in 2024, usage increased by an average of **66%** between February and April.

```
  Month        A.101a (Hours / % YoY) A.101b (Hours / % YoY)
-----------------------------------------------------------------
  2025-01:  239.45h (-1.1%)         201.99h (-3.5%)
  2025-02:  253.57h (+22.4%)        228.63h (+54.4%)
  2025-03:  287.78h (+78.1%)        269.05h (+103.4%)
  2025-04:  229.74h (+40.3%)        191.65h (+40.2%)
```

Listing 6.3: Monthly usage hours of the IT Lab compared to the same months in the previous year.

These results strongly suggest that the policy change successfully increased the IT Lab's usage. As this question is treated as a separate instance, the time estimate includes data fetching. The full details, including GraphQL queries and raw output, are available in Section D.5.

Estimated time spent: 3 hours.

Chapter 7

The LLM Test Results

This chapter evaluates the LLM solution's capability to answer data-related insight questions. Initially, the models selected in Section 5.2 are tested, leading to the identification of a final model in Section 7.1. Subsequently, the chosen solution is assessed using the questions listed in Section 6.1, and its responses are compared against expert answers provided in Section 6.3. The expert's ability to address all questions underscores the data's potential for generating insights.

7.1 Final Model Selection

As described in Section 5.2, four language models were shortlisted based on benchmarks, model size, and general capabilities. While these criteria provide an initial indication of performance, selecting the best model for production requires practical evaluation on real-world tasks.

As the Evaluation Strategy, Part 1 of Question 1 (see Section 6.3.1) was selected as the primary benchmark. This question involves identifying the busiest days of the year in the building `TMV25`, which requires:

1. Calling the `_get_current_time_` endpoint to determine the current year.
2. Using the `_introspect_schema_` endpoint to understand the available data.
3. Querying the `buildings` endpoint with `TMV25` as the ID and requesting all `telemetryData` from each `entrance`, starting from the beginning of the year.
4. Parsing the telemetry data to calculate visitor counts per day.
5. Sorting these counts in descending order and returning the top 10 days.

This task is challenging due to the nested data structure, large telemetry volume, and need for cross-day aggregation and tool use. Some interactions with each model are listed in Appendix E.

7.1.1 Qwen3

Qwen3 performed well in identifying required steps, including schema introspection, query formulation, and temporal reasoning. It:

- Understood tool use and made correct GraphQL queries.
- Called the right query and got back the required data.
- Attempted Python script generation for aggregation tasks.

However, it ultimately failed to deliver a complete result due to:

- Exceeding the context window during large telemetry data processing.
- Inability to iteratively debug or correct execution logic in external code.

7.1.2 LLaMA 4

LLaMA 4 demonstrated basic schema understanding and could generate plausible query templates. However, it struggled to:

- Use tools effectively, often returning Python-style pseudocode instead of actionable queries.
- Maintain state across multi-step interactions due to prompt truncation.
- Complete the aggregation logic required for the task.

Its failure to execute the required data analysis made it unsuitable for this use case.

7.1.3 Watt Tool

Although the Watt Tool model showed excellent awareness of available tools and could suggest accurate tool calls, it:

- Lacked the capability to execute them.
- Only returned static suggestions without interaction.

Due to its static-only nature, it was disqualified from further evaluation and deemed unusable for this usecase.

7.1.4 Mistral

Mistral demonstrated several promising capabilities during the task:

- It exhibited an understanding of the overall multi-step process required, including data aggregation and sorting.
- It showed initiative by invoking the `_get_current_time_` endpoint and attempting schema introspection.
- It correctly identified the need to use the `requests` library for making API calls.

Despite these strengths, Mistral's output suffered from critical issues that ultimately impeded task completion:

- It consistently generated invalid GraphQL queries, referencing nonexistent fields such as `name` on the `Building` type and `visitorCount` on the `TelemetryData` type.
- It failed to correctly introspect and apply the schema structure, leading to logically flawed queries.
- The generated Python scripts included persistent syntax errors and incorrect use of query structures.

While Mistral demonstrated a solid understanding of the intended process, its inability to produce syntactically and semantically valid GraphQL queries, combined with persistent scripting errors, ultimately made it ineffective for successfully completing the task. It often generated strong initial results but consistently veered off course when asked to fix small bugs or make minor improvements.

7.1.5 Conclusion

This initial testing leads to the creation of the following overview table (see Table 7.1). The table uses checkmark syntax to provide a visual summary of how effectively each model approached solving the first question.

Each model was tested across at least three to four independent conversations. Conversations were continued until the author determined that the LLM's responses were no longer progressing or providing useful direction.

	Qwen3	**Mistral**	**LLama4**	**Watt Tool**
Able to call tools	✓	✓	✓	×
Capable of multi-step reasoning ("deepthink")	✓	±	×	×
Correct GraphQL query construction	✓	±	×	×
Produced meaningful answer	×	×	×	×
Avg. response time	5m 11s	4m 56s	1m 46s	1m 41s
Median response time	4m 18s	2m 47s	1m 6s	59s

Table 7.1: An overview of model performance on question 1.

Both Qwen3 and Mistral demonstrated a good understanding of the task and were able to produce Python code that was close to functioning correctly, with only minor syntax issues. However, Mistral had more difficulty recovering from its own mistakes. When asked to fix small problems, it often lost track of the earlier context and drifted away from its initially promising answers. In contrast, Qwen3 handled follow-up interactions more effectively by retaining context and making focused corrections.

Although Qwen3 was slower than the other models, it ultimately proved to be the most reliable. It responded better to user input and showed a slightly deeper understanding of the problem through its more deliberate reasoning. For these reasons, all further experimentation was carried out exclusively with Qwen3. The slower performance was an acceptable tradeoff given the significantly higher reliability and consistency in task completion.

7.2 LLM Solution Evaluation

Next, `Qwen3` is used to solve questions 1 through 5. The evaluation in Table 7.2 summarizes its performance using checkmark symbols to indicate where the LLM succeeded, partially succeeded, or failed. The figure covers all five questions, with particular emphasis on how well the LLM understood the tasks, constructed queries, handled data, and generated accurate and extendable code.

For questions 4 and 5, the data retrieval process had already been completed by the expert using custom Python tools, due to the complexity and volume involved. Therefore, Qwen3 was only expected to analyze the provided data, not to extract it. The structure of the input data was explicitly known to the LLM, allowing it to focus on generating meaningful and functional Python code for analysis.

Qwen3	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**
Understood the question	✓	✓	✓	✓	±
Created the right query	✓	✓	✓	±	×
Got the required data	±	✓	✓	✓	✓
Analysed the data into an answer	±	✓	✓	✓	✓
Got a complete and correct answer	×	×	✓	±	±
Able to extend the program	×	±	✓	×	✓

Table 7.2: An overview of `Qwen3`'s performance to solve the questions

In Question 1, Qwen3 was able to understand the question and correctly query the time tool to determine the starting year for a year-to-date analysis. It then constructed and executed the appropriate query. Unfortunately, this led to an input prompt that exceeded the context window that Ollama can handle. Two additional attempts were made, this time requesting a Python script. However, these attempts were not successful and resulted in buggy output.

Question 2 involved a single long attempt. While it took some time to construct the correct query—and one query resulted in a server error despite being theoretically valid, the final result used the `rooms` endpoint instead of the `buildings` endpoint. Although functional in this context, this approach did not align with the specific request to retrieve data for the `TMV25` building.

An answer was eventually achieved, but it did not reach the quality of the expert answer. Despite being adaptable to follow-up instructions, it lacked essential correctness. The produced output is displayed in Listing 7.1.

```
Utilization Analysis Results:
Total meeting rooms analyzed: 237
Average daily utilization: 0.08 hours
Peak room utilization: 24.0 hours
Peak room name: TMV25 C.004 - Auditorium
Any room at full capacity (8h+): Yes
```

Listing 7.1: The output of `Qwen3`'s Python program to solve question 2.

This result is problematic. The number of meeting rooms is incorrect, as it included all rooms rather than just meeting rooms. While the script attempted to filter by type,which is uncluded in the dataset, but does not hold whether a room is a meeting room. The actual meeting room status should be inferred from whether the room has an email address. As a result, the average daily utilization and peak usage figures were misleading and did not add meaningful value. Therefore, this attempt received a crossmark for completeness and correctness.

In contrast, Question 3 proceeded very smoothly. As shown in Section 6.3.3, the expert also found this task relatively straightforward. The first attempt tried to solve the task directly in the LLM but ran into context limitations in Ollama. A subsequent attempt, in a fresh conversation and requesting a Python solution, was highly successful. One prompt resulted in a fully functional script and a correct answer. A follow-up request for more detailed yearly information was handled flawlessly. The LLM completed the initial task in 8 minutes and 7 seconds, clearly outperforming the expert in this case.

For Question 4, the LLM correctly analyzed the PIR sensor data and identified active periods based on `1.0`–`0.0` transitions. It produced a working Python script that sorted sensors by usage time. However, the output was hard to interpret due to raw seconds and a lack of room names. Attempts to query the BMS GraphQL API to map sensor IDs to room names failed due to incorrect request formats and endpoint mismatches. As shown in Listing 7.2, the code was functional but stranded, producing results with sensor IDs only and no readable time or room context. Later efforts to improve the code failed, as did attempts to start over from scratch and try again.

```
Least used rooms (sensor ID, active time in seconds):
Sensor 235558: 467095.00 seconds
Sensor 235564: 2663663.00 seconds

Most used rooms (sensor ID, active time in seconds):
Sensor 176740: 35440115.00 seconds
Sensor 179983: 38964240.00 seconds
```

Listing 7.2: The output of `Qwen3`'s Python program to solve question 4.

For **Question 5**, the LLM was tasked with detecting whether a functional change to the IT lab led to increased activity. It successfully extracted the `externalIDs` for rooms A.101a and A.101b using a Python script that parsed a nested JSON file. Initially, the LLM struggled, failing to construct the correct GraphQL query to find the IDs. However, when asked to write a script using a local file, it managed the task effectively.

The LLM then analyzed PIR trend logs for these sensors, calculating active durations between `1.0` and `0.0` transitions. This led to the result in Listing 7.3. While Qwen3 did produce a working program, it forgot previously gathered information, like the sensor IDs and data folder, when writing the second script. Furthermore, the analysis used January 2023 as the threshold instead of the correct date (January 2025), failing to utilize the time tool for dynamic year detection.

```
{
"176558": {
  "pre_avg_hours": 242.06,
  "post_avg_hours": 236.03,
  "increase_percent": -2.49
},
"176567": {
  "pre_avg_hours": 153.33,
  "post_avg_hours": 209.59,
  "increase_percent": 36.69
}
}
```

Listing 7.3: The initial output of `Qwen3`'s Python program to solve question 5.

The code, while functional, did not offer a clear or meaningful comparison. It simply splits data into "before" and "after" the change, without a temporal breakdown or better granularity. After further clarification, the LLM produced a refined and structured script to compare monthly usage before and after January 2025. As shown in Listing 7.4, the final result matches the expert output in Listing D.20, but expressed in seconds here.

```
Sensor 176567:
  Pre-Avg (2024): 522043.20h
  Post-Avg (2025): 754536.00h
  Overall Change: 44.54%
  Monthly Changes:
    Month 1: 753914.00h → 727156.00h (-3.55%)
    Month 2: 533177.00h → 823083.00h (54.37%)
    Month 3: 476187.00h → 968587.00h (103.40%)
    Month 4: 492066.00h → 689940.00h (40.21%)
    Month 5: 354872.00h → 563914.00h (58.91%)
Sensor 176558:
  Pre-Avg (2024): 650658.00h
  Post-Avg (2025): 849692.40h
  Overall Change: 30.59%
  Monthly Changes:
    Month 1: 871533.00h → 862004.00h (-1.09%)
    Month 2: 745553.00h → 912859.00h (22.44%)
    Month 3: 581764.00h → 1036012.00h (78.08%)
    Month 4: 589443.00h → 827048.00h (40.31%)
    Month 5: 464997.00h → 610539.00h (31.30%)
```

Listing 7.4: The final output of `Qwen3`'s Python program to solve question 5.

To compare, the entire conversation spanned **53 minutes**, of which the LLM actively reasoned for 12 minutes and 20 seconds. In contrast, the expert spent approximately 1 hour reasoning through the same question, not including the time required to obtain the data. This places the LLM's performance on par with the expert in terms of analytical time, despite the expert having prior familiarity with the task context. However, the LLM needed some expert help to get the right answer.

Chapter 8

Discussion

As outlined in Section 2.2, the central question initially posed was whether existing systems and sensors in smart buildings could be leveraged to gain meaningful insights into building usage. The concept of interfacing this data with an LLM emerged only later, as introduced in Chapter 5. This discussion will first evaluate the expert's results in Section 8.1, followed by an assessment of the LLM's performance in Section 8.2.

8.1 Expert Analysis and Insights

The expert successfully addressed all questions posed in collaboration with Innovate. These questions were carefully constructed with available data access in mind.

For **question 1**, the busiest days were estimated using laser sensor data. However, since these sensors do not cover a fourth entrance at Innovate, but only the three main doors, the resulting data may be an underestimate. To investigate the cause of peak activity, only Outlook meeting data was considered. This relied on the `formParticipants` field, which represents expected rather than actual attendance. Despite these limitations, the data still provided useful insights.

Question 2 was more straightforward. Assuming all meetings are correctly registered in the room calendar, which is standard procedure, the results can be considered reliable. These findings were particularly valuable as they countered the assumption that there are not enough meeting rooms available.

Question 3, however, yielded a significant underestimation. Major events were not accounted for, and the number of meetings alone says little about their importance or the number of attendees. While the task was completed quickly, the analysis fell short of addressing the deeper question of whether Innovate is attracting more external collaboration or whether existing collaboration simply moved into the new facilities.

Question 4 encountered challenges due to the large volume of data. A potential improvement would be to implement a cache system that continuously logs room activity based on BMS PIR sensors, making future analysis more efficient. Alternatively, lighting system data, when available, could improve precision. The results, while demanding to produce, were promising and likely valuable for Innovate's management.

Question 5 delivered particularly interesting insights, showing that a functional change had the desired effect. Since the analysis compared relative changes before and after the modification, the results were more robust against potential data inaccuracies.

8.2 Evaluating the LLM

The performance of the LLM was notably mixed. Setting up the model was complex, and despite access to powerful hardware, response times were relatively slow. Discussions with CLAAUDIA revealed that the bottleneck was not GPU speed but rather the interconnect between GPUs, as usage often remained below 20%. Nevertheless, response times remained within acceptable bounds for the experiment.

The initial goal—to have the LLM answer questions directly within a prompt without writing a Python script—quickly proved unfeasible. While the model could retrieve data using queries, it frequently ran into context window limitations. Although these parameters could be adjusted in Ollama, the data-intensive nature of the questions made such adjustments impractical. The query endpoint itself performed well and could be valuable in scenarios requiring smaller, focused data retrieval. However, the use case here required long-term analysis, making such an approach unsuitable.

As a result, the strategy shifted toward having the LLM generate Python programs. Given Python's strong support for data processing and the LLM's aptitude for code generation, this made sense. In many cases, the initial code output was impressive, and when it worked, extending or adapting the script was relatively easy. However, if bugs were present or additional logic was needed, progress often stalled. The LLM frequently forgot previously obtained context, such as specific years or endpoint URLs, and failed to reuse it in subsequent code.

Overall, the LLM was capable of writing functional Python scripts, but typically only when given one task at a time, such as in question 3, or after breaking down tasks into steps, as was done in question 5. More complex requests, such as those in question 1, or tasks that required combining code with external data queries, were consistently unsuccessful. It also never demonstrated a fundamentally different approach to solving a problem, instead trailing behind the expert in both reasoning and creativity.

In conclusion, while the experiment showed that the LLM could answer simpler data questions independently or with minor guidance, it struggled with complex tasks, even when aided by expert knowledge. It often failed to complete the full pipeline from data to insight. Even when it succeeded, the time saved was marginal. The most promising path forward is likely for experts to use LLMs as assistants for generating Python scripts, especially when the model has access to the GraphQL schema and can understand small, well-scoped examples. With this setup, the LLM could help rapidly scaffold analysis tools that the expert then completes or validates.

Conclusion

This project demonstrates that integrating heterogeneous sensor and system data from a smart building, such as FMS, BMS, door counters, coffee machines, and meeting calendars, can provide actionable insights that support evidence-based decision-making. The successful creation of a unified, hostable data graph for AAU Innovate proves the feasibility of connecting multiple systems into a coherent structure that management can use for ongoing monitoring and strategic planning.

The project's first achievement is getting access to certain data systems, both procedural and technical. This was proven to be a sometimes difficult challenge. As well as documenting the different systems and the status/ possibilities with them. Next, the integrated GraphQL federation was created. Making an easy, documented way of data collection at AAU Innovate possible, with possibilities to extend to other buildings at AAU. As well a the creation of a system architecture which is scalable for new systems to integrate in, when rolled out or access is achieved. Apart from a platform, which can be used for future research, it is a proof of concept, which can be used to roll out data collection over all of AAU, as imagined by AAU building support.

To validate the value of this integration, five relevant data-driven questions were formulated in collaboration with AAU Innovate. These were then solved by the report's author, acting as the expert. Importantly, all five questions led to actionable insights, clear evidence of what becomes possible only through the unified data scheme. These results establish a strong benchmark and proof-of-concept for how integrated building data can empower better decisions and deeper understanding of real usage patterns.

In exploring the potential of using LLMs as an interface to this data, results were more mixed. While the LLM could not handle complex queries directly within prompts, due to limited memory and context size, it performed well in generating Python scripts for single, focused tasks. These scripts occasionally reached expert-level quality and produced valuable outputs, particularly when tasks were broken into smaller steps.

However, limitations remained. The LLM struggled with multi-step logic, remembering previously gathered context, and chaining tools together effectively. It did not propose novel approaches and consistently lagged behind expert performance in more complex cases. Despite this, for simpler questions, the time-to-solution was often comparable to human effort, suggesting potential as a useful coding assistant.

These findings suggest that LLMs may not yet replace expert data analysts but could significantly enhance their workflow. Used interactively, and especially with access to structured help, such as GraphQL schema awareness or retrieval-augmented generation (RAG), LLMs could accelerate scripting and lower the barrier to insight.

Although the LLM results were limited, the system still proved highly valuable to AAU Innovate. The answers were presented to them and led to positive feedback and useful insights, demonstrating the potential of the acquired data access and platform. While a non-technical user may not yet be able to independently extract meaningful insights using only the LLM, a technically proficient student assistant could leverage the platform to explore new questions as they arise or to investigate observed trends more deeply. A particularly valuable aspect is the ability to use the data to reflect on policy changes, as illustrated by question 5.

Future Work

Looking ahead, improvements can be made across several dimensions. The MCP server could be extended with more tools beyond just the `query` function, allowing LLMs to operate more autonomously. While stronger models like Gemini or OpenAI's GPT currently outperform local open-source models, they cannot be deployed self-hosted due to licensing and privacy constraints. Nonetheless, they offer a glimpse of what open-source models will likely achieve in the near future as they continue to improve.

The system architecture also makes it straightforward to integrate additional data sources, such as lighting systems or upcoming machine usage data, enabling richer insights into room activity and operational efficiency. Making computation-heavy metrics, such as active room hours or daily building occupancy, available directly in the GraphQL schema, and optionally caching them server-side, could significantly enhance both performance and user experience.

Future research may extend this platform toward optimization, e.g., improving heating strategies using not only BMS data but also calendar and entrance data. It could also investigate how RAG or pre-defined code snippets can guide LLMs toward more accurate, context-aware answers. Another direction involves designing "shortcuts" that offload heavy computations to the server, making frequent queries, such as the number of people in the building on a given day, efficient and scalable.

In summary, this work proves that with the right data architecture, valuable building insights are within reach. While LLMs are not yet ready to take over the analyst's role, they show promise as co-pilots—especially when the system is tailored to their strengths.

References

[1] K. Lawal and H. N. Rafsanjani, “Trends, benefits, risks, and challenges of IoT implementation in residential and commercial buildings,” *Energy and Built Environment*, vol. 3, no. 3, pp. 251–266, 2022.

[2] J. Aguilar, A. Garcès-Jimènez, N. Gallego-Salvador, J. A. G. De Mesa, J. M. Gomez-Pulido, and À. J. Garc\ia-Tejedor, “Autonomic management architecture for multi-HVAC systems in smart buildings,” *IEEE Access*, vol. 7, pp. 123402–123415, 2019.

[3] B. Dong, V. Prakash, F. Feng, and Z. O'Neill, “A review of smart building sensing system for better indoor environment control,” *Energy and Buildings*, vol. 199, pp. 29–46, 2019.

[4] S. D’Oca, T. Hong, and J. Langevin, “The human dimensions of energy use in buildings: A review,” *Renewable and Sustainable Energy Reviews*, vol. 81, pp. 731–742, 2018.

[5] F. Wang *et al.*, “Predictive control of indoor environment using occupant number detected by video data and CO2 concentration,” *Energy and Buildings*, vol. 145, pp. 155–162, 2017.

[6] T. Leephakpreeda, “Adaptive occupancy-based lighting control via grey prediction,” *Building and environment*, vol. 40, no. 7, pp. 881–886, 2005.

[7] N. Maynard and D. Grant, “Smart Buildings & the Battle for Sustainability,” Basingstoke, Hampshire, UK, Mar. 2022. [Online]. Available: http://www.juniperresearch.com/

[8] Density, “Unused Space in Buildings Generates 22M Tons of CO\textsubscript{2}.” [Online]. Available: https://www.density.io/resources/how-companies-can-take-climate-action-with-occupancy-data

[9] R. K. Garrett, “Should We Worry That Half of Americans Trust Their Gut to Tell Them What’s True?.” [Online]. Available: https://theconversation.com/should-we-worry-that-half-of-americans-trust-their-gut-to-tell-them-whats-true-84259

[10] T. Stobierski, “Data-Driven Decision-Making.” [Online]. Available: https://online.hbs.edu/blog/post/data-driven-decision-making

[11] K. Bäcklund, P. Lundqvist, and M. Molinari, “Showcasing a digital twin for higher educational buildings: developing the concept toward human centricity,” *Frontiers in Built Environment*, vol. 10, p. 1347451, 2024.

[12] Z. Ni, C. Zhang, M. Karlsson, and S. Gong, “Leveraging Deep Learning and Digital Twins to Improve Energy Performance of Buildings,” in *2023 IEEE 3rd International Conference on Industrial Electronics for Sustain-*

able Energy Systems (IESES), IEEE, Jul. 2023, pp. 1–6. doi: 10.1109/ieses53571.2023.10253721.

[13] S. Anik, X. Gao, N. Meng, P. Agee, and A. McCoy, "A Cost-Effective, Scalable, and Portable IoT Data Infrastructure for Indoor Environment Sensing." [Online]. Available: https://arxiv.org/abs/2110.14042

[14] D. Hugo *et al.*, "A smart building semantic platform to enable data re-use in energy analytics applications: the Data Clearing House." [Online]. Available: https://arxiv.org/abs/2311.11630

[15] A. Floris, S. Porcu, R. Girau, and L. Atzori, "An iot-based smart building solution for indoor environment management and occupants prediction," *Energies*, vol. 14, no. 10, p. 2959, 2021.

[16] Z. Chen, C. Jiang, and L. Xie, "Building occupancy estimation and detection: A review," *Energy and Buildings*, vol. 169, pp. 260–270, 2018, doi: https://doi.org/10.1016/j.enbuild.2018.03.084.

[17] S. S. Kumar, R. Chandra, and S. Agarwal, "A Real-Time Approach for Smart Building Operations Prediction Using Rule-Based Complex Event Processing and SPARQL Query." [Online]. Available: https://arxiv.org/abs/2309.10822

[18] K. H. Andersen *et al.*, "Exploring occupant detection model generalizability for residential buildings using supervised learning with IEQ sensors," *Building and Environment*, vol. 254, Apr. 2024, doi: 10.1016/j.buildenv.2024.111319.

[19] I. P. Mohottige, H. H. Gharakheili, T. Moors, and V. Sivaraman, "Modeling Classroom Occupancy Using Data of WiFi Infrastructure in a University Campus," *IEEE Sensors Journal*, vol. 22, no. 10, pp. 9981–9996, May 2022, doi: 10.1109/jsen.2022.3165138.

[20] V. Bellavista-Parent, J. Torres-Sospedra, and A. Perez-Navarro, "New trends in indoor positioning based on WiFi and machine learning: A systematic review," in *2021 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, IEEE, Nov. 2021, pp. 1–8. doi: 10.1109/ipin51156.2021.9662521.

[21] Y. Wang and L. Shao, "Understanding occupancy and user behaviour through Wi-Fi-based indoor positioning," *Building Research & Information*, vol. 46, no. 7, pp. 725–737, 2018.

[22] A. Innovation, "AAU Innovation - Data." Accessed: Oct. 05, 2023. [Online]. Available: https://www.touch.innovate.aau.dk/aau-innovation-data

[23] FM:Systems, "Top Office Space Trends for Hybrid Work Success." Accessed: Apr. 02, 2025. [Online]. Available: https://fmsystems.com/blog/top-office-space-trends-hybrid-work-success/

[24] Microsoft Learn, "Create and manage room mailboxes in Exchange Server." Accessed: Jul. 03, 2025. [Online]. Available: https://learn.microsoft.com/en-us/exchange/recipients/room-mailboxes?view=exchserver-2019

[25] Microsoft Corporation, "List events - Microsoft Graph v1.0." Accessed: Apr. 02, 2025. [Online]. Available: https://learn.microsoft.com/en-us/graph/api/group-list-events?view=graph-rest-1.0&tabs=http

[26] Schneider Electric, "EcoStruxure Building Operation." Accessed: Apr. 03, 2025. [Online]. Available: https://www.se.com/uk/

en/product-range/62111-ecostruxure-building-operation/?parent-subcategory-id=26426255&filter=business-2-building-automation-and-control

[27] S. P. Melgaard *et al.*, "Fault Detection in AHU: A Walkthrough for Implementation in a Danish Educational Building." 2024.

[28] N&W GLOBAL VENDING GROUP, "OPERATOR MANUAL Wittenborg 9100 ES 2FB." Via Roma 24 - 24030 Valbrembo (BG) Italy, 2017. Accessed: Oct. 27, 2023. [Online]. Available: https://f.hubspotusercontent20.net/hubfs/6668578/Manuals/Wittenborg9100-3_Manual.pdf

[29] Multiteknik, "Kundetæller – kom i gang på under en time!." Accessed: Apr. 03, 2025. [Online]. Available: https://www.multiteknik.dk/produkter/kundetaeller-til-butikker/

[30] NVC Lighting, "What is DALI-8." Accessed: May 22, 2025. [Online]. Available: https://www.nvcuk.com/technical-support/view/what-is-dali-8

[31] Zumtobel Group, "LITECOM REST Handbook." Accessed: May 22, 2025. [Online]. Available: https://resources.zumtobel.com/assets/downloads/litecom/manuals/LITECOM_REST_HB_en.pdf

[32] M. Vega-Barbas, M. Álvarez-Campana, D. Rivera, M. Sanz, and J. Berrocal, "AFOROS: A Low-Cost Wi-Fi-Based Monitoring System for Estimating Occupancy of Public Spaces," *Sensors*, vol. 21, no. 11, p. 3863, Jun. 2021, doi: 10.3390/s21113863.

[33] S. Liu, Y. Zhao, F. Xue, B. Chen, and X. Chen, "DeepCount: Crowd counting with WiFi via deep learning," *arXiv preprint arXiv:1903.05316*, 2019.

[34] J. Dai, M. Wang, B. Wu, J. Shen, and X. Wang, "A Survey of Latest Wi-Fi Assisted Indoor Positioning on Different Principles," *Sensors*, vol. 23, no. 18, 2023, doi: 10.3390/s23187961.

[35] A. Le Floch, R. Kacimi, P. Druart, Y. Lefebvre, and A.-L. Beylot, "A comprehensive framework for 5G indoor localization," *Computer Communications*, vol. 228, p. 107968, 2024, doi: https://doi.org/10.1016/j.comcom.2024.107968.

[36] GraphQL Foundation, "GraphQL: A query language for your API." Accessed: May 02, 2025. [Online]. Available: https://graphql.org/

[37] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," 2000. Accessed: May 02, 2025. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[38] G. Bello, "GraphQL vs. REST." Accessed: May 02, 2025. [Online]. Available: https://blog.postman.com/graphql-vs-rest/

[39] M. DeBergalis, "The Future of MCP is GraphQL." Accessed: May 22, 2025. [Online]. Available: https://www.apollographql.com/blog/the-future-of-mcp-is-graphql

[40] "Federation." Accessed: May 02, 2025. [Online]. Available: https://www.apollographql.com/docs/graphos/schema-design/federated-schemas/federation

[41] The gqlgen team, "gqlgen." Accessed: May 02, 2025. [Online]. Available: https://gqlgen.com/

[42] S. Khayalian, "Docker vs. Docker Compose: Simple and Fun Explanation." Accessed: May 02, 2025. [Online]. Available: https://medium.com/@ShantKhayalian/docker-vs-docker-compose-simple-and-fun-explanation-4811582127f7

[43] Python Package Index, "pywin32." Accessed: May 23, 2025. [Online]. Available: https://pypi.org/project/pywin32/

[44] Microsoft Learn, "Application object (Outlook)." Accessed: May 23, 2025. [Online]. Available: https://learn.microsoft.com/en-us/office/vba/api/outlook.application

[45] Microsoft Learn, "Outlook Object Model Overview (Outlook VBA reference)." Accessed: May 23, 2025. [Online]. Available: https://learn.microsoft.com/en-us/office/vba/api/overview/outlook/object-model

[46] "ThingsBoard." Accessed: May 02, 2025. [Online]. Available: https://thingsboard.io/

[47] T. Sassmannshausen, "How to get your company's data into an LLM: a story of fine-tuning and RAG," *Medium*, Jan. 2024, Accessed: May 13, 2025. [Online]. Available: https://medium.com/@till.sassmannshausen/how-to-get-your-companys-data-into-an-llm-b6e2729a81a1

[48] Prompt Engineering Guide, "Retrieval Augmented Generation (RAG) for LLMs." Accessed: May 13, 2025. [Online]. Available: https://www.promptingguide.ai/research/rag

[49] Prompt Engineering Guide, "Function Calling with LLMs." Accessed: May 13, 2025. [Online]. Available: https://www.promptingguide.ai/applications/function_calling

[50] Anthropic, "Introducing the Model Context Protocol." Accessed: May 12, 2025. [Online]. Available: https://www.anthropic.com/news/model-context-protocol

[51] E. Lisowski, "MCP Explained: The New Standard Connecting AI to Everything." Accessed: May 12, 2025. [Online]. Available: https://medium.com/@elisowski/mcp-explained-the-new-standard-connecting-ai-to-everything-79c5a1c98288

[52] Ollama, "qwen3." Accessed: May 13, 2025. [Online]. Available: https://ollama.com/library/qwen3

[53] Ollama, "llama4." Accessed: May 15, 2025. [Online]. Available: https://ollama.com/library/llama4

[54] Ollama, "mixtral." Accessed: May 15, 2025. [Online]. Available: https://ollama.com/library/mixtral

[55] Ollama, "mistral-large." Accessed: May 15, 2025. [Online]. Available: https://ollama.com/library/mistral-large

[56] Nous Research, "Hermes 3." Accessed: May 15, 2025. [Online]. Available: https://nousresearch.com/hermes3/

[57] Ollama, "command-r-plus." Accessed: May 15, 2025. [Online]. Available: https://ollama.com/library/command-r-plus

[58] Ollama, "dbrx." Accessed: May 15, 2025. [Online]. Available: https://ollama.com/library/dbrx

[59] The Mosaic Research Team, "Introducing DBRX: New State-of-Art Open LLM," *Databricks Blog*, Mar. 2024, Accessed: May 15, 2025. [Online]. Available: https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm

[60] fireshihtzu, "watt-tool-70B-formatted." Accessed: May 21, 2025. [Online]. Available: https://ollama.com/fireshihtzu/watt-tool-70B-formatted

[61] Mistral AI team, "Mixtral of experts," *Mistral AI Blog*, Dec. 2023, Accessed: May 15, 2025. [Online]. Available: https://mistral.ai/news/mixtral-of-experts

[62] A. Gomez, "Introducing Command R+: A Scalable LLM Built for Business," *Cohere Blog*, Apr. 2024, Accessed: May 15, 2025. [Online]. Available: https://cohere.com/blog/command-r-plus-microsoft-azure

[63] Mistral AI team, "Large Enough," *Mistral AI Blog*, Jul. 2024, Accessed: May 15, 2025. [Online]. Available: https://mistral.ai/news/mistral-large-2407

[64] R. Teknium, J. Quesnelle, and C. Guang, "Hermes 3 Technical Report." [Online]. Available: https://arxiv.org/abs/2408.11857

[65] blurrah, "mcp-graphql." 2025.

[66] T. Balarabe, "What is Ollama? Running Large Language Models Locally," *Medium*, Mar. 2025, Accessed: May 13, 2025. [Online]. Available: https://medium.com/@tahirbalarabe2/what-is-ollama-running-large-language-models-locally-e917ca40defe

[67] mark3labs, "mcp-go." Accessed: May 13, 2025. [Online]. Available: https://github.com/mark3labs/mcp-go

[68] "GeForce RTX 4090 Graphics Card." Accessed: May 28, 2025. [Online]. Available: https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4090/

[69] "High Performance Computing (HPC)." Accessed: May 28, 2025. [Online]. Available: https://hpc.aau.dk/

[70] "AI Lab System Overview." Accessed: May 28, 2025. [Online]. Available: https://hpc.aau.dk/ai-lab/system-overview/

[71] "NVIDIA Tesla T4 GPU." Accessed: May 28, 2025. [Online]. Available: https://www.nvidia.com/en-us/data-center/tesla-t4/

[72] "uCloud Providers." Accessed: May 28, 2025. [Online]. Available: https://hpc.aau.dk/ucloud/providers/

[73] "NVIDIA A40 GPU." Accessed: May 28, 2025. [Online]. Available: https://www.nvidia.com/en-us/data-center/a40/

[74] "NVIDIA L40 GPU." Accessed: May 28, 2025. [Online]. Available: https://www.nvidia.com/en-us/data-center/l40/

[75] K. K. Se, "🦸🏻‍♂️#14: What Is MCP, and Why Is Everyone – Suddenly!– Talking About It?." Accessed: May 12, 2025. [Online]. Available: https://huggingface.co/blog/Kseniase/mcp

[76] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions," *arXiv*, Apr. 2025, Accessed: May 12, 2025. [Online]. Available: https://arxiv.org/abs/2503.23278

[77] "Core architecture - Model Context Protocol." Accessed: May 12, 2025. [Online]. Available: https://modelcontextprotocol.io/docs/concepts/architecture

[78] "Transports - Model Context Protocol." Accessed: May 12, 2025. [Online]. Available: https://modelcontextprotocol.io/docs/concepts/transports

Appendix A

AAU Innovate's Layout

This section details the layout of the AAU Innovate building, which serves as the case study for this report. The building comprises five levels, ranging from the basement to the third floor. The detailed floor plans for each level are illustrated in the following figures: Figure A.2, A.3, A.4, A.5 and A.6 To provide context for the floor plans, Figure A.1 shows the exterior view of the AAU Innovate building.

Figure A.1: The AAU Innovate building viewed from the outside.

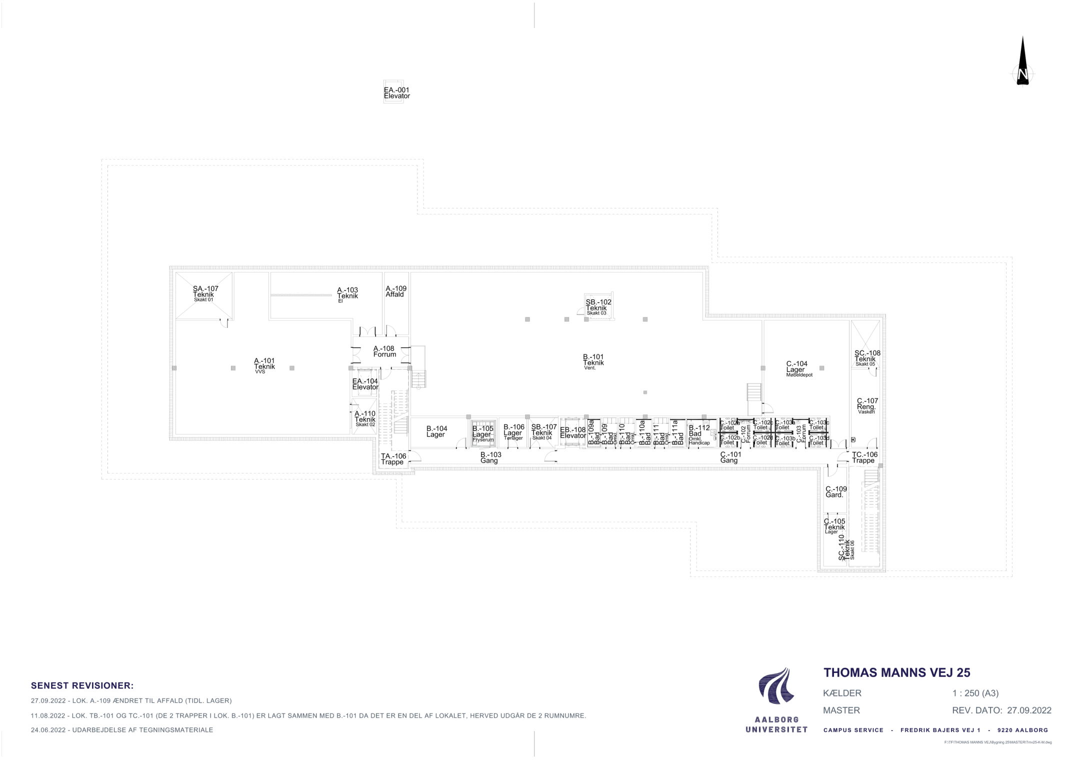

Figure A.2: The floorplan of the basement floor of the AAU Innovate building.

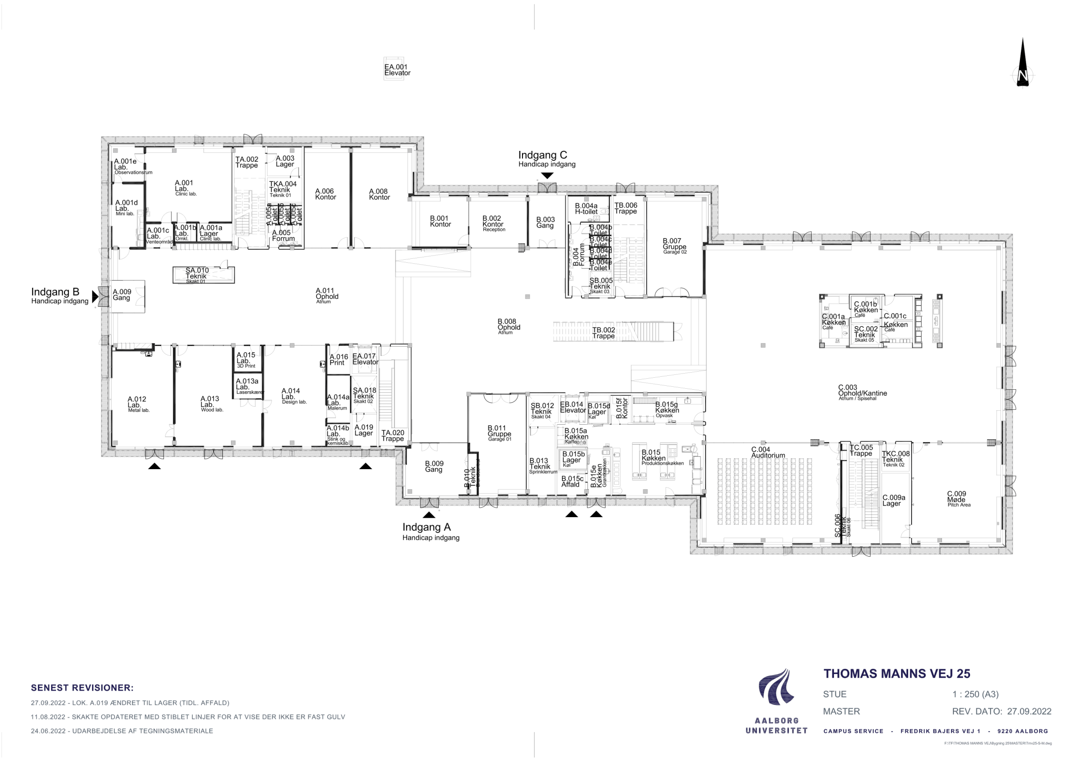

Figure A.3: The floorplan of the ground floor of the AAU Innovate building.

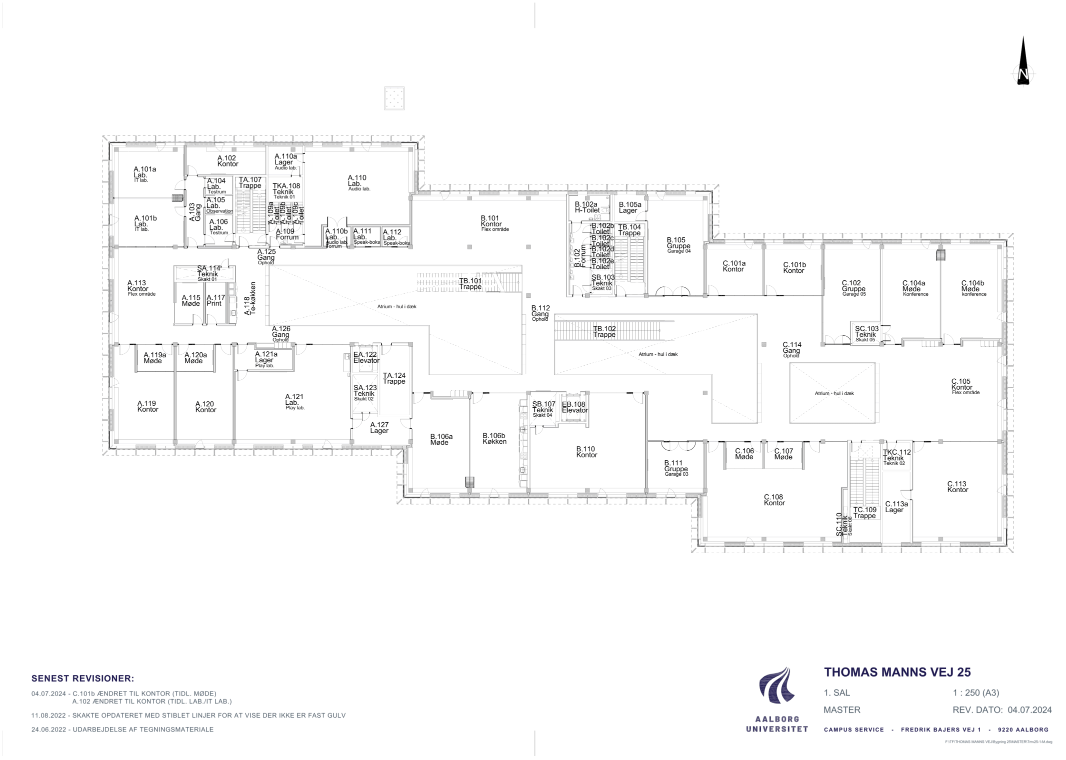

Figure A.4: The floorplan of the first floor of the AAU Innovate building.

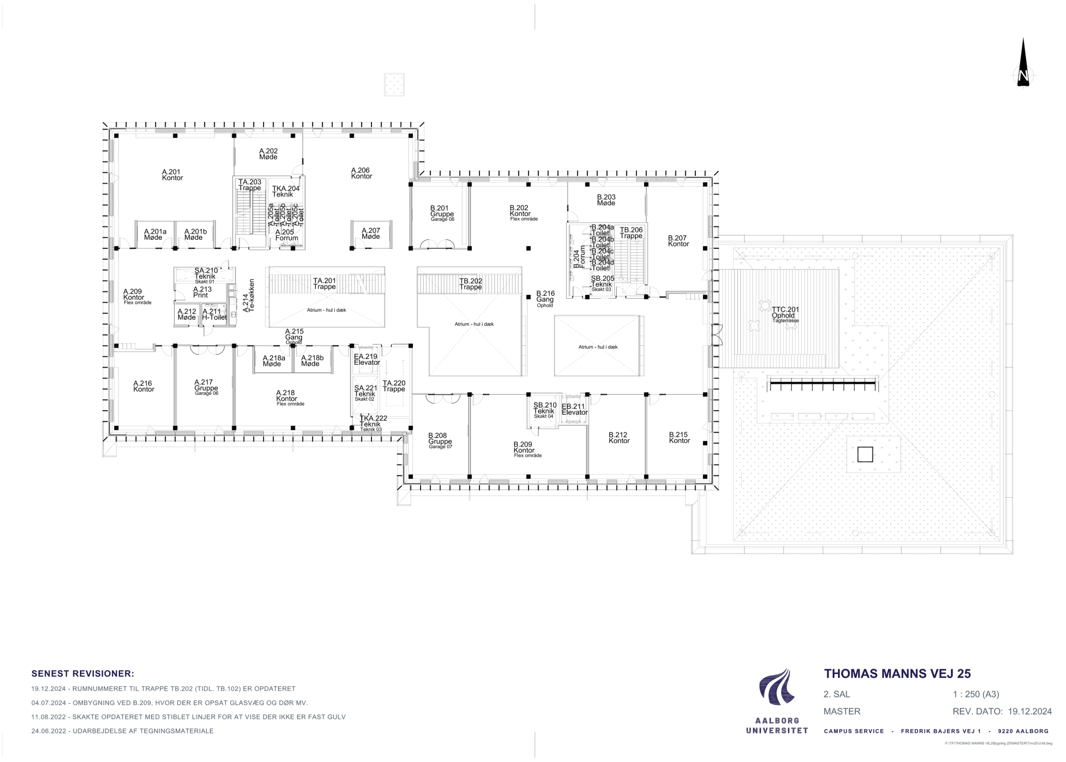

Figure A.5: The floorplan of the second floor of the AAU Innovate building.

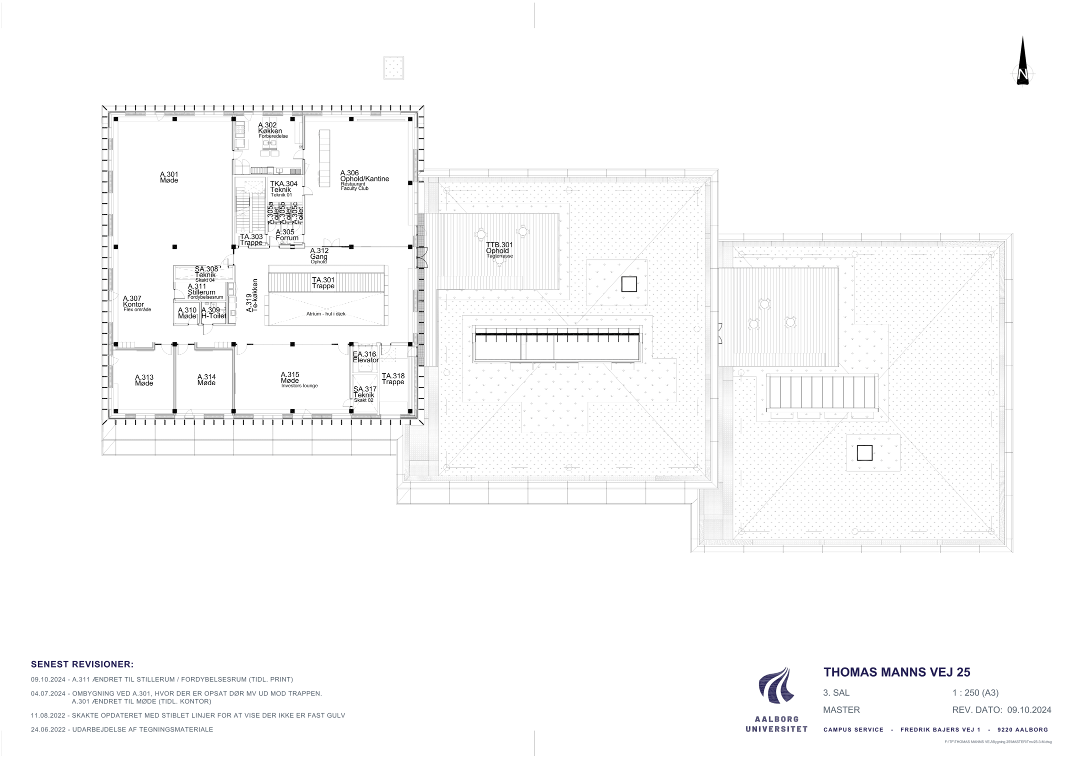

Figure A.6: The floorplan of the third floor of the AAU Innovate building.

Appendix B

Hosting the Data Application

The data application is hosted using Docker containers, orchestrated through Docker Compose to manage inter-container networking and simplify service configuration. This setup is illustrated in Listing B.1.

```
# Stage 1: Builder
FROM golang:1.24-alpine AS builder

# Define an argument for the application source directory relative to the
context
ARG APP_SRC=.

WORKDIR /app

# Copy only necessary files first for layer caching
COPY ${APP_SRC}/go.mod ${APP_SRC}/go.sum ./
RUN go mod download

# Copy the specific application source code using the build argument
COPY ${APP_SRC} ./

# Build the application binary, ensure static linking
RUN CGO_ENABLED=0 GOOS=linux go build -ldflags="-w -s" -o /app-binary .

# Stage 2: Runner
FROM alpine:latest

# Create a non-root user for security
RUN adduser -D -u 10001 appuser
WORKDIR /app

# Copy the compiled binary from the builder stage
COPY --from=builder /app-binary .
# Change ownership if using a non-root user
RUN chown appuser:appuser /app/app-binary

# Switch to the non-root user
USER appuser
```

Listing B.1: The Dockerfile used by all the GoLang individual graphs

Each Go-based microservice is built using a multi-stage Dockerfile. The first stage uses the `golang` image to compile the source code into a statically linked binary, optimizing for size and performance. In the second stage, the compiled binary is transferred to a lightweight `alpine` image. For enhanced security, a non-root user is created and assigned ownership of the binary. This practice reduces the attack surface in production environments.

The gateway, a Node.js application, uses a straightforward setup based on the official Node.js image. It installs dependencies via `npm ci` for reproducible builds, and then copies over the core application file. This configuration is shown in Listing B.2.

```
# Use a specific, minimal Node.js image
FROM node:22-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy package.json and package-lock.json (or yarn.lock, pnpm-lock.yaml)
first
# This allows Docker to cache this layer, speeding up builds if only code
changes
COPY package*.json ./

# Install dependencies
RUN npm ci --production

# Copy the rest of the application code
COPY ./server.js ./server.js
```

Listing B.2: The Dockerfile for the NodeJS gateway.

To coordinate all services, a `docker-compose.yaml` file is used. This file defines how containers should be built and configured, including port mappings, environment variables, and command overrides. It also specifies volumes and context paths for individual services. A portion of this configuration is shown in Listing B.3. Running `docker compose up` brings up the entire system, whether locally or on a remote server, thanks to Docker's platform-independent nature.

```yaml
services:
  gateway:
    build:
      context: ./service-gateway
    ports:
      - "4000:4000"
    environment:
      - APP_LISTEN_PORT=4000
    command: ["node", "server.js"]

  doorcounters:
    build:
      context: .
      args:
        APP_SRC: ./service-doorcounters
    ports:
      - "4001:4001"
    environment:
      - APP_LISTEN_PORT=4001
    env_file:
      - ./service-doorcounters/.env
    command: ["./app-binary"]

...All other subgraph systems

  coffee:
    build:
      context: .
      args:
        APP_SRC: ./service-coffee
    ports:
      - "4005:4005"
    environment:
      - APP_LISTEN_PORT=4005
    env_file:
      - ./service-coffee/.env
    command: ["./app-binary"]
```

Listing B.3: The Docker Compose file defines the interaction between all the containers.

Appendix C

The Model Context Protocol

The Model Context Protocol (MCP) is an open standard developed by Anthropic to facilitate seamless integration between LLMs and external tools, data sources, and services [50], [51], [75], [76]. Introduced in late 2024, MCP addresses the complexities associated with integrating AI systems into real-world applications by providing a standardized protocol for communication between AI assistants and various resources.

C.1 Motivation and Problem Statement

With the growing adoption of AI-driven applications, the need to connect models to a wide variety of external tools, services, and data sources has introduced significant complexity. Traditionally, every AI application (N) required bespoke integrations with each tool or system (M), resulting in an N×M integration problem. This pattern is inherently unscalable and creates major barriers to rapid AI development and deployment.

Prior to the introduction of MCP, developers relied on ad hoc methods such as manual API wiring, plugin-based interfaces, or specialized agent frameworks to enable tool interactions. These approaches not only increased development overhead but also resulted in fragmented, fragile, and hard-to-maintain systems.

As illustrated in Figure C.1, provided by X. Hou, Y. Zhao, S. Wang, and H. Wang [76], each tool integration previously required defining custom interfaces, authentication mechanisms, and data handling logic. In contrast, MCP offers a universal, standardized protocol for AI-to-tool communication. This abstraction significantly reduces integration effort by allowing AI agents to interact with diverse tools through a shared, structured interface.

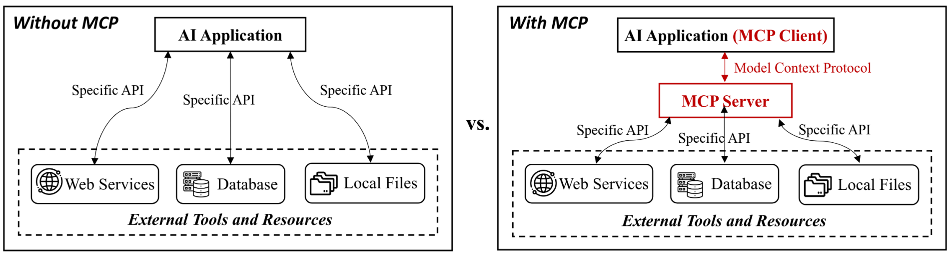

Figure C.1: Comparison of tool invocation with and without MCP [76].

By eliminating the need for tool-specific wiring and enabling dynamic discovery and invocation of external capabilities, MCP improves scalability, reduces duplication of effort, and promotes interoperability. This shift lays the foundation for more flexible, maintainable, and autonomous AI systems capable of handling real-world workflows.

C.2 Architecture Overview

MCP follows a modular client-server architecture designed to standardize and simplify interactions between AI applications and external tools, services, and data sources. The architecture comprises three main components:

- **MCP Client:** Typically embedded in the AI system, the client is responsible for initiating requests, managing interactions, and interpreting responses. It represents the interface through which the model accesses external capabilities.
- **MCP Server:** An independent service that exposes the functionality of a specific external tool or dataset. It translates structured requests from the client into concrete actions, such as file operations, API calls, or database queries.
- **Services:** The actual external systems (e.g., file systems, APIs, databases) that the MCP server interfaces with to perform the requested operations.

Communication between the client and server is conducted using structured data formats, primarily JSON, to ensure clear semantics, cross-language compatibility, and ease of parsing.

Furthermore, MCP defines three key *interaction primitives* that enable standardized and modular tool usage:

- **Resources:** Represent referenceable data objects such as files, images, documents, or records that can be retrieved or manipulated.
- **Tools:** Represent operations that can be invoked by the AI agent, including APIs, scripts, or workflows (e.g., search, transform, send message).
- **Prompts:** Represent task-specific templates or instructions that guide how AI models should interact with tools or resources, including inputs, outputs, and formatting.

MCP servers can be implemented in a variety of languages such as Python, TypeScript, Java, or C#, using open-source SDKs. They support both local (on-device) and remote (cloud-hosted) deployments, enhancing flexibility in use cases ranging from personal assistants to enterprise-scale agents. Servers handle authentication, capability registration, and return structured responses compatible with LLM reasoning workflows.

C.3 Protocol Overview

The Model Context Protocol (MCP) defines a robust, extensible communication protocol that connects LLMs with external tools, resources, and workflows [77], [78]. It is designed to support secure, real-time, bi-directional communication and is capable of dynamic tool discovery, invocation, and structured result retrieval. The protocol promotes clarity and standardization through schema-based interfaces and interoperable message formats.

C.3.1 Communication Model

MCP's communication is based on a structured message exchange process that enables AI agents to understand and interact with tool servers:

1. **Initialization:** The MCP client initiates a connection by requesting the server's capabilities, which include metadata about exposed tools, accessible resources, and prompt templates.
2. **Capability Exchange:** The server responds with a manifest describing its interface and supported operations. This enables the model to plan interactions based on what is available.
3. **Active Message Exchange:** Once initialized, the system enters an operational state in which the client and server exchange requests, results, notifications, and error messages. This enables real-time task orchestration, asynchronous event handling, and flexible agent-driven control.

This design promotes modularity and general-purpose use, allowing AI agents to interact with a wide variety of tools and services using the same messaging and discovery patterns.

C.3.2 Transport and Message Semantics

MCP adopts JSON-RPC 2.0 as its wire format, enabling cross-language interoperability and structured communication. Four primary message types are defined:

- **Request:** A message sent by a client or server that expects a response.
- **Result:** A successful response to a request.
- **Error:** A structured error response, with codes and messages.
- **Notification:** A one-way message that does not expect a reply.

Transports handle the delivery of these messages. MCP supports multiple transport mechanisms:

- **Standard Input/Output (stdio):** Best suited for local tools or CLI-based integrations, using `stdin` and `stdout` streams to send and receive messages.
- **HTTP with Server-Sent Events (SSE):** Ideal for web-based or remote tools, using POST for sending messages and SSE for receiving server-side updates.

Both transports adhere to the same JSON-RPC structure and may be extended or replaced with custom transports that conform to the defined interface.

SDKs in Python and TypeScript abstract these layers, allowing developers to focus on tool implementation rather than low-level messaging details.

C.3.3 Connection and Server Lifecycle

The MCP connection lifecycle is structured into three phases:

- **Initialization:** A version negotiation handshake occurs, during which the client and server exchange supported features and capabilities. The client sends an `initialize` request, the server responds, and a final `initialized` notification is sent to confirm readiness.

- **Active Communication:** Once initialized, both parties can exchange requests and notifications as needed. The protocol supports bidirectional communication, tool invocation, progress updates, and event handling.
- **Termination:** Either side can close the session due to task completion, error conditions, or disconnection. Graceful shutdown via a `close` command ensures resource cleanup.

The server itself also follows a lifecycle with three operational phases:

- **Creation:** The server is configured with metadata (name, version, tool list, prompts, permissions), deployed with source code and manifests, and optionally registered for discovery. Security checks, such as code integrity verification, are essential during this phase.
- **Operation:** The server handles incoming requests and notifications, executes tool actions, and ensures safe interaction via sandboxing and access control. It may also interpret multi-step commands or slash-style queries.
- **Update:** This phase includes version control, privilege reassessment, and deprecation of insecure configurations. Proper update management prevents configuration drift, ensures continuity, and supports long-term maintainability.

Understanding both the protocol and server lifecycle is crucial for secure and scalable implementation of MCP-based AI systems. By combining dynamic discovery, structured messaging, and flexible transport options, MCP enables powerful model-to-tool integrations across a wide range of environments.

Appendix D

The Expert Test Results

This section provides a more detailed account of the expert's work, expanding on Section 6.3. It describes the process and results for each question in depth in the subsections below.

D.1 Question 1: Identifying the Busiest Days and Their Causes

Understanding the usage patterns of the building is essential for evaluating how space and resources are utilized. One of the key questions investigated in this report is:

> Which have been the busiest days of the year and overall? Can we identify which events were organized on those days?

The first step is to retrieve the unique identifier for the relevant building. This is done using a GraphQL query (shown in Listing D.1) that returns all available buildings along with their addresses.

```
query GetBuildingIDQuery {
  buildings {
    id
    address
  }
}
```

Listing D.1: GraphQL query to fetch all buildings and their addresses.

Manual inspection of the results yields the building ID `TMV25`. Using this identifier, entrance telemetry data is collected, limited to the current calendar year, and excluding unnecessary metadata for efficiency. The used query is shown in Listing D.2.

```
query GetBuildingIDQuery {
  buildings(ids: ["TMV25"]) {
    entrances {
      telemetryData(startTime: "2025-01-01T00:00:00Z") {
        timestamp
        value
      }
    }
  }
}
```

Listing D.2: GraphQL query to fetch entrance telemetry data for building TMV25 from 2025-01-01 onward.

The query returns a nested JSON structure containing hourly visitor counts per entrance. Since manual analysis is impractical, a Python script is used to aggregate the data by date. Hourly values are first summed into daily totals per entrance, then combined across all entrances. The five days with the highest total foot traffic are extracted, as shown in Listing D.3.

```
2025-04-29: Total = 1057, Per Entrance = {0: 354, 1: 80, 2: 623}
2025-04-03: Total = 1027, Per Entrance = {0: 408, 1: 181, 2: 438}
2025-04-01: Total = 930, Per Entrance = {0: 281, 1: 184, 2: 465}
2025-05-02: Total = 869, Per Entrance = {0: 215, 1: 105, 2: 549}
2025-05-15: Total = 801, Per Entrance = {0: 427, 1: 150, 2: 224}
```

Listing D.3: The top 5 busiest days at AAU Innovate in 2025

To determine the cause of these peaks, scheduled events and their expected attendance are examined for each corresponding date. This is done using the query shown in Listing D.4, with the `startTime` and `endTime` adjusted to match the date in question.

```
query getEventsOnCertainDay {
  buildings(ids: ["TMV25"]) {
    floors {
      name
      rooms {
        id
        name
        email
        events(startTime: "2024-10-02T00:00:00Z",
endTime:"2024-10-02T23:59:59Z") {
          departmentBreakdown {
            attendeeCount
          }
          formParticipants
          subject
          start
          end
        }
      }
    }
  }
}
```

Listing D.4: The GraphQL query, used to query all events on a certain day.

This leads again to a JSON output, which is read out and analysed in another Python processing program, especially written for this purpose. For the first 3 days, the results are shared in Listing D.5, Listing D.6, and Listing D.7.

- 2025-04-29:

```
AAU Energy Research Day 2025: 1000 attendees
Månedsmøde (morgenmad fra 8:45 - mødestart kl. 9.00): 53 attendees
AI Nordjylland Netværksmøde temarække 2: 50 attendees
Åben Workshop 3: 50 attendees
```

Listing D.5: The events with the most attendees on 2025-04-29.

This clearly shows that the AAU Energy Research Day 2025 was the main driver for the high foot traffic.

- 2025-04-03:

```
Besøg af VL-gruppe: 25 attendees
IAS PBL TAP: 16 attendees
Møde i Tværgående Sikkerhedsorganisering på AAU: 15 attendees
Aftagerpanel for Internationale Relationer og GRS: 12 attendees
LabSTEM: 10 attendees
Møde i styregruppen - GIRAF - Generativ, Interaktiv og Refleksiv Ai i
Faglig forankring🦒: 10 attendees
```

Listing D.6: The events with the most attendees on 2025-04-03.

No single large event is visible here; instead, the day includes several small to medium-sized events.

- 2025-04-01:

```
ALF-møde: 72 attendees
Game Tech Academy Partnermøde: 40 attendees
Drøftelse af dagsorden for kommende SAU møde: 13 attendees
```

Listing D.7: The events with the most attendees on 2025-04-01

A combination of two moderately sized events seems to explain the elevated visitor numbers.

To answer the broader question of the busiest days overall, the data query is extended to include all available telemetry records starting from early 2022. Applying the same aggregation method reveals the five days with the highest total visitor counts across the full dataset, as shown in Listing D.8.

```
2024-10-02: Total = 1109, Per Entrance = {0: 547, 1: 169, 2: 393}
2024-09-18: Total = 1085, Per Entrance = {0: 641, 1: 111, 2: 333}
2025-04-29: Total = 1057, Per Entrance = {0: 354, 1: 80, 2: 623}
2025-04-03: Total = 1027, Per Entrance = {0: 408, 1: 181, 2: 438}
2023-04-25: Total = 1015, Per Entrance = {0: 286, 1: 86, 2: 643}
```

Listing D.8: The top five busiest days at AAU Innovate since 2022.

To illustrate the connection between high traffic and scheduled activities, only the events registered on the most visited day, 2024-10-02, are presented below in Listing D.9. The same analytical approach applies to the other dates.

```
Science Day: 1200 attendees
ALF møde: 70 attendees
AAU Startup Program [indsæt workshop-navn] (AAL 3-24) : 30 attendees
NTNU: 30 attendees
```

Listing D.9: Events scheduled at AAU Innovate on 2024-10-02.

The results confirm that a single large-scale event, `Science Day`, served as the primary driver of foot traffic, supported by three medium-sized events. Together, these contributed to the highest recorded visitor count in the building's history.

D.2 Question 2: Meeting Rooms Usage Analysis

Meeting rooms are central to daily activities at AAU Innovate — from hosting project meetings to facilitating external collaborations. Understanding how these rooms are used can help optimize resources and ensure that space is allocated effectively. To analyze usage patterns, both the Outlook meeting data and the BMS system's PIR sensor data were considered. However, due to the imperfect placement and reliability of the PIR sensors, only the Outlook calendar data was used in this analysis.

To explore usage trends, all meeting data from the start of 2022 until now was retrieved. This was done using a GraphQL query (see Listing D.4), with `startTime` set to `2022-01-01T00:00:00Z` and `endTime` omitted, defaulting to the current time. The `durationMinutes` field was included to make parsing more efficient.

A typical workday is defined as lasting from 08:00 to 16:00, that is, 8 hours or 480 minutes. Daily room usage percentages were thus calculated by dividing the total minutes a room was booked by 480. As a result, some rooms may show occupancy rates above 100%, indicating overlapping bookings or back-to-back meetings. A truncated result can be seen in Listing D.10.

```
Room: TMV25-Stue-A.006
2024-09-17: 1050.0% occupied (5040 min)
2023-10-12: 712.5% occupied (3420 min)
2025-11-04: 425.0% occupied (2040 min)
2024-04-26: 412.5% occupied (1980 min)
2025-04-15: 393.8% occupied (1890 min)
2023-05-11: 350.0% occupied (1680 min)
...

Room: TMV25-Stue-C.004
2024-06-21: 2593.8% occupied (12450 min)
2023-07-24: 1500.0% occupied (7200 min)
2023-07-17: 1500.0% occupied (7200 min)
...
```

Listing D.10: The peak occupancy moments of each meeting room at AAU Innovate

These records highlight the peak usage days per room, showing that at certain times, bookings far exceeded the expected maximum. However, to understand general trends, average usage since the beginning of 2022 was calculated for each room. These averages assume a constant 5-day workweek (excluding holidays and special periods).

```
Average usage per room (from start 2022):
TMV25-Stue-C.004: 49.10% average usage
TMV25-Stue-C.009: 38.27% average usage
TMV25-1. Sal-C.104a: 37.73% average usage
TMV25-1. Sal-C.104b: 36.23% average usage
TMV25-3. Sal-A.314: 26.34% average usage
TMV25-2. Sal-A.202: 25.92% average usage
TMV25-3. Sal-A.313: 25.90% average usage
TMV25-2. Sal-B.203: 22.32% average usage
TMV25-3. Sal-A.301: 20.20% average usage
TMV25-3. Sal-A.315: 18.85% average usage
TMV25-Stue-A.006: 15.44% average usage
TMV25-2. Sal-A.218b: 11.20% average usage
TMV25-1. Sal-C.107: 10.78% average usage
TMV25-1. Sal-C.106: 8.28% average usage
TMV25-1. Sal-A.119a: 2.78% average usage
TMV25-1. Sal-A.120a: 2.14% average usage
TMV25-1. Sal-A.115: 1.47% average usage
TMV25-1. Sal-A.111: 0.83% average usage
TMV25-1. Sal-B.106a: 0.83% average usage
TMV25-2. Sal-A.207: 0.55% average usage
TMV25-1. Sal-A.112: 0.28% average usage
TMV25-3. Sal-A.310: 0.15% average usage
TMV25-2. Sal-A.201a: 0.08% average usage

Overall average usage across all rooms: 15.46%
```

Listing D.11: The total average usage of each meeting room at AAU Innovate

While this provides a general overview, it does not reflect temporal developments or account for the fact that the building only opened in January 2022 and has not yet reached full capacity. To better capture these dynamics, the data was segmented into half-year intervals to reveal trends over time. Listing D.12 illustrates how room usage has evolved from 2022 onwards.

```
--- 2022-H2 ---
TMV25-1. Sal-C.104a: 41.87% average usage
TMV25-1. Sal-C.104b: 35.04% average usage
TMV25-Stue-C.004: 32.84% average usage
TMV25-Stue-C.009: 25.52% average usage
TMV25-2. Sal-A.202: 23.74% average usage
TMV25-3. Sal-A.313: 21.77% average usage
TMV25-3. Sal-A.314: 19.35% average usage
TMV25-2. Sal-B.203: 14.27% average usage
TMV25-3. Sal-A.315: 10.79% average usage
TMV25-2. Sal-A.218b: 7.05% average usage
TMV25-1. Sal-C.106: 5.68% average usage
TMV25-1. Sal-C.107: 2.83% average usage
TMV25-2. Sal-A.201a: 0.52% average usage
TMV25-1. Sal-A.115: 0.25% average usage
TMV25-Stue-A.006: 0.05% average usage
TMV25-1. Sal-A.119a: 0.05% average usage
TMV25-1. Sal-B.106a: 0.05% average usage

...

Form-wide average for 2022-H2: 14.22%
Form-wide average for 2023-H1: 27.14%
Form-wide average for 2023-H2: 32.32%
Form-wide average for 2024-H1: 26.49%
Form-wide average for 2024-H2: 25.29%
Form-wide average for 2025-H1: 23.84%
```

Listing D.12: The average usage, total and for each room, of AAU Innovate's meeting rooms, segmented into half-year intervals.[3]

These values show that overall usage peaked in the second half of 2023 and has gradually declined since then. It is important to note that this does not directly reflect the number or quality of activities at Innovate — rather, it offers an operational insight into space utilization and potential areas for improvement.

Finally, peak usage moments were analyzed by dividing each workday into 1-hour intervals and comparing the number of rooms occupied in each interval against the total number of meeting rooms. For this, only rooms with a non-empty `email` field were considered. This granular approach helps identify specific times of high demand and can assist with future planning, for example, by highlighting periods when booking a room might be difficult or when shared spaces are underutilized.

[3]The data for 2025 is not yet final, as this data is requested on the 20th of May, and only events which are already in the calendar now are taken into account for the usage calculation.

```
2024-05-13 13:45–14:45: 14 rooms occupied out of 26
2024-05-13 14:00–15:00: 14 rooms occupied out of 26
2023-10-23 10:15–11:15: 14 rooms occupied out of 26
2023-10-23 10:30–11:30: 14 rooms occupied out of 26
2024-09-17 12:45–13:45: 14 rooms occupied out of 26
2024-09-17 13:00–14:00: 14 rooms occupied out of 26
2025-04-22 08:45–09:45: 14 rooms occupied out of 26
2025-04-22 09:00–10:00: 14 rooms occupied out of 26
2025-03-10 13:00–14:00: 13 rooms occupied out of 26
2024-11-14 09:15–10:15: 13 rooms occupied out of 26
```

Listing D.13: The top 10 busiest meeting room 1-hour periods at AAU Innovate.[4]

Listing D.13 shows that rumors about meeting room overusage are not supported by the data. Throughout the existence of AAU Innovate, there has not been a single instance, according to the Outlook calendar used to book rooms, where all meeting rooms were simultaneously occupied. While it is true that not every room serves the same purpose or offers the same facilities, a peak occupancy of 14 out of 26 suggests that there is still significant headroom in the system. This is reassuring, especially given that Innovate is a newly constructed building, designed with flexibility and future growth in mind.

D.3 Question 3: Meetings with External Partners

As mentioned in Section 6.1, collaboration with external partners is important for AAU Innovate. Although meetings do not give the full picture, they can be an indication of it.

How many meetings have we hosted where an external partner was present?

First, all meetings from the start of 2022 are requested using a similar query to Listing D.4, with `startTime` set to `2022-01-01T00:00:00Z` and `endTime` omitted, defaulting to the current time. The arguments are also changed slightly. Only `start` is needed to find the year the meeting was in, but `isExternal` is added to `departmentBreakdown` to see whether an event has external people invited. The results are shown in Listing D.14.

```
2022: 245 meeting(s) with external participants
2023: 1146 meeting(s) with external participants
2024: 823 meeting(s) with external participants
2025: 268 meeting(s) with external participants
```

Listing D.14: The number of meetings an external partner was invited into the meeting at AAU Innovate each year.

Here, the same pattern as in Section D.2 is seen, where fewer meetings were organised in 2024 compared to 2023. Note that these numbers assume that external partners are invited into the meeting, where the meeting room is invited into. For example, for bigger events, this is not the case, and this is thus not seen in the data. The actual numbers will thus be higher.

[4]The data for 2025 is not yet final, as this data is requested on the 20th of May, and only events which are already in the calendar now are taken into account for the usage calculation.

D.4 Question 4: Analyzing Room Usage

Efficient use of resources is important, and while some rooms may appear underused yet remain essential, mapping out room activity is a valuable tool to optimize building usage, room function, and layout.

> Which rooms are underused or show the least activity year to date? And which rooms show the most activity?

At the time of writing, access to lighting system data is not available. Instead, the BMS system is used. Although the lighting data might later offer more accuracy, the current process remains applicable and can easily be repeated once access is granted.

The initial step involved querying all sensors via Listing D.15. Unfortunately, the BMS does not provide sensor types, making it necessary to manually identify PIR sensors. Using domain knowledge, a Python script filtered sensors whose `sourcePath` contained `PIR`. This returned a subset of `externalIDs`, which could then be used to extract sensor data. Additional filtering was implemented to handle cases where multiple PIR logs existed for the same room. Specifically, only sensors with both `pir` and `TM025` in the `sourcePath` were selected. This is a fragile but effective method, relying on consistent naming conventions used during the Innovate building setup.

```
query getRoomActivity {
  buildings(ids: ["TMV25"]) {
    floors {
      name
      rooms {
        name
        sensors {
          externalID
          sourcePath
          unit
        }
      }
    }
  }
}
```

Listing D.15: The initial query to identify all available sensors in the building.

Attempting to query sensor values for all rooms at once failed due to response size limits. The BMS system provides no metadata to distinguish sensor types, and sensor naming varies across buildings. Therefore, data was fetched in smaller batches, using the filtered list of relevant `externalIDs` from the PIR sensors.

A starting date of `2022-09-01` was chosen based on observed data availability. Data was then collected month-by-month for each room using an iterative script, referencing the query in Listing D.16. This process alone took several hours due to the size and granularity of the data.

```
query getAllSensors {
  sensors(ids: [...selected ids]) {
    externalID
    sourcePath
    unit
    values(startTime: "2022-09-01T00:00:00Z") {
      timestamp
      value
    }
  }
}
```

Listing D.16: GraphQL query to retrieve value data for selected PIR sensors.

The next step involved calculating the number of hours a room was active. This was defined as the duration between a `1.0` value (motion detected) and the next `0.0`. Consecutive `0.0` values and any non-binary values were ignored. A second Python script batched this processing, computing monthly active hours per room. These results were then correlated with room data from Listing D.15 using the `externalID` as a key.

Finally, a third Python script analyzed the total active hours to determine the most and least used rooms. Results from September 2022 to the present are shown in Listing D.17, and from January 2025 to now in Listing D.18.

```
Top 5 Most Active Rooms:
A.314: 15291.6 hours
B.106b: 13163.9 hours
A.115: 12156.9 hours
B.008: 12005.1 hours
B.002: 10822.9 hours

Top 5 Least Active Rooms:
A.014a: 1492.5 hours
A.001b: 1040.1 hours
A.310: 784.3 hours
B.208: 646.7 hours
B.-111: 129.8 hours
```

Listing D.17: Top 5 most and least used rooms from 2022-09 until now at AAU Innovate.

```
Top 5 Most Active Rooms:
A.115: 3238.5 hours
B.106b: 2193.3 hours
A.314: 2151.7 hours
B.008: 1829.4 hours
A.215: 1766.4 hours

Top 5 Least Active Rooms:
A.106: 158.3 hours
A.001b: 96.9 hours
B.-111: 25.0 hours
A.310: 0.0 hours
A.011: 0.0 hours
```

Listing D.18: Top 5 most and least used rooms from 2025-01 until now at AAU Innovate.

Overall, the data collection and analysis took approximately 3 to 4 hours, highlighting both the complexity of the dataset and the importance of automation and structured querying in working with the BMS system.

D.5 Question 5: IT Lab Change Detection

The final analysis investigates whether a noticeable change in the number of visitors to the IT Lab can be observed, indicating a successful policy change. Anecdotal evidence and casual observation suggest increased activity. The policy change was implemented over the course of January 2025.

> Innovate changed the function of the IT lab to attract more users. Can a measurable increase in activity be seen, and if so, how much more is it being used?

To evaluate this, the floor plan of AAU Innovate was consulted to identify the room IDs corresponding to the IT Lab. While room names are not included in the dataset, the room type `LABORATORIUM` provided a helpful clue. This investigation identified `A.101a` and `A.101b` as the rooms used for the IT Lab. These rooms are typically used together and are rarely divided.

The same approach as outlined in Section D.4 was used to extract relevant PIR data from the BMS system, filtering only these two rooms and their corresponding PIR sensors.

The monthly usage was then mapped using a Python program, calculating the month-over-month differences. The results are shown in Listing D.19.

```
2023-01: 345.34h 266.13h
2023-02: 411.06h (+19.0%) 197.42h (-25.8%)
2023-03: 390.13h (-5.1%) 188.08h (-4.7%)
2023-04: 521.53h (+33.7%) 103.99h (-44.7%)
2023-05: 406.34h (-22.1%) 249.42h (+139.8%)
2023-06: 312.57h (-23.1%) 197.06h (-21.0%)
2023-07: 179.65h (-42.5%) 71.27h (-63.8%)
2023-08: 199.06h (+10.8%) 94.51h (+32.6%)
2023-09: 222.10h (+11.6%) 149.70h (+58.4%)
2023-10: 231.16h (+4.1%) 204.37h (+36.5%)
2023-11: 223.95h (-3.1%) 225.40h (+10.3%)
2023-12: 208.91h (-6.7%) 195.04h (-13.5%)
2024-01: 242.09h (+15.9%) 209.42h (+7.4%)
2024-02: 207.10h (-14.5%) 148.10h (-29.3%)
2024-03: 161.60h (-22.0%) 132.27h (-10.7%)
2024-04: 163.73h (+1.3%) 136.69h (+3.3%)
2024-05: 129.17h (-21.1%) 98.58h (-27.9%)
2024-06: 166.54h (+28.9%) 72.74h (-26.2%)
2024-07: 148.45h (-10.9%) 51.38h (-29.4%)
2024-08: 178.40h (+20.2%) 104.90h (+104.2%)
2024-09: 172.25h (-3.4%) 85.36h (-18.6%)
2024-10: 147.77h (-14.2%) 76.56h (-10.3%)
2024-11: 173.90h (+17.7%) 132.20h (+72.7%)
2024-12: 175.52h (+0.9%) 105.62h (-20.1%)
2025-01: 239.45h (+36.4%) 201.99h (+91.2%)
2025-02: 253.57h (+5.9%) 228.63h (+13.2%)
2025-03: 287.78h (+13.5%) 269.05h (+17.7%)
2025-04: 229.74h (-20.2%) 191.65h (-28.8%)
2025-05: 169.59h (-26.2%) 156.64h (-18.3%)
```

Listing D.19: Monthly usage hours of the IT Lab at AAU Innovate. Room `A.101a` on the left, `A.101b` on the right.

Listing D.19 shows a significant increase in usage starting in January 2025, with continued growth in February and March. Although April still shows elevated usage, May appears to decline. Since the data for May was gathered before the end of the month, it is likely incomplete. Nevertheless, usage levels remain clearly above those from the previous year. To ensure a fair comparison and minimize the impact of vacation periods and other irregularities, a year-over-year comparison is provided in Listing D.20.

```
  Month        A.101a (Hours / % YoY) A.101b (Hours / % YoY)
----------------------------------------------------------------
  2024-01:  242.09h (-29.9%)       209.42h (-21.3%)
  2024-02:  207.10h (-49.6%)       148.10h (-25.0%)
  2024-03:  161.60h (-58.6%)       132.27h (-29.7%)
  2024-04:  163.73h (-68.6%)       136.69h (+31.4%)
  2024-05:  129.17h (-68.2%)        98.58h (-60.5%)
  2024-06:  166.54h (-46.7%)        72.74h (-63.1%)
  2024-07:  148.45h (-17.4%)        51.38h (-27.9%)
  2024-08:  178.40h (-10.4%)       104.90h (+11.0%)
  2024-09:  172.25h (-22.4%)        85.36h (-43.0%)
  2024-10:  147.77h (-36.1%)        76.56h (-62.5%)
  2024-11:  173.90h (-22.3%)       132.20h (-41.3%)
  2024-12:  175.52h (-16.0%)       105.62h (-45.8%)
  2025-01:  239.45h (-1.1%)        201.99h (-3.5%)
  2025-02:  253.57h (+22.4%)       228.63h (+54.4%)
  2025-03:  287.78h (+78.1%)       269.05h (+103.4%)
  2025-04:  229.74h (+40.3%)       191.65h (+40.2%)
  2025-05:  169.59h (+31.3%)       156.64h (+58.9%)
```

Listing D.20: Monthly usage hours of the IT Lab compared to the same months in the previous year.

This year-over-year comparison confirms a significant increase in usage during 2025. Excluding January and May (due to ongoing changes and incomplete data), the average increase between February and April is approximately **66%**, indicating that the policy change achieved its intended effect.

Appendix E

LLM Output Snippets

This section presents representative snippets from each tested LLM to illustrate the types of responses they generate. Although many more interactions were conducted, this overview focuses on two key prompts for each model: the instruction to `'introspect the GraphQL schema'` and a step-by-step guide on how to solve question 1, shown in Listing E.1.

These two examples already reveal significant differences between the models. For some, additional instructions were later added, such as performing the task within a Python script. Most responses have been truncated for brevity; where this is the case, it is indicated using `'[]'`.

```
me> Determine the busiest days of the year in the TMV25 building, I have a
step by step plan for you to follow and answer the question. You have
access to all the required data.

  1. use the time endpoint to get the current time, use this to determine
the current year. this endpoints returns the current time, which you can
use to infer what year it is
currently.
  2. Now use the entrances values on the buildings query to get information
about all the entrances of each building. Here telemetryData can be
requested, returning the number of visitors for that door during one hour.
Use as a starttime the start of the year you just figured out in step 1.
Endtime is an optional parameter here, as defined in the schema and will
default to now, just like requested.

  3. This return will be a highly nested structure, witch the amount of
people entering/leaving the building during that hour. To get the amount of
visitors the building had in a certain day, the individuals values for each
hours should be summed together for each day. This sum of values (for each
hour) should be summed with the values of each of the 3 doors. All this
together results in the number of visitors in the TMV25 building that day

  4. Lastly, I want you to sort each day on number of visitors and return
the 10 most visited days, acoording to this data, return the amount of
visitors and the day it happend.
```

Listing E.1: The input given to each model to test its ability to solve question 1.

E.1 Qwen 3

Listing E.2 and Listing E.3 show the relative good understanding of Qwen3 of the questions. Especially its `deepthink` mode, which is somewhat seen in Listing E.3, but further truncated, is interesting.

```
me> introspect the graphql schema

server logs> 2025/05/26 14:10:02 INFO 🔧 Using tool name=mcp-graphql-aau-
innovate-data-server__introspect-schema

qwen> Looking at the schema, there are several types like BeverageCount,
BeverageMachine, Building,
Entrance, Event, Floor, Room, Sensor, etc. The Query type has various
fields to retrieve data such as
sensors, beverageMachines, getEntrances, buildings, floors, rooms, and
events.

If the user wants to query data, they might need to know the available
fields and their parameters.
For example, to get beverage machine data, they can use the
beverageMachines query with optional
machine IDs. Similarly, events can be fetched using the events field on the
Room type, specifying a
time window.

I should probably prompt the user to ask a specific question or what they
want to query. Maybe they
need help constructing a GraphQL query, like getting all beverage machines,
checking telemetry data
for an entrance, or events in a room. Since the schema is large, guiding
the user to specify their
needs would be helpful. I can offer examples of possible queries based on
the schema to assist them.

Now that we have the schema, what specific data would you like to query?
Here are some examples based
on the schema:

1. Get all
buildings:
[... correct query]

2. Get telemetry data for an entrance:
[... correct query]

3. Get events in a room:
[... correct query, on `rooooms` endpoint]

4. Get beverage machine statistics:
[... correct beverages query]

Would you like to execute any of these queries, or do you have a different
request?
```

Listing E.2: The output of Qwen 3, when asked to inspect the GraphQL schema.

```
 First, I need to get the current time to determine the current year. The
user mentioned using the time
endpoint for this. The function time-server__get_current_time requires a
timezone. Since the example
uses Europe/Copenhagen as the default, I'll use that unless told otherwise.
Once I get the current
time, I can extract the year from it.

Next, I have to use the entrances values on the buildings query to get
telemetry data for all
entrances of TMV25. [...] I'll have to use the mcp-graphql-aau__buildings
query [... way more thinking]

Now, putting it all together in the required format with the tools
available.

Step 1: Get Current Time and Year I'll first retrieve the current time to
determine the year.
[... correct time tool call, listed]

Step 2: Query Building Entrances Next, I'll fetch telemetry data for all
entrances of TMV25 from the start of the year to the present.

  {
    "tool": "mcp-graphql-aau__buildings",
    "parameters": {
      "query": "query ($id: String!, $startTime: String!) { buildings(id:
$id) { entrances { id telemetryData(startTime: $startTime) { timestamp
value } } }}",
      "variables": {
        "id": "TMV25",
        "startTime": "2023-01-01T00:00:00Z"
      }
    }
  }

Step 3: Process Data I'll aggregate hourly values per day across all
entrances:
10. Parse timestamps into dates (e.g., 2023-01-01).
11. Sum all values for each date.

Step 4: Sort and Return Top 10 Days Finally, I'll sort the days by total
visitors and return the top
12.
  {
    "top_busiest_days": [
      {"date": "2023-03-15", "total_visitors": 1245},
      [... truncated]
    ]
  }

Note: The actual dates and values depend on the telemetry data retrieved.
This example illustrates the
final output format.
```

Listing E.3: The output of Qwen3, when asked to solve question 1 with a step-by-step guide.

```
First, I need to understand the data structure. From the previous schema
provided, there's a Building type with entrances, and each entrance has
telemetry data. The telemetry data probably includes timestamps and counts.
So, the steps would involve querying all entrances for TMV25, then getting
their telemetry data over a specific period, summing up the counts per day,
and finding the day with the highest total.
[... truncated thinking]

So, the plan is:
1. Find the entrance IDs for TMV25.
2. For each entrance, get all telemetry data entries (which include hour
and count) over a certain period.
3. Sum the counts for each day across all entrances.
4. Determine which day has the highest total.

[... truncated thinking, plan getting updated along the way]
[... truncated step by step with code snippets and query info + final plan]

### Python Program
  import requests
  from datetime import datetime
  from collections import defaultdict

  # Configuration
  GRAPHQL_URL = "https://aau-innovate-api.example.com/graphql"
  HEADERS = {
      "Authorization": "Bearer YOUR_ACCESS_TOKEN",
  }

  # Helper function to execute GraphQL queries
  def run_graphql_query(query, variables=None):
    [... function to send query]

  # Step 1: Get entrances for TMV25
  building_query = """[... entrances id + name query]"""
  building_data = run_graphql_query(building_query)
  entrances = building_data["data"]["building"]["entrances"]

  # Step 2: Define date range (example: January 2023)
  start_date = "2023-01-01T00:00:00Z"
  end_date = "2023-01-31T23:59:59Z"

  # Step 3: Aggregate telemetry data by day
  telemetry_query = """query EntranceTelemetry [... correct query]"""

  for entrance in entrances:
    [... send query for each found entrance + calculate daily counts]

  # Step 4: Find the busiest day
  busiest_day = max(daily_counts, key=daily_counts.get)
  print(f"The busiest day was {busiest_day} with
{daily_counts[busiest_day]} total entries.")
  [... truncated explanation and notes]
```

Listing E.4: The output of Qwen 3, when asked to solve question 1 in a Python script.

Listing E.4 shows a Python script with a solid structure that, apart from a few minor syntax errors, should have worked. All compiler-detected bugs were resolved, but the program still failed to run due to an incorrect input argument in the query. Qwen 3 correctly identified this issue in the following response, but then began generating unrelated scripts in an attempt to debug the query, failing to retain the schema or the previous context.

Interestingly, it chose to query the data separately for each entrance. While not strictly necessary, since the backend could handle all entrances at once, this could be seen as a thoughtful design choice and even an improvement over the expert's approach.

E.2 LLaMa 4

Listing E.5 and Listing E.6 show example interactions with the LLaMa 4 LLM.

```
me> introspect the graphql schema

server log> INFO 🔧 Using tool name=mcp-graphql-aau-innovate-data-
server__introspect-schema

Llama>  <|python_start|>query { buildings { id rooms { id name } } }<|
python_end|>.
```

Listing E.5: The output of LLaMa 4, when asked to inspect the GraphQL schema.

Listing E.5 shows that LLaMa 4 successfully called the required tool, but then returned with a query wrapped in some Python tags. Although not wrong in itself, it is a rather unseen response compared to other LLMs. Also note that the returned query is wrong, as a building has `floors`, which have `rooms`. `rooms` cannot be queried on `buildings` themselves. And it thus read the introspect response wrong and did not get the schema correctly.

```
llama> <|python_start|>import datetime import
requests

def get_current_time(): response
=
time_server__get_current_time(timezone="Europe/Copenhagen") return
response

[... same for other exposure tools, truncated]

  query =
f"""
  query {{
    buildings(name: "TMV25")
{{
      entrances
{{
        telemetryData(startTime: "{start_time}")
{{

timestamp

peopleCount
        }}
      }}
    }}
  }}
  """

  response =
query_graphql(query)
  data = response['data']['buildings'][0]
['entrances']

  [... truncated wrong data parsing]
busiest_days = get_busiest_days() for day, visitors in
busiest_days:
print(f"{day}: {visitors}")<|python_end|>


me> This code is unusable, with some function calls that are tools
available to you, not in a standard python program


llama> <|python_start|>query { buildings { id name } }<|python_end|>
```

Listing E.6: The output of LLaMa 4, when asked to solve question 1 with a step-by-step guide.

Listing E.6 shows that although LLaMA 4 understood it needed to create a Python script using the `requests` module to send a query, there are several issues with the code. It attempted to call MCP endpoints that are accessible to the LLM itself, but incorrectly tried to do so from within Python. While parts of the query it constructed were partially correct, elements like `peopleCount` do not exist in the schema, making the query invalid.

When following up on this code and pointing out the misuse of MCP-style requests, the response was essentially a non-answer. Further attempts (not shown in Listing E.6) resulted in unrelated Python scripts, as if the LLM had lost track of the entire context of the conversation.

E.3 Watt Tool

Listing E.7 and Listing E.8 show an example of an interaction with the Watt Tool LLM. Here can be seen that Watt Tool cannot send tool requests, but only output the ones it would send.

```
me> inspect the graphql schema

watt-tool>  [mcp-graphql-aau-innovate-data-server__introspect-
schema(argument="")]
```

Listing E.7: The output of Watt Tool, when asked to inspect the GraphQL schema.

```
me> Determine the busiest days of the year in the TMV25 building, I have a
step by step plan for you to follow and answer the question. You have
access to all the required data.

  1. use the time endpoint to get the current time, use this to determine
the current year. this endpoints returns the current time, which you can
use to infer what year it is
currently.
  2. Now use the entrances values on the buildings query to get information
about all the entrances of each building. Here telemetryData can be
requested, returning the number of visitors for that door during one hour.
Use as a starttime the start of the year you just figured out in step 1.
Endtime is an optional parameter here, as defined in the schema and will
default to now, just like requested.

  3. This return will be a highly nested structure, witch the amount of
people entering/leaving the building during that hour. To get the amount of
visitors the building had in a certain day, the individuals values for each
hours should be summed together for each day. This sum of values (for each
hour) should be summed with the values of each of the 3 doors. All this
together results in the number of visitors in the TMV25 building that day

  4. Lastly, I want you to sort each day on number of visitors and return
the 10 most visited days, acoording to this data, return the amount of
visitors and the day it happend.

Note that the tool is setup this was that all variables should be included
into the query string, so don not work with graphql variables to get the
data through the tools

Inspect the schema after step 1 to understand the structure of the schema,
before getting to step 2

Instead of calling the data query tool yourself for all the data, write a
pyhton program that will do so and will answer the question, which were the
10 busiest days this year in the tmv25 building. Use the time and
introspect, and maybe even test the query to create a pyhton progam that
will answer this question without problem. Bake as much info as you know
into the program

watt-tool> [time-server__get_current_time(timezone="Europe/Copenhagen")]
```

Listing E.8: The output of Watt Tool, when asked to solve question 1 with a step-by-step guide.

E.4 Mistral

Listing E.9, Listing E.10, and Listing E.11 show example interactions with the Mistral LLM.

```
me> introspect the graphql schema

mistral>  To introspect the GraphQL schema, I need a specific argument to
pass to the function. Could you please provide the necessary argument?

me> the argument is test

[... truncated]

server logs> INFO 🔧 Using tool name=mcp-graphql-aau-innovate-data-
server__introspect-schema

mistral> query($dummyArgument: String!) {
    beverageMachines(machineIDs: [$dummyArgument]) {
      id
      name
      beverageMachines {
        id
        name
        beverageCounts(startTime: "2023-01-01T00:00:00Z", endTime:
"2024-01-01T00:00:00Z") {
          timestamp
          value
        }
      }
    }
  }
```

Listing E.9: The output of Mistral, when asked to inspect the GraphQL schema.

Listing E.9 shows that Mistral was able to introspect the schema and construct a correct query.

In Listing E.10, it can be seen that Mistral, with some additional guidance, managed to call the query, target the correct data, and retrieve a response. Although it failed to calculate daily visitors and got stuck comparing the raw hourly values, using only the range up to January 6th, it correctly focused on the year 2025, which it had previously determined using the `getTime` tool.

```
  mistral> The current year is 2025. Now I will gather information about
all the entrances of each building using
  the buildings query.
  [...]

  server logs> INFO 🔧 Using tool name=mcp-graphql-aau-innovate-data-
server__query-graphql
INFO 🔧 Using tool name=mcp-graphql-aau-innovate-data-server__query-graphql

  mistral> The data you've provided appears to be a nested JSON structure
containing timestamped values. To
  analyze this data, let's break it down step by step:
  1. Top Level Structure: The top-level object contains an array with one
element.
  2. First Nested Array: Inside the first nested array, there is another
array of objects.
  3. Innermost Objects: Each innermost object has two keys: timestamp and
value.

  ### Key Observations:
  • The timestamps are in ISO 8601 format (e.g., "2024-01-05T23:00:00Z").
  • The values are integers, ranging from 0 to 30.

  ### Patterns and Insights:
  4. Time Range: The data spans from January 1, 2024, to January 6, 2024.
  5. Value Distribution: Most of the values are zero, with occasional non-
zero values.
  6. Peak Times: There are notable peaks on specific dates and times:
    • January 2, 12:00 PM has a value of 30.
    [listed more peak times]

  ### Potential Analysis:
  • Daily Activity: The non-zero values suggest there is some activity or
events occurring at specific times of the day.
  • Peak Hours: Midday (around 12:00 PM) seems to be a peak time for
activity, especially on January 3 and 5.
  • Consistency: There are consistent zero values during night hours,
indicating no activity during these periods.

  ### Visualization Suggestions:
  • Line Chart: A line chart with timestamps on the x-axis and values on
the y-axis can show the variation over time.
  • Heatmap: A heatmap can visualize the intensity of values across
different times and dates, highlighting peak periods.
  [... Pyhton script to visualize the data in a graph]

  ### Conclusion:
  This JSON data represents a time series with varying values at different
times of the day. Analyzing
  this data can help identify patterns, peak hours, and periods of
inactivity. Visualizing the data
  using tools like pandas and matplotlib can provide deeper insights into
these patterns.
```

Listing E.10: The output of Mistral, when asked to solve question 1 with a step-by-step guide.

Listing E.10 looked very promising, but the LLM spiraled off track after being asked to make small changes or optimizations.

```
Got the current time. The year is 2025.

Now, we need to fetch the entrances data for the TMV25 building for this
year. Let's start by inspecting the GraphQL schema to understand its
structure and how we can query the required data. Here is a Python program
that will perform the necessary steps to determine the busiest days of the
year in the TMV25 building:

  import requests
  from datetime import datetime, timedelta
  import json

  GRAPHQL_ENDPOINT = "https://your-graphql-endpoint.com/graphql"

  # Step 1: Get the current year
  current_year = 2025

  # Step 2: Query to get information about all entrances of each building
  entrances_query = f"""[correct entrance query, using `current_year`]"""

  def run_query(query):
    [... function to run the graphql query, but using 'visitors']

  entrances_data = run_query(entrances_query)

  # Step 3: Process the data to sum visitors for each day
  visitors_by_day = {}
  for building in entrances_data['data']['buildings']:
      if building['name'] == 'TMV25':
          for entrance in building['entrances']:
              for telemetry in entrance['telemetryData']:
                  timestamp =
datetime.fromisoformat(telemetry['timestamp'].replace('Z', '+00:00'))
                  date_str = timestamp.date().isoformat()
                  if date_str not in visitors_by_day:
                      visitors_by_day[date_str] = 0
                  visitors_by_day[date_str] += telemetry['visitors']

  # Step 4: Sort the days by number of visitors and get top 10 busiest days
  sorted_days = sorted(visitors_by_day.items(), key=lambda x: x[1],
reverse=True)
  top_10_busiest_days = sorted_days[:10]

  print("Top 10 Busiest Days in TMV25 Building:")
  for day, visitors in top_10_busiest_days:
      print(f"{day}: {visitors} visitors

[... explanation and notes]
```

Listing E.11: The output of Mistral, when asked to solve question 1 in a Python script.

Listing E.11 shows a program that, at first glance, appears quite reasonable. It follows the requested steps and constructs a somewhat correct query. However, when the program was executed and the resulting error was returned to Mistral, it failed to handle the issue, losing this relatively good answer and spiraling off course. In subsequent attempts, Mistral was never able to reach a similarly promising result again.