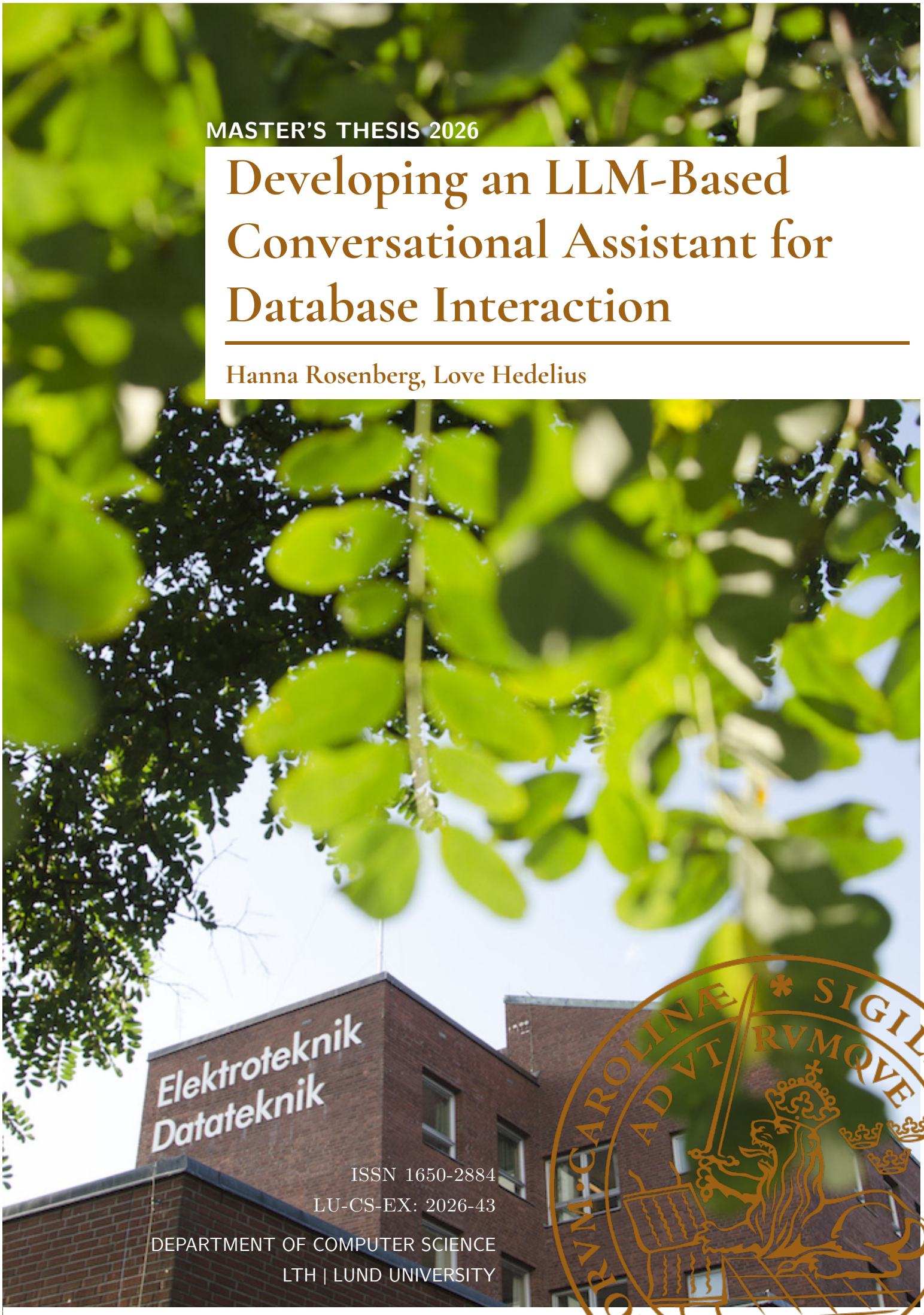
MASTER'S THESIS 2026

Developing an LLM-Based Conversational Assistant for Database Interaction

Hanna Rosenberg, Love Hedelius



ISSN 1650-2884
LU-CS-EX: 2026-43

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2026-43

Developing an LLM-Based Conversational Assistant for Database Interaction

Utveckling av en LLM-baserad konversationsassistent för databasanvändning

Hanna Rosenberg, Love Hedelius

Developing an LLM-Based Conversational Assistant for Database Interaction

Hanna Rosenberg
ha4086ro-s@student.lu.se

Love Hedelius
lo7002he-s@student.lu.se

June 22, 2026

Master's thesis work carried out at Axis Communications AB.

Supervisors: Damir Knezevic, damir.knezevic@axis.com
Douglas Persson, douglas.persson@axis.com
Elizabeth Bjarnason, elizabeth.bjarnason@cs.lth.se

Examiner: Alma Orucevic-Alagic, alma.orucevic-alagic@cs.lth.se

Abstract

Analysing data collected from devices is a valuable way for technology companies to gain insights and improve their products. However, using the data effectively can be difficult due to factors such as large volumes, complex database structures and the need for proficiency in query languages or analytics tools. With their combination of accessibility and capability, large language models (LLMs) offer a promising means of lowering this barrier. In this thesis, we use a Design Science Research approach to develop a conversational assistant with the purpose of helping users utilise a large analytics database more easily. Based on the technological landscape and the needs of intended users, we designed and implemented a solution in several parts, and we evaluated them systematically to assess the effectiveness of each. The final product is an LLM-based agent system connected to a Model Context Protocol (MCP) server and with access to knowledge base files through retrieval-augmented generation. Results from systematic evaluation and user testing show that the assistant successfully answers questions about the database and is considered useful, but fails on some tasks that are advanced or involve subtle pitfalls. We conclude that LLM-based assistants have potential for making data analytics more accessible, and that further development could make the implemented solution a valuable tool in practice.

Keywords: large language models, analytical databases, Model Context Protocol, NLIDB, SQL generation

Acknowledgements

First and foremost, we would like to thank our LTH supervisor, Elizabeth Bjarnason, for her involvement and guidance during this project. Her academic advice and insightful feedback have provided valuable input in all phases of the thesis and greatly contributed to its quality. We also want to thank Axis Communications and the Diagnostics and Data Management team for the warm welcome and the opportunity to conduct our thesis project in such an inspiring environment. We especially extend our most sincere appreciation to Douglas Persson and Damir Knezevic, our supervisors at Axis. Their unwavering support and generosity with time and knowledge have been incredibly helpful in every aspect of our work. Finally, we are thankful to the interview and user testing participants for graciously sharing their thoughts and experiences.

Contents

Chapter 1
Introduction

Using diagnostics data to drive decision-making is an increasingly common and important activity in modern technology companies. Having access to relevant data and being able to extract information and identify patterns from it enables companies to gain insights which can be used to guide future decisions. However, this requires a certain amount of knowledge and skill which some lack. Making use of the available data requires some familiarity with the database it is stored in, and in some cases knowledge of database query languages [1] and statistical methods [2]. The challenge of using a database often grows with its size and complexity [3], as navigating the structure becomes more difficult and performance problems become more likely [4]. To make data more accessible for people lacking specialised analytical skills or understanding of the relevant database, a more abstracted and user-friendly interface for the database can be beneficial.

One possible way of making a database easier to use is to allow users to interact with it through natural language, as that is the primary means by which humans communicate with each other, and thus easy for most people to use. Because of the convenience of natural language for humans, several systems have been developed to interpret it and represent its semantic content in a form usable by computers [5]. One of the approaches is to use a large language model (LLM), which is trained on large amounts of text in order to detect patterns and thus be able to generate responses with human-sounding language and content relevant to the prompt it is given [6]. The advancement of LLMs has in recent years been fast, and their potential usefulness for a wide range of tasks has become apparent [7]. Many organisations and individuals therefore use such models, often through chat interfaces, to assist in their daily work and potentially help save time [8]. With their ability to process and produce natural language but also handle large amounts of information much quicker than a human, LLMs could thus be useful for helping employees navigate and utilise company databases without having expert knowledge of them, thereby facilitating data-driven decision-making. This thesis explores that prospect by considering the situation at a large, international company and implementing an LLM-based solution to help them increase data usage.

1.1 Problem Context

The thesis project was carried out at Axis Communications AB, which is a Swedish company developing surveillance cameras and other equipment for physical security. When in use by consenting customers, these devices collect and send anonymous diagnostics data to the company, where it is used to perform analyses and help improve products and services. The data is of numerous and varied types, but examples include product temperatures, memory and CPU usage statistics, and device configuration settings. It is stored in a column-oriented analytical database at Axis, based on the open-source database management system ClickHouse [9]. Due to the many types of data being recorded, the large number of devices contributing with data and the frequency with which they take measurements, the database contains thousands of columns, distributed across hundreds of tables, and trillions of rows. Furthermore, the majority of the data is raw and unprocessed, and it contains a relatively high concentration of missing or implausible values due to limitations in the collection process. These factors increase the difficulty of understanding the database and using it to draw accurate conclusions.

The database can be accessed programmatically using Structured Query Language (SQL), but there is also a data catalogue for browsing the data and a business intelligence tool for querying the database and creating visualisations, as shown in Figure 1.1. All employees in the R&D organisation of Axis have access to these tools and are encouraged to use them. However, the data catalogue offers limited help in gaining a high-level overview of the database, while the business intelligence tool has a steep learning curve for new users and can be difficult to utilise effectively without SQL knowledge. The Diagnostics and Data Management (DDM) team at Axis, which is responsible for the database and its contents, therefore provides support to employees at other departments who want to use the data, by answering questions and offering introductory workshops. To reduce the amount of time spent on this, the employees of DDM want to make it easier for others to use the database without their help, and they see an LLM-based system with a natural language interface as a promising means to accomplish this.

To make adoption as easy as possible for users and minimise the effort needed for user interface development and deployment, a suitable approach is to implement the solution as an extension of one of the company's existing systems. Axis has an internal LLM-based chatbot used across the company, which has both general-purpose capabilities and increased knowledge about Axis compared to publicly available assistants. It can answer questions based on its pretrained knowledge, files uploaded to it by the user, information found on the internet, and certain internal Axis documents, but it does not have access to the diagnostics data stored in the database or its documentation. The chatbot uses a locally hosted instance of Qwen3.5 [10] as its underlying large language model. The user interface is built on LibreChat [11], which is an open-source, self-hosted platform for interacting with LLMs and has several features that can be used to extend the functionality of the system.

1.2 Purpose

The goal of this thesis is to explore how an LLM-based chat assistant can facilitate independent use of a large and complex database by users who lack expertise in either that specific

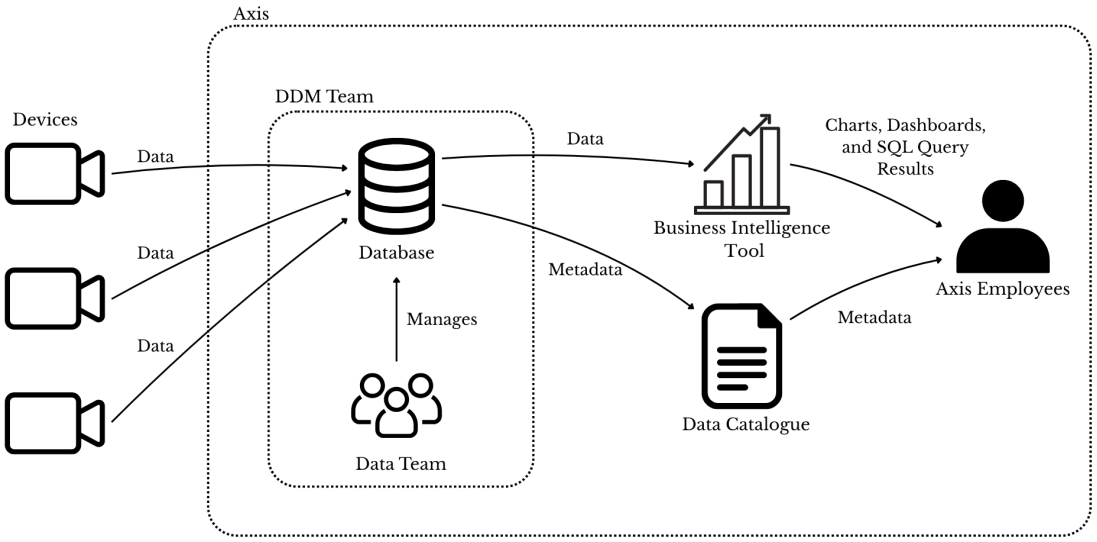


Figure 1.1: Overview of how the diagnostics data is made accessible to Axis employees. Data is collected from devices and stored in a database managed by the Diagnostics and Data Management (DDM) team. Other Axis employees can access the data through two web-based tools: a business intelligence tool for querying the database and creating visualisations, and a data catalogue for browsing metadata.

database, database systems in general, or data analytics. Through a case study conducted at Axis, it aims to investigate how such a system can support more accessible and user-friendly data analytics. The work focuses particularly on identifying the needs and challenges of intended users, and on exploring different approaches for augmenting an LLM with database knowledge in order to improve the usefulness of the assistant. Additionally, it examines how different solution designs affect performance and usefulness. The thesis thus aims to answer the following research questions:

- **RQ1**: How do the intended users currently work with the database, and what barriers do they face in doing so?
- **RQ2**: How can an LLM-based assistant be designed and implemented to help users navigate and query the diagnostics database?
- **RQ3**: How does the implemented solution perform?
  - **RQ3a:** In terms of accuracy, relevance, and response time?
  - **RQ3b:** In terms of perceived usefulness for users?

In responding to these questions, we hope to contribute to a better understanding of how LLMs can be used to facilitate data-driven practices both at Axis and in other settings. By adding research and empirical evidence within the subject of user-friendly database usage

and analytics, our work can provide a foundation for further studies and practical implementations of LLM-based database assistants.

1.3 Division of Work

Both authors were active and worked collaboratively in most parts of this thesis. The work itself, including the design of the assistant, was carried out together, while the writing of the report was organised so that the first author wrote more about the interviews and the second wrote more about the technical implementation. Both authors contributed to and reviewed each other's work throughout. The generative AI tools ChatGPT and Claude were used to revise text and to provide suggestions for improvements, including assistance with word choices, grammar, and advice for how to structure the text. Additionally, GitHub Copilot was used on some occasions to review code or find problems. Scopus AI was used to search for literature, and interview recordings were transcribed using Whisper. The authors always inspected the output before using it, and take full responsibility for the contents of this thesis.

1.4 Thesis Outline

The remainder of this report is organised into chapters, each focusing on a different aspect of the thesis. Chapter 2 introduces the technical concepts relevant to our work, while Chapter 3 gives an overview of the research that has previously been done to solve problems similar to ours. In Chapter 4 we describe the method that we used to conduct the thesis, and in Chapter 5 we present the results. Finally, we discuss our findings in Chapter 6 and draw our conclusions in Chapter 7.

Chapter 2
Theoretical Background

This chapter provides background information that is relevant for understanding the key concepts of the thesis. Since the aim of the thesis is to use techniques based on large language models to facilitate usage of a database, the main topics described are databases and large language models.

2.1 Databases

A database can be defined as a self-describing collection of integrated records, meaning that it contains not only the data itself but also the associated metadata and the relationships between data entities [12]. The data can be any representation of a fact, value or idea, such as a number, a text string, or an image, but has no meaning without the metadata to describe its context [12]. Understanding how data is organised and structured in a database is fundamental to querying and retrieving it effectively [1]. The database at the case company in this thesis is a relational database that is designed for online analytical processing (OLAP) [13] workloads, uses column-oriented storage and is queried through SQL. Therefore, each of these concepts is described in this section.

2.1.1 Relational Databases

A relational database is a database that organises data into tables composed of columns and rows, where a row represents an entry in the form of a set of related values [14]. The structure of a relational database is described by its schema, which defines how data is organised in terms of table names, columns, data types, and the relationships between tables [15]. The schema thus describes the structure of the database, but does not contain the data itself [15]. When querying a relational database, understanding of the schema is necessary, since the user needs to know which tables and columns exist and how they relate to each other [1].

2.1.2 OLAP and Column-Oriented Storage

Relational databases can be designed in different ways depending on the type of workload they are intended to support [4]. The database at the case company is designed for online analytical processing (OLAP), which governs how the data is stored, organised, and queried. OLAP systems are designed for analytical work over large datasets, with the aim of extracting knowledge to support decision making [4]. The queries against such databases are typically complex, ad hoc, and they often aggregate over large volumes of data [13]. In contrast, online transactional processing (OLTP) systems, which are traditionally associated with relational databases, are used for day-to-day operations and handle many small, high-speed transactions, such as updating customer records or processing sales [16, 4, 17]. The key differences between the two are that OLAP systems usually work with much larger amounts of data, that insertion happens in batches rather than continuously, and that the data represents historical information intended for analysis rather than a current operational state [4].

To support analytical workloads, OLAP systems often use column-oriented storage rather than the row-oriented storage that is typical for OLTP [4]. In column-oriented databases, all values in the same column are stored together physically, while a row-oriented database stores all values in the same row together [18]. The latter is efficient for transactional workloads where entire records are frequently accessed [4], but storing data based on columns enables more efficient aggregation operations, such as calculating sums and averages [18]. Column-oriented storage can also improve storage efficiency through column-wise compression [18], and is especially useful when queries frequently access a small subset of columns across many rows [4]. This is typical in analytical use cases such as those at the case company. However, column-oriented storage also has downsides, as write operations such as inserts, updates, or deletions are usually slower than in row-oriented systems [18]. Similarly, joining tables is less effective in column-oriented databases, and they are therefore often less normalised than traditional relational databases, with some information duplicated across tables [18].

2.1.3 Structured Query Language

Structured Query Language (SQL) is the standard language used to interact with relational databases [14]. It was designed to be relatively easy to use, even for users without extensive knowledge of database implementations [19]. SQL is a declarative language, meaning that a query specifies what data should be retrieved but not how that retrieval should be performed [19]. The task of determining how to execute the query is instead handled by the database system's query optimiser [14]. While the language is standardised, different database management systems implement different variants of SQL, which can have slight variations in syntax and available features but share the same fundamental operations [20]. Data retrieval is primarily performed using the `SELECT` statement, which specifies which columns should be returned from a table, together with clauses such as `FROM`, which specifies which table to extract data from, and `WHERE`, which defines conditions that restrict which rows are included in the result [14]. Data from different tables can be combined using `JOIN` operations, and rows can be grouped using `GROUP BY`, often combined with aggregation functions such as `COUNT`, `SUM`, and `AVG` [14]. In addition to `SELECT` for querying data, core operations of SQL include `INSERT`, `UPDATE`, and `DELETE`, which are used to modify the data, and there are also commands for tasks such as defining database structures and managing access privileges [14].

2.2 Large Language Models

Large language models (LLMs) are a class of machine learning systems that are trained on a large amount of text data to process and generate human-like language [6]. They achieve this by taking a text sequence as input and predicting the most probable characters to follow, based on the training data [21]. The introduction of the transformer architecture in 2017, along with increases in computational power and the amount of available training data, have allowed this operation to scale in ways that have made the models capable of many tasks [22]. In this section we describe some aspects of LLMs relevant to this thesis. We first provide an overview of how they work, what they can be used for, and what their drawbacks are, and then we introduce concepts related to customising and augmenting them for specific tasks.

2.2.1 Technical Overview

LLMs are based on transformers, which are a type of neural network, i.e. a computational model where nodes organised in layers propagate signals to each other [22]. The signals are influenced by trainable parameters, which are adjusted to minimise a loss function of the output [22]. The key advantage of transformers is that they enable this training to be parallelised since they, unlike previous architectures, do not rely on processing the text sequentially [6]. This allows the models to be scaled up significantly, with the number of parameters in an LLM typically being in the billions or trillions [6]. When processing text, the LLM first splits the input sequence into tokens, which can be words, parts of words or punctuation [22]. These are then converted to vectors in a space with hundreds or thousands of dimensions, which allows the mathematical operations in the neural network to be applied to them so a prediction of the next token can be calculated [22]. Based on the error of this prediction, the parameters are tweaked through back-propagation [21]. This process is known as self-supervised learning, and it is another important aspect of LLMs since it does not require labelled training data [22]. After the model is pre-trained with self-supervised learning to learn language, it can be fine-tuned with supervised learning to perform specific desired tasks such as answering questions [22]. Finally, reinforcement learning with human feedback is used to further refine the model's outputs according to what is preferred by humans [22].

2.2.2 Applications and Limitations

The level of performance that LLMs can reach in a wide range of tasks have brought about their adoption in many different sectors, especially for repetitive work that would otherwise consume a large amount of a human's time [7]. The models can for example process large amounts of textual or numerical information quickly and present relevant parts or summaries of it to a human, potentially saving significant time [7]. Another use case is to increase availability of services by enabling automated handling of requests and questions, thereby reducing the need for users to wait for a human operator [7]. Furthermore, LLMs can generate material like text, media, or code, which can both save time and in some cases also increase quality and reduce errors [7]. Using LLMs for certain tasks can therefore help organisations scale their activity efficiently, take more informed decisions, reduce costs and allow employees to focus on more complex tasks [7].

However, there are several concerns related to the widespread adoption of LLMs [7]. Some of them are ethical, societal or environmental [7], but there are also challenges related to the quality of the models' output, which limit their usefulness in situations where correctness is important [23]. One of the biggest problems is the tendency of LLMs to produce responses that sound plausible and are presented as facts, but are incorrect or inconsistent with context or sources [24]. These errors are often referred to as hallucinations, although there is a lack of consensus on an exact definition of that term [25] and criticism towards its use [26]. Hallucinations are especially prevalent when inputs are long, as LLMs have limited context windows and experience degrading performance when that limit is approached or exceeded [27]. An aspect that compounds this problem is the tendency of LLMs to rarely signal uncertainty or disclose missing information, instead appearing confident even when wrong [28]. Another issue is their unwillingness to ask for clarification if the prompt is unclear, and instead make potentially false assumptions about the intent [29]. Moreover, the ability of LLMs to follow instructions and produce responses that are in line with what the user wants can become unsatisfactory as instructions get less clear [30] or more complex [31]. Similarly to uncertainty regarding correctness, the problem of not signalling doubt can also apply to instruction-following [32].

2.2.3 Adapting LLMs to Specific Domains

Despite their usefulness for many general tasks, LLMs can be less suited for situations where more specific knowledge and skills are needed [33]. Since training custom models for each situation is often infeasible due to the resources required, many organisations instead adapt existing models to their specific domains [33]. One of the methods used to do this is fine-tuning, where an existing model is trained further with data that is more specific to the task the model should learn [33]. There are several variants of fine-tuning, suitable for different situations, but what they have in common is that they modify the LLM itself, by altering some or all of its parameters [33].

An alternative to this is to use a general model without alterations, and instead use adjustments to the input to improve performance at a specific task [34]. This can either be done by adjusting the original prompt or by augmenting it dynamically based on relevant information sources [34]. The process of refining the prompt to promote better outputs is called prompt engineering [35], and is an important activity since LLMs have been shown to be sensitive to small changes in the input [36]. Carefully designed and formulated prompts can make a significant difference in making an LLM produce responses which are more accurate, relevant and in line with desired behaviour [35, 37]. In order for an LLM to provide an answer or perform a task as well as possible, the question or instruction should be precise, unambiguous, and structured in a logical manner with punctuation and delimiters [37]. There are also many prompting techniques which describe more specific strategies to achieve the desired results [36]. Common elements among these are to include examples of question-answer pairs similar to what the model should produce (known as few-shot prompting), to encourage the model to explicitly state the steps taken to perform a task (known as chain-of-thought prompting) and to assign the model a role which aligns with the desired behaviour [36, 35]. Many of the techniques can also be applied to the system prompt [38], which is the instruction of desired general behaviour which is prepended at the start of every conversation [37].

The other way to improve the input that the LLM receives is to add a knowledge retrieval step at the start of the inference process, where external information is fetched based on the original prompt and given to the LLM along with that prompt [34]. One technique for this, which has the advantage of being flexible and working well for natural language sources, is retrieval-augmented generation (RAG) [35]. To use RAG, the information to be used is first split into chunks, which are converted to embeddings in a vector database, similar to those that the LLMs also use to represent text [35]. When a prompt is given to the system, the RAG mechanism first vectorises it using the same embedding model and calculates which chunks are most semantically related to the prompt [35]. The ones that are ranked as most relevant are retrieved and inputted to the main LLM along with the original prompt, which provides it with potentially useful context and information [35].

2.2.4 Agents and Tool Use

While capable of many tasks, LLMs on their own are restricted to taking an input context and generating an output based on it [39]. A solution to this is agents, which are autonomous software systems that can perform tasks by reasoning about the problem, planning the solution and accessing external resources [40]. Agents are often based on LLMs, but extend the capabilities of stand-alone LLMs by operating iteratively in several steps, between which they can observe the result of the previous action [41]. In an LLM-based agent, the language model serves as the central reasoning component, but the reasoning ability can be enhanced compared to the original model [39]. Agents can also be more adaptable than ordinary LLMs, continuously learning from their task, correcting their own mistakes and adjusting their behaviour based on varying circumstances [39]. This combined with the agents' ability to plan multiple steps ahead makes them capable of acting with high autonomy [39].

One of the most important aspects of LLM-based agents is their ability to interact with their environment by using tools [40]. This is achieved by providing the LLM with descriptions of the available tools along with the user's prompt [42]. Based on this, the model generates a structured output specifying which tool to use and with what parameters, and the agent executes the call through an application programming interface (API) [43]. Tools can allow agents to access up-to-date and potentially non-public information that was not included in the model's training data, which standard LLMs cannot do [44]. They can also address the limitation that language models have when it comes to logical tasks such as mathematical calculations [45]. This increases the probability of the agent retrieving the information it needs rather than hallucinating [46]. Tool use can therefore lead to more accurate and reliable LLM systems, especially in contexts with specialised terms or concepts, high demands on precision, and a need for information processing and quantitative reasoning [47]. Finally, it also allows agents to take actions that have an effect on the world, as opposed to only generating outputs, which significantly extends their area of application [39].

2.2.5 Model Context Protocol

The Model Context Protocol (MCP) is a universal protocol used to standardise how AI applications discover and use capabilities provided by external systems [48]. Prior to such a standard, custom integrations for each system were needed in order to connect the applications to other programs [48]. MCP addresses this by providing a common interface through

which systems can expose their capabilities for agents and other systems to discover and invoke [48]. The MCP architecture consists of three components: the host, the client, and the server [48]. The host is the application that receives user requests and manages connections to external servers, creating a dedicated MCP client for each such connection [48]. Each client maintains a connection to an MCP server, which is the program responsible for exposing capabilities that the client can invoke on behalf of the host [48]. The three main types of capabilities that MCP servers can offer are tools, prompts, and resources [48]. Tools are executable functions that enable the host to perform actions [48], as described in the previous subsection. MCP prompts, which are reusable interaction templates, and resources, which are data sources that provide information, can also be used [48] but are not supported by all client applications [49].

Chapter 3
Related Work

The relative inaccessibility of some databases is a problem that has been studied previously, and LLMs have been used in solutions before. This chapter reviews existing approaches with the aim of situating our work within the broader research landscape. It begins by introducing the general challenge of making databases accessible to users and continues with a review of key techniques explored in the literature, including schema filtering, fine-tuning, user-friendly schema representations, SQL examples, task decomposition, and human-in-the-loop interaction.

Enterprise databases are often complex and users may not know which tables or columns are relevant to their question [50]. Wang et al. [50] note that in real-world scenarios involving large databases, users may need to spend considerable time just identifying and selecting the appropriate databases and tables before they can formulate a query, and the authors therefore emphasise the need for systems that let users ask questions without having knowledge of the entire database structure. Natural language interfaces to databases (NLIDBs) aim to make databases more accessible by allowing users to query data using natural language rather than writing SQL manually [50]. With the development and increasing availability of LLMs, building such systems has become more accessible, as pre-trained models can be used without the need to train domain-specific models from scratch.

Existing research has explored several approaches to improving LLM-based SQL generation, which we outline in the following paragraphs. One of the approaches is **schema filtering**, which addresses the issue of large database schemas exceeding LLM context windows by selecting only relevant parts to send to the LLM. Wang et al. [50] propose a framework where an AI agent with access to a map of the database identifies the tables relevant to the user's natural language query and then feeds only those tables to the LLM. This improved the accuracy of generated SQL queries in their experiments. Similarly, Yu et al. [51] introduce dynamic context-aware schema mapping, which feeds a simplified map of the database into the LLM. This map helps the model select relevant schemas and tables while avoiding context-window overflow or information overload.

Another approach to handling large database schemas is presented by Nguyen et al. [52]

who use **fine-tuning** of the LLM rather than filtering the schema. Their framework, SQLong, augments the training data by adding randomly selected, irrelevant items from the database alongside the relevant ones. This way, the model learns to distinguish relevant schema elements from noise. The original training data consists of a natural language question, the relevant database schema information, sample rows and the target SQL query. The augmentation is done to train the model on handling long context SQL generation tasks, with results showing significant improvements compared to models fine-tuned without augmentation. While the fine-tuning with SQLong improved performance, results still show a decrease in accuracy when the context length increases, suggesting that the fine-tuning with an augmented dataset does not fully address the challenges associated with large database schemas.

Beyond filtering or fine-tuning, another approach to improving SQL generation is to provide the LLM with a more **user-friendly schema representation**. Nascimento et al. [53] suggest providing the LLM with views where tables and columns are renamed to terms closer to end-user vocabulary, and also to predefine commonly used joins. This makes it easier for the LLM to link natural language queries to the actual data. Their results showed that this approach substantially improved SQL generation accuracy compared to using the raw schema.

While the previous approaches focus on what schema information to provide to the LLM, another approach is to supplement the schema with concrete **examples of natural language questions and their corresponding SQL queries**. Nascimento et al. [53] explore retrieval-augmented generation (RAG), where the LLM is provided with database-specific example questions and correct SQL queries as guidance. Combining the RAG technique with LLM-friendly views yielded the highest accuracy in their experiments, indicating that LLMs perform best when they are supported with both simplified, human-readable schema representations and concrete examples. Ojuri et al. [54] similarly explored the use of concrete examples to improve the accuracy of LLM-generated SQL queries, comparing two approaches: few-shot learning, where example questions and their corresponding correct SQL queries are included directly in the prompt, and fine-tuning, where the model is further trained on such examples. While fine-tuning yielded higher accuracy, it requires substantially more resources and effort, making few-shot learning a more accessible alternative when such investment is not feasible. Another approach is utilising previously executed SQL queries to give the LLM examples that encompass domain-specific logic. Jang and Li [55] propose a framework where each example, called a query capsule, consists of a query context and an SQL template. The query context is a natural language description of the SQL template, and the SQL template is an SQL query where values have been replaced with placeholder names. When the user poses a question, RAG is used to retrieve a relevant query capsule. The question, retrieved query capsule, and the full database schema are fed into the LLM as a prompt, and it generates SQL. Successful SQL queries are saved in a pool, to be used in future query capsules to utilise the business-logic and information that is not obvious from only studying the schema. The framework also includes a test suite which tests if a query capsule actually improves SQL accuracy before adding it to the query capsule pool. The evaluation of the framework showed improved performance across different test datasets, but the authors also note that query capsules can introduce unnecessary complexity into queries.

Rather than improving the input given to the LLM, some approaches focus on **decomposing the SQL generation task** into smaller sub-problems. Wang et al. [56] introduce a framework called MAC-SQL, consisting of one core agent and two auxiliary agents that are

activated when necessary. The core agent decomposes the natural language question into sub-questions and sub-queries. When the relevant schema, needed to answer the main question, is longer than a certain threshold, the selector agent retrieves a smaller subset of the relevant schema to avoid problems such as overflowing context windows. The other auxiliary agent has the possibility to execute the final SQL query produced by the core agent and refine it based on potential error messages. The authors' evaluation shows that the multi-agent approach consistently outperforms single-agent baselines across different difficulty levels. Pourreza and Rafiei [57] take a different approach to decomposing tasks. Rather than using multiple agents, their framework DIN-SQL decomposes the task into four modules, each implemented as a separate prompt to the same LLM. First, a schema linking module identifies relevant tables, columns, and values from the natural language question. This output is then used by a classification and decomposition module that categorises the question based on how complex the query will be, and identifies any required joins or sub-queries. The classification determines which prompting strategy is used for SQL generation, where harder queries receive more intermediate reasoning steps. Finally, a self-correction module reviews the generated SQL for potential errors. Evaluation showed consistent improvements over basic few-shot prompting, with the largest improvements on more difficult queries.

A further challenge for NLIDB systems is ambiguity in user queries, where the same natural language expression can refer to multiple different entries in the database. For example, Yu et al. [51] describe a database containing geographical locations, where multiple entries can share the same name. Such ambiguity is difficult for an LLM to resolve on its own. Their proposed solution is **human-in-the-loop interaction**, where the system first identifies all potential matches for the user's query and presents them as a list of alternatives. The user then selects the intended option, which the system uses to generate the correct SQL query. Including this step significantly improved the resolution of ambiguous place names, doubling the success rate on a specialised dataset.

Overall, existing research shows that it is possible to improve the performance of natural-language-to-SQL systems using techniques such as schema filtering, fine-tuning, user-friendly schema representations, SQL examples, task decomposition, and human-in-the-loop interaction. These approaches address technical challenges related to large schemas, ambiguity, and complex queries, and demonstrate that LLM-based systems can generate more accurate SQL when given the right context and support.

However, most work still focuses on SQL generation as a standalone task. Much less attention is given to earlier stages in the interaction process, such as helping users understand what data is available, find the right tables, and navigate complex databases. This suggests a need for systems that support the full process of working with data, rather than focusing solely on SQL generation. At the same time, it remains unclear how well the techniques described in this chapter perform on large, real-world databases. Most of the papers reviewed evaluate their systems on standard benchmarks such as Spider or BIRD, which do not fully reflect the scale of enterprise databases. Recent work similarly points out that achieving natural-language-to-SQL systems suitable for industrial use remains an open challenge [58]. This thesis contributes to this area by designing, implementing, and evaluating an LLM-based database assistant for a real enterprise database.

Chapter 4
Method

The method of this thesis is based on Design Science Research (DSR), following the framework described by Runeson et al. [59]. DSR is a research paradigm that aims to improve real-world practice by designing and evaluating solutions to practical problems, while producing generalisable knowledge that can be applied beyond the specific context studied. This is done through iterative cycles of problem conceptualisation, solution design, and validation. With inspiration from Widell and Sadrian [60], we adapt DSR into three concrete steps to fit the scope of our work: *Problem Conceptualisation*, *Solution Design and Implementation*, and *Solution Evaluation*. Figure 4.1 shows an overview of the method.

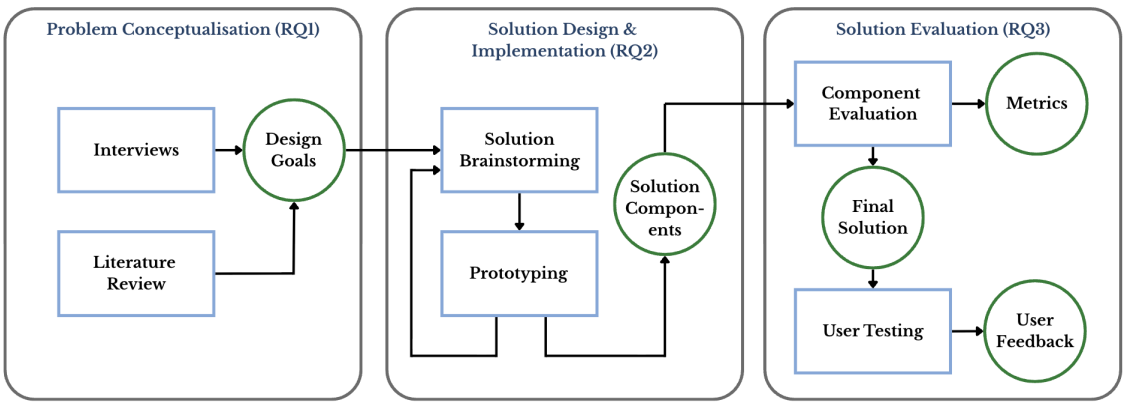


Figure 4.1: Overview of the research method used in this thesis. The method consists of three steps: *Problem Conceptualisation*, *Solution Design and Implementation* and *Solution Evaluation*. The rectangles represent activities and the circles represent their outputs.

In the first step, Problem Conceptualisation, we identified current practices and challenges at the case company and investigated existing research on LLM-based database tools. This step resulted in a set of design goals which were carried into the next step, Solution

Design and Implementation, where we developed a set of components of our solution. In the third and final step, Solution Evaluation, we systematically evaluated different configurations of the solution to identify the best-performing combination of components. This final solution configuration was then assessed qualitatively through user feedback.

4.1 Problem Conceptualisation

The Problem Conceptualisation step focused on understanding current practices at the case company and identifying barriers to working with the data (RQ1), in order to define a set of design goals. The step consisted of a literature review and interviews with intended users of our assistant. The interviews gave insights into the challenges users face when working with the data and the information they need from the database, while the literature review showed how challenges similar to ours have been addressed in previous work, which inspired our solution design in the next step.

4.1.1 Literature Review

We performed a review of scientific literature to explore the problem domain and identify potential solutions to the problem we were studying. The goal of the literature review was to study the topic of natural language interaction with databases, and how LLMs can assist in this. We used the Lund University library discovery service, Finn [61], and the citation database Scopus [62] to find relevant literature. Finn was chosen because it provides access to the full set of resources available through Lund University's libraries. Scopus was chosen for its extensive coverage of peer-reviewed literature and its inclusion of abstracts, which allowed us to efficiently assess the relevance of papers. In addition to standard keyword searches, we used Scopus AI, which allows natural language prompts to identify relevant articles. We divided the work between us, with the first author focusing more on databases and text-to-SQL solutions and the second author reading more about LLMs. We searched using keywords related to our research area, such as "NL2SQL" or "RAG", and iteratively refined our searches by adjusting and adding keywords as we discovered relevant terminology and concepts in the papers we found.

To assess the relevance of an article, we first read the abstract to determine whether it was relevant to our research area. If the content seemed relevant, we looked at the publication year and how many citations it had. We had no exact criteria for either publication year or number of citations, but newer and more cited articles were favoured as they were deemed more relevant and more likely to reflect the current state of the art while also having been validated by the research community. For topics that have evolved rapidly in recent years, specifically those related to LLMs, we prioritised recency particularly highly, and used only sources from 2023 or later. Articles that passed the initial screening were added to a list of selected articles. We then read them all more thoroughly, and summarised their contents. Based on this, we selected the articles that were most relevant to use in the thesis. As the thesis progressed, some articles initially deemed relevant were excluded as our understanding of the area deepened, while new sub-areas of interest emerged, leading us to conduct additional searches and update our selection throughout the writing process. In the end, 55 articles were included in the literature review, which provided us with a deeper understanding of the

research area and allowed us to situate our solution in relation to current research approaches and emerging technologies.

4.1.2 Interviews

We conducted interviews with employees at the case company to gain an understanding of how they worked with the company's diagnostics database, what challenges they faced, and how a database assistant could help them. We chose a semi-structured format for the interviews, due to the interview participants having different backgrounds and levels of knowledge about the database and its contents. The semi-structured format enabled us to let the conversation take different directions, reorder questions, adjust formulations, and ask follow-up questions as needed. This section describes the interview activity in four parts: how participants were selected, how the interview guide was designed, how the interviews were conducted, and how the collected data was analysed.

The selection of interview participants was based on suggestions from one of our supervisors at the case company, who provided a list of 15 employees from other teams that could be potential users of our solution. Six of these employees were part of an initiative to increase data usage across the organisation, meaning they had hands-on experience working with the database, while the rest had taken part in an introductory course on how to work with the data. We reached out to all people on this list, explained the background of our thesis, and asked if they were willing to participate in an interview. Upon acceptance, we booked a meeting at a time that suited them. The first seven to accept were selected for an interview, of whom two were part of the data initiative. The number of interviews was decided together with our supervisor at LTH, who considered it sufficient for the scope of this thesis, while also being feasible given our time constraints. An overview of the interview participants can be found in Table 4.1.

We wrote a set of interview questions covering the information we wanted to gather, starting with some simpler questions to ease into the conversation. The questions were written to address RQ1 and covered areas such as the participants' general experience working with data, their familiarity with the database and its contents, and how they currently use and access data, including what specific information they retrieve. We also asked about what difficulties they encounter when working with the diagnostics data, potential use cases that they currently do not make use of, and how they think an LLM-based assistant could help them with both of these. We spent considerable time trying to make the questions suitable for participants with different backgrounds and varying levels of prior knowledge, which ultimately reinforced our choice of semi-structured interviews as a way to adapt our wording to the situation. We first went over the questions with our supervisor at LTH, who advised us to reduce the number of questions and instead use checklists or sub-questions to make sure all relevant aspects of a question were covered. After that, the interview guide was piloted with one of our supervisors at the case company, who, based on their knowledge of the participants and the problem context, made some suggestions to make the questions clearer for our target group. The interview guide that we finally arrived at and used for the interviews can be found in Appendix A .

All interviews were held in person, with one participant at a time and both authors present. During an interview, one of us was primarily responsible for asking the questions while the other took notes and only intermittently asked clarifying questions or follow-ups.

We alternated between these roles across interviews. To make sure we were in a private environment where everyone could speak freely, we booked private meeting rooms at the company premises. Before starting, participants were asked to consent to us recording the interview, which they all did. We also informed them that they would remain anonymous in the thesis to make sure that they felt free to express their opinions. The interviews took 20 to 30 minutes each, and were afterwards transcribed to make sure we had a complete record of what had been said. The transcription was done using OpenAI's Whisper, an open-source speech recognition tool that was run locally on a personal computer. Running the tool locally meant that the audio files never had to be uploaded to any external server, which was important given that participants had been promised anonymity. The transcripts were then divided between the two authors and reviewed for errors by listening back to the audio recordings and correcting any mistakes manually.

Table 4.1: Overview of interview participants. Familiarity with the database is based on self-assessments – participants who at least stated that they were familiar with most tables relevant to their team are labelled as "Fair" while participants who claimed to only know about very small parts of the database are labelled as "Limited".

Participant	Role	Years at Case Company	Familiarity With Database
I1	Test developer	5	Fair
I2	Software developer	4	Limited
I3	Test developer	4.5	Limited
I4	Test developer	7	Fair
I5	Test developer	1.5	Fair
I6	Software developer	2.5	Fair
I7	Test developer	3	Fair

To analyse the interview data, we performed a thematic analysis following the guide by Ahmed et al. [63]. As a first step, we read through our notes and the transcripts from the interviews to familiarise ourselves with the data and get an overall understanding of what the participants had shared. We then began coding the transcripts as guided by this approach, labelling text excerpts with short descriptive codes that capture their content. We used a combination of deductive and inductive codes. The deductive codes were defined in advance by both authors, based on the main themes of the interview guide such as current data usage, barriers encountered, and wishes for an LLM-based assistant. These served as a starting point for the coding. In addition, inductive codes emerged during the coding process, capturing aspects of the data that fell outside the deductive codes. We went through the transcripts and assigned codes to relevant excerpts using the comment function in Google Docs. The coded excerpts were then exported to a spreadsheet and listed alongside their assigned codes. The coding was done partly individually and partly together, with disagreements and ambiguous excerpts discussed until consensus was reached. We then grouped the codes into broader themes, merging codes that overlapped. This process was iterative, meaning themes and codes were revised several times as we reviewed them against the full set of excerpts, until we arrived at a final set of themes and codes that we felt adequately captured the data.

4.1.3 Identifying Design Goals

We defined a set of design goals to guide our solution design, which were derived from the user needs identified in the thematic analysis and informed by techniques described in related work. Both authors went through and discussed the identified user needs and barriers, and for each one considered whether it could realistically be addressed by our solution. Needs that were deemed both relevant and feasible were formulated as design goals. For example, a commonly reported barrier was difficulty finding relevant data, which led to the design goal of supporting data discovery. Some goals, such as handling ambiguity, were also informed by the literature review and our own assessment of what would be necessary for a reliable assistant. Some of the user needs identified in the interviews were not deemed feasible to implement due to technical limitations or time constraints, and were therefore excluded from the design goals.

4.2 Solution Design and Implementation

The second step, Solution Design and Implementation, addressed how an LLM-based assistant could be implemented to help users navigate and query the diagnostics database (RQ2). Based on the design goals identified in the previous step, we identified potential approaches to a solution, which we then developed through iterative prototyping and testing. This section describes these activities, as well as some technical aspects of the approach.

We decided to base our solution on the Agent Builder functionality of the LibreChat platform. This enabled us to develop it in several different parts, such as tools in an MCP server, and combine them by connecting them to an agent. We therefore created the assistant as a system of one or more LibreChat agents, and used the features of Agent Builder to integrate the parts that we implemented. Apart from MCP tools, we added files, a system prompt, and a handoff between agents in this way. In the rest of this report, these various parts of the solution are referred to as *elements*, and they are described in detail in Section 5.3. An element is a single tool, a single file, or a single section of the system prompt, and the handoff between agents is considered one element.

We designed and implemented the solution based on the design goals identified in the Problem Conceptualisation step. Because of the exploratory nature of the problem and the need to refine the solution based on observed performance, we chose an iterative prototyping approach for this. The Solution Design and Implementation step therefore contained multiple development cycles, where each cycle focused on a specific element. We spent a significant amount of time discussing and brainstorming potential solutions to address the design goals and the problems that we had observed when experimenting with earlier versions of the assistant. When we had an idea that we deemed promising enough to pursue, we started a development cycle where we implemented the idea in the form of some element. The development work was carried out by both authors, either through pair programming or with one author implementing an element and the other reviewing it. Most of the work involved writing code, for example implementing MCP tools or writing scripts to generate files based on resources at the case company. There were however also some non-coding tasks, like writing natural language instructions for the assistant.

After each development cycle, the implemented element was tested by both authors

through interaction with the assistant. The testing consisted of two types of prompts. First, we asked more targeted low-level questions aimed at testing the specific functionality of the element, for example verifying that the assistant could correctly list all columns in a table when a tool for that had been implemented. Second, we asked questions that were more similar to what a user at the case company might ask, in order to see if the implemented element seemed to help the assistant answer these. These questions were based on examples and use cases described by the interview participants, such as asking what the average temperature is for devices running a certain firmware version. The purpose of this testing was formative rather than summative, meaning that the emphasis was on identifying weaknesses and refining the design rather than on producing reproducible performance measurements. As such, individual test prompts were not systematically recorded. Based on the results of these informal tests, the solution was refined in subsequent iterations. In some cases, this involved modifying existing elements, while in others, alternative elements were developed to better fulfil the intended functionality. An element was considered complete when it produced correct results for the majority of our test queries and fulfilled its intended functionality, which meant that the number of development cycles varied for each element.

The prototyping step was concluded when the majority of the design goals had been addressed by at least one element. The aim was not to fully implement every aspect of the solution, but to explore whether and how an LLM-based assistant could help users interact with the database. We therefore considered this sufficient to proceed to the evaluation step. Some design goals were however not fully realised due to time constraints.

4.3 Solution Evaluation

The third step, Solution Evaluation, focused on finding a final version of the solution, as well as assessing how well that final solution performed (RQ3). The step was therefore divided into two parts. The purpose of the first part was to determine a final configuration of the solution by identifying which elements improved performance. We investigated this with a comparative configuration evaluation, the result of which was a final version of the assistant. When this was found, we moved on to the second part, which focused on user satisfaction. In this part of the evaluation, intended users tested the final solution and provided feedback on how well it met their needs.

4.3.1 Comparative Configuration Evaluation

In order to assess what contributed with value to the solution, we aimed to determine which elements made the assistant perform better when added. We achieved this with a systematic evaluation of different configurations, each with a different combination of elements included. Each configuration was evaluated by asking it a set of pre-defined questions and assessing the accuracy, relevance, and quickness of its response. The configuration that performed the best in this evaluation was chosen as the final version of the assistant.

To keep the number of combinations to test manageable, we **grouped the implemented elements**, i.e. the individual tools, files, system prompt sections, and the handoff between agents, into seven *components*. The grouping was done so that elements with a similar purpose and related implementations were part of the same component. During the evaluation,

a component was treated as a unit, meaning that its elements were either all included or all excluded. To further reduce the number of combinations, we split the components into three evaluation groups based on how they were intended to contribute to performance: by incorporating knowledge of the database, by improving SQL quality, or by steering the behaviour of the agent. This allowed us to compare the components against others in the same group instead of evaluating all components simultaneously, which reduced the number of configurations from 128 to 16. While this was necessary to keep the evaluation feasible, it is possible that it prevented the identification of beneficial interaction effects between components from different groups. Table 4.2 shows the seven components, the elements that they consisted of, and the evaluation group they were placed in.

Table 4.2: The grouping of elements into components and components into evaluation groups. Elements in monospace font are either MCP tools (underscores, no extension), files (underscores and a file extension), or sections in a system prompt (Markdown headings prefixed with `#`).

Component	Descriptive Name	Elements	Group
A	SQL-based exploration tools	`get_tables_in_schema`, `get_columns_in_table`, `get_table_comment`, `get_column_comment`, `get_table_sample`	1
B	Contract-based exploration resources	`table_names_and_descriptions.md`, `get_table_contract`	1
C	Abbreviations dictionary	`abbreviations_explanations.json`	1
D	Query log tool	`get_previous_queries`	2
E	Structured query tool	`execute_query`	2
F	System prompt	`# Role`, `# Context`, `# Rules`, `# Tools`, `# Example`	3
G	Multi-agent system	*The separation of capabilities across two agents*	3

The comparative configuration evaluation was **carried out in three phases**, each one focusing on one group of components. In phase 1, we assessed components A, B and C by creating eight agents, one for each possible subset of these components. Each agent was given the same set of evaluation questions, and the combination of components that resulted in the highest score was identified. This optimal subset, denoted $S^*_{ABC} \subseteq \{A, B, C\}$, was taken as part of the final solution, and was therefore fixed for the remainder of the evaluation. In phase 2, we tested the four combinations of components D and E in a similar manner, with S^*_{ABC} included in every agent alongside the components under evaluation. The subset $S^*_{DE} \subseteq \{D, E\}$ with the highest score in phase 2 was then fixed along with S^*_{ABC} in phase 3, where the best subset $S^*_{FG} \subseteq \{F, G\}$ was identified. Since one of the components evaluated

in phase 3 was the separation into multiple agents, some of the configurations in that phase technically consisted of two linked agents rather than a single one, but this did not affect how they were used. The final configuration was chosen to be an agent or agent system with the components identified in the three phases, i.e. described by $S^*_{ABC} \cup S^*_{DE} \cup S^*_{FG}$. Figure 4.2 shows the three evaluation phases and how the components were systematically included and excluded. It should however be noted that although we did not evaluate component F until phase 3, we did include a system prompt in all configurations in the two earlier phases. The reason for this was our belief that the performance would be much lower without the system prompt and that the evaluation of the other components would yield less representative results without the added instructions.

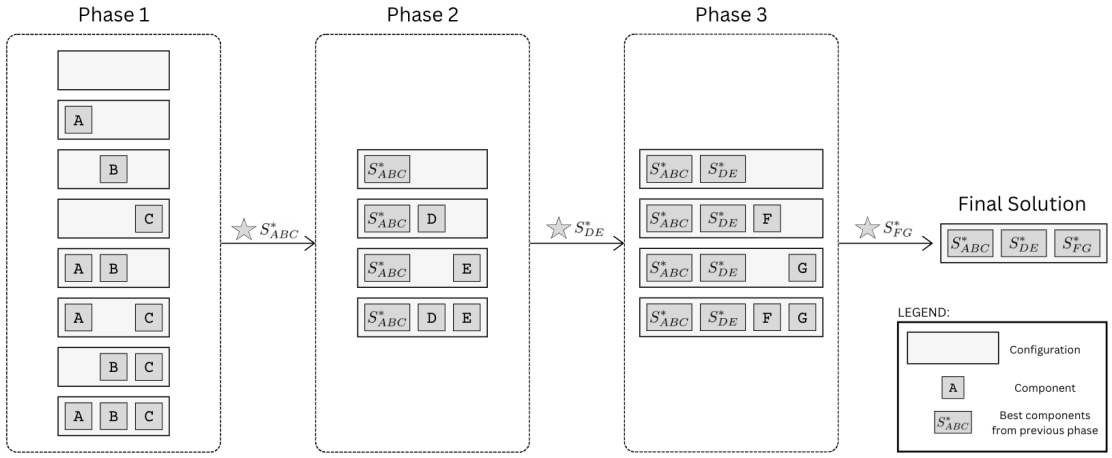


Figure 4.2: Overview of the method used in the comparative configuration evaluation. The figure shows which combinations of components were tested in each phase and how the best-performing components were fixed in the following phases.

We tested the agents by giving them **a set of evaluation questions** and observing the answers. Since one of the key challenges of the project was the size and complexity of the database at the case company, existing benchmark datasets were too simple to be suitable for this. Furthermore, some parts of our solution were specifically designed for the company's database. In order for the evaluation to be representative of the actual intended use of the assistant, we therefore judged that the agents needed to be tested with questions about the specific database. With guidance from our supervisors at the company, we thus wrote 30 evaluation questions, which we attempted to make varied and relatively realistic. This was done based on examples given and wishes expressed in the interviews, as well as on questions that our supervisors had previously been asked from other employees. Since the main purpose of the components in evaluation group 1 was to give the assistant knowledge of the database's structure and contents, we tested them with 15 questions related to exploring the database. The components in evaluation group 2 were instead primarily intended to improve the assistant's ability to write SQL queries, and the 15 questions used in phase 2 were therefore focused on retrieving actual data from the database. In phase 3, all 30 questions were used to get as complete a picture as possible of the assistant's abilities.

Within the two categories, **the questions were of two different types**. Twelve questions

in each category were designed to have a short and objective answer, which could be quickly and automatically verified by comparing to a known truth. The answers to the exploration questions of this type, which in most cases was the name of a table or column, were found while writing the questions and confirmed by our company supervisors. For the objective type of data retrieval questions, many of which had a single number as the right answer, we wrote SQL queries that correctly answered them, which our supervisors also verified. During the evaluation, we ran these queries right before giving the question to the assistant, to make sure we had an up-to-date correct answer. The remaining three questions in each question category were more open-ended and required a longer and more nuanced answer. For these, no correct answer was decided in advance and the responses were instead assessed subjectively by the supervisors. Since they were not restricted to having a short and unambiguous answer, the open-ended questions could be varied more than the others, and we tried to choose them in a way that would reveal whether the agents had the behaviour that the intended users desired. All questions used in the evaluation are listed in Table 4.3.

To evaluate any given combination of components, an agent was created which included those components. For each question, we then started a new conversation with that agent, submitted the prompt, and observed the response. We also measured the response time using the Network tab of Google Chrome's DevTools, where we could read off the timestamps of the initial request and of the final received byte in the response. Due to time constraints, each question was run only once per configuration. The questions of the objective type included an instruction to respond with only the answer, without any extraneous explanation or elaboration. They were initially planned to be automatically verified using a testing framework, but due to limitations related to accessing the agents programmatically, we decided to do it manually in the user interface instead. The agent's answers to these objective questions were recorded in a spreadsheet where we compared them with the correct answers, which in the case of the data retrieval questions were retrieved right before by running the corresponding solution query. For the questions of the more subjective type, we exported the entire responses as Markdown files and gave them to our supervisors at the case company. With their expertise in the domain, they graded the responses on a scale of 0 to 3 based on how accurately and relevantly they answered the question, as well as on to what extent they showed desired behaviour like admitting uncertainty and being transparent about assumptions. Inadequate responses were given the grades 0 and 1, with the completely wrong or actively misleading ones receiving 0, while the grade 2 was given to helpful but not perfect responses and the grade 3 to near-perfect ones. In contrast to the objective questions, the responses to these manually graded questions were judged not only on the final answer but also on the method used to produce it. This meant that an agent giving the right answer through coincidence or using an inefficient approach could be separated from an agent handling the task optimally.

After having answered all applicable questions, **each agent was given a *response quality score*** based on the number of correct answers and the grades from the subjectively judged questions. In order to give both types of questions equal weight, we normalised the scores from each type by dividing by their respective maximum possible scores. This resulted in a score between 0 and 1 from each type of question, and the total response quality score was calculated as their sum. This score, which could thus take values between 0 and 2, was the main factor for determining the best-performing configuration. However, the response times were also considered to promote quicker agents over those who would make the user wait. We defined no exact rule for how to factor in response time, as we planned on using the metric to

detect large differences in efficiency or to distinguish between otherwise equivalent results.

4.3.2 User Testing

We conducted user testing of our final solution to collect qualitative feedback from intended users. This was done through a survey where we first let the participants test our assistant, and then answer a questionnaire about their experience. We chose this approach because it allowed multiple participants to test the assistant and submit feedback at their own convenience. The user testing activity consisted of three parts: participant selection and recruitment, survey design, and data analysis.

The people invited to participate in our user testing were the same seven employees who had participated in the interviews earlier in the study. This was because they were seen as potential users of our solution, and because they were already familiar with the project which meant that less introduction was needed. It was also this group's expressed barriers and needs that informed the design goals and ultimately our solution, and therefore we deemed it appropriate to evaluate the solution with their feedback. We reached out to the selected group of employees, informed them about the user testing and asked if they wanted to participate. All seven agreed, but due to time constraints and technical issues related to making the assistant accessible for others, only four could complete the testing within the time frame we had allocated for it. They were instructed to test the assistant as little or as much as they wanted and to ask any types of questions, but preferably something relevant to their actual work. Upon acceptance, we followed up with instructions on how to access the assistant and a link to the questionnaire.

The survey design was performed by both authors and resulted in a questionnaire consisting of three open-ended questions. In the first, participants were asked to summarise what questions or tasks they had tested the assistant on. This was included to provide us with context for the rest of the responses and facilitate understanding of the assistant's strengths and weaknesses. The second question asked about the participants' general experience using the assistant. To give some guidance without constraining their answers, we added some examples of aspects they could include in their answer. These included whether they found the assistant useful, whether it, to their knowledge, answered the questions correctly, whether it seemed to have varying performance on different types of tasks, and whether it showed desirable behaviour. The intention was not to limit their answers to these points, but to give some inspiration for what they could include. In the last question, we asked the participants for any additional comments. The collected survey data was read by both authors and analysed to understand what the assistant handled well, where it performed worse, how useful it was considered, and what extensions or improvements were desired.

Table 4.3: The questions used for the comparative evaluation, partially anonymised for confidentiality reasons.

ID	Question	Evaluation	Phases
E1	Which table should I use if I want to look at ACAP usage across devices? Answer with just one table name.	Verified	1, 3
E2	Which table should I use if I want to make a dashboard with product temperatures? Answer with just one table name.	Verified	1, 3
E3	Is there any data about how cameras switch between modes for filming in different lighting conditions? If there is, answer with just one table name.	Verified	1, 3
E4	Give me all the tables that have information about body worn cameras. Answer with only the table names.	Verified	1, 3
E5	Which column in the table <TABLE NAME> contains information about the speed of fans? Answer with only the column name.	Verified	1, 3
E6	I want to know if a device uses a network share. Can you tell me which table and column to use? Answer with only the table name and the column name.	Verified	1, 3
E7	I am interested in knowing the extent to which our devices use the Session Initiation Protocol. Which table and column should I use? Answer with one table name and one column name only.	Verified	1, 3
E8	What kind of product does <TABLE NAME> have information about? Answer with one of the following alternatives, and nothing more: recorders, intercoms, speakers, wearables, thermal cameras.	Verified	1, 3
E9	What combination of columns uniquely identifies a row in the table <TABLE NAME>? Answer with only the column names.	Verified	1, 3
E10	Which one of these columns do all tables include: <COLUMN NAME>, <COLUMN NAME>, <COLUMN NAME>, <COLUMN NAME>, <COLUMN NAME>? Answer with only one of the column names.	Verified	1, 3
E11	What is VMD? Answer with only the expansion of the acronym.	Verified	1, 3
E12	What software version does a device at least need to run the <PROGRAM NAME> plugin? Answer with just the version number.	Verified	1, 3
E13	How do I count the number of unique ACAPs? Are there multiple ways to do it?	Judged	1, 3

ID	Question	Evaluation	Phases
E14	I am looking for data related to audio recording in devices. What is there in the database?	Judged	1, 3
E15	Explain the <TABLE NAME> table, what it contains and how it can be used. What is the difference compared to <TABLE NAME>?	Judged	1, 3
D1	Which different names are there in the column <COLUMN NAME> in the table <TABLE NAME>? Answer with only the names.	Verified	2, 3
D2	What is the average maximum allowed music volume in the <TABLE NAME> table? Answer with only the volume.	Verified	2, 3
D3	What is the highest wattage that the device <SERIAL NUMBER> has ever reported? Answer with only the wattage.	Verified	2, 3
D4	What percentage of devices have ever had a configured text overlay count of more than 1? Answer with only the percentage.	Verified	2, 3
D5	Give me the standard deviation of all the reported durations of <DATABASE NAME> data collection from C3003-E products in the last year. Answer with only the standard deviation.	Verified	2, 3
D6	On what date was the software version for the device <SERIAL NUMBER> last changed, and what version did it change from and to? Answer with only the date and the software versions.	Verified	2, 3
D7	What is the product name of the device that has analysed the most frames for video object detection? Answer with only the product name.	Verified	2, 3
D8	What is the difference between the highest and lowest temperature that the device <SERIAL NUMBER> had in March this year? Answer with only the difference.	Verified	2, 3
D9	What were the 10 most common ACAPs running yesterday? Answer with only a list of the ACAPs in order, and use their human-friendly names.	Verified	2, 3
D10	How many devices in the <TABLE NAME> table have reported the Entry List feature as enabled in their latest data entry? Answer with only the number.	Verified	2, 3
D11	Find the P4708-PLVE device that has had the lowest minimum amount of unused physical memory of all P4708-PLVE devices since the start of 2026. What was the latest error message in the syslog of that device before that minimum memory point? Answer with only the message.	Verified	2, 3

ID	Question	Evaluation	Phases
D12	For each hour (in the UTC timezone) of the 17th of April 2026, how many times did watchdog force a reboot on the device <SERIAL NUMBER>? Answer on the format 00:00 - 00:59: <number_of_reboots_first_hour>, 01:00 - 01:59: <number_of_reboots_second_hour>, and so on.	Verified	2, 3
D13	How common is it that our devices have SD cards, and which sizes of SD cards are most common?	Judged	2, 3
D14	Based on how the number has changed until now, estimate how many devices will be contributing with data to the <DATABASE NAME> database in 12 months.	Judged	2, 3
D15	Give me an overview of the ERR syslog messages that the subsystem httpd has produced on W800 devices in the last week. Are there groups of similar messages, which of those groups are most common, and has that changed over the week? Can you find any other patterns?	Judged	2, 3

Chapter 5
Results

The activities described in the previous chapter led to results of different types, and these results are presented here. This includes a description of the current situation related to usage of the diagnostics database, derived from the interviews, as well as the design goals that we identified based on that and the literature review. We also present the implemented assistant, the results from the comparative configuration evaluation leading to a final version of that assistant, and the feedback received from users who tested it.

5.1 Data Usage at the Case Company

The interviews conducted with employees at the case company addressed how users currently work with the company's diagnostics database and what challenges they face (RQ1), and also informed the design goals presented in the next section. This section reports the findings from three areas: current data usage, barriers encountered when working with the data, and user needs for a database assistant. All participants had technical backgrounds and worked with code, but varied in their experience with the diagnostics database and the tools used to access it. Section 4.1.2 contains a description of the interview methodology, including an overview of the interview participants.

5.1.1 Current Data Practices

The intended users of our assistant currently use the diagnostics database in varying ways. Their routines are influenced by several factors, such as different roles and varying levels of experience with the database and the tools used to access it. This section presents some aspects of the current usage, starting with the primary means of accessing the data and followed by patterns of tool use, strategies for exploring the database, and the main goals driving data usage.

Participants described one **main way of accessing the data**, through a web-based business intelligence tool that has both an SQL editor for writing queries directly and a drag-and-drop interface for creating charts and dashboards. Some also reported using a web-based data catalogue where users can look up metadata such as table and column descriptions and data ownership. For most users, the business intelligence tool is the only interface through which they can query the database, but one interviewee also had direct access to the underlying database and could query it directly.

Participants showed different **patterns of tool use**. Some did not use any tools themselves and relied on visualisations set up by others. Some lacked SQL experience and created graphs in the drag-and-drop interface, and others combined the two, using SQL for quick queries and the drag-and-drop interface for more complex analysis. I3 checked dashboards weekly with their team without knowing what was happening behind them, simply consuming the visualised data that someone else had set up. I7 described how the DDM team had helped their team set up a dashboard, which I7 later extended with additional charts and customisations. Dashboards and charts were described as well-suited for ongoing monitoring. Several participants mentioned having recurring team meetings where the team reviews the same dashboard regularly to track how key metrics are developing over time. Pre-built dashboards were described as useful because they can be configured once and then revisited regularly without having to set them up again each time, making them accessible to people regardless of technical expertise. SQL queries served a different purpose, as they were used to find quick answers in raw data. I1 said that dashboards are needed if you want to monitor data, but SQL queries are better for getting specific information quickly. I5 similarly noted that the drag-and-drop interface worked better for more complex analytical work, but relied on SQL for simpler queries. In general, employees who were more familiar with the database tended to be more active in their tool use, while less familiar participants relied more on consuming pre-built dashboards.

A few participants described their **strategies for exploring the database** and finding relevant data. They relied primarily on table and column names and descriptions as a starting point. One of the more experienced participants, I1, described their process: first looking at table and column names and descriptions, then moving on to looking at the actual data through SQL queries. They said that using SQL queries usually works well for exploring and finding data, by starting with broad queries and narrowing it down step by step. If these steps were not enough, the participant would go into the code responsible for collecting the data, and try to figure out from there what the data is about and how it is structured. Both I4 and I5 mentioned using the data catalogue as a way to explore available data. I1 and I5 reported that table and column names and descriptions usually provided enough information to locate relevant data, while I4 noted that the names were not always intuitive.

Two main **goals for using the data** emerged from the interviews: gathering customer insights and monitoring product health. In both cases, interview participants described wanting to identify trends and patterns in the data to support their decision-making. Regarding customer insights, participants described wanting to understand how and to what extent their features are being used by customers, in order to make informed decisions about where to focus their efforts. Both I3 and I5 exemplified this, saying that if the data shows that a feature does not seem to be widely used, it can be prioritised lower in the testing of features. I3 also described wanting to see how their features are actually being used, to create test cases that accurately reflect real usage rather than having to make up numbers. The second com-

mon goal was monitoring product health, which means using the data to detect problems in code or hardware. Examples mentioned in the interviews include identifying specific devices that appear to have problems, discerning if customers are uninstalling a particular feature, or seeing whether a new software update creates longer restart times.

5.1.2 Barriers

The interview participants had varying perceptions on the challenges associated with working with the diagnostics data. From our analysis, barriers emerged in three parts of working with the data: finding relevant data, understanding and interpreting the data, and querying and analysing the data. A fourth category, structural barriers, affected the work more broadly.

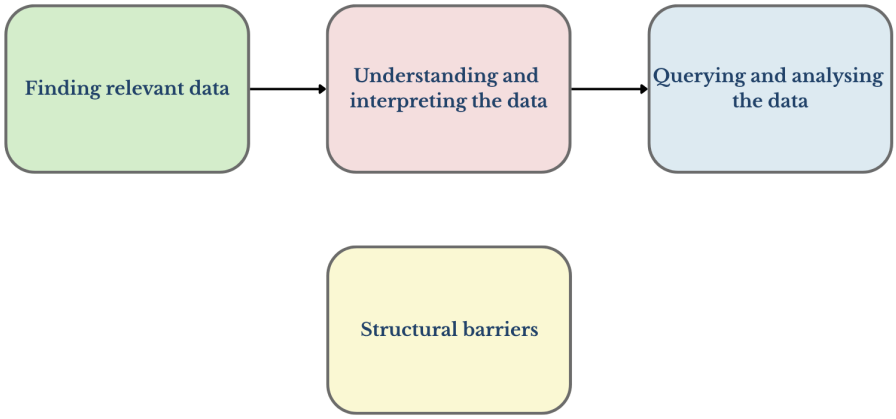


Figure 5.1: The four areas of barriers identified in the interviews.

The first area, **finding relevant data**, was the most commonly reported challenge. The size of the database left several participants feeling overwhelmed and unsure of where to start, particularly those with less experience with the data. I3 likened it to a bucket of Lego dumped on the floor, saying they knew the pieces were there but not what they could build with them. I5 described the database as a "data swamp" where it is easy to get lost, saying they felt more comfortable building a chart when they knew which data to use than exploring the database to find the right data. I1 and I4 also reported finding relevant data as one of the most challenging parts of working with the data. Part of what makes navigation difficult is the similarity between columns, which can contain very similar data and be difficult to tell apart. This challenge was not limited to less experienced database users, as those with more experience also described difficulties related to finding relevant data in unfamiliar parts of the database. This barrier motivated a first design goal, which was that the assistant should help users discover relevant tables and columns in the database.

The second area, **understanding and interpreting the data**, presented challenges after participants had found the data they were looking for. I4 described this as their biggest challenge, stating that since they do not know how many of the devices in the field actually report data, it is hard to know how representative it is and to draw reliable conclusions from it. I7 also reported understanding what the data means as a challenging part of working with the diagnostics database. Beyond this, some participants struggled in a more basic way with

just understanding the data and how it can be used. This barrier led to a design goal of data understanding, meaning the assistant should help users understand the database structure and what the data represents.

The third area, **querying and analysing the data**, which most used the business intelligence tool for, was a challenge for some but not all. I4 reported getting stuck when using the drag-and-drop interface, stating it can be difficult to get past errors that arise when creating a chart. I2 also reported not knowing how to use the tool, describing it as much more complicated than expected. I6, on the other hand, reported having no problem with the drag-and-drop interface, saying they learned it early on at the case company, and the biggest issue they face is the time it takes for the tool to load the data from the database. I7 said that compared to finding relevant data and understanding it, creating a chart is the easier step. Overall, querying and analysing the data seemed to be more of a barrier for those who had less experience with the business intelligence tool, while those who had learned how to use it found it satisfactory.

The fourth area, **structural barriers**, covers aspects that were not tied to any specific step in the workflow but affected data work more broadly. One such aspect was employees forgetting what they had learned. I7 reported having gone to an introduction for learning how to use the data and the business intelligence tool, but had forgotten what they had learned and felt that they would have to start over. Another aspect that was commonly mentioned was time and priorities. Many interviewees said that they want to use the data more and think it can be useful in many ways, but they stick to basic visualisations due to a lack of time. Multiple participants mentioned having more important things to do, and working with the diagnostics data was described as a peripheral activity rather than part of their core work. Not all participants shared this experience, however. I5 reported that working with the data was a recognised part of their responsibilities.

Beyond the barriers themselves, participants also described how they sought help when facing problems. Many already relied on AI tools to resolve errors when making charts or to come up with SQL queries that fit their purpose. Others reached out to the DDM team when facing problems they did not know how to solve on their own, such as figuring out whether setting up a particular dashboard was possible.

5.1.3 User Needs for the Assistant

The interview participants described their desired capabilities for a database assistant. These included help with finding and understanding data, querying and analysing data, and also some other features. They also discussed how they would approach trusting the assistant's outputs.

Regarding **finding and understanding data**, multiple participants wanted the assistant to help them find relevant data in the database. I5 gave the example of wanting to ask if there is data about temperature, and having the assistant locate it in the database. I3 similarly expressed a desire to ask about some type of measurement and have the assistant answer whether the database contains information about that. I5 felt this capability would be especially valuable when exploring unfamiliar parts of the database. Beyond locating the data, interviewees also wanted the assistant to help them interpret the data once they had found it. I3 said that the assistant being able to help them find and understand data would be valuable on its own, even without any other capabilities.

Regarding **querying and analysing the data**, the needs varied between participants. I3, for example, wanted to be able to ask straightforward questions and get direct answers, presented as tables or values. I1, I2, and I7 wanted the assistant to create charts, for example visualising how a value changes over time, but I7 would also accept instructions on how to create charts instead. Regarding SQL generation, preferences diverged. I1 and I7 wanted the assistant to generate SQL that they could run themselves, while I2 and I3 preferred not to see SQL at all, and to instead just get an answer. I5 suggested making this configurable, noting that colleagues without SQL knowledge would benefit from the assistant querying the database and presenting results directly. Overall, the theme was that the assistant should reduce manual work. These needs informed design goals related to data retrieval and SQL assistance.

In addition to the capabilities described above, participants also mentioned **additional features**. One suggested feature by I1 was that the assistant should have information about the specific user environment, such as which database management system version was in use, allowing it to adapt suggestions to that. I2 wanted the assistant to have knowledge of the entire data workflow, from writing data collection code on the device to visualising the data in the business intelligence tool, in order to guide users through each step. I1 wanted the assistant to validate its own outputs, for example by testing SQL before presenting it, instead of users having to test every query and provide the assistant with potential error messages. From this, we identified output evaluation as a design goal. I2 also mentioned that the assistant could suggest what data might be worth analysing, which led to a design goal of analysis suggestion. Another design goal, analysis execution, was motivated by the need of I6. They wanted help with analytical tasks, such as identifying new error logs from the past two weeks that had not appeared previously, since their team currently spent considerable time doing this manually. Finally, I7 suggested integrating the assistant directly into the chart-making tool so it could see the user's environment and context. Overall, participants prioritised simplicity and saving time above all else. I7 also noted that they did not expect the assistant to be perfect, but wanted it to handle basic tasks reliably. I3 said that it was important for the assistant to communicate its progress, so that users do not have to wonder whether it had crashed or stalled when processing longer requests. I4 suggested that the assistant should check whether a similar dashboard already exists before helping create a new one, to avoid duplication across teams as the number of users grows.

Participants had different attitudes towards **trusting and verifying the assistant's outputs**. Some said they would not fact-check the data the assistant provided, since they lacked sufficient knowledge of the data beforehand to do so effectively. Others took a more cautious approach, saying they would verify everything the assistant produced. For example, SQL queries could be checked, though this required understanding both the data and SQL to confirm whether it matched what they wanted.

5.2 Design Goals

Based on the interview findings and technical constraints, the following design goals were formulated for the assistant:

- **Data discovery**: The assistant should help users find relevant tables and columns.

- **Data understanding**: The assistant should help users understand the database structure and what the data represents.
- **Data retrieval**: The assistant should help users retrieve data from the database.
- **SQL assistance**: The assistant should assist users in formulating correct and effective SQL queries.
- **Analysis execution**: The assistant should help users answer analytical questions that go beyond retrieving raw data, such as aggregations, comparisons, or trends.
- **Analysis suggestions**: The assistant should help users identify relevant analyses based on available data and their goals.
- **Output evaluation**: The assistant should assess the correctness of its own queries and results before presenting them to the user.
- **Ambiguity handling**: The assistant should ask clarifying questions when the task is not straightforward.

5.3 The Implemented Solution

The implemented solution is an LLM-based assistant in the form of an agent system, with tools, files, and instructions for incorporating database knowledge and guiding SQL generation, as illustrated in Figure 5.2. It is based on the design goals presented in the previous section. The agent system was created in LibreChat's Agent Builder panel, and many of its elements were added through an MCP server, which we developed and connected to the agents. The tools in the server provide access to external systems such as the diagnostics database. Other elements were configured directly in the Agent Builder panel, for example by uploading a file. To use the assistant, users write prompts in a chat interface, and these prompts are sent to the LLM along with a system prompt and information about the agent capabilities. If instructed to do so by the output of the LLM, the system can then invoke one of the MCP tools, search the files, or transfer to another agent. As part of the evaluation described in Section 4.3.1, we grouped all the implemented elements of the solution into seven components, each containing one or more elements. This section describes the elements of the solution, and for clarity they are presented in the same groupings as those used in the evaluation, with one subsection per component.

5.3.1 SQL-Based Exploration Tools

The first component consists of five MCP tools with the purpose of giving the agent access to information about the structure and contents of the database. They achieve this by connecting to the database and, with the exception of one tool, querying its `system` schema, which contains metadata about the schemas, tables and columns. All these tools rely on prewritten, parametrised SQL queries to retrieve the relevant information. When an agent invokes one of them, the schema, table or column of interest is specified using arguments, and the query is executed with these as parameters. If the query successfully returns a result, that result is sent

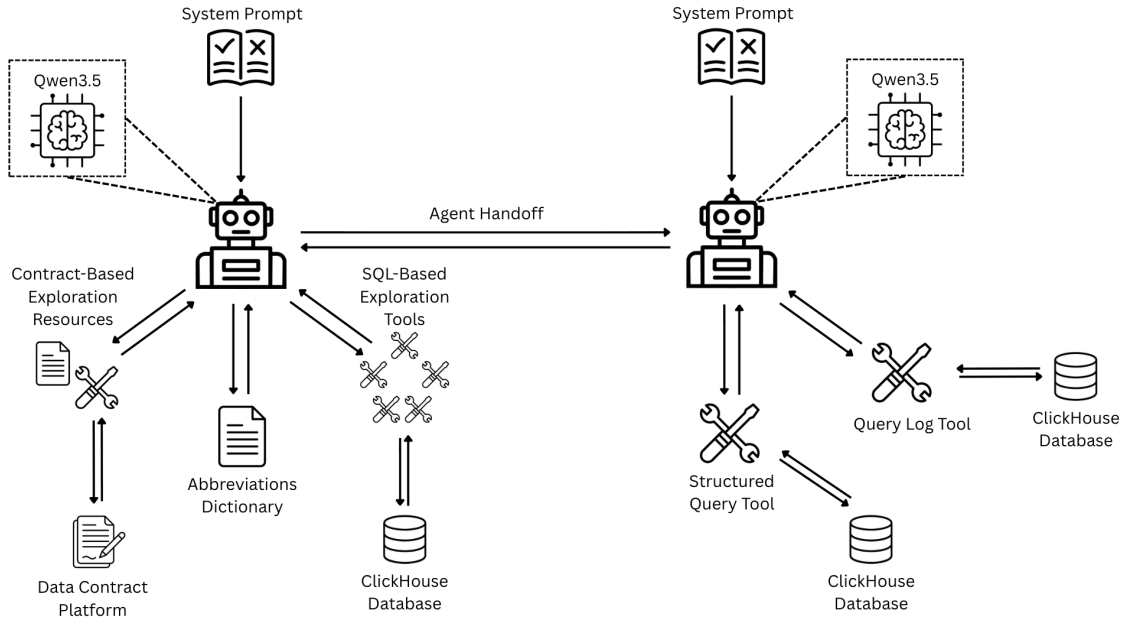


Figure 5.2: The complete implemented solution, with the elements grouped into the components that were used for evaluation. The two system prompts are considered the same component and the separation of the elements across two agents is also considered a component. Qwen3.5 is the LLM that the assistant uses, and the contract platform and the database are systems that the elements communicate with, rather than elements or components themselves.

back to the agent and included as context for the LLM, providing information that guides the next action. In the case that the query causes an error in the database or returns an empty result, information about this is relayed to the agent through an error message, allowing it to adjust and produce a corrected tool call.

Four of the tools use the metadata tables in the `system` schema, more specifically `system.tables` and `system.columns`, to retrieve information about the database. One of them is called `get_tables_in_schema`, and as the name implies, it is used to reveal which tables a given schema contains. It takes the schema name as input, and returns a list of tables which are listed as belonging to that schema. A similar tool is `get_columns_in_table`, which lists columns that exist in the table specified by the arguments. There are also two tools for retrieving more descriptive information about the tables and columns, using the `comment` column of the two metadata tables. They are called `get_table_comment` and `get_column_comment`, and like the previous tools, they have arguments which specify the filtering conditions in the query to the respective metadata table.

The fifth tool, `get_table_sample`, is the only one of the SQL-based exploration tools that does not use the metadata in the `system` schema. Instead, it queries the table of interest, which is specified using the arguments in the same way as for the other tools. The purpose of this tool is to provide the LLM with an example of what actual data in the table looks like, to potentially reveal information that was not apparent from the column names or comments. Therefore, it uses a `SELECT *` statement to retrieve values from all columns of the table. In an attempt to make the results as useful as possible, the query uses filtering conditions to

ensure that returned rows have been recently inserted and come from three different devices. Furthermore, those devices are chosen so each represents a distinct product model, to increase the probability of capturing variety between types of products. To inform the agent of how entries vary within the same device, five rows are returned for each.

5.3.2 Contract-Based Exploration Resources

Another tool in the MCP server is `get_table_contract`, which integrates with the case company's platform for data contracts. A data contract is a formal specification of a dataset that contains information such as table and column names, descriptions, data types, and ownership. Like the SQL-based exploration tools, the purpose of this tool is also to provide the agent with knowledge about the tables and columns, but using a different source of information than the tools in the previous subsection. The tool accesses the API of the contract platform and calls an endpoint which returns the data contract for a single table. When invoking the tool, the agent specifies the target table with arguments, and the tool fetches the contract of that table from the API endpoint. Upon a successful request, the contract is parsed and extracted from the response, and returned to the agent in JSON format. If an error occurs, the error message is sent to the agent instead, informing it about what went wrong.

To complement the `get_table_contract` tool, another resource based on the data contracts is included in the assistant. This is a Markdown file called `table_names_and_descriptions.md`, which summarises the contracts by listing all tables and their descriptions. This provides an overview of the tables which for example can be used to identify a table potentially containing the sought data, after which `get_table_contract` can be used to examine the table more thoroughly. The file is uploaded to the File Search capability in LibreChat's Agent Builder, allowing the agent to use RAG to fetch the most relevant parts of it. Like `get_table_contract`, `table_names_and_descriptions.md` also has the contract platform API as its source of information. It was created with a Python script which retrieves all contracts, parses them, extracts the names and descriptions, and generates the file in the new format. If a table is added or removed, or a description is changed, the file has to be generated anew by running the script.

5.3.3 The Abbreviations Dictionary

The implemented solution also includes the file `abbreviations_explanations.json`, which is a JSON file acting as a dictionary for abbreviations that could be encountered when interacting with the database. Its purpose is to give the agent the correct interpretation of terms, which may either not have been seen during training of the LLM or may have different meanings depending on the domain. This information could be useful both when explaining something in the database to the user and when understanding of the contents is needed to produce a correct query. The file is based on a resource at the case company's intranet, which was exported and converted to JSON through Python code. Like `table_names_and_descriptions.md`, the abbreviations file is not updated automatically and has to be regenerated to reflect any changes in the source. Also similarly to that file, the contents of this file are retrieved by the agent through RAG.

5.3.4 The Query Log Tool

In addition to tools for giving the assistant domain information, we also implemented tools with more focus on the SQL generation aspect. While knowledge of the database's composition is important when querying it and not just when explaining it to the user, we also attempted to improve the quality of generated queries by adding tools with that specific purpose. The first of these is `get_previous_queries`, which uses the `query_log` table in ClickHouse's `system` schema to fetch queries that have previously been executed against the database, and gives them to the LLM as examples. The reasoning behind this is that provided examples of how a table, column, or function is used can increase the probability of them being used correctly in the queries generated by the model.

In order to obtain the statements that are most likely to be helpful, the results are filtered to only include `SELECT` queries that executed successfully, and that were initialised by a client rather than by another statement. Furthermore, the tool only returns queries within a certain length range. We set the lower boundary of this range to 40 characters to filter out very short and trivial queries, and the upper boundary to 2000 characters to avoid overly long and complicated queries produced automatically by systems. The tool has four arguments, which are used to further narrow down the set of matching queries. The first two, `schema` and `table`, are required to use the tool, and they limit the results to queries that include the specified schema and table. The other two, `columns` and `functions`, are optional. If `columns` is included in the call, the tool returns only queries that involve at least one of the columns in the argument. If `functions` is included, the queries are limited to those that use at least one of the ClickHouse functions included in the argument. To increase the variation of the returned queries, they are deduplicated so that queries which are identical or only differ by values of literals only appear once, and additionally so that no more than three queries from the same user are included. The ten most recent queries satisfying these criteria are returned from the tool to the agent.

5.3.5 The Structured Query Tool

Another MCP tool we developed with the aim of improving SQL quality is `execute_query`. This tool takes a JSON object called a *query plan* as input, which it uses to construct an SQL query that is then executed against the database. An example of a query plan and the SQL query it corresponds to is shown in Figure 5.3. The structure of the query plans is defined in the docstring of the tool, and needs to be followed in order for a query to be successfully generated. The rationale behind letting the LLM output a structured plan instead of a raw SQL query is primarily that the explicit format encourages systematic consideration of each part of the query, potentially reducing the risk of mistakes. In the implementation of the tool, the query plan is first validated, and if it is invalid, the agent is informed of the problems through error messages which are designed to be as specific and detailed as possible. Since the required structure of the query plan is complex, the aim of the validation messages is to maximise the probability that the LLM generates a valid query plan on the next attempt. If the query plan follows the specified rules, a query string is incrementally assembled based on the keys that are present in the plan and their values. The generated query is then executed against the database and the result is returned to the agent. To reduce the risk of sending excessively large amounts of data to the LLM and exceeding its context window, a default

`LIMIT` clause restricting the result to 100 rows is added to the query if no limit is specified.

The structure of the query plans is designed with the aim that all queries that the assistant needs to run should be representable with a query plan. To this end, it includes fields for most of the commonly used ClickHouse SQL clauses. In the docstring, these fields are listed, and the allowed formats for each of their values are described along with what they represent in the resulting query. For many of the fields, the values involve an *expression*, a concept which is also defined in the tool's docstring. A query plan expression is a JSON object nested inside the query plan, with `type` as one of its keys. It is used to represent an SQL construct such as a literal, a function call, or an operator. The value of `type` determines what the expression represents, and thus what additional fields are required to fully specify it. In addition to the structured types supported by the tool, there is also the possibility to insert raw SQL code through the expressions. This is however explicitly discouraged in the docstring, and should only be used when the structure otherwise cannot represent the desired query.

Since some way to query the database is needed in order to answer any data retrieval questions, the assistant would lose much of its functionality if `execute_query` were removed. However, the complexity of the tool is not necessarily needed. We therefore implemented an alternative tool called `execute_any_query`, which takes any string as an argument and tries to execute it as an SQL statement. Protection against unsafe commands is handled by restrictions on the account that the MCP server uses to access the database. The purpose of this alternative tool was to determine whether the more constrained `execute_query` offered any benefit in comparison. For phase 2 of the evaluation, we therefore made `execute_query` component E, and replaced it with `execute_any_query` in the configurations that did not include that component.

5.3.6 The System Prompt

In addition to the tools and files used to give the assistant capabilities, we also wrote a system prompt intended to control its actions. These instructions are automatically added at the start of every conversation, in between the system prompt of the LLM and the prompt entered by the user. The system prompt is based on what we, our supervisors, and the interviewed intended users considered desirable behaviour. We also included information and instructions addressing some of the common mistakes that we observed when testing versions of the assistant that did not have the system prompt. To increase adherence to these instructions and steer the agent towards better performance, we considered established best practices for prompting and used several common prompt engineering techniques, including few-shot, chain-of-thought, and role-based prompting. The system prompt was adapted slightly to fit each configuration, but most of the content was the same across all versions, including the one used in the final solution. This includes a `# Role` section where the agent is assigned a role and a high-level task, a `# Context` section providing relevant information about the database, and a `# Rules` section listing a number of rules that the agent should always follow. In addition to this, we wrote short instructions for what each of the tools and files should be used for, and included these in a `# Tools` section in the agents with the corresponding components. Finally, we also provided an `# Example` section, with a question that the agent might get, a chain-of-thought reasoning process with the steps to handle that question, and a correct SQL query for answering it. The version of the system prompt that was used in the final solution can be seen in Appendix B.

```
{
  "select": [
    "column_a",
    {
      "expr": {
        "type": "func",
        "name": "count",
        "args": [
          "column_b"
        ],
        "distinct": true
      },
      "alias": "distinct_count"
    },
    {
      "expr": {
        "type": "case",
        "when": [
          {
            "condition": {
              "type": "binary_op",
              "op": ">",
              "left": {
                "type": "func",
                "name": "sum",
                "args": [
                  "column_c"
                ],
                "distinct": false
              },
              "right": {
                "type": "literal",
                "value": 100
              }
            },
            "result": {
              "type": "literal",
              "value": "high"
            }
          }
        ],
        "else": {
          "type": "literal",
          "value": "low"
        }
      },
      "alias": "level"
    }
  ],
  "from": "example_schema.table_one",
  "joins": [
    {
      "table": "example_schema.table_two",
      "type": "LEFT",
      "on": {
        "type": "binary_op",
        "op": "=",
        "left": "table_one.column_d",
        "right": "table_two.column_d"
      }
    }
  ],
  "where": {
    "type": "binary_op",
    "op": "AND",
    "left": {
      "type": "in",
      "left": "column_e",
      "right": [
        "value_1",
        "value_2"
      ]
    },
    "right": {
      "type": "binary_op",
      "op": ">",
      "left": "table_two.column_f",
      "right": {
        "type": "literal",
        "value": 10
      }
    }
  },
  "group_by": [
    "column_a"
  ],
  "order_by": [
    {
      "expr": "distinct_count",
      "direction": "DESC"
    }
  ],
  "limit": 20
}
```

```
SELECT
    column_a,
    count(DISTINCT column_b) AS distinct_count,
    CASE
        WHEN sum(column_c) > 100 THEN 'high'
        ELSE 'low'
    END AS level
FROM example_schema.table_one
LEFT JOIN example_schema.table_two ON table_one.column_d = table_two.column_d
WHERE column_e IN ('value_1', 'value_2')
    AND table_two.column_f > 10
GROUP BY column_a
ORDER BY distinct_count DESC
LIMIT 20
```

Figure 5.3: An example query plan (top) and the SQL query that the structured query tool generates from it (bottom).

5.3.7 The Multi-Agent System

Finally, we turned the agent into a multi-agent system by splitting the previously implemented components between two agents. This separation is intended to give each agent a smaller and more focused task compared to when everything is handled by the same agent, which could improve performance. In the multi-agent version of the assistant, the first agent has the components for providing knowledge of the database, i.e. those belonging to group 1 in Table 4.2. The components for improving SQL quality and querying the database, i.e. those in group 2, are instead included in the second agent. Both agents have a system prompt that is largely the same as that of the single-agent version, with minor adjustments made to reflect their respective capabilities. From a user perspective there is little difference between the multi-agent system and the single-agent version, as prompts can be given to the first of the two agents in the same way that they are to the single agent. The agents themselves handle the switching between each other based on when which capabilities are needed, and all outputs appear in the same response, as if given by a single agent. The implementation of this uses LibreChat's Agent Handoff functionality, where each agent is added to the other as handoff agent. It also includes a short instruction to each agent describing when the task should be handed over to the other. The multi-agent system is considered a component in itself, with the absence of the component being defined as the use of a single agent containing all elements.

5.4 Results of Comparative Configuration Evaluation

In the first part of the Solution Evaluation step, we asked a set of evaluation questions to different configurations of the developed assistant in order to determine which of our implemented components contributed to improving its performance. The result of this was a final solution in the form of an agent system with some of the components. The components that the assistant could perform equally well or better without were excluded from the final version. In this section, the results of this comparative evaluation between components are presented, structured according to the three phases where the three groups of components were evaluated. A more detailed breakdown of the results, including per-question scores for each configuration, is provided in Appendix C. For convenience, the configurations are referred to with names reflecting the components they include, with a zero representing the absence of components. For example, in the phase where components A, B, and C are compared, the configuration with no components is referred to as Agent 0, the configuration with only component A is referred to as Agent A, and the configuration with all three components under evaluation is referred to as Agent ABC.

5.4.1 Results From Phase 1

Phase 1 of the evaluation focused on determining which subset of the components used for incorporating domain knowledge performed best on questions related to database exploration. The components under evaluation were thus the SQL-based exploration tools (component

A), the contract-based exploration resources (component B), and the abbreviations dictionary (component C). This meant that eight agents were tested in this phase. Among the questions of the objective type, E1–E12 in Table 4.3, there were none that were answered correctly by all eight agents. The question with the most correct answers was E11, which only Agent B failed to answer correctly. One question, E6, had no correct answers across the eight agents. However, Agent BC and Agent ABC were closer than the others, giving answers that were just subtly wrong. There were some instances of agents with added components answering questions wrongly that agents with a smaller subset of those components succeeded on. Agent 0 answered E8 correctly, which neither Agent A nor Agent C did. Similarly, adding component B to Agent 0 resulted in question E11 going from correct to incorrect. The responses to the questions of the subjective type varied, with each question receiving every grade from 0 to 3 at least once across the agents. The average grades across agents were relatively similar for all three questions, ranging from 1 for question E13 to 1.875 for question E15. Also on these questions, there were occurrences of agents receiving lower grades compared to counterparts with fewer components on the same questions. E13 was the most noteworthy question in this regard, as all three single-component agents achieved lower grades than Agent 0, and Agent ABC only reached grade 1.

Table 5.1 shows the score that each configuration achieved and the average time elapsed before their responses came. The results showed a large spread in the number of correctly answered objective questions, between 1 and 11 out of a maximum of 12, and in the total grade from the subjectively judged responses, between 0 and 7 out of a maximum of 9. Meanwhile, most configurations had an average response time between 5 and 7 seconds, with lower averages being achieved by the agents with the least capabilities and the lowest response quality scores. The agents with the highest response quality score were Agent AB and Agent ABC. Out of these, Agent ABC had a slightly lower average response time. Furthermore, adding component C increased response quality in Agent A, from 14 to 15, and in Agent B, from 13 to 16. Therefore, we deemed the configuration including all three components, Agent ABC, to be the winning combination in phase 1, meaning that $S^*_{ABC} = \{A, B, C\}$ using the notation introduced in Section 4.3.1.

5.4.2 Results From Phase 2

In phase 2, the contribution of the query log tool (component D) and the structured query tool (component E) was assessed by evaluating four different configurations with questions that should be answered by querying the database. Since Agent ABC had performed the best in phase 1, components A, B, and C were included in all configurations used in phase 2, while components D and E were varied. The results from this phase showed lower variation compared to phase 1. Five of the objective questions (D1, D2, D3, D4, and D5) were answered correctly by all four agents, while three (D6, D8, and D12) were answered incorrectly by all. Thus, the results of the agents differed in only four of the twelve objectively verified questions. On the subjectively judged questions, D13–D15, no response reached the top grade and eleven of the twelve total responses received either grade 1 or grade 2 out of 3. D15 was the question that the agents handled best on average, with all four responses receiving the grade 2. Meanwhile, the responses to D13 received an average grade of 1 and the responses to D14 received an average grade of 1.25 across the four configurations. Different questions showed different indications regarding whether the components D and E increased performance. The

Table 5.1: Summary of the results in phase 1 of the comparative configuration evaluation. Since the twelve exploration questions of the objective type were asked once each, the maximum score from those was 12. The three subjective exploration questions were also asked once each, and graded from 0 to 3, so the maximum score from those was 9. The response quality score was calculated as the ratio of correct answers to the number of objective questions plus the ratio of the grade sum to the maximum total grade possible, meaning it had a maximum value of 2.

Included Components	Correct Answers (E1–E12)	Grade Sum (E13–E15)	Response Quality Score	Avg. Response Time
None	2 / 12	2 / 9	0.39 / 2.00	3.7 s
A	9 / 12	5 / 9	1.31 / 2.00	9.7 s
B	7 / 12	6 / 9	1.25 / 2.00	5.3 s
C	1 / 12	0 / 9	0.08 / 2.00	2.6 s
A, B	11 / 12	6 / 9	1.58 / 2.00	6.7 s
A, C	10 / 12	5 / 9	1.39 / 2.00	6.4 s
B, C	9 / 12	7 / 9	1.53 / 2.00	6.8 s
A, B, C	11 / 12	6 / 9	1.58 / 2.00	5.7 s

best response to D14 was given by the agent with neither component, referred to as Agent ABC0, while D7 was only answered correctly by the agent with both components, Agent ABCDE.

The scores and response times for phase 2 of the evaluation are shown in Table 5.2. The performance was relatively similar across all configurations, with the highest number of correct answers to the objective questions only being two greater than the lowest, and the best-performing agent on the subjectively judged questions only achieving two total grade points more than the worst performer. When comparing the total response quality scores, the best configuration was Agent ABC0, which had the joint highest score on both types of questions. This most minimal configuration also had the lowest average response time across all questions. Based on these results, we concluded that neither component D nor component E improved the performance of the assistant, and that they would therefore not be included in the rest of the testing or part of the final solution. Thus, $S^*_{DE} = \{\}$.

5.4.3 Results From Phase 3

The third and final phase of the comparative configuration evaluation was performed to assess the components that are related to affecting the assistant's behaviour rather than adding new capabilities. This meant that we compared configurations with the system prompt to configurations without it, and compared the multi-agent system to the single agent. A system prompt had been used in the previous phases to make sure the other components could be evaluated fairly, but in phase 3 we removed it from some configurations to observe if the prompt actually improved performance. Therefore, the system prompt became component F in this phase, while the separation into two different agents was evaluated as component

Table 5.2: Summary of the results in phase 2 of the comparative configuration evaluation. Since the twelve data retrieval questions of the objective type were asked once each, the maximum score from those was 12. The three subjective data retrieval questions were also asked once each, and graded from 0 to 3, so the maximum score from those was 9. The response quality score was calculated as the ratio of correct answers to the number of objective questions plus the ratio of the grade sum to the maximum total grade possible, meaning it had a maximum value of 2.

Included Components	Correct Answers (D1–D12)	Grade Sum (D13–D15)	Response Quality Score	Avg. Response Time
A, B, C	8 / 12	5 / 9	1.22 / 2.00	30.9 s
A, B, C, D	7 / 12	3 / 9	0.92 / 2.00	41.3 s
A, B, C, E	6 / 12	5 / 9	1.06 / 2.00	42.8 s
A, B, C, D, E	8 / 12	4 / 9	1.11 / 2.00	53.3 s

G. Since these components were intended to improve behaviour and performance on both exploration tasks and data retrieval tasks, the configurations were tested with all 30 questions rather than just those from one of the categories. Because the combination of components A, B, and C was the best in phase 1 and the agent with neither of the components D nor E was the best in phase 2, the configurations used in phase 3 all included A, B, and C but not D or E.

The results from the individual questions contained both differences and similarities compared to the earlier phases. Of the 24 questions of the objective type, there were eleven (E1, E4, E5, E11, E12, D1, D2, D3, D4, D5, and D10) that all configurations answered correctly and four (D6, D7, D8, and D12) that none succeeded on. Thus, there were just nine objective questions that revealed any differences between the different configurations. E6, which was the only objective exploration question that all agents in phase 1 failed on, was answered correctly by two of the four configurations in phase 3, specifically the two with a system prompt included. The results of the data retrieval questions were relatively similar compared to phase 2, as all the questions with a uniform result across agents in phase 2 also had a uniform result across agents in phase 3. However, in this phase, D10 was answered correctly by all configurations and D7 by none, which was not the case in phase 2. From the subjectively graded questions, the most noteworthy result was that the best response to E14 was given by the configuration with neither component F nor component G, referred to as Agent ABC00. On the rest of these questions, the assistant generally seemed to perform equally well or better with at least component F included.

In Table 5.3, the aggregated results for each configuration in phase 3 are presented. They show that the two agents with component F, i.e. the system prompt, performed better than the two agents without that component. The difference was largest on the objectively verified questions, where especially Agent ABC0FG gave many correct answers. This led to it achieving the highest total response quality score despite Agent ABC0F performing slightly better on the subjectively judged questions. The response times show that between the two configurations with a system prompt, the single-agent system Agent ABC0F was slightly

faster on average. However, we deemed this difference small enough to not outweigh the response quality score. We therefore considered Agent ABC0FG to be the best-performing configuration in phase 3, which meant that $S^*_{FG} = \{F, G\}$.

Table 5.3: Summary of the results in phase 3 of the comparative configuration evaluation. Since the 24 questions of the objective type, twelve from each category, were asked once each, the maximum score from those was 24. The six subjective exploration questions, three from each category, were also asked once each, and graded from 0 to 3, so the maximum score from those was 18. The response quality score was calculated as the ratio of correct answers to the number of objective questions plus the ratio of the grade sum to the maximum total grade possible, meaning it had a maximum value of 2.

Included Components	Correct Answers (E1–E12, D1–D12)	Grade Sum (E13–E15, D13–D15)	Response Quality Score	Avg. Response Time
A, B, C	12 / 24	10 / 18	1.06 / 2.00	37.8 s
A, B, C, F	16 / 24	12 / 18	1.33 / 2.00	24.1 s
A, B, C, G	12 / 24	7 / 18	0.89 / 2.00	24.1 s
A, B, C, F, G	20 / 24	11 / 18	1.44 / 2.00	27.8 s

5.4.4 The Selected Final Solution

Based on the comparative configuration evaluation, the contributing components were determined to be A, B, C, F, and G. Thus, the final solution is a multi-agent system with two agents which both have a system prompt. The first agent in the system has the SQL-based exploration tools, the contract-based exploration resources, and the abbreviations dictionary included. Since neither component D nor component E were shown to improve performance, the second agent has, apart from the system prompt, only the tool `execute_any_query`, mentioned in Section 5.3.5. This enables the agent to query the database, but adds no functionality for improving the generated SQL queries. In Table 5.4, the results on each individual evaluation question for the final solution are presented.

5.5 Feedback From User Testing

User testing provided insights into how the assistant was perceived and used by intended users, highlighting both its strengths and its limitations. The participants were instructed to test the assistant freely and submitted feedback through an open-ended questionnaire, as described in Section 4.3.2. Therefore, the tested use cases varied, and although some common themes can be identified in the feedback, the comments had limited overlap. This section summarises participants' experiences, including successful use cases, encountered challenges, perceived usefulness, and suggested improvements.

Table 5.4: The result on each evaluation question achieved by the configuration chosen as the final solution. Questions E1–E12 and D1–D12 were verified against a known correct answer while E13–E15 and D13–D15 were graded on a scale of 0 to 3.

Question ID	Correct	Grade	Response Time
E1	Yes	N/A	1.1 s
E2	Yes	N/A	1.1 s
E3	Yes	N/A	3.8 s
E4	Yes	N/A	2.0 s
E5	Yes	N/A	1.4 s
E6	Yes	N/A	10.1 s
E7	Yes	N/A	2.9 s
E8	Yes	N/A	2.8 s
E9	Yes	N/A	4.6 s
E10	Yes	N/A	10.2 s
E11	Yes	N/A	1.5 s
E12	Yes	N/A	1.8 s
E13	N/A	3	9.2 s
E14	N/A	2	8.5 s
E15	N/A	3	7.7 s
D1	Yes	N/A	24.9 s
D2	Yes	N/A	7.4 s
D3	Yes	N/A	10.3 s
D4	Yes	N/A	34.2 s
D5	Yes	N/A	9.9 s
D6	No	N/A	122.8 s
D7	No	N/A	18.7 s
D8	No	N/A	11.6 s
D9	Yes	N/A	68.0 s
D10	Yes	N/A	6.8 s
D11	Yes	N/A	97.8 s
D12	No	N/A	23.4 s
D13	N/A	1	24.0 s
D14	N/A	1	37.0 s
D15	N/A	1	267.8 s

Participants reported that the assistant handled a range of tasks successfully. Examples included answering questions about which tables contain certain types of data and which devices had the highest reported temperatures among a certain population. Another task that the assistant was given was to calculate the average duration of calls from intercom products. The employee who tried this found the response to be clear, well explained, and nicely presented, and said that the answers seemed reasonable but did not verify them. One participant reported that most of their questions were answered correctly, but that they did not double-check the answers to all of them. Another used it to generate SQL, and reported that

the assistant could provide a working query. Furthermore, multiple participants mentioned that it retrieved data relatively quickly, which was another positive aspect.

There were however also limitations encountered during the testing. One participant experienced connection issues that interrupted their conversations and prevented them from testing everything they wanted. The same employee reported simpler queries being handled well, but more advanced use cases, such as optimising an SQL query, failing. The assistant also reportedly produced some queries that contained errors. In particular, it failed to correctly handle SQL queries that included templating, a mechanism where the query contains placeholders that are filled in before the query runs, thus not fulfilling one participant's use case. The assistant was also reported using data from 2024 when more recent data would have been more helpful. One participant highlighted that the assistant lacked access to a dataset that was particularly relevant to their work, which limited its usefulness. A third participant reported that although response times were generally fast, calculating averages took longer than other types of queries.

Overall, the participants perceived the assistant as useful, but also suggested some improvements. One said it gave them interesting and fun facts that they can use in their daily work. Another stated that they would probably use it occasionally, but noted that it would be even more useful if it had access to another dataset that their team uses. A third emphasised that this is exactly the type of thing that is needed to make data more accessible at the case company. One participant wanted the assistant to be integrated into the business intelligence tool that most teams use to visualise data, suggesting it could be used to leave AI-generated comments on their dashboards with an analysis of the data, for example by finding and reporting unusual values. Another participant suggested that the assistant could be better at asking clarifying questions, such as which SQL environment is used, rather than presenting too much information at once and letting the user specify which parts are relevant.

Chapter 6
Discussion

In this chapter, we analyse and discuss the results presented in the previous chapter, including what factors influenced them and what the implications of the results are. The chapter is organised in four sections, the first three of which roughly correspond to each of the three research questions. The fourth section covers some of the identified threats to the reliability and generalisability of the results.

6.1 Analysis of User Needs and Barriers (RQ1)

The interviewed potential users of our assistant worked with the case company's diagnostics database in different ways and faced different barriers in doing so. Understanding of this was an important first step in designing a useful solution, and it guided us in the development process. This section analyses the findings related to how the interview participants use the diagnostics data and what difficulties they encounter, and discusses what they mean for the design and the benefit of an LLM-based database assistant.

Based on our findings, there was **no standard way in which participants used the database**. Some were active users, meaning they interacted with the data themselves and created their own visualisations, whereas other interviewees only consumed what others had made and discussed it in meetings. Overall, the most common way to draw insights from data was through dashboards in the business intelligence tool. Employees were generally familiar with their own teams' dashboards, but there did not seem to be a lot of cross-team collaboration on data analysis. This suggests that valuable insights may not reach beyond the teams that produced them, something a database assistant aware of existing dashboards could help address. Participants who were more familiar with the database were more content with their current situation and more independent in their pursuit of solutions when they encountered problems, whereas the less familiar participants expressed that they did not use the data as much as they wanted. Overall, the participants were positively inclined towards using the case company's diagnostics data more in their work.

Among the **barriers participants faced**, the most common was finding relevant data in the database. Much of the existing research on natural language interfaces to databases focuses on the SQL generation step, but our findings suggest that the earlier steps, such as finding relevant data, are at least equally important. Two observations support focusing on this barrier. First, since the business intelligence tool where most employees access the data has a drag-and-drop interface, SQL is not strictly required to visualise the data. Once users have located and understood the data, mouse-based interaction may therefore be sufficient to extract value from the data. This means that it may be more beneficial to focus solution development efforts on tasks such as helping the user find relevant data, explore the database, and understand its contents. Second, even users who were more familiar with the database reported feeling lost in parts of the database they did not frequently use. There might thus be unrealised potential in the data that is not used because employees do not realise it exists and could be of use to them, which again suggests that data discovery could be a valuable focus for the assistant. This focus could also be particularly helpful for less experienced users, who described not knowing where to start working with the database. Although data discovery might be a good starting point, supporting experienced users in use cases that are more advanced than database exploration is also valuable since they may produce insights that benefit the wider organisation, and share their knowledge with colleagues.

While finding relevant data was a common barrier, participants had different ideas about **what a database assistant should help them with**. Some wanted advanced SQL help, others wanted help with more basic tasks, such as understanding what kind of data a table contains, and some wanted the assistant to create dashboards or be integrated into the business intelligence tool. Ideally, a database assistant should support all of these needs, but designing a single system that does so is difficult given how much the needs differed. However, one recurring theme was that several participants did not have enough time to allocate to working with the data, and one had even forgotten how to use the business intelligence tool because it had been too long since they last used it. This suggests that, above all, the assistant should be a time-saving tool for employees at the case company.

6.2 Design Decisions and Effects (RQ2)

To implement a solution that would provide as much help to the users as possible, we explored several aspects of LLM-based assistants. Some approaches were deemed infeasible, unsuitable or insufficiently promising, while others were pursued and investigated. In this section, we discuss how the assistant was affected by the components we implemented, as well as the consequences of some of the limitations and decisions.

In addition to user needs and design goals, **the implementation of the assistant was also influenced by factors** such as time constraints and the existing technical environment at the case company. For example, we were limited to using the company's instance of Qwen3.5, which we could not fine-tune or modify in any other way. Additionally, the goal of making the final solution usable as part of the company's existing chatbot application meant that we based it on LibreChat, which in turn had consequences on the design. LibreChat does not support usage of static resources or prompt templates from MCP servers, so those capabilities were not considered in the solution and our MCP server contained only tools. We also based the elements not related to MCP on the features of LibreChat, like File Search and

Agent Handoff. The usage of File Search for RAG led to us having limited control of how the uploaded files were chunked. This meant that table descriptions and term definitions were not always kept intact, which may have reduced the effectiveness of the RAG functionality in our solution. Moreover, some unexpected behaviour related to the switching between agents was sometimes observed when we informally experimented with the final version of our assistant, and a possible reason for this is that LibreChat's Agent Handoff was a beta feature at the time of this thesis. In addition to the consequences for the implementation, the usage of LibreChat also affected the evaluation process somewhat, as it was not straightforward to access all agent capabilities programmatically. This led to the evaluation being carried out manually, which limited the total number of tests that could be performed within the available time.

In the results of the comparative configuration evaluation, presented in Section 5.4, we noted that **adding components sometimes seemed to hurt performance**. In some cases this was an overall effect, such as Agent C achieving a lower total response quality score than Agent 0, and in other cases it just applied to individual questions. One question that Agent C, which had access to the abbreviations dictionary, performed worse on than Agent 0, which had no tools or files, was E8. Based on the behaviour we observed, a reason for this may be that an agent without capabilities is more likely to answer without having all the required information, while an agent with tools or files tends to not answer if its capabilities cannot be used for the task. In the case of E8, which was a multiple choice question, this may have benefitted Agent 0 as the correct answer could be guessed. On the other hand, Agent A, Agent B, and Agent C all achieved lower grades than Agent 0 on question E13 because of an opposite effect. On this question, which was one of the questions requiring a longer answer, all three configurations with a single component gave responses that contained false claims or misinformation. They were therefore given the grade 0 by our supervisors at the case company, while Agent 0 produced a response which was neither sufficiently helpful nor misleading and therefore received the grade 1. In this case, the added capabilities appeared to make the assistant more prone to making unwarranted claims beyond the correct information, while the less equipped version answered more conservatively. Our results thus provide further evidence that the behaviour of LLMs can be inconsistent across tasks and difficult to predict, and that increased access to information, for varying reasons, does not always lead to improved performance.

Despite not leading to better responses on every individual question, we concluded from the comparative configuration evaluation that **most components improved the assistant's performance**. The results indicate that incorporating information about the database's structure and contents is beneficial for properly answering exploration questions, as both Agent A and Agent B performed significantly better than Agent 0 in phase 1. Combining information from both metadata tables and data contracts seemed, based on the results of Agent AB, to enhance performance further compared to using just one of the sources. Providing explanations and definitions of domain-specific terms did not help on its own, but in combination with other components it seemed to provide more of a marginal benefit than a disadvantage. The file we used was scoped to the case company rather than the database specifically, and it is possible that including more terms appearing in the database would result in a stronger positive impact. Finally, the results of phase 3 suggest that both the system prompt and the multi-agent system contributed to improving the assistant, but to different degrees. The results of the configurations with a system prompt imply that it was a valuable component,

while splitting the capabilities across two agents appeared to have a smaller, but still positive, effect. Overall, we find that specialised domain knowledge and tailored instructions are effective for creating customised LLM-based assistants for situations similar to the one studied in this thesis. This is broadly consistent with previous findings, such as those by Wang et al. [50] and Yu et al. [51], which are covered in Chapter 3.

The **components that did not seem to help performance** were the query log tool and the structured query tool, as no configuration with these tools performed better than Agent ABC0, which did not include them. Part of the reason for the query log tool not being helpful may have been that the agents that had it seemed to use it relatively rarely, despite being encouraged to do so by the system prompt. In the situations where the tool was used, the retrieved queries may not have been relevant enough to provide any guidance. This could perhaps be improved by adjusting how the retrieved queries are chosen. It is also possible that giving previous queries as examples is simply not an effective approach in our situation, despite the positive results that Jang and Li [55] achieved in part through the use of a pool of successful queries. Similarly, the structured query tool did not appear to lead to better queries compared to letting the LLM generate SQL freely. We observed some instances of the agents generating invalid query plans and thus receiving an error message instead of a result. This may have been a contributing factor to decreased performance. However, the primary reason for the tool not being helpful seemed to be that letting the model generate query plans instead of queries did not reduce the risk of erroneous omissions in the way that we hoped.

6.3 Performance of the Final Solution (RQ3)

The final version of the assistant was tested both with the evaluation questions, like the rest of the configurations, and by some potential users at the case company. Based on the outcomes of these activities, we made observations related to the assistant's performance. This section covers some of the noted strengths and weaknesses, and identifies improvements that could be made in order to increase usefulness.

The configuration that was chosen as the final solution was the one that in phase 3 of the configuration evaluation was referred to as Agent ABC0FG. It answered all exploration questions of the objective type correctly, suggesting that it is **capable of accurately informing users** of where desired data can be found and other facts related to the database's structure and contents. On the three exploration questions where the responses were assessed by our case company supervisors, the final solution received the highest grade on two questions and the second highest grade on the third. This is further evidence of the assistant's exploration strengths, and indicates that it can also produce longer and more explanatory responses that are both correct and relevant to the asked question. This aspect was reported as helpful in the user feedback. The assistant's ability to generate queries and retrieve actual data from the database was also tested by the users, who perceived it as satisfactory in many cases. The results from the objective data retrieval questions in the configuration evaluation partly support this, as the final configuration gave correct answers to most of them. Response times were around or below ten seconds on the majority of evaluation questions and were described as adequately fast in most situations by the users, so latency did not seem to be a major problem of the assistant. On the whole, the solution was viewed as useful by the participants

in the user testing, and the majority of the design goals seemed to be at least partly achieved.

There were however also **flaws observed in the assistant** and situations where it did not perform well enough to be helpful. These were noticed both in the configuration evaluation and the user testing, and they were primarily related to querying the database. In the configuration evaluation, the final version of the assistant received the grade 1 of 3 on all three data retrieval questions evaluated by our supervisors, and also failed on four of the objective questions in the category. Failure to produce adequate SQL queries was also reported by one participant in the user testing who tried to use the assistant for an advanced task. Based on our observations of the assistant's behaviour and the assessments by our supervisors, we identified some specific situations where performance was lower. Some questions were missed due to the assistant using subtly wrong tables or columns, such as the wrong type of timestamp. It also wrongly omitted filtering conditions in some cases where they were not explicitly mentioned in the question, and sometimes based parts of its response on incorrect assumptions instead of verifying them. Finally, it handled dates poorly, which led to inaccurate answers on some questions where that was important. A reason for this was that we did not implement a specific tool or other capability for checking the current date and time, but with proper usage of ClickHouse's date- and time-related functions this should not have been strictly necessary. On question D9, for example, the assistant successfully used these functions to identify which date the word "yesterday" referred to.

To make the assistant perform better and address the observed weaknesses, there are some **potential improvements** that could be made. One simple addition that would likely have a significant impact is to implement a tool that returns the current date and time. This would presumably lead to less errors related to dates, as it would provide an explicit way of retrieving needed information about it. More correct usage of the tables and columns could potentially be achieved by extending the `# Context` section of the system prompt, providing more information about the database and guidance for how to use it. The rest of the system prompt could also be refined and optimised to find the version that elicits the most desired behaviour in the assistant, such as a greater tendency to verify assumptions. Additionally, a more sophisticated implementation of RAG, where each chunk corresponds to one table description, may increase the probability of the retrieved chunk containing helpful information. The concept of separating the capabilities across multiple agents could also be explored further, for example by splitting the task across a larger number of more specialised agents. A separate fact-checking agent could also be added to reduce the risk of the assistant presenting false claims. Another potential upgrade is to modify the structured query tool, the current version of which did not improve performance, by for example simplifying the structure, providing better instructions for its usage, or taking inspiration from the query capsule approach suggested by Jang and Li [55] and described in Chapter 3. Finally, the method of providing a representation of the database where tables and columns are renamed to match user vocabulary, as suggested by Nascimento et al. [53], could be explored in order to increase the assistant's ability to find the right data.

6.4 Threats to Validity

Awareness of validity threats is important to ensure the transparency and credibility of any study. In this section, we discuss the circumstances, biases, and other factors that may have

affected our study and its findings. Some threats affect the reliability of our results, while others are related to the degree to which the results are applicable to other situations.

The Problem Conceptualisation step was where the basis of the thesis was laid, and issues there may have affected decisions later in the project. During the literature review, the two authors did not systematically cross-check each other's references to be included in the report. This could have affected the reliability of the literature review due to a potential selection bias, and a possible consequence is that we did not have a complete picture of the technological landscape when moving on to development. The interviews conducted also introduce possible threats to validity. Firstly, the selection of interview participants was done based on a list of employees provided to us by one of our supervisors at the case company, where the first seven to respond were selected for participating in the interviews. This introduces the risk of self-selection bias, as early responders may have a particularly strong or otherwise not representative opinion, and therefore the results are not necessarily generalisable to the whole company. Secondly, even though the participants were promised anonymity, there may have been a reluctance to fully speak their mind, which could have affected their responses and thus the reliability of our findings.

In the Solution Design and Implementation step, there were also factors that influenced the outcome of our work. In addition to the limitations and constraints mentioned in Section 6.2, our choice of elements to implement was also affected by our knowledge of available techniques, from before the start of the thesis and from the literature review, as well as potential biases in judging what would be effective. This can be seen as a threat to reliability, as other authors approaching the same problem may have arrived at a different design. Threats to generalisability can also be identified in our implementation, since the solution was specifically designed for the diagnostics database at the case company. While many of the ideas could be applied to other databases, the implementations are specific and would need to be modified to different degrees in order to be used in other contexts. For example, the SQL-based exploration tools rely on the database having metadata tables containing certain information, while the tool `get_table_contract` requires easily accessible data contracts. Furthermore, the solution was based on the characteristics and challenges of the specific database and may be less helpful for databases with different properties, such as a considerably smaller number of tables.

There are several factors that should be considered when interpreting the results of **the comparative configuration evaluation**. One important aspect is that we tested the configurations with custom evaluation questions rather than a standard benchmarking dataset. This was deemed necessary to make the evaluation representative of actual usage, but it means that the results cannot be interpreted in absolute terms or directly compared with those of other studies, which reduces their generalisability. We also wrote the evaluation questions after developing the solution, which meant that we were aware of which capabilities it had. Even though we based the questions on use cases from the intended users and our supervisors, this order of operations introduces a threat to reliability, as there is a risk that we unintentionally wrote questions informed by our knowledge of the implementation. Creating the evaluation dataset before development would have alleviated this risk. The fact that several of the questions were either answered correctly by all configurations or by none of them also suggests that the difficulty perhaps should have been adjusted to get more informative results. Perhaps the most significant limitation of our evaluation is that each agent was given each question only once. The outputs produced by an LLM are not deterministic, and random

chance can therefore influence the quality of an agent's response. For example, the agents with components D and E may have achieved lower scores in phase 2 due to coincidence, rather than because those components hurt performance. Similarly, there is a high risk that the measured response times were influenced by other factors than the components included in the tested configuration. While difficult to eliminate entirely, the effects of both these threats to reliability could have been reduced by testing each configuration multiple times, but time constraints and limitations related to programmatic interaction with the agents made this difficult. We may also have been able to learn more from the comparative configuration evaluation if we tested each element individually and if we compared all combinations instead of dividing the evaluation into three phases, but that was not feasible in the available time.

Lastly, there are also threats impacting the validity of the results from **the user testing**. One of them is the risk that the users were not entirely honest in their feedback, for example out of a desire to be polite. To be able to contact respondents for follow-up questions if needed, we did not make the questionnaire for the feedback anonymous, and this may also have affected the responses. They may also have been influenced by the phrasing of our questions, despite our efforts to make them as open-ended as possible so as not to limit the feedback to any type. Another risk is that the users thought the assistant's responses were accurate even if they were not, and that their feedback was therefore better than it should. All these factors are threats to the reliability of the results, but there is also a threat to their generalisability. Similarly to the interviews, the user testing was performed by a relatively small and partly self-selected group of employees, whose needs, use cases, and opinions are not necessarily the same as the rest of the intended users.

Chapter 7
Conclusions

In this thesis, we have investigated how an LLM-based conversational assistant can help users utilise a large diagnostics database, by developing such a solution at a case company. Our work includes understanding the needs of the assistant's potential users, implementing a system intended to meet those needs, examining the effectiveness of different techniques and approaches, and assessing the usefulness of the final solution. This chapter summarises the findings of the thesis in an attempt to answer the research questions, and closes with concluding remarks on the implications of the work and directions for future research.

Based on our investigation of the **current practices and barriers** related to working with the diagnostics database (RQ1), we found that the intended users primarily access the database through a business intelligence tool, most often using dashboards to draw insights from the data. The frequency with which they actively use the tools and the degree to which they do so independently varies significantly between the users. Four areas of barriers emerged from the interviews: finding relevant data, understanding and interpreting it, querying and analysing it, and structural barriers such as limited time. Out of these, finding relevant data was the most commonly reported barrier and affected users of all experience levels. The intended users were generally open to using the diagnostics data more in their work, but felt hindered by these difficulties, which motivates the development of a solution that can help users overcome the barriers.

The process of developing a solution for the reported problems, and the comparative configuration evaluation where different components of it was compared, provided insights into **how an LLM-based assistant can aid database usage** (RQ2). We found that incorporating knowledge about the database, such as table structure, descriptions and explanations, was effective in creating a helpful assistant. Giving it specific instructions for how to act was also a useful approach for achieving responses of high quality. Furthermore, our findings indicate that the usage of multiple agents with specialised subtasks may improve performance compared to a single agent responsible for the entire process. In contrast, our implemented elements for guiding SQL generation showed no positive effect and were deemed redundant.

The evaluation of the final solution (RQ3) showed that it generally responded accurately

and relevantly on tasks related to exploring the database, such as finding the right table based on a description of the desired information. When it came to retrieving data, the assistant answered many questions correctly but showed more weaknesses compared to when performing exploration tasks. It sometimes used the wrong tables or columns in queries, and occasionally operated on false assumptions which could have been verified using its capabilities. The user testing showed that participants found the assistant useful, and the response times, which we measured to around or below ten seconds for most of the evaluation questions, were reported as satisfactory. Overall, the results suggest that the final solution is useful for database exploration and more straightforward data retrieval questions, but has limitations on more advanced data retrieval use cases.

Our work benefits both the case company and the wider technology and research community. At the company, the developed assistant can, perhaps after some tweaks, be deployed and used by employees to help them use the diagnostics database more easily, thus facilitating data-driven work. This was seen as beneficial by the participants in the user testing, who described the assistant as useful and several of whom expressed a desire to use it moving forward. The company can also benefit from the added insights into the challenges that users of the database experience, as well as into what approaches seem to be worth exploring further for alleviating those challenges. Similarly, other organisations and researchers can also learn from our findings related to how LLMs can be used to make analytical databases more accessible. This thesis contributes with research in several rapidly expanding areas related to LLM-assisted data analytics, and provides experimental data on the effectiveness of some approaches. While developed for querying as well, our solution places significant focus on the exploration part of database usage, and performs well on tasks of that type. It therefore helps fill the research gap that we identified in the literature, where most work focused on SQL generation and did not address the earlier stages of database interaction to the same extent. Our assistant is also tested in a real-world enterprise setting, contributing to addressing the unresolved challenges recognised by Floratou et al. [58].

Future work could be directed at implementing the aspects described in Section 6.3, such as a tool for retrieving the current date and time, an improved RAG implementation, and more specialised agents in a multi-agent system. The structured query tool could also be improved to address the issues encountered with more advanced data retrieval questions. Moreover, investigating whether other LLMs can yield significantly better performance than Qwen3.5 would be an interesting next step. Beyond improving the existing system, features for creating charts or dashboards could be explored to enable users to draw insights directly from the assistant's outputs. To better understand the usefulness and performance of the assistant across different roles and use cases, the evaluation could be extended by testing it against a broader set of users across the case company, and with a larger set of evaluation questions. A comparative evaluation at a more granular level than ours would also be valuable, as it would provide deeper insights into what details are most important for performance. Additionally, comparing all component combinations rather than only those within the same group would increase reliability of the results by reducing the risk of missing helpful interactions between components in different groups. In general, the challenge of how to appropriately evaluate domain-specific LLM-based systems is one that warrants further research. A standardised method for assessing how well-suited an agent is for real-world company use, as well as for comparing different configurations systematically, would be valuable for future projects within the area.

References

[1] Anna Fariha and Alexandra Meliou. Example-driven query intent discovery: Abductive reasoning using semantic similarity. *Proc. VLDB Endow.*, 12(11):1262–1275, July 2019.

[2] Nishant Wanjari, Aashka Gupta, Reshma Gulwani, and Aditi Chhabria. Statistics in data science. In *Advanced Mathematics in Computing, Communication and Security*, pages 357–379. Wiley, 2025.

[3] Xue Wang, Xuan Zhou, and Shan Wang. Summarizing large-scale database schema using community detection. *Journal of Computer Science and Technology*, 27(3):515–526, 2012.

[4] Martin Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, 2017.

[5] Anand Mohan Vimal, Meenakshi Srivastava, Pratibha Maurya, and Ranjana Rajnish. A comprehensive survey of natural language processing: Milestones and future directions. In *Proceedings of the 1st International Conference on Data Science and Intelligent Network Computing (ICDSINC)*, pages 70–75, 2025.

[6] Minghao Shao, Abdul Basit, Ramesh Karri, and Muhammad Shafique. Survey of different large language model architectures: Trends, benchmarks, and challenges. *IEEE Access*, 12:188664–188706, 2024.

[7] Zina Chkirbene, Ridha Hamila, Ala Gouissem, and Unal Devrim. Large language models (LLM) in industry: A survey of applications, challenges, and trends. In *Proceedings of the 21st IEEE International Conference on Smart Communities: Improving Quality of Life Using AI, Robotics and IoT (HONET)*, pages 229–234, 2024.

[8] Zachary Schillaci. LLM adoption trends and associated risks. In Andrei Kucharavy, Octave Plancherel, Valentin Mulder, Alain Mermoud, and Vincent Lenders, editors, *Large Language Models in Cybersecurity: Threats, Exposure and Mitigation*, pages 121–128. Springer Nature Switzerland, 2024.

[9] ClickHouse. `https://clickhouse.com/`, n.d. Accessed: 2026-06-12.

[10] Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, February 2026. Accessed: 2026-06-12.

[11] LibreChat: The open-source AI platform. https://librechat.ai/, n.d. Accessed: 2026-06-12.

[12] Kristi L. Berg, Tom Seymour, and Richa Goel. History of databases. *International Journal of Management & Information Systems*, 17(1), December 2012.

[13] Alberto Abelló and Oscar Romero. On-line analytical processing. In *Encyclopedia of Database Systems*. Springer, 2017.

[14] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 7th edition, 2020.

[15] IBM. What is a database schema? https://www.ibm.com/think/topics/database-schema, n.d. Accessed: 2026-05-04.

[16] Jan L. Harrington. The database environment. In *Relational Database Design and Implementation*, pages 3–26. Elsevier, 4th edition, 2016.

[17] Anja Bog. *Benchmarking Transaction and Analytical Processing Systems: The Creation of a Mixed Workload Benchmark and its Application*. In-Memory Data Management Research. Springer, 2014.

[18] Terry Halpin and Tony Morgan. NoSQL and other nonrelational databases. In *Information Modeling and Relational Databases*, pages 875–933. Morgan Kaufmann, 3 edition, 2024.

[19] Donald D. Chamberlin. Early history of SQL. *IEEE Annals of the History of Computing*, 34(4):78–82, 2012.

[20] Mohammed Salahat, Muhammad Ahsan Raza, Zuha Khawar, Javaid Ahmad Malik, Hammad Raza, Harish K G Nair, and Muath Jarrah. Analysis of query processing on different databases. In *Proceedings of the International Conference on Business Analytics for Technology and Security (ICBATS)*, 2023.

[21] Amalio Telenti, Michael Auli, Brian L. Hie, Cyrus Maher, Suchi Saria, and John P.A. Ioannidis. Large language models for science and medicine. *European Journal of Clinical Investigation*, 54(6):e14183, June 2024.

[22] Raj Bridgelall. Unraveling the mysteries of AI chatbots. *Artificial Intelligence Review*, 57(4):89, March 2024.

[23] Yi Yang, Yitong Ma, Hao Feng, Yiming Cheng, and Zhu Han. Minimizing hallucinations and communication costs: Adversarial debate and voting mechanisms in LLM-based multi-agents. *Applied Sciences*, 15(7):3676, 2025.

[24] Ernests Lavrinovics, Russa Biswas, Johannes Bjerva, and Katja Hose. Knowledge graphs, large language models, and hallucinations: An NLP perspective. *Journal of Web Semantics*, 85:100844, 2025.

[25] Negar Maleki, Balaji Padmanabhan, and Kaushik Dutta. AI hallucinations: A misnomer worth clarifying. arXiv preprint arXiv:2401.06796, 2024.

[26] Tanner Stening. What are AI chatbots actually doing when they 'hallucinate'? here's why experts don't like the term. `https://news.northeastern.edu/2023/11/10/ai-chatbot-hallucinations/`, 2023. Accessed: 2026-04-06.

[27] Jasmin Jarsania and Vivek Patel. Chain-of-memory: Taming hallucinations in million-token models with verifiable retrieval and causal tracing. In *Proceedings of the International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD)*, pages 1–5, 2025.

[28] Shuai Qin, Lianke Zhou, Liu Sun, and Nianbin Wang. Do large language models know when they lack knowledge? *Electronics*, 15(2):253, 2026.

[29] Yiqiu Shen, Laura Heacock, Jonathan Elias, Keith Hentel, Beatriu Reig, George Shih, and Linda Moy. ChatGPT and other large language models are double-edged swords. *Radiology*, 307:e230163, January 2023.

[30] Hyeonseok Moon, Jaehyung Seo, Seungyoon Lee, Chanjun Park, and Heuiseok Lim. Find the intention of instruction: Comprehensive evaluation of instruction understanding for large language models. arXiv preprint arXiv:2412.19450, 2025.

[31] Qianyu He, Jie Zeng, Wenhao Huang, Lina Chen, Jin Xiao, Qianxi He, Xunzhe Zhou, Jiaqing Liang, and Yanghua Xiao. Can large language models understand real-world complex instructions? *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):18188–18196, March 2024.

[32] Juyeon Heo, Miao Xiong, Christina Heinze-Deml, and Jaya Narain. Do LLMs estimate uncertainty well in instruction-following? arXiv preprint arXiv:2410.14582, 2025.

[33] Ammar Mohammed and Rania Kora. A comprehensive overview and analysis of large language models: Trends and challenges. *IEEE Access*, 13:1–1, January 2025.

[34] Zirui Song, Bin Yan, Yuhan Liu, Miao Fang, Mingzhe Li, Rui Yan, and Xiuying Chen. Injecting domain-specific knowledge into large language models: A comprehensive survey. arXiv preprint arXiv:2502.10708, 2025.

[35] Xinyuan Qiu, Honghua Chen, Xin Wang, Xiaolong Cui, and Weitong Ji. A survey on domain-specific large language models. In *Proceedings of the 6th International Conference on Machine Learning and Computer Application (ICMLCA)*, pages 908–916, 2025.

[36] Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. The prompt report: A systematic survey of prompt engineering techniques. arXiv preprint arXiv:2406.06608, 2025.

[37] Daniel Koch, Andreas Kohne, and Nils Brechbühler. *Prompt Engineering in the Enterprise—An Introduction: Competitive Advantages through Generative AI and Large Language Models*. Springer Fachmedien Wiesbaden, 2025.

[38] Thea Lovise Ahlgren, Helene Fønstelien Sunde, Kai-Kristian Kemell, and Anh Nguyen-Duc. Assisting early-stage software startups with LLMs: Effective prompt engineering and system instruction design. *Information and Software Technology*, 187:107832, 2025.

[39] Yaxin Tang, Yijia Liu, Jiahe Lan, Zheng Yan, and Erol Gelenbe. Security of LLM-based agents regarding attacks, defenses, and applications: A comprehensive survey. *Information Fusion*, 127:103941, 2026.

[40] Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. AI agents under threat: A survey of key security challenges and future pathways. *ACM Comput. Surv.*, 57(7):182, February 2025.

[41] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Trans. Inf. Syst.*, 43(6):155, September 2025.

[42] Shengqian Qin, Yakun Zhu, Linjie Mu, Shaoting Zhang, and Xiaofan Zhang. Metatool: Unleash open-world function calling capabilities of general-purpose large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30653–30677, July 2025.

[43] Weikai Xu, Chengrui Huang, Shen Gao, and Shuo Shang. LLM-based agents for tool learning: A survey. *Data Science and Engineering*, 10:533–563, June 2025.

[44] Kinjal Basu. Bridging knowledge gaps in LLMs via function calls. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM)*, pages 5556–5557, 2024.

[45] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. arXiv preprint arXiv:2302.04761, 2023.

[46] Yilun Kong, Jingqing Ruan, YiHong Chen, Bin Zhang, Tianpeng Bao, Shiwei Shi, Qing Du Guo, Xiaoru Hu, Hangyu Mao, Ziyue Li, Xingyu Zeng, Rui Zhao, and Xueqian Wang. TPTU-v2: Boosting task planning and tool usage of large language model-based agents in real-world industry systems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 371–385, November 2024.

[47] Adrian Theuma and Ehsan Shareghi. Equipping language models with tool use capability for tabular data analysis in finance. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2024.

[48] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (MCP): Landscape, security threats, and future research directions. arXiv preprint arXiv:2503.23278, 2025.

[49] Anthropic. Example clients. `https://modelcontextprotocol.io/clients`, 2026. Accessed: 2026-05-10.

[50] Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. DBCopilot: Natural language querying over massive databases via schema routing. In *Proceedings of the 28th International Conference on Extending Database Technology (EDBT)*, 2025.

[51] Chenglong Yu, Yao Yao, Xiang Zhang, Geyuan Zhu, Yanduo Guo, Xiaowei Shao, Mariko Shibasaki, Zhihui Hu, Dai Liangyang, Qingfeng Guan, and Ryosuke Shibasaki. Monkuu: A llm-powered natural language interface for geospatial databases with dynamic schema mapping. *International Journal of Geographical Information Science*, 40(2):588–609, 2025.

[52] Dai Quoc Nguyen, Cong Duy Vu Hoang, Duy Vu, Gioacchino Tangari, Thanh Tien Vu, Don Dharmasiri, Yuan-Fang Li, and Long Duong. SQLong: Enhanced NL2SQL for longer contexts with LLMs. arXiv preprint arXiv:2502.16747, 2025.

[53] Eduardo Nascimento, Gustavo Coelho, Lucas Feijó, Yenier Izquierdo, Grettel García, Aiko Oliveira, João Pinheiro, Antony Seabra, Antonio Furtado, Luiz Paes Leme, Melissa Lemos, and Marco Casanova. On the construction of text-to-SQL tools based on large language models for real-world relational databases. In M. Marchiori and F. García Peñalvo, editors, *Web Information Systems and Technologies. WEBIST 2023*, volume 543 of *Lecture Notes in Business Information Processing*. Springer, 2025.

[54] Samuel Ojuri, The Anh Han, Raymond Chiong, and Alessandro Di Stefano. Optimizing text-to-SQL conversion techniques through the integration of intelligent agents and large language models. *Information Processing & Management*, 62(5):104136, 2025.

[55] Jisoo Jang and Wen-Syan Li. Retrieval-augmented NL2SQL generation with data-centric query capsules for enterprise applications. In *Proceedings of the 2025 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region (SIGIR-AP)*, pages 123–132, 2025.

[56] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. arXiv preprint arXiv:2312.11242, 2025.

[57] Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. arXiv preprint arXiv:2304.11015, 2023.

[58] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex Van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, K. Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. NL2SQL is a solved problem…not! In *Proceedings of the 14th Conference on Innovative Data Systems Research (CIDR)*, January 2024.

[59] Per Runeson, Emelie Engström, and Margaret-Anne Storey. The design science paradigm as a frame for empirical software engineering. In Michael Felderer and Guilherme Horta Travassos, editors, *Contemporary Empirical Methods in Software Engineering*. Springer, 2020.

[60] Milla Widell and Philip Sadrian. Designing a domain-specific language to address challenges in automated web testing. Master's thesis, Lund University, 2025.

[61] Finn. `https://finn.lub.lu.se`, n.d. Accessed: 2026-06-12.

[62] Scopus. `https://scopus.com`, n.d. Accessed: 2026-06-12.

[63] Sirwan Khalid Ahmed, Ribwar Arsalan Mohammed, Abdulqadir J. Nashwan, Radhwan Hussein Ibrahim, Araz Qadir Abdalla, Barzan Mohammed M. Ameen, and Renas Mohammed Khdhir. Using thematic analysis in qualitative research. *Journal of Medicine, Surgery, and Public Health*, 6:100198, August 2025.

Appendices

Appendix A
Interview Guide

The following questions were used in the interviews conducted with employees at the case company, translated to Swedish for participants proficient in that language. The numbered questions served as the primary structure of the interview, while the unnumbered bullet points were used as potential prompts if needed. The interview methodology is described in Section 4.1.2 and the results are presented in Section 5.1.

General

1. What is your name?
2. What is your educational background?
3. What is your role here at Axis and how long have you worked here?
4. What are your tasks on a typical working day?
5. How much experience do you have working with databases, writing SQL or in some way making quantitative analyses of data?

Current Usage

6. Were you aware that there is a database with collected diagnostics data from Axis' devices?
7. Are you aware that you have access to the DDM database?
8. How well do you feel that you know what data the DDM database contains?
9. Can you describe how you currently use the DDM database?
    - Do you use the DDM data at all?
    - How often do you use it?
    - What do you use the data for?
    - How do you go about finding where the data you are interested in is located?
    - How do you go about extracting the data you are interested in?

10. Can you give an example of specific information that you retrieve from the database, or that you would like to retrieve? It could for example be: "how many devices of model X do we collect data from?", "what is the average temperature of devices of model X over the last month?" or "what is the distribution of memory usage for process Y?".
    - Do you have an example of a question you have answered, a chart you have created or similar?
11. What parts of working with the data do you find difficult or challenging?
    - Identifying what data is relevant?
    - Extracting information from the data?
    - Interpreting the information into insights?
    - Presenting or making decisions based on the information?

Future Usage

12. Do you see any applications for the DDM data that you don't currently make use of?
    - Do you have an example of a question you would like to answer, a chart you would like to create or similar?
    - Is there something in particular that prevents you from doing this?
13. In what way do you think an LLM-based assistant could help you use the DDM data more easily and effectively?
    - What would you like the assistant to help you with?
14. What do you think is the most important aspect for you to want to use the assistant?
    - What would make you trust what it says?

Ending

15. Is there anything we have not talked about yet that you think is relevant for us to know?
16. Do you have anyone in mind that you think would be good for us to talk to about this?

Appendix B
System Prompt

Here, the system prompt used in the final version of the assistant is presented. Since the final configuration consisted of two agents, there were two versions. The sections `# Role`, `# Context`, and `# Rules` were the same in both, and these are shown in Listings B.1, B.2, and B.3 respectively. The `# Tools` section was different for the two agents as they had different tools, and the two variants can be seen in Listings B.4 and B.5. The `# Example` section, which is found in Listing B.6, was only included in the agent capable of executing custom queries.

Listing B.1: The `# Role` section of the system prompt, partly anonymised for confidentiality reasons

```
# Role
You are a helpful assistant with expert knowledge in the
<DATABASE NAME> database. Your task is to help the user navigate and
explore the <DATABASE NAME> database, and to answer questions based on
the data in it.
```

Listing B.2: The `# Context` section of the system prompt, partly anonymised for confidentiality reasons.

```
# Context
<DATABASE NAME> is a ClickHouse database where time-series diagnostics
data collected from Axis devices is stored. Data is collected on
average <COLLECTION FREQUENCY> (in most tables) from around
<DEVICE COUNT> devices, leading to very large volumes of data. The data
is raw and can contain a high degree of missing or implausible values.
In some tables, the serial number combined with the timestamp uniquely
identifies a row. Other tables have several rows per device and
timestamp, so more columns are needed to uniquely identify a row. Some
of the columns contain metadata about the device or the collection, and
the values in these are duplicated across all tables.
```

Listing B.3: The `# Rules` section of the system prompt.

```
# Rules
- Always try to find the information you need instead of making
assumptions. If finding the necessary information is impossible, state
your assumptions clearly or say that you lack information.
- Never guess or assume table or column names. Always make sure to use
the exact names that are used in the database.
- Always admit if you are unable to answer a question, instead of
making up an answer.
- Always tell the user if you are unsure of your answer, instead of
sounding confident regardless.
- Never add extra information that is not based on what you find in the
database or the documentation and present it as facts. If you want to
add explanations, definitions or comments on your own, clearly signal
to the user that they are your guesses.
```

Listing B.4: The `# Tools` section of the system prompt given to the first of the two agents in the multi-agent system.

```
# Tools
- Use the tools get_tables_in_schema and get_columns_in_table to get an
overview of which tables and columns exist.
- Use the tools get_table_comment and get_column_comment to gain more
understanding of what the tables and columns contain.
- Use the tool get_table_sample to get a glimpse of the contents of a
table, for example to see the data types of columns or to get an
indication of what makes a row unique.
- Use the file table_names_and_descriptions.md to see what tables there
are and what their descriptions are.
- Use the tool get_table_contract to get more detailed information
about a specific table, such as which columns it has.
- Use the file abbreviations_explanations.json to look up the meaning
of abbreviations you encounter.
```

Listing B.5: The `# Tools` section of the system prompt given to the second of the two agents in the multi-agent system, partly anonymised for confidentiality reasons.

```
# Tools
- Use the tool execute_any_query to execute an SQL query against the
database.
```

Listing B.6: The `# Example` section of the system prompt given to the second of the two agents in the multi-agent system.

```
# Example
Here is an example of how a question about the data can be translated
to an SQL query, including one possible thought process to construct
it. Note that different tables can have different structures and that
not all questions are necessarily similar to this.

**Question:**
"For each product name, how many unique devices have had their
/usr/local flash usage exceed 80% at any point in the last 30 days?
Only show product names with at least 10 such devices, ordered by
decreasing number of devices."

**Thought process:**
1. I must find which table or tables in the database contain data that
can be used to answer this question. I will use one/some of my
available exploration tools for this.
- **Result:** Finds that the table `<TABLE NAME>` in the schema
`<SCHEMA NAME>` has all the relevant and necessary information for this
specific question.
2. I must execute a query to answer this question, so I should use the
execute_any_query tool. I will write a SELECT query against the table I
found. I'll start by identifying the source table in the FROM clause.
- **Result:** Writes `FROM <SCHEMA NAME>.<TABLE NAME>`.
3. I must identify what should be outputted as the result. The question
asks about the number of unique devices, so I must count the number of
unique devices satisfying some conditions. I will use the SQL function
`count` for this. Since there might be multiple rows for each device, I
must add DISTINCT to this count. The question says that the product
name should also be outputted along with the number, so I must include
that column in the SELECT clause as well.
- **Result:** Writes `SELECT <COLUMN NAME>, count(DISTINCT
<COLUMN NAME>)`.
4. I must interpret the filtering conditions in the question and
translate them to SQL. Since three of the conditions apply to rows
rather than groups of rows, I should put them in a WHERE clause as
pre-aggregation filtering conditions.
- One condition is that I should only look at data from the last 30
days. I saw that the table has a `<COLUMN NAME>` column, so I can use
date arithmetic and ClickHouse's `now()` function to compare that with
the current datetime and an interval of 30 days.
- The question is only interested in the flash usage for /usr/local. I
saw that the column `<COLUMN NAME>` can have the values '/usr/local' or
'/lib/persistent' so I should use this column to filter.
- I should only include rows where the usage exceeds 80%. I have seen
that in the column `<COLUMN NAME>` I can find the number of used 1k
```

```
blocks and that in the column ‘<COLUMN NAME>‘ I can find the total
number of 1k blocks. I interpret the usage that the question asks about
as the ratio between these two numbers. This is an assumption that I
should tell the user about when I answer. For an accurate calculation I
should cast both values to floats. Then I can divide ‘<COLUMN NAME>‘ by
‘<COLUMN NAME>‘ and check if that is above 0.8, since that is the
decimal form of 80%.
- **Result:** Writes ‘WHERE <COLUMN NAME> >= toDateTime(now() -
toIntervalDay(30)) AND <COLUMN NAME> = ’/usr/local’ AND
toFloat64(<COLUMN NAME>) / toFloat64(<COLUMN NAME>) > 0.8‘.
5. The question asks about some number for each product name, so I
should use a GROUP BY clause to group the results based on the column
that contains product names.
- **Result:** Writes ‘GROUP BY <COLUMN NAME>‘.
6. The fourth and final filtering condition in the question is a
condition that applies to groups of rows. I should translate this to a
post-aggregate filtering condition using HAVING. The groups should be
filtered based on the number of devices. This is the quantity that I
calculate with the ‘count‘ function in step 3, so I should add an alias
to that. This alias can now be used in the HAVING clause.
- **Result:** Adds ‘AS device_count‘ to the SELECT clause and writes
‘HAVING device_count >= 10‘.
7. I must check if any specific ordering is required for this query.
The question says that the results should be ordered by decreasing
number of devices, which is the quantity I added an alias for. The
alias can be used to sort the result with an ORDER BY clause. I should
also specify descending order, since that is what the question asked
for.
- **Result:** Writes ‘ORDER BY device_count DESC‘.
8. Now I can assemble the complete query from the clauses I’ve built
and call execute_any_query with it.
- **Result:** Receives an output table with product names and device
counts as a response from the tool.
9. Now I have a potential answer to the question. I should check if it
looks reasonable and if possible verify it with some other source of
information. Then I can present the answer to the user, along with an
explanation of my method and any assumptions I made.
- **Result:** Finds that the answer seems reasonable and delivers it to
the user.

**Query:**
SELECT <COLUMN NAME>, count(DISTINCT <COLUMN NAME>) AS device_count
FROM <SCHEMA NAME>.<TABLE NAME>
WHERE <COLUMN NAME> >= toDateTime(now() - toIntervalDay(30))
AND <COLUMN NAME> = ’/usr/local’
AND toFloat64(<COLUMN NAME>) / toFloat64(<COLUMN NAME>) > 0.8
GROUP BY <COLUMN NAME>
HAVING device_count >= 10
ORDER BY device_count DESC
```

Appendix C
Results on Individual Questions

This appendix contains the results that each configuration in the comparative configuration evaluation achieved on each individual evaluation question. The questions can be seen in Table 4.3 and the aggregated results are presented in Section 5.4.

Table C.1: The scores achieved by each configurations on the evaluation questions in phase 1. The column headings indicate which components were included in each configurations. Questions E1–E12 were verified as either correct or incorrect, thus scoring 0 or 1, while E13–E15 were judged on a scale of 0 to 3.

	None	A	B	C	A, B	A, C	B, C	A, B, C
E1	0	1	1	0	1	1	1	1
E2	0	1	1	0	1	1	1	1
E3	0	1	0	0	1	1	1	1
E4	0	1	1	0	1	1	1	1
E5	0	1	1	0	1	1	1	1
E6	0	0	0	0	0	0	0	0
E7	0	1	1	0	1	1	1	1
E8	1	0	1	0	1	1	1	1
E9	0	1	0	0	1	1	0	1
E10	0	1	0	0	1	1	0	1
E11	1	1	0	1	1	1	1	1
E12	0	0	1	0	1	0	1	1
E13	1	0	0	0	2	1	3	1
E14	0	3	3	0	2	2	1	3
E15	1	2	3	0	2	2	3	2

Table C.2: The response times (in seconds) taken by each configuration to answer the evaluation questions in phase 1. The column headings indicate which components were included in each configurations.

	None	A	B	C	A, B	A, C	B, C	A, B, C
E1	1.6	5.5	1.4	2.7	1.4	1.2	1.1	1.4
E2	7.3	3.5	14.2	3.4	1.4	2.2	1.5	2.5
E3	12.6	2.2	3.0	2.3	3.2	3.7	2.4	4.4
E4	1.8	12.9	2.5	2.0	2.2	7.2	2.1	2.2
E5	1.1	14.3	2.3	1.7	1.5	2.3	1.6	2.1
E6	1.3	20.6	5.1	2.3	11.1	11.6	19.4	10.4
E7	1.1	15.8	3.8	2.3	4.3	3.9	4.0	4.7
E8	2.3	2.5	1.7	2.0	5.2	8.9	13.6	3.0
E9	1.7	4.4	1.7	2.1	3.7	4.1	1.5	2.4
E10	1.9	14.8	6.4	2.6	17.8	11.0	6.2	10.9
E11	0.5	2.5	1.7	1.5	2.4	2.6	2.1	1.5
E12	1.8	9.0	1.5	2.6	1.7	6.8	1.4	1.3
E13	6.1	13.5	7.4	5.3	9.9	10.2	6.9	10.4
E14	3.8	10.1	14.9	3.9	10.0	11.0	21.8	12.3
E15	11.0	14.2	11.4	3.1	24.7	9.9	16.5	15.3

Table C.3: The scores achieved by each configurations on the evaluation questions in phase 2. The column headings indicate which components were included in each configurations. Questions D1–D12 were verified as either correct or incorrect, thus scoring 0 or 1, while D13–D15 were judged on a scale of 0 to 3.

	A, B, C	A, B, C, D	A, B, C, E	A, B, C, D, E
D1	1	1	1	1
D2	1	1	1	1
D3	1	1	1	1
D4	1	1	1	1
D5	1	1	1	1
D6	0	0	0	0
D7	0	0	0	1
D8	0	0	0	0
D9	1	1	1	0
D10	1	1	0	1
D11	1	0	0	1
D12	0	0	0	0
D13	1	0	2	1
D14	2	1	1	1
D15	2	2	2	2

Table C.4: The response times (in seconds) taken by each configuration to answer the evaluation questions in phase 2. The column headings indicate which components were included in each configurations.

	A, B, C	**A, B, C, D**	**A, B, C, E**	**A, B, C, D, E**
D1	2.9	1.8	8.7	4.8
D2	8.4	5.1	4.2	4.9
D3	5.0	5.5	18.0	13.1
D4	13.2	6.1	20.4	36.2
D5	6.9	9.0	12.3	11.0
D6	105.2	86.7	45.9	52.4
D7	6.4	9.5	7.4	58.0
D8	5.9	7.1	20.9	23.9
D9	16.9	18.7	12.9	51.2
D10	6.6	8.7	67.4	60.4
D11	117.4	63.9	64.6	123.8
D12	15.3	32.6	16.9	17.7
D13	26.5	271.5	91.9	162.0
D14	27.0	27.9	67.3	50.3
D15	99.7	64.7	183.2	129.2

Table C.5: The scores achieved by each configurations on the evaluation questions in phase 3. The column headings indicate which components were included in each configurations. Questions E1–E12 and D1–D12 were verified as either correct or incorrect, thus scoring 0 or 1, while E13–E15 and D13–D15 were judged on a scale of 0 to 3.

	A, B, C	A, B, C, F	A, B, C, G	A, B, C, F, G
E1	1	1	1	1
E2	0	1	0	1
E3	0	1	0	1
E4	1	1	1	1
E5	1	1	1	1
E6	0	1	0	1
E7	0	0	1	1
E8	0	0	0	1
E9	0	1	0	1
E10	1	1	0	1
E11	1	1	1	1
E12	1	1	1	1
E13	1	2	1	3
E14	3	2	2	2
E15	2	3	3	3
D1	1	1	1	1
D2	1	1	1	1
D3	1	1	1	1
D4	1	1	1	1
D5	1	1	1	1
D6	0	0	0	0
D7	0	0	0	0
D8	0	0	0	0
D9	0	0	0	1
D10	1	1	1	1
D11	0	0	0	1
D12	0	0	0	0
D13	0	0	0	1
D14	2	3	1	1
D15	2	2	0	1

Table C.6: The response times (in seconds) taken by each configuration to answer the evaluation questions in phase 3. The column headings indicate which components were included in each configurations.

	A, B, C	A, B, C, F	A, B, C, G	A, B, C, F, G
E1	1.2	4.4	1.3	1.1
E2	0.5	4.2	0.7	1.1
E3	2.3	9.6	2.6	3.8
E4	1.8	6.5	2.0	2.0
E5	4.7	2.1	2.0	1.4
E6	23.8	15.2	10.9	10.1
E7	6.1	9.4	1.5	2.9
E8	1.4	3.7	1.2	2.8
E9	9.4	6.5	1.6	4.6
E10	36.4	15.3	0.5	10.2
E11	5.7	2.1	1.8	1.5
E12	34.3	2.9	3.2	1.8
E13	8.2	14.4	9.5	9.2
E14	6.6	9.3	6.2	8.5
E15	14.4	17.7	15.9	7.7
D1	2.3	1.8	4.4	24.9
D2	3.2	4.6	5.7	7.4
D3	28.6	12.6	16.7	10.3
D4	34.4	13.0	24.8	34.2
D5	14.5	8.7	10.3	9.9
D6	157.7	48.9	104.4	122.8
D7	15.3	10.2	12.3	18.7
D8	4.9	6.9	11.2	11.6
D9	13.9	13.0	16.2	68.0
D10	2.7	11.7	8.8	6.8
D11	377.5	74.0	22.7	97.8
D12	18.2	14.8	19.8	23.4
D13	81.9	44.3	16.2	24.0
D14	21.4	52.1	22.4	37.0
D15	200.0	282.6	366.7	267.8

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2026-06-08

EXAMENSARBETE Developing an LLM-Based Conversational Assistant for Database Interaction
STUDENTER Hanna Rosenberg, Love Hedelius
HANDLEDARE Elizabeth Bjarnason (LTH)
EXAMINATOR Alma Orucevic-Alagic (LTH)

Kan AI göra företagsdatabaser lättare att använda för fler anställda?

POPULÄRVETENSKAPLIG SAMMANFATTNING **Hanna Rosenberg, Love Hedelius**

Många företag samlar stora mängder data som kan ge värdefulla insikter och ligga till grund för viktiga beslut. Informationen lagras ofta i stora och komplexa databaser som kan vara svåra att använda och förstå. Vi har därför utvecklat en AI-assistent som låter anställda ställa frågor om en företagsdatabas genom att chatta med den.

Detta examensarbete utfördes hos ett globalt teknikföretag som vill öka användningen av sin insamlade produktdata, som de lagrar i en väldigt stor databas. För att använda datan krävs god kännedom om databasens struktur, så att man kan hitta och förstå informationen, samt kunskap om databasspråket SQL eller andra verktyg som finns för att komma åt datan. För att underlätta användningen av datan skapade vi därför en AI-assistent som man kan ställa frågor till på vanligt människospråk. Assistenten har kunskap om och tillgång till databasen, och kan därmed både hjälpa anställda att själva använda den och även direkt svara på frågor om datan.

För att utveckla AI-assistenten använde vi oss av en stor språkmodell som vi adderade ytterligare information och förmågor till. Den fick tillgång till databasen via en uppsättning verktyg för att bland annat hämta information om databasens innehåll och struktur, ta del av tidigare SQL-frågor som körts samt formulera och köra sina egna SQL-frågor. Den fick även tillgång till filer med företagsspecifik information och en överblick av databasens tabeller, och vi skrev en systemprompt för att styra dess beteende. Vi delade upp förmågorna på två olika agenter som var specialiserade på varsin del av assistentens uppgift.

För att avgöra vilka av alla dessa komponenter som bidrog till att göra assistenten bättre jämförde vi olika kombinationer av dem. Detta gjorde vi genom att ställa samma frågor till olika versioner av assistenten och bedöma svaren. Vi ställde både utforskande frågor om databasens innehåll, som exempelvis "Vilken tabell ska jag använda för att göra ett diagram över produkters temperatur?", och sådana som handlade om att hämta ut konkret data, som till exempel "Vad är den högsta uppmätta strömförbrukningen hos enhet XXX?". Det visade sig att alla komponenter som var till för att ge assistenten information om databasen eller styra dess beteende var fördelaktiga, medan SQL-generering var lika bra att låta språkmodellen sköta själv.

Assistenten testades även av användare, som uppgav att de var nöjda och att den skulle vara användbar i deras arbete. Den presterade någorlunda på datahämtningsfrågor men var särskilt bra på att utforska databasen, vilket vi tidigare identifierat som de anställdas största svårighet med databasen och även något som många tidigare forskningsarbeten fokuserat mindre på. Vårt arbete kan därmed bidra till både ökad dataanvändning på företaget och allmän förbättrad kunskap om användarvänlig databasinteraktion.