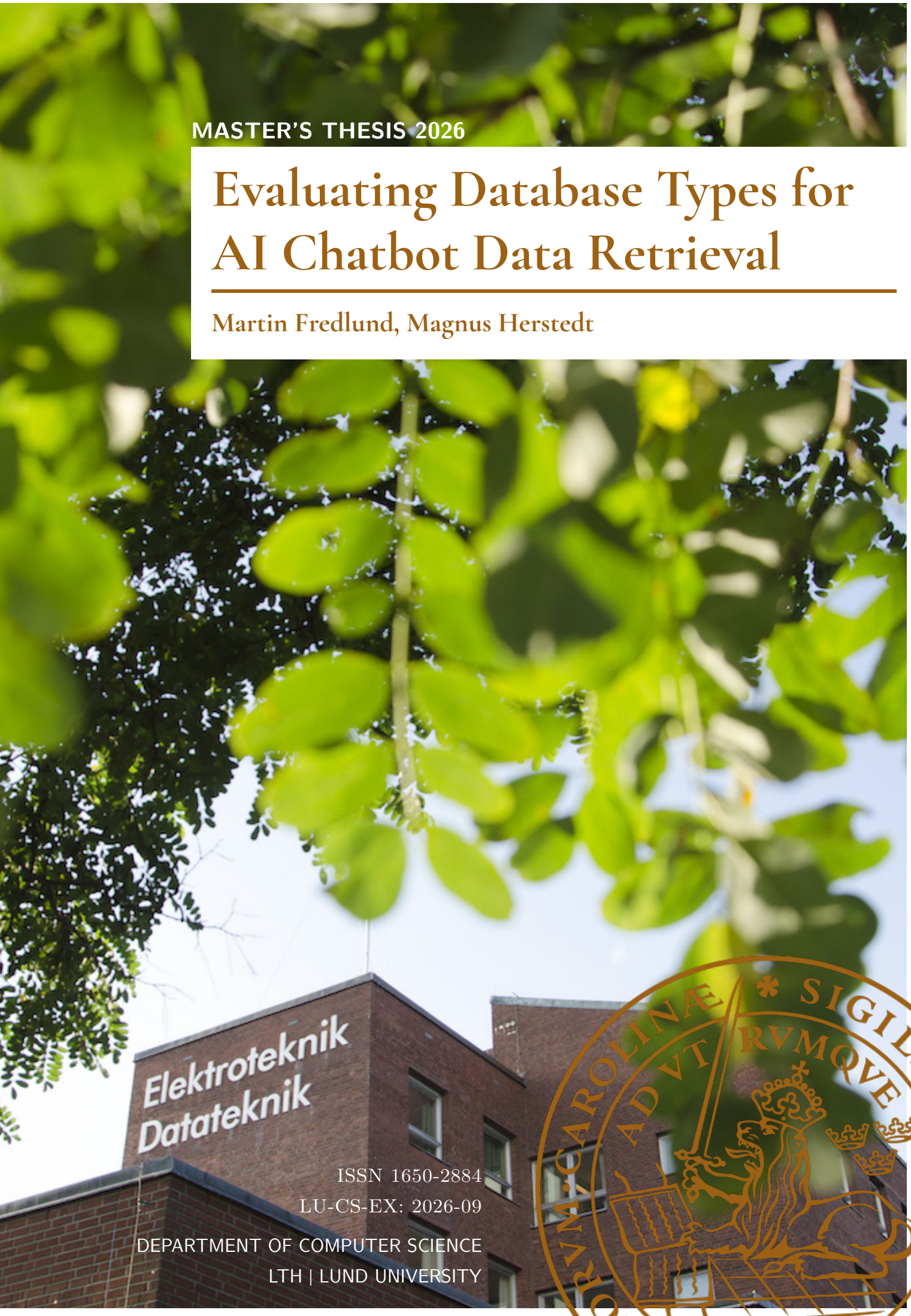
MASTER'S THESIS 2026

Evaluating Database Types for AI Chatbot Data Retrieval

Martin Fredlund, Magnus Herstedt



ISSN 1650-2884
LU-CS-EX: 2026-09

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2026-09

Evaluating Database Types for AI Chatbot Data Retrieval

Utvärdering av Databastyper för Datahämtning i AI-chattbottar

Martin Fredlund, Magnus Herstedt

Evaluating Database Types for AI Chatbot Data Retrieval

(A CRM System Perspective)

Martin Fredlund
martinfredlund@live.se

Magnus Herstedt
mg.herstedt@gmail.com

April 1, 2026

Master's thesis work carried out at Lime Technologies.

Supervisors: Per Fryking, per.fryking@lime.tech
Lars Bendix, lars.bendix@cs.lth.se

Examiner: Per Andersson, per.andersson@cs.lth.se

Abstract

Customer Relationship Management (CRM) solutions increasingly integrate AI chatbots. The requirements for information retrieval are complex, making the selection of the optimal database type critical. This thesis analyzes how different database types compare in their retrieval capabilities within a CRM context. To solve the initiating problem we constructed two research questions. The first RQ examines how CRM system requirements shape the implementation of an chatbot. Our second RQ investigates the performance differences between alternative database types based on the system requirements.

In our first phase we conducted interviews with stakeholders to gain insight into use cases and system requirements. These insights were utilized in Phase 2 that was an experimental phase over four iterations in which each completed iteration was evaluated and new exploratory ideas were carried on.

The results demonstrated that while graph databases generally delivered strong overall performance and high accuracy by utilizing graph traversal, vector databases outperformed them in certain scenarios specifically on retrieval from unstructured data and semantic prompts.

These findings suggest that a combined approach with a graph and vector database is a suitable choice for this context. The evaluation demonstrates that minimizing tool calls is the most critical factor for designing an efficient retriever. Since the context must be repeatedly transmitted back and forth with each tool call, excessive calls significantly degrade overall performance and latency.

Keywords: Retrieval Augmented Generation, Graph Database, Vector Database, Chatbot, Customer Relationship Management, Model Context Protocol, Large Language Models

Acknowledgements

First and foremost we would like to thank our supervisor Lars Bendix for his guidance throughout this thesis. When concluding our work we finally know WTFP, have no CBL and did definitely not develop a prototype.

We would also like to thank Lime Technologies for the opportunity to conduct this thesis. Thank you to Per Fryking for your support and thanks to everyone that made our time at Lime so enjoyable!

Declaration of Generative AI Usage:
During our thesis we used several generative AI tools including Gemini, Chatgpt, Claude. Utilized to proofread text, assist with vocabulary selection, and improve overall sentence clarity.

Contents

Chapter 1

Introduction

This thesis was carried out at Lime Technologies (Lime), a Swedish tech company currently investing in AI. At the beginning of this thesis, their AI team was working on integrating both AI agents and an AI driven chatbot into Lime's CRM system. However, there is an uncertainty regarding whether they should adopt alternative database types as opposed to their current SQL structure. Our research will contribute to the technical understanding of how different database types that utilize Retrieval Augmented Generation (RAG) and the Model Context Protocol (MCP) can be part of future solutions for Lime's unique data structures, as well as other companies with similar data structures, goals, and challenges.

This thesis will address the challenges and opportunities of alternative database types for a chatbot integrated in a CRM system developed by Lime. This was prompted by the initiating problem of whether migrating towards alternative database types as opposed to Lime's current SQL database could be beneficial for their AI development. This thesis also investigates how MCP and RAG can be incorporated. These techniques were chosen based on their relatively new introduction to the AI domain, as well as the mutual interest in them from the stakeholders of this thesis. Although there is a great deal of contemporary research on these techniques, their applicable capabilities and implementation possibilities for CRM data structures can provide new insights into that field.

This thesis builds on current research of how different database types can be useful for AI solutions and how this connects to a CRM system perspective. To answer this, we constructed two RQs. The first RQ examines the system requirements that define the implementation of a CRM integrated chatbot. Our second RQ will evaluate alternative database types given the context and compare them based on the system requirements gathered from the first RQ.

The approach for this thesis was iteratively experimental in order to generate broad insights in our research field. We conducted experiments over four iterations in which each completed

iteration was evaluated and new exploratory ideas were carried on. This approach made future experimental choices simpler and allowed for flexibility throughout the thesis.

Our RQs allow us to solve the initiating problem by generating use cases and system requirements to guide the database experiments. These experiments were evaluated against each other based on performance metrics obtained from the system requirements. The results of the evaluation will provide quantitative guidance on different solutions that Lime can utilize in their chatbot development.

This thesis is structured into six chapters. In Chapter 2, we give underlying background information that relates to the initiating problem, as well as our research questions, methodology, and theoretical foundation. In Chapter 3, we present the results of our first research question that addresses use cases and system requirements. Chapter 4 presents the results and details of our four experiments. In Chapter 5, we reflect on our methodology, discuss our results, bring related work into this thesis context, and lastly propose future work. Our last chapter concludes our findings and insights that emerged from this thesis.

Chapter 2

Background

To understand the context surrounding this thesis, there are several areas that need to be clarified. Without a clarification it is difficult to understand why specific decisions were taken and the motivation for them. Therefore, by presenting and motivating our background we ensure a shared vision of what this thesis intends to accomplish.

This chapter will begin with an introduction of the company, followed by a presentation and analysis of the problem that prompted our research questions. Then the methodology of this project will be described and motivated. Lastly, the theoretical foundation on which this project was built will be presented.

2.1 Lime Technologies

There are several technical challenges related to enterprise software and the Lime CRM system brings their own unique constraints and possibilities that impact development. Therefore, understanding the context surrounding the technical challenges at Lime is important as they define the research and decisions taken in this thesis. By understanding these challenges we ensure that our research is coherent with the technical context at Lime.

We start by briefly introducing the company, followed by Lime's CRM system which is the product in which the work of this thesis is carried out, and lastly, give an insight into the current state of the AI projects at Lime.

2.1.1 History and Product

Lime is a Lund based Software-as-a-Service company with around 500 employees that was founded in 1990 with the original name of Lundalogik. Their primary product is a CRM

system, which is the product we will be working on during this thesis. The CRM system assists in managing all the interactions with customers, by compiling and presenting data from different communication channels that are integrated in the system, and also automates processes that would have to be done manually otherwise.

The CRM system developed by Lime is designed to be highly modular and adaptable as a core design principle from the start. This means that the product can change in visual appearance, system behavior, and database structure. They achieve these custom-built solutions by having Lime's application consultants build the solution for their customers with the adaptable building blocks that Lime has developed and additional custom implementations to suit each customer's special requests. Lime's process is comparable to building with Lego, where their engineers create Lego blocks and then have their application consultants put them together based on the customer's configuration.

2.1.2 AI at Lime

The rise of AI in recent years has resulted in increased demands from Lime's customers for AI integration in the CRM system. At the beginning of this thesis, Lime lacked integrated AI features in their CRM systems. In order to stay relevant and contemporary with their competitors, Lime was pushing for AI solutions that meet customers demands. Currently, they have two ongoing features, a simple chat service with limited contextual knowledge, and tools to create workflow agents that can automate tasks such as summarizing or creating titles.

The objective of the team is to integrate AI features into the system that brings real value to customers. A feature requested by stakeholders is a chatbot with contextual awareness and the possibility of tool use. However, there are several challenges and pitfalls that the chatbot will need to overcome in order to retrieve information from customer databases in an efficient and correct way.

2.2 Problem Analysis

Following the overview of the company context and the current AI environment at Lime, it is necessary to convert these general observations into the initiating problem that this thesis aims to resolve. Without a clearly defined problem, it is difficult to go from general understanding to focused analysis.

Consequently, this section will present the initiating problem and describe its connection to the surrounding context. Finally, the initiating problem will be developed into RQs that this thesis will answer.

At the beginning of this thesis, the Lime CRM system did not offer extensive AI solutions. This has been a business decision, as Lime has chosen not to be an early adopter. However, customers are increasingly pushing Lime to invest in AI, as the lack of such functionality can become a deal breaker for some when choosing a CRM system. Therefore, to stay relevant and competitive, Lime has invested heavily in the development of AI solutions within its CRM system.

An area of interest at Lime has been GPT-based technologies, which has prompted the company to explore the potential of an integrated chatbot. At the beginning of this thesis, Lime's chatbot was in early development and offered limited functionality. It operated only on one object within the CRM system and lacked system-wide retrieval, which restricted its capabilities. Additionally, the chatbot lacked proper tool use through its MCP server, further limiting its performance.

In the future, Lime plans to develop a chatbot that has the power to access and retrieve data from the entire CRM system. Achieving this requires the ability to retrieve the correct data from the database. However, Lime's current SQL database structure is not naturally suitable for RAG-based retrieval, making the topic a central investigation point within the AI team. Lime therefore questions whether other database types might better support the chatbot.

However, evaluating database types is not a purely technical exercise, it is heavily dependent on the complexity of the retrieval use cases. A database that performs well for simple retrieval might fail when faced with multi-layered, relational queries.

Therefore, the purpose of our research is centered on two parts. The first part concerns the system requirements that define the implementation of the chatbot. Understanding the Lime environment will shape the chatbot's possibilities and place it in a realistic and context-specific environment. By studying customer scenarios, the system requirements can also be refined to better address Lime-specific challenges. The second part involves exploring different database types to determine whether current solutions are suitable for a chatbot or if other options are more viable.

Research Questions

To address the core problems and knowledge gap at Lime in their AI mission, the following research questions were formulated from our problem statement:

RQ1: What system requirements define the implementation of a text-based chatbot using an off-the-shelf LLM in the Lime CRM solution?

RQ1a: What use case scenarios exist and how do they shape the system requirements?

RQ1b: How does the Lime CRM system structure influence the system requirements?

RQ2: How do different database types compare in terms of performance, accuracy, and scalability when used for data retrieval, regarding the system requirements.

2.3 Methodology

To address the RQs, this study employs a two-phase methodology. Phase 1 focuses on establishing a foundation by uncovering system requirements through stakeholder interviews and generating use cases for the study. Building on this foundation, Phase 2 consists of an iterative experimental approach to evaluate the comparative performance of database types and retrieval techniques. By structuring the methodology in this way, the use cases and system requirements identified in the first phase directly shape the technical explorations of the second phase.

This section details the interview methods used in Phase 1, followed by a comprehensive description of the experimental design utilized in Phase 2.

2.3.1 Phase 1 - RQ1

Establishing the answer to the first research question (RQ1) was critical to this study, as the identified system requirements directly influenced the exploratory work in Phase 2 and grounded the experiments within the Lime context. Without proper system requirements, we would lack a method to conclude whether our experiments successfully addressed the initiating problem. Consequently, this section discusses the methodology used to answer RQ1 and its sub-questions. We utilized a two-stage interview approach, first conducting general interviews to gain a basic understanding of Lime's AI mission, followed by technical interviews to derive specific insights into how the AI team viewed the requirements for the chatbot.

Since the chatbot was in early development and no usage data was available there were limited options for us to gather information surrounding the AI products. We adopted semi-structured interviews with company stakeholders as the primary method to gain information about Lime's AI mission and to answer RQ1a. The interviews were roughly 30 minutes each and we chose to focus on three key areas: What is the Lime AI mission? How do different stakeholders view AI? What are interesting use cases for AI at Lime? By understanding these questions, we will have well motivated use cases going forward and ground any experiments in Phase 2 on these interviews. To ensure that we capture all critical insights, the initial interviews were transcribed and analyzed. Any identified knowledge gaps were then used to shape the subsequent technical interviews, allowing for a more complete understanding of the stakeholder's and the company's perspectives.

Since the initial interviews were broader and did not include the AI team, we needed a deeper technical understanding to ground our research with technical feasibility and close the knowledge gaps from the initial interviews. The Lime AI team consisted of software engineers and the AI product owner, with whom we discussed to ensure a complete understanding of the AI project at Lime. These interviews provided valuable insight into which evaluation parameters were highest priority as well as their wishes for what the chatbot should be able to do and technical areas of interest that we could consider exploring. Our discussions also involved how certain areas of the Lime CRM system were more or less suitable for certain retrieval methods, such as the benefits of vectorization for certain Lime objects that carry larger amounts of semantic data. These interviews were not as structured as the initial ones and consisted of discussions with technical stakeholders.

In Figure 2.1 the first phase presents the process of generating use cases and system requirements by conducting two sets of interviews. As the same figure suggests, these findings would help ground the experiments in Phase 2.

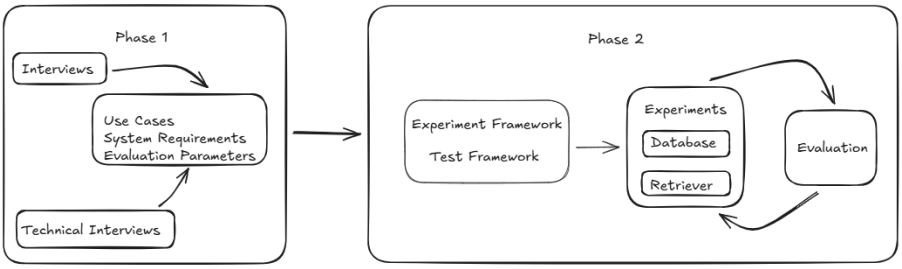


Figure 2.1: Overview of the method

2.3.2 Phase 2 - RQ2

To address RQ2, Phase 2 was designed as an iterative exploratory phase examining the comparative performance of different database types and retriever techniques. This phase directly responded to the knowledge gap identified in the initial problem analysis, which was a lack of understanding of alternative database types and retrieval methods for the chatbot. Our experimental approach concluded the specific results of each database type in relation to the evaluation parameters established in Phase 1, thus providing guidance for database selection decisions at Lime.

The exploratory phase followed an iterative experimental design as illustrated in Figure 2.1, where the findings of each experiment shaped subsequent iterations. This adaptive methodology allowed efficient exploration of alternative techniques without an excessive time investment in less promising approaches. The phase consisted of four stages that progressively evolved toward a comprehensive understanding of database and retriever performance characteristics.

We began by developing a modular framework designed to facilitate seamless component substitution for our iterations. The framework architecture allowed replacements of MCP servers, connected database instances, and retriever implementations. To validate that the framework met project requirements, we implemented a small-scale graph database with a corresponding retriever and conducted initial validation manually through the chat interface to ensure basic functionality.

Following framework validation, we developed an automated testing framework to enable systematic evaluation across iterations. The testing framework was designed to collect and visualize data according to the evaluation parameters established in Phase 1, which enabled quantitative comparison across different database configurations. We prioritized automated testing after the initial iteration to balance early validation needs with the requirement for a more extensive comparative evaluation in later experiments, without manual review for each test case.

The experimental phase consisted of independent, time-boxed iterations. Each iteration experiment was conducted by selecting database types or retriever techniques to further inves-

tigate based on previous findings. The experiment was implemented within the framework and an automated test suite shaped specifically for the experiment was executed. Performance was then evaluated against the evaluation parameters and we decided the scope and focus of the next iteration. Throughout this process, the experiments explored different database types such as vector databases, graph databases, and retrieval techniques.

The final iteration served as a culminating experiment combining techniques and databases identified in earlier iterations. This culminating experiment evaluated the performance of a retrieval system integrating multiple database types and compared this integrated approach to the isolated retrieval methods tested in previous iterations. The goal was to assess whether combining techniques yielded benefits or introduced complexity costs. That would inform Lime's decision to decide between deploying a unified multi-database architecture or selecting a single database type. This adaptive and findings-driven approach enabled efficient exploration of possible solutions while maintaining proper evaluation through automated testing and structured evaluation of each iteration.

2.4 Theoretical Foundation

To evaluate the chatbot, it is necessary to understand the capabilities and architectural constraints of the underlying technology within the CRM environment. The purpose of this section is to establish a shared technical baseline on the specific components included in this thesis. This theoretical foundation is essential for the reader to critically assess the design choices later in this thesis.

This section introduces the general characteristics of LLMs. Then we introduce the characteristics of RAG and how the MCP can be utilized for tool usage. Finally, we define the distinct characteristics of relational, vector, and graph databases.

2.4.1 Large Language Models

Introduction to LLMs

The LLMs used in this thesis are based on decoder-only architecture trained through next-word prediction, where the model learns to predict the next word based on preceding context [15, 17, 23]. Training datasets consist of billions of words collected primarily from publicly available sources [17].

The additional functionalities in LLMs that go beyond language prediction are called emergent properties that arise from increasing model parameters. It is defined by properties that are not shown in small models. The new properties of the LLM cannot be predicted but are observed only after a critical scale is reached. After reaching the critical scale, the performance increases linearly for that task. Properties that are shown to follow the emergent properties are complex skills such as mathematical reasoning or multi-step processes [17].

The performance of LLMs depends on three factors, the number of parameters, the size of the dataset on which it is trained, and the amount of computing used for training [12][17]. Performance is linked to the scale of the LLM which leads to a contentious increase of size for

each new model trained. The LLM training process is very expensive and the total training cost of GPT-4 (hardware and energy) was above 40 million USD [4].

Limitations of LLMs

Hallucination is the most widely known limitation of LLMs. So far, LLMs have been trained to create coherent and predictable text, but there is no algorithm that forces the generated text to be factually accurate [11]. There are two types of hallucination, a model can be factually wrong that means responding with an incorrect answer, or it can be faithfulness wrong, that means that the LLM does not follow the instructions or context provided by the user [17].

A fundamental limitation of LLMs is their knowledge cut-off date. Because the models are not trained on data generated after this date, they lack awareness of recent events and updated information. When asked about recent topics, the LLM is prone to hallucinating responses [26].

Negative biases towards minority groups have been seen in LLMs, this is a result of the datasets the LLMs are trained on. This demonstrates that it is important to exclude sources of poor quality from the training dataset. There are also other types of bias present in LLMs, the large size of training data from US results in a bias towards it in LLMs. There are efforts to create methods to de-bias the models after training, but they have not reached a sufficient level yet [17].

A problem with LLMs is that their knowledge is limited in specific knowledge domains. A domain could be medicine or internal information about an organization. This derives from training datasets consisting primarily of public information. That public data lacks the specificity required for specific domains [15].

With the connection between scale and performance, the drive to increase the datasets of human-generated text data will soon include all public human text data. After that, the further increase of the dataset is unclear and widely discussed [25]. There are attempts to train LLMs with synthetic text data, which means that the data are created by an LLM but in several cases it has resulted in a substantial decrease in performance and the model forgetting skills it previously knew [17].

2.4.2 Model Context Protocol

The Model Context Protocol (MCP) is an open source protocol developed by Anthropic that allows compatible LLMs to perform tasks by enabling custom access to external tools and resources that one incorporates [16]. Before the release of MCP applications, APIs were required to connect directly to the AI application. This process becomes increasingly complicated as you add more tools or resources. This in turn limits scalability and adds to its underlying complexity [9]. This problem led to the development of MCP which solves the underlying issues by introducing a universal protocol that eliminates the need for custom API integration. In Figure 2.2 the difference between performing API connections to AI applications with and without MCP is shown. This overview also highlights some of the architecture that enables the capabilities of MCP.

There are three core components that the MCP architecture incorporates: the MCP host, MCP client and MCP server. Together, they provide a variety of capabilities, but an appropriate example for this thesis is the facilitation of tools. A typical interaction between the components is that an application, such as Claude Desktop or a chatbot acts as a host that the user interacts with. The host contains an MCP client that manages connections to one or more MCP servers. The LLM, running within or accessed by the host, can then invoke these servers through the client to access their capabilities. The MCP servers themselves are independent programs that facilitate certain capabilities, such as tools [16, 9].

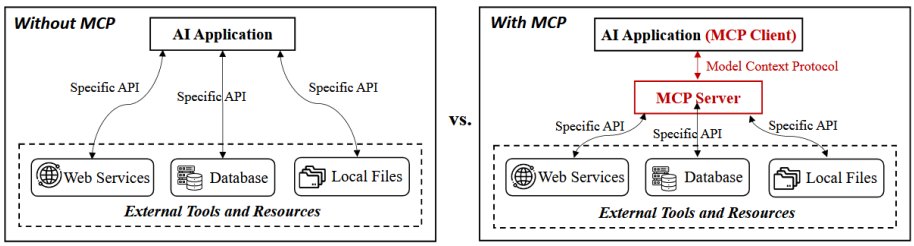


Figure 2.2: Overview of the MCP architecture (reproduced from [9]).

2.4.3 Retrieval-Augmented Generation

Retrieval-augmented generation is a technique to handle the inherent limitations of LLMs, which has a tendency to hallucinate, especially in domain specific context that the model was not trained on. RAG was first introduced by Facebook AI in 2020, later re-branded as Meta AI, and has been a highly studied technique since [13][1]. RAG is one of the solutions that has shown promising results in reducing the tendency of LLMs to hallucinate by introducing up-to-date knowledge without additional training [7]. In this section, we will introduce how RAG achieves this and the components needed.

From a surface-level perspective, RAG is a very easy solution to understand, it adds context to the prompt written by the user to give contextual knowledge to the LLM. To perform this task, traditional RAG is built on two components that together store, locate, and produce the context required to answer the current question.

The first step to implement a RAG solution is to collect the information that should be part of the knowledge base. The information selected to be part of the additional contextual knowledge is converted into text format. After that, all text is divided into chunks of smaller text sections and processed by an embedding model that creates a vector representation of the text chunks. The vector is then stored in a database based on the vector representation [17].

After a database with the required knowledge base is implemented, the next step is to retrieve the most relevant text chunks. That is performed when the user requests a query, using the same embedding model on the user query, the most relevant text chunks can be located with a proximity check between the vector representation of the user query and the text chunks located in the database [17].

The retrieved text chunks are then included in the user query, with this solution the LLM gets additional information about the user query directly into the prompt. One of the benefits

of RAG is that no additional LLM training is required to introduce new domain-specific knowledge [7].

2.4.4 Databases

To grasp the scope of this thesis, one must understand the underlying foundations of different database types and the structures they store. A database's configuration dictates its suitability for various data retrieval approaches. This section explores this relationship, beginning with an introduction to Lime's current database architecture and retrieval methods. We then present information on alternative database types, specifically focusing on those best suited for integration with LLMs.

Databases at Lime

Each of the Lime's customers has a custom database that is created according to the requirement of that solution. Due to the adaptive nature of CRM solutions, the database configuration is modified to fulfill certain solution requirements. The current database type for solutions at Lime is an SQL database, but the user communicates with the database with a custom transpiler called Lime query instead of SQL queries [22]. With Lime looking more into integrating AI solutions, it becomes natural to reflect on whether the relational database with a custom middle layer is well suited for AI integration, which was what initiated this thesis, as stated in the initial problem.

Graph database

The concept of graph databases originates from graph theory that represents data as nodes (vertices) and edges (relationships). Directed relationships indicate how two nodes are connected and add semantic value to the graph. For example, two nodes representing a teacher and a school could have a directed relationship from the teacher to the school that displays *WORKS AT*. The different nodes can also have descriptive properties such as physics teacher or university which adds further value to the semantic interpretation of the node [14].

In a graph database, both the nodes and the relationships can store data, for example, the relationship *WORKS AT* could include the salary and the start date of the employment. In a dataset that includes many relations graph databases are well suited [14].

Vector database

Vector databases are designed to store vector embeddings that are representations of data. This design enables efficient storage and retrieval of unstructured data. The use cases for vector databases can be to store text segments that can be searched with semantically similar text for retrieval [20].

Vector embedding is the central part of a vector database, it's the algorithm that creates the vector from the data that is stored in the database. A vector embedding is a numerical representation that can be performed on multiple types of data. The embedding captures the semantics from the data and allows for searching on vectors that are similar. The same

embedding method is used for the retrieval, and the corresponding vector embedding is used to retrieve the most similar items from the database [21].

Chapter 3

Phase 1: Establishing the System Requirements

In order to evaluate the databases in the experiments conducted in Phase 2, use cases and system requirements were necessary. Without them, the experiments cannot be grounded in the initiating problem. Gathering the use cases required broad interviews with appropriate stakeholders knowledgeable in customer expectations and Lime's AI vision, followed by generating system requirements to apply on the Lime CRM system and generate evaluation parameters that could benchmark the experiments.

In this chapter, we present the results of the first phase of the research. We begin by presenting the findings from the initial interviews and the technical focused interviews with key stakeholders associated with Lime's AI mission. Lastly, we summarize the system requirements and present the evaluation parameters for Phase 2.

3.1 Initial Interviews

Since the chatbot was a project in early development, there was no fixed specification or scope to guide it. This was important because the lack of clarity pushed us to pursue a clearer grasp of expectations and general company wishes. To bridge this knowledge gap, it was necessary to gain a comprehensive understanding of the Lime CRM ecosystem and how stakeholders envision an integrated chatbot. This foundational knowledge was key in grounding the project in practical, real-world applications. This became important to ensure that the research aligned with the Lime AI vision. Therefore, we conducted initial interviews to gain broad insight into what functionality Lime wants to achieve with the chatbot.

From the interviews, we understood that there was no consensus of what functionality stake-

holders expected. This was mostly due to that different stakeholders at Lime had different perspectives and technical understandings. For example, employees in the sales department had experience working with the CRM on a daily basis, selling the product, and knowledge of what AI features the customers want. While employees from the marketing team contributed knowledge of what functions would be good for marketing and align with Lime's public image.

After analyzing the interview transcripts, three general types of functionalities could be concluded from the combined interest of the stakeholders.

- **Retrieval from documents**

  A specific request was for data retrieval from documents which are objects inside the CRM system. Users commonly have documents, such as contracts, product images, etc. saved. Implementing retrieval that can perform various tasks on these documents was commonly brought up during the interviews, ideas of creating summaries and drafting new documents could be interesting use cases.

- **Create and send email**

  A more concrete and practical request for the chatbot was to assist with time consuming daily email tasks. A salesman specifically mentioned the tedious tasks of creating customer focused emails, synonymous with the Lime brand, repeatedly on a daily basis.

- **Internal LLM integration**

  Whilst discussing more with stakeholders who were familiar with AI and chatbots in particular, they expressed the benefits they saw in effectively combining their work with LLMs. This led to discussions about whether it would be better for employees to use chatbots within the CRM to avoid copy pasting information between the CRM and an LLM. This is similar in theory to what Lime was already trying to do for customers, but the difference being this solution focusing on internal work at Lime and training an LLM towards making customer communication more effective.

This information became useful in our discussions with the technical stakeholders, but also to reflect on the scope as it would become quite wide if not limited. Some of the ideas were less concrete, which was not optimal. However, these ideas became the starting point for the technical interviews and guided us to decide areas of interest.

3.2 Technical Interviews

Although the initial interviews discussed potential use cases for the chatbot, they did not provide detailed information about the current technical state of the chatbot and how the AI team envisioned it. This becomes important because the scope of the research had to be focused to avoid the risk of missing technical factors that should be included in the scope. Therefore, this section presents the results of our interviews with technical stakeholders to understand the Lime CRM landscape and also presents how that aligned the expectations on the project scope.

Based on the list of general types of functionality identified in the initial interviews and with a primary focus on retrieval, we determined which user scenarios were the most relevant to our research. These scenarios were then further discussed with the AI project owner. This step ensured that our study's starting point was aligned with the broader AI initiative and focused on the most significant areas.

These technical discussions confirmed two primary areas for further analysis regarding retrieval capabilities. Firstly, the SQL database contains numerous objects. Analyzing methods to retrieve these objects could significantly enhance the LLM's contextual awareness. Secondly, the CRM solution stores documents from different sources, ranging from text files in PDF format to images in different format. It was therefore interesting to analyze methods for retrieving information from unstructured documents to further improve the LLM's context.

The interviews also helped us narrow the scope of the research. Discussions about the email functionality was brought up but was deemed too simple and not something that neither the AI team or we reasonably thought needed any deep-dive research. Instead, it was deemed as a nice to have functionality. Regarding internal LLM integration, it was deemed too wide and difficult to research.

3.3 System Requirements

In order to evaluate the chatbot, the need for underlying system requirements was essential. Without concrete requirements, it would have been difficult to guide the research and conclude whether certain techniques could be useful for Lime in the future. This is important to ensure that our research addresses the problem statement. Therefore, this section defines the system requirements and evaluation parameters for the chatbot given the Lime context.

The technical interviews provided insight into the most critical technical requirements. These can be categorized into two types, evaluation parameters used to guide the exploratory phase, and broader system requirements. Although the latter affect the experiments of this study less, they remain important for real-world implementation.

The evaluation parameters guiding this study are response accuracy (correctness). As the most critical metric, this will be assessed by comparing the output to a pre-defined correct response. In addition to correctness, two additional quantitative parameters were identified. The first is response time (latency), which is expected to have the greatest effect on user experience after correctness. The second is token consumption, which does not directly affect the user, it however dictates the operational cost. Therefore, optimizing for lower token usage is necessary to ensure that the system remains cost-effective.

Complementing these evaluation parameters, the system analysis highlighted system requirements that will be addressed in the discussion. Although not the primary focus of the exploratory phase, they are relevant for the final solution. The first is access control and permissions, it is vital to maintain systems coherent and ensure that the chatbot does not expose data that the user should not have read permission for. The second requirement is scalability, considering both how data volume affects performance and how the solution can be scaled to support all CRM customers.

Chapter 4

Phase 2: Experiment and Evaluations

The system requirements and use cases generated in Phase 1 established the scope of this research. To determine suitability for the Lime CRM environment and guide the company's future AI endeavors, these experiments evaluated alternative database types and techniques. The experimental phase was structured into four distinct iterations to systematically assess the viability of various database types for the Lime chatbot.

This chapter presents all the experiments chronologically, with a focus on the purpose, the outcome, and the insights carried on to future experiments as a result of our evaluations. Iteration 1 focuses on establishing a framework for the chatbot to showcase the possibility of evaluation and a basis for future experiments. Iteration 2 experiments on the same foundation as in Iteration 1, but explores the limitations of the graph database and retriever according to the evaluation parameters. Iteration 3 is divided into two parts, one that introduces vectorization of parts of the existing graph database and the other vectorized documents and images and stores them in a vector database. Iteration 4 was the final experiment, integrating elements from previous iterations to examine how they interacted and performed in unison.

4.1 Iteration 1 - Initial Framework

In order to be able to experiment, we had to build a framework. Therefore, Iteration 1 focused on establishing a framework with a functional graph database integrated with the chatbot, which created possibilities for future experiments.

In this section, we will present our findings from the setup and architecture that enable the initial experiments for the chatbot. This includes the creation of a graph database, retrieving capabilities from the chatbot to the database, and the tools that make it possible. Lastly, we present the results and brief conclusions drawn from Iteration 1.

4.1.1 Starting Point

The starting point for this project was a hackathon project from the AI team at Lime. This project sets the initial environment for our project and works as the foundation for our framework.

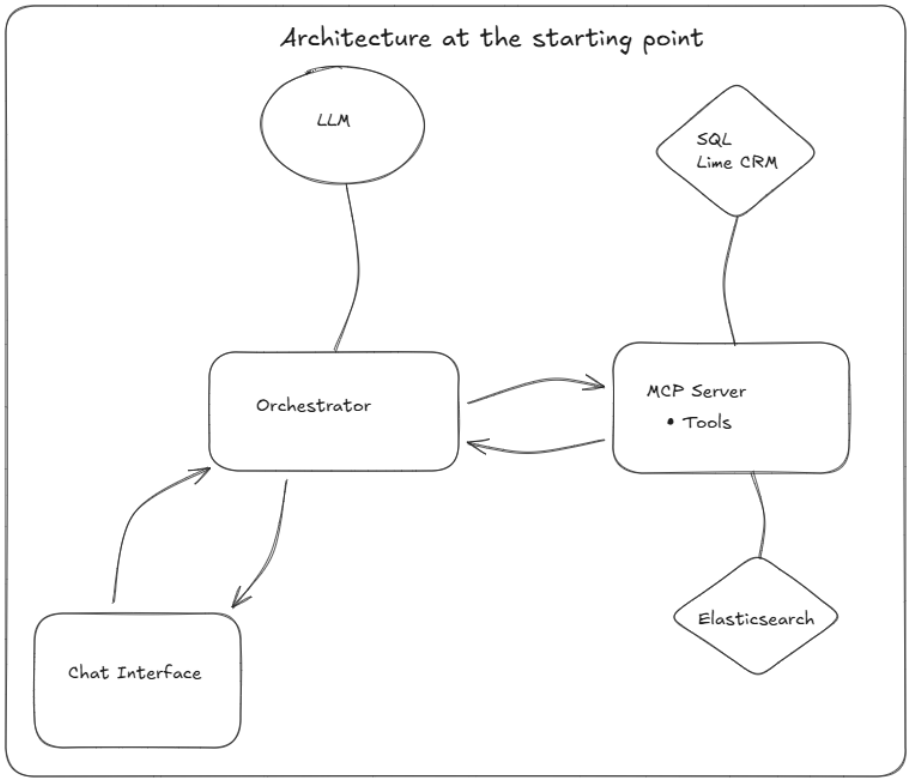


Figure 4.1: Overview of the starting point architecture

Figure 4.1 illustrates the system architecture at the starting point. This setup features a modular design where the Orchestrator and the MCP server function as the primary nodes.

The Orchestrator acts as the "brain" of the system. It connects to the Lime CRM via a text chat interface, links to the LLM infrastructure via a cloud-hosted LLM service, and connects to the MCP server.

The MCP server works as the retriever through tool execution. As shown in the figure, it connects to the SQL database for the Lime CRM. The role of the MCP server in the framework was to retrieve data from the database. The tools relied on an internal query abstraction used within the CRM platform. The tools in the MCP server are called by the Orchestrator. Additionally, the setup includes Elasticsearch which serves as a free-text search engine. Elasticsearch was included in the starting point and will be a part of the culminating experiment.

The execution flow begins when a user submits a question via the chat interface. This triggers the Orchestrator to forward the query to the LLM, accompanied by a list of available tools from the MCP server and their respective descriptions. This list provides the LLM with context on when and how to use each tool and what results to expect.

Based on this input, the LLM determines it's action. Typically requesting a tool call with defined parameters. The Orchestrator routes this request to the MCP server, which executes the tool, retrieves data from the database and returns the result. The LLM then evaluates this new information to decide its next action. This cycle repeats until the LLM generates a final answer or reaches the tool usage limit. Finally, the Orchestrator presents the answer to the user through the chat interface.

This architecture was retained for the framework because its modular design supports targeted changes and experimentation on individual components. The Orchestrator will be kept mostly intact and will not be further discussed in this thesis because it does not affect the performance according to the evaluation parameters. The same reasoning was used for the selection of the LLM, we used Claude Sonnet 4 for this research [2]. Although all LLMs are not equal in performance, since this study focused on experimenting and evaluating database types, experimenting with LLMs would shift that focus and was not something we prioritized.

4.1.2 Database

The first iteration began with data analysis of the existing databases used for testing within Lime. The reason for using a test database was that it does not include sensitive customer data. This was performed to identify a starting database that would be suitable for the first iteration. We decided to select a small sample database for Iteration 1. This database had no approval process before we could start using it, which was the deciding factor, but we did not expect this database to be a long term solution.

At the starting point of this thesis, Lime had a solid understanding of the limitations of using an SQL database for the chatbot. This made it relevant to explore alternative database types and their potential benefits. The other decision surrounding the databases in Iteration 1 was which provider of graph databases to choose. We selected Neo4j Aura because it combines an intuitive user interface with a cloud-hosted solution, allowing for rapid experiments.

After the selection of Neo4j, an analysis was needed to determine what should be migrated to the new Neo4j Aura graph database and how the SQL tables were interconnected. Specifically, core entities such as Persons and Companies were mapped to nodes, while foreign keys were converted into relationships. Based on the analysis, we decided to perform the migration manually through the Neo4j interface to be able to get it functional quickly. Although the initial data set was too small to demonstrate retrieval performance at scale, it was sufficient for the first iteration because the target for Iteration 1 was to show that the framework worked.

4.1.3 Retriever

For retrieval, we created an MCP server that was connected to the existing Orchestrator used in the starting point. As mentioned in the background, an MCP server is a framework used to enable LLMs to perform tool calls on an external server. In the framework, the LLM was not directly connected to the MCP server but through the Orchestrator. The MCP server was used to enable different retrieval tools depending on the prompt inserted into the LLM. When we switched to a graph database there was a need for creating tools that works with

that database. Without these tools it would not be possible to retrieve all the data from the graph database. This motivated us to create tools for different types of retrieval from the graph database which combined allowed for full access to all the data.

The MCP server was connected to the Aura database and managed all database connections. This architectural choice ensured that the same Orchestrator could be used for all iterations of the experimental phase because functionality in the Orchestrator is not dependent on the specific MCP server.

4.1.4 Tools

An MCP server functions by having a suite of tools that can be called by an LLM. In this context, the functionality required was to enable retrieval from the new graph database. The challenge was that the LLM has no prior knowledge of the graph database's specific schema or content. To address this, multiple tools had to be created.

For the first iteration, the tools were developed to enable correct retrieval from the graph database in the simplest way. The goal was to validate the sufficiency of the framework and confirm that retrieval was possible. Consequently, the tools were designed to guide the LLM through a specific sequential order.

The MCP server included three logical categories of tools, and in total five tools designed to be used in a order.

- **Structure Discovery:** The process begins with **list_limetypes**. This tool exposes all nodes and relationship types that exist in the database. It provides the necessary structural context without retrieving actual data records. Once the LLM identifies the relevant node type based on the user's prompt, it calls **get_limetype_properties** and this tool returns all properties types for that node, completing the LLM's understanding of the database schema.
- **Data Retrieval:** With sufficient understanding of the structure, the LLM can proceed to retrieve data. The tool **retrieve_value_list** returns a key-value list (ID and property value) for all entries of a specific node type and a specific property. The LLM uses this to match the user's prompt to a specific database entry ID.
- **Context Expansion:** Finally, once the specific node ID is located, the LLM requests data using **get_limeobject** (All data from one node entry) and **get_related_limeobjects** (To retrieve all entries that have a relation with the specific node entry)

This five-tool pattern successfully enabled the system to retrieve all data and answer user queries. This approach relied heavily on the LLM's logical reasoning and was not designed to be evaluated according to the evaluation parameters, but it successfully demonstrates the functional feasibility of the framework.

4.1.5 Evaluation Methodology

In order to validate that the chatbot could retrieve data across the graph database in the framework, we conducted small scale testing. The testing was performed manually by writing

prompts into the chat interface and validate the answers. The prompt could be "Give me information about company X" or "What is person X phone number?". This allowed us to observe how the LLM reasons and select appropriate tools for a prompt. This process also validated that the architecture was successful. As mentioned above, performance was not the focus for this experiment, so the testing only evaluated accuracy. The testing, although small scale, gave valuable feedback on how the LLM reasons and how it conducts its tool call selection.

4.1.6 Results and Conclusions

From the testing of the framework, the results show that the chatbot can give correct answers to direct questions about information in the graph database through the chat interface. From these results, we can conclude that a framework that supports future experiments has been created.

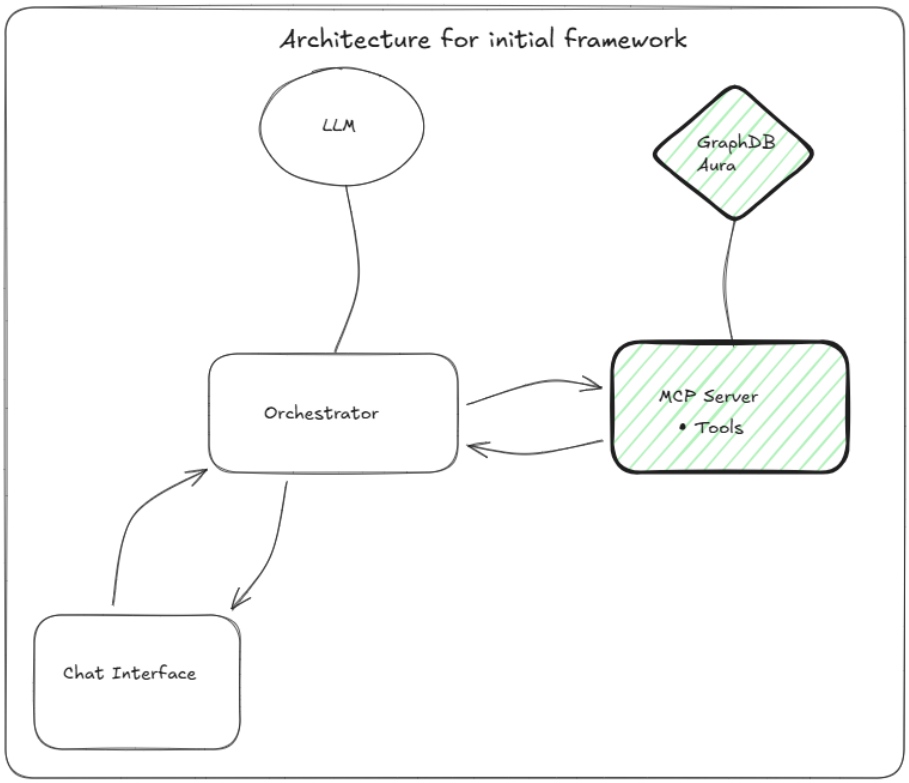


Figure 4.2: Overview of the architecture for Iteration 1

Figure 4.2 illustrates the progression from the starting point, shown in Figure 4.1, to the finished initial framework. Highlighted in green are the parts of the architecture that underwent a change. Most notably is the introduction of the graph database, hosted by Neo4j Aura, which replaced the SQL database. The change of databases also led to the introduction of an MCP server with tools that work specifically for a graph database, as mentioned in earlier sections.

As depicted in the figures, Elasticsearch was decoupled. This was a temporary decision that was motivated by that we wanted the LLM to solely run queries towards the graph database

because evaluating it in isolation would provide deeper insights. However, evaluating the retrieval possibilities with Elasticsearch will be part of the final iteration.

Consequently, the first iteration gave valuable information that defined the experiments of Iteration 2. Firstly, a new database with sanitized and more realistic data was needed. This was due to the fact that the initial database we received was small in data size and riddled with unrealistic data. To properly evaluate further iterations, we would need to find a dataset with more sanitized data. Data such as persons named only *test* or lots of empty objects, reduce the relevant data that can be retrieved. Another issue was that a limited database would reduce the potential information and data gathered from later iterations when we wanted to conduct prompts that cover a wide variety of scenarios.

Another takeaway for succeeding iterations was the importance of a testing framework to conduct automatic tests. The current testing setup was to input the prompts directly into the chat interface one by one. This was not only time-consuming, but also reduced the accuracy of the evaluation because of no structured logging of token consumption and latency.

4.2 Iteration 2 - GraphDB Experiment

The first iteration demonstrated the viability of the initial framework, but did not explore the limitations of graph databases. Since Iteration 1 did not experiment with the performance according to the evaluation parameters of the graph database, the performance limit and behavior of this technique are not yet known. Therefore, the experiment in this iteration seeks to add automatic testing to the framework which facilitate the performance evaluation and also experiment with the limitations and behavior of data retrieval from a graph database.

Consequently, this section details the addition of automatic testing to the framework and the retrieving experiments, including a new database structure, new tools, tool refinement, and few-shot prompting.

4.2.1 Test Framework

The testing of Iteration 1 highlighted the need for a more robust evaluation system due to the lack of reliable test data on the evaluation parameters and the excessive manual effort. This evoked two additions to the testing process.

A test framework that was designed to support a modular selection of test suites, allowing new test suites to be tested seamlessly without altering the framework. The test suites required updating for each iteration to ensure that new experiments were tested accordingly.

In addition, the automated test infrastructure was designed to connect directly to the Orchestrator and bypass the chat interface of the CRM solution. This makes it possible to run prompts automatically and collects essential test data for the evaluation parameters without affecting the results. This new infrastructure greatly increased our test capacity and reduced the manual effort to test new iterations, but more importantly, it generated accurate metrics, leading to more data-driven insights of the experiments.

4.2.2 Database

For Iteration 2, we identified that the initial graph database lacked the depth required for our evaluation and would not suffice as a realistic testing environment due to the lack of objects. Therefore, we adopted a larger and more complex data set that was previously utilized for demonstrations at Lime and contained ten times as many objects as the previous test database. Data quality was highly variable, ranging from complete workflow simulations to basic test data. To improve the quality of the data and make it more realistic, we performed data sanitation prior to migration. This process reduced low quality synthetic data thus rendering the database more similar to a production database. The use of a production database would be optimal for our tests, but GDPR restrictions limited us to the internal test database.

With the sanitized database, the tables were migrated to a new Neo4j Aura instance and mapped to the corresponding graph entities. The tables were converted to nodes and foreign keys were converted to relationships between nodes. Migration resulted in substantially fewer nodes compared to the original table count. The reason for that was that multiple tables can often be merged into one node and also tables used for mapping information can be converted into relations or additional labels for existing nodes.

4.2.3 Tool Refinements

The tool structure from the first iteration was extended with a text-to-cypher tool. Cypher is the query language used for Neo4j graph databases. This tool allowed the LLM to create custom queries and possibly solve a wider variety of prompts. This tool would work similarly to a query generating tool previously developed by Lime engineers for their SQL database. In addition, a relationship retrieval tool was added, which enabled extensive data retrieval from the relationships between nodes. The last focus was to experiment with the tools and analyze their behavior, mostly by working with the tool descriptions. This included few-shot prompting and more detailed guides of proper usage of each tool, which gave insight into how descriptions and examples impact tool behavior.

Relationship Tool

With the new Aura database for Iteration 2, data stored on relationships between nodes were introduced to reduce the overall complexity of the database. This required us to create a new tool for the MCP server that returns the properties for a specific relationship type. This tool was designed in the same way as **get_limetype_properties** from Iteration 1. By keeping the tools to retrieve information about nodes and relations similar, we maintained a standard for structure retrieval. The new tool is called **get_relationship_properties**.

Text-to-Cypher

An insightful experiment was the introduction of a new tool called **make_query**. This was a text-to-Cypher tool for the retriever, which enabled the LLM to write and execute its own Cypher queries on the database. The motivation for this tool was to investigate whether a query generating tool could work for the graph database, which would be interesting for the Lime engineers since they had a query generating tool for the SQL database. The text-

to-Cypher tool had only read access, the reason for this was that the MCP server should solely be used for retrieving in this experiment. This introduction fundamentally changed the retrieval process, previously the tool usage started with understanding the data structure in the graphDB followed by comparing data in nodes with **retrieve_value_list** to understand which node to retrieve. The new process still begins with understanding the data structure, but can afterwards execute self written Cypher queries to retrieve the data from the database. This improved the range of retrievals that it could perform and also reduced the number of tool calls required to retrieve information on average.

However, this tool introduced the risk of hallucinated queries. This results in retrieval failures, increased response time, unnecessary token consumption, and wasted tool calls. To improve correctness of the text-to-Cypher tool use, we experimented with few-shot prompting.

Few-Shot Prompting

Few-shot prompting is a commonly used technique to improve the correctness of tool usage. The approach works by providing examples of intended tool behavior within the tool description, thereby guiding the model both in how the tool should be used and in which situations it should not be applied [24].

In this experiment, we evaluated the impact of few-shot prompting by varying the number of examples included in the tool descriptions. In particular, introducing three distinct examples in the description of the text-to-Cypher tool significantly increased the rate of valid Cypher queries generated by the model. Each example consisted of a natural language prompt paired with the corresponding Cypher query.

However, the experiments also indicated that overly detailed tool descriptions can negatively affect retrieval performance, likely caused by an overload of contextual information that the LLM does not process accurately. Based on these findings, we decided to limit the use of few-shot examples to the text-to-Cypher tool in the current experiment.

4.2.4 Results from Experimental Observations

During Iteration 2, the performance of the chatbot was improved by experimenting and fixing two observations. Although small, their impact became significant. We want to highlight that even small changes can bring ample performance gains.

- **Max tool call awareness**

  A problem that would repeatedly occur was for the Orchestrator to abort when reaching the limit of ten tool calls. This limit was set to avoid token wastage in the case that the LLM keeps trying to find an answer and does not succeed. This became linearly expensive, since each tool call contained the history of the previous tool calls. This drawback is caused by the stateless memory model of the LLM which requires the full history to be resent for every tool call. The theory of why this did occur was that the LLM was simply unaware that it had a limit to adhere to. Therefore, by informing it about the limit and that it should avoid using more than seven tool calls, we noticed a decrease in the number of prompts that reached the limit. We set the limit to seven,

since experiments with different prompts rarely required more than seven tool calls to properly answer the question. Therefore, if the LLM decides to go over seven tool calls there are often other problems and it should instead try to formulate an answer and not continue calling tools.

- **Description differentiation of objects**

  The Lime CRM system separates person objects and coworker objects by the distinction of a person being on the customer side whilst a coworker is employed at the same company as the user. This confused the LLM and could be noticed when the chatbot was prompted to find certain people. It could return answers of a coworker not being in the system due to it interpreting a coworker as a person and therefore not locate the correct node. This was solved by informing the LLM of this situation in the description of the tool to retrieve nodes with a certain category.

These observations demonstrated that descriptions were an effective method of guiding the LLM through system limitations and database structures. Although this approach was advantageous because it required no code modifications or structural changes, it relied on natural language descriptions, which made the system's behavior more difficult to test, verify, and predict.

4.2.5 Evaluation Methodology

The evaluation was conducted with the test framework and provided information for individual tests and the entire test suite. To use the test framework effectively, we created a test suite that targets the different parts of the retrieval capability. After reasoning about the most critical capabilities, we concluded that there were five distinct categories of questions on which we wanted to focus the testing. After deciding to have five different categories of questions, we needed to decide on the total number of tests that should be included. Increasing the number of tests increased the run time for the test, but also the data quality. The evaluation was limited to 15 questions to maintain a reasonable running time of approximately five minutes, which is necessary given that the non-deterministic behavior of the LLM requires multiple iterations. It was also important that each category had several questions in order to base the results of the different categories on more than one test.

Here is a short summary of the categories and some examples of tests from the test suite.

- **Direct look-ups:** These are prompts that the chatbot without reasoning can check in one node in the graph database.
  Example: How much value is associated with the deal Lime CRM Full Digitalization project at panduro hobby ab?

- **Graph traversal:** These are tests that test if the chatbot can traverse the graph (Find information about neighboring nodes).
  Example: Which people work at Lime Technologies AB in Lund?

- **Filtered queries:** These are tests that required the chatbot to perform filtering on all nodes for a specific requirement.
  Example: How many of Magnus Fagerlunds deals are valued over 100,000?

- **Aggregation & analytics:** These tests challenge the reasoning and retrieval of the chatbot in more complex scenarios.
  Example: Rank the top three deals won by salespersons excluding Magnus Fagerlund
- **Error handling & ambiguity management:** These tests how the chatbot handles questions about information that does not exist in the database or questions that are related to data but are misleading.
  Example 1: Get data for company SAAB AB (SAAB AB does not exist in the database)
  Example 2: Show me information on Viktor Eriksson (There exist two persons named Viktor Eriksson)

The entire test suite can be found in Appendix A.1.

4.2.6 Results and Conclusions

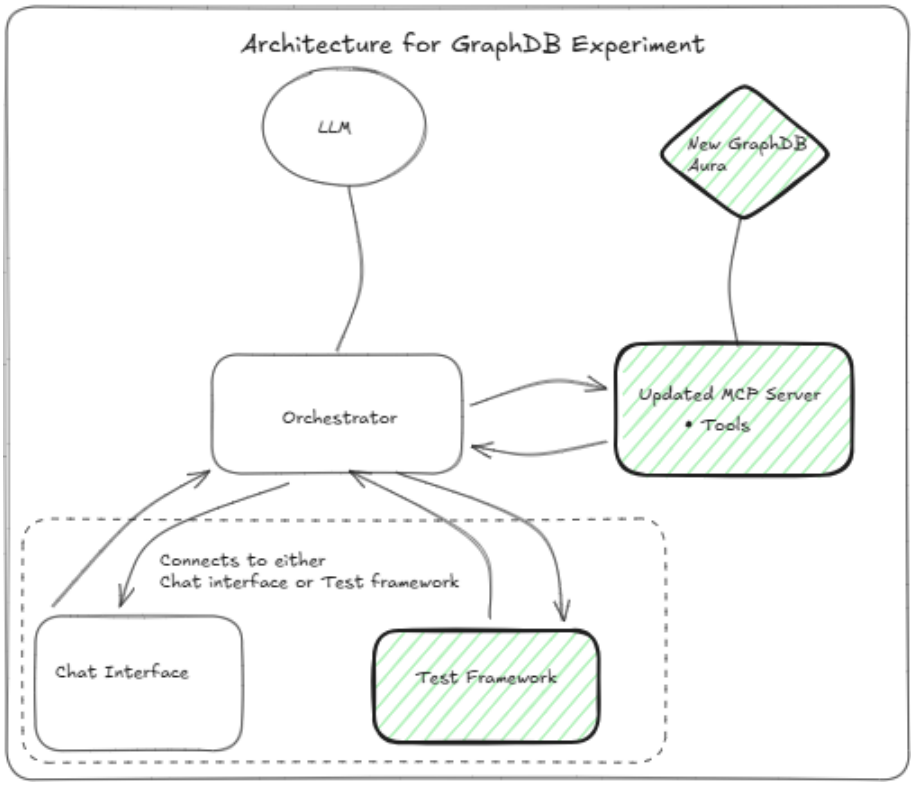


Figure 4.3: Overview of the architecture for Iteration 2

Figure 4.3 illustrate the architecture at the end of the graphDB experiment, this iteration includes the new Aura database, updated MCP server and an update to the framework with the addition of a test framework.

From Table 4.1 the data shows that the accuracy was at a mean value of 87% and with a specific test prompt that failed all iterations. The consistency of the accuracy was good, with a range between 80% - 93.3%. The test case that failed every iteration was testing ambiguity. It was *"Show me information on Viktor Eriksson"* and the issue was that the chatbot found one Viktor Eriksson and proceeded to give information about that person. Without giving information on the fact that there are multiple people with the same name in the system.

Table 4.1: Evaluation for Iteration 2

Metric	1	2	3	4	5	Min	Max	Mean
Accuracy (%)	86.7	86.7	93.3	86.7	80.0	80.0	93.3	86.7
Latency (s)	309	295	315	319	285	285	319	304.6
Token Usage	492,160	446,299	493,688	500,523	425,630	425,630	500,523	471,660
Tools Called	62	56	58	61	54	54	62	58.2

Regarding latency, the response time for an average test question was around 20 s. This response time allowed room for improvement and will be further investigated in future experiments.

Table 4.1 shows that the average tool usage was 3.8 calls per test case, a number driven by the minimum of two required calls needed to understand the structure of the database.

At this stage there are few conclusions to be drawn from the token consumption. When we compare token usage with the chatbot at the starting point, illustrated in Figure 4.1. Tests on the chatbot from the starting point had 970,000 tokens used for the same test suite. In that context, we could conclude that Iteration 2 had a 50% improvement from the starting point.

There was also a perceived correlation between the token usage, latency, and the number of tools called. For example, the fifth test run had the lowest number of tool calls (54), and it also used the fewest tokens and achieved the fastest latency. Overall, the data show a trend that as the number of tool calls increases, both token usage and latency tend to increase as well. However, this relationship is not entirely consistent across all runs. Notably, the fourth test run made fewer tool calls than the first run, yet it consumed more tokens and had higher latency.

It is important to note that there are several factors that can impact token usage and latency besides tool calls. Therefore, it is interesting to conduct further research on the impact tool calls have on token usage and latency.

From this experiment, we concluded that we have explored the potential of this technique. An interesting finding from this iteration is a perceived correlation between the number of tool calls and the latency, as well as token usage. This motivates further experiments on techniques that can reduce the number of tool calls and improve overall performance.

4.3 Iteration 3 - Vectorization Experiment

From the previous experiment, we found that there may be a correlation between the number of tool calls and the overall performance in regards to latency and token usage. Therefore, we explored whether reducing the number of tool calls would improve the performance according to the evaluation parameters. Vectorization was a method of validating this theory, as it allowed for direct data retrieval without requiring the model to navigate the database structure through multiple exploratory steps. Furthermore, this technique also offered the possibility to address a use case identified in Phase 1, the ability to retrieve information from unstructured data sources, such as PDFs and images.

Therefore, this section is divided into two parts. First, we present the node vectorization experiment for existing graph nodes to evaluate our previous finding about the correlation between token usage, latency, and token calls. Secondly, we apply vectorization on unstructured data to explore one of the identified use cases from Phase 1.

4.3.1 Vectorization of Database Nodes

One of the conclusions of Iteration 2 was the perceived correlation between the number of tool calls and the performance in the context of latency and token usage. Although the correctness of the answers was promising in Iteration 2, many prompts required several tool calls before a correct answer was presented. Therefore, we investigated whether vectorization of database nodes could reduce the number of tool calls for some of the prompts and if that consequently would decrease their token usage and latency. This technique was chosen because it connected both our interest in methods for reducing tool calls and was a primary area of interest from the Lime AI engineers for us to research.

Use Case Scenario and Test Data

In addition to the potential tool call reduction, vectorization enables an interesting use case for a CRM chatbot. When working inside a CRM system you might need to find objects that are contextually similar in different ways. This is possible with similarity search, which allows for the comparison of vectorized data and can be used to return one or more objects similar to the given input. This specific use case is most valuable for objects that carry a significant amount of semantic data, since that data can have different content, but carry a similar meaning. One promising object within the Lime CRM system to conduct similarity search on was the lead objects, since they have a description field that can contain semantically rich data. The descriptions of most lead objects in the database consisted of one to two sentences detailing the customer's intent. These texts portrayed various individuals reaching out to a fictive company that sold coffee-related products. Consequently, the leads varied in perceived urgency, intent, and overall seriousness, creating objects that were different but also possible to group into different categories depending on the given description. The discrepancy between objects allowed for more or less similar objects, which would allow the similarity search to work against a wide variety of vectors.

Vector Embeddings

The lead descriptions were represented as vector embeddings in order to capture their semantic meaning. For this process, we used **all-MiniLM-L6-v2** [10], a model from the Sentence Transformers framework [18]. This choice was motivated by the model being balanced in both speed and quality and would be sufficient to start experimenting with. To create the vector embeddings for the leads we used a script that combined the name, company, and description into a single string. We chose these fields because they provided information of the lead object for the similarity search and carried a sufficient amount of semantic meaning. Leads that lacked or had too short descriptions were not given embeddings, since they would not benefit from similarity search to the extent that we strived for. The strings were then passed through the Sentence Transformer model and processed to obtain their vector

representation. Each embedding was then stored as a property in their respective lead in the Aura database from the previous experiment. This was a one time setup that was based on the current data we had in the database, but would require to be executed again with any change to the processed data of the leads.

Vector Search Tool

Once the vector embeddings were stored in the database the focus shifted to experimenting with the MCP tool which would conduct the similarity search. The new tool, **vector_search_leads** would take a prompt and convert it into a vector embedding with the same model previously used for the leads. The tool would then search through the vector index in our Aura database for its most similar embedding using cosine similarity as its distance metric. Since more than one lead may be of interest the vector search tool retrieves a set amount of leads. The retrieval was conducted with similarity score, where a high score suggests that two leads are similar based on the prompt. The LLM can then process this information and present the findings to the user. The tool was given a description that details its usage and performance rules similar to descriptions of tools from previous experiments. Few-shot experiments with the description were conducted to analyze the behavior of when the tool did and did not have examples on which to base its reasoning. Without any examples, the LLM struggled to understand whether or not a prompt required the vector search tool or if another tool was more appropriate. Prompts with the goal of conducting exact property matches did not require similarity search as they did not need any comparison between objects, instead **make_query** would be a faster and more efficient way for those types of prompts. Adding this information, as well as examples where similarity search with the vector search tool was the right approach, improved overall performance.

Evaluation Methodology

The vector search tool was tested with a new test suite that included five tests that targeted the vector search tool and lead objects. The prompts tested a few scenarios for which similarity search could be useful. Such as its ability to find leads with synonyms to the given prompt and variations of the prompt, as well as intent-based prompts that conveyed a perceived interest. The test suite was limited to five prompts since it was difficult to find examples of questions where we could validate the expected output. We based the accuracy on whether the output exactly matched with the expected results, but since similarity search generates a ranked set of possible candidates, it proved difficult to evaluate. Generating prompts that produced consistent results is therefore more difficult than in previous test suites. Spending additional time generating a larger test suite could provide more detailed insight into the vector search tool, but our goals of this experiment could still be achieved with a smaller test suite, since we could gather sufficient data on its behavior.

An example of a prompt was *"Find leads who requests immediate purchases"* which would test the tools ability to properly rank leads which showed this specific intent despite not expressively using the words immediate or purchase. This allowed testing to focus on semantic understanding rather than retrieval with strict matching of the words in the prompt, which demonstrates the strongest use case for similarity search. The entire test suite can be found in Appendix A.2

Results and Conclusions

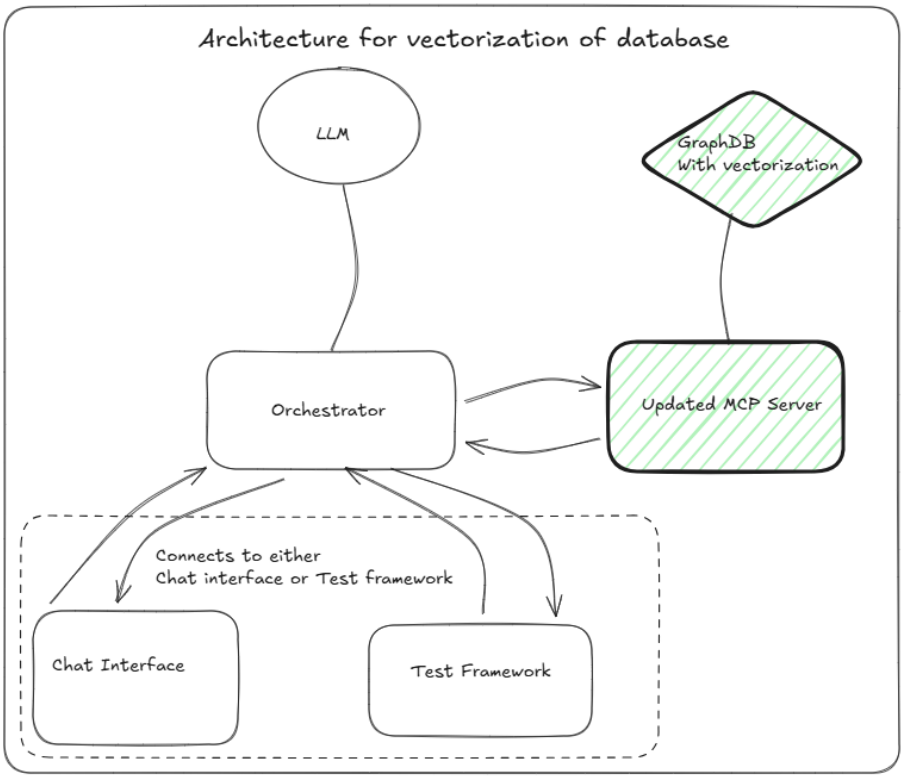


Figure 4.4: Overview of the architecture for Iteration 3.1

Figure 4.4 show the architecture for this experiment and highlighted in green are the changes made compared to the previous experiment. Most of the leads in the graph database were updated with embeddings which were stored as properties in their respective node in the graph database. The MCP server was consequently updated with the vector search tool that enables similarity search for the lead objects.

For the vectorization experiment, we wanted to gather insights into whether vectorization would work in the Lime CRM context for a certain object, which if possible, could motivate vectorization of larger parts of the system in the future. We also wanted to investigate the finding from Iteration 2 and the effects that the vector search tool had on reducing tool calls and how that shaped the performance.

Table 4.2: Evaluation results of vectorization on leads

Metric	1	2	3	4	5	Min	Max	Mean
Accuracy (%)	80.0	100.0	100.0	100.0	80.0	80.0	100.0	92.0
Latency (s)	65	66	64	61	67	61	67	64.8
Token Usage	33,740	33,120	33,660	33,661	34,279	33,120	34,279	33,692
Tools Called	5	5	5	5	5	5	5	5.0

Table 4.2 presents the result of the vectorization experiment on the similarity search test suite, whilst Table 4.3 presents the results of the graphDB experiment on the similarity search

Table 4.3: Evaluation results of vectorization experiment on Iteration 2

Metric	1	2	3	4	5	Min	Max	Mean
Accuracy (%)	60.0	60.0	60.0	60.0	40.0	40.0	60.0	56.0
Latency (s)	142	119	140	115	162	115	162	135.8
Token Usage	170,767	137,892	179,252	140,812	191,032	137,892	191,032	163,951
Tools Called	22	19	23	18	24	18	24	21.2

test suite. These results present how the different experiments perform against the same test suite consisting of prompts that target the similarity search. The tables indicate a clear performance gain in both the latency and token usage across all test runs. The discrepancy in tool calls between the experiments is substantial, where the vector search tool is only called once for each of the prompts and still delivers a high level of accuracy. When comparing the mean values for the tables we can see that the vectorization experiment is close to twice as fast, uses around five times fewer tokens, and four times as few tool calls. These numbers continue to strengthen the correlation between the number of tool calls and the latency and token usage.

The difference in accuracy between iterations was most significant when the prompts lacked specificity. The vector search excelled at looking for synonyms and intent from the prompts, which Iteration 2 struggled with. This demonstrates the power of similarity search when it comes to prompts that require a semantic interpretation. It further strengthens our finding that reducing the number of tool calls has a substantial impact on latency and token usage.

It is important to note that the results contain bias, as the vector search tool was specifically built to answer questions similar to those in the test suite. The tool is limited in its capabilities and the performance might not be as impressive if a generic vector search tool was used instead covering all objects and a larger variety of test cases. This was not tested in this iteration, but is a clear next step for validating vector search on a broader level. There are some considerations as to whether you can limit the generic tool to not include objects which have little to none semantic data, as they might not benefit from similarity search to the same extent. Another consideration could be to create vector search tools for each object type in the database instead of a generic vector search tool. However, this would introduce dozens of similar tools that will increase the context window, risk improper usage by the LLM, and violate code principles.

The goal of the experiment was to show the feasibility of vector search and implementing a tool that works for one object was sufficient to prove it. The results show that vectorization of nodes is possible given the Lime CRM context since the vector search tool is called and acts as intended and achieves a high level of accuracy. However, the benefits of a full scale vectorization of the nodes remain uncertain.

It is also important to note that objects in the CRM system do not include the same amount of semantic data as the lead objects which makes them less suitable, but not unthinkable candidates for vectorization. This is because titles, company names, and personal names can contain enough semantic meaning to motivate vectorization in some cases. The vectorization experiment is therefore a success in that it works as intended given the limitations and further

strengthens our finding from Iteration 2. However, it does not provide any certainty as to whether vectorization of nodes would perform well with a generic vector search tool.

4.3.2 Vectorization of Documents

A use case identified in Phase 1 was the ability to retrieve information from unstructured files. Using the context found in documents, such as PDFs and images, could enable new retrieval possibilities. To explore this use case, this section experiments with the application of vectorization of unstructured data.

Consequently, the section details the selection of the database and the specific documents used. This is followed by a description of the data pipeline covering the entire process from ingestion to retrieval. Finally, the testing methodology, results, and brief conclusions are presented.

Database

To effectively store text segments with their respective embedding, a new type of database had to be introduced. This type of database is a vector database that is specialized in storing vectorization. For the selection of the database provider, we choose Pinecone. The reason for this was that the surrounding architecture is similar to Aura with a cloud-hosted solution. Pinecone is also integrated with the embedding model that was used for this experiment.

Selection of data

The data set used for the first part of Iteration 3 was created from demonstrations by stakeholders at Lime and therefore lacked unstructured data types that a production database would have. To enable the possibility of this experiment, synthetic unstructured data was introduced into the system in the form of PDF documents and images.

PDF documents were generated using an LLM. These documents were linked to existing companies in the database to ensure cohesion, but the semantic content was not aligned with the data in the SQL database. To generate fully aligned and accurate documents, a large part of the experiment resources would be spent on that. The discrepancies between the unstructured and structured data do not undermine the technical validity of the experiment. Because this experiment is motivated primarily by the mechanics of retrieving unstructured context, the specific semantics of the documents remain secondary to the retrieval performance.

A set of images ranging from marketing materials (campaigns, events) and product photos was included in the solution. In total 49 documents were included in this experiment. Although this document count was sufficient to establish the feasibility of the retrieval process, evaluating performance at scale was outside the scope of this experiment.

Data Processing

The pre-processing pipeline differs between PDFs and images because images lack extractable text, they require additional steps to enable vectorization.

This section explains the data processing method for PDF documents, followed by the data processing for images.

- **PDF Documents** We utilized a fixed-size chunking strategy, segmenting text into blocks of 1000 characters with a 200-character overlap. The fixed size chunking was selected because of its simplicity, the issue with fixed-size is that context might be lost in the division between chunks. That is why the overlap is critical to preserving semantic context, ensuring that sentences split between chunks remain understandable in a minimum of one the text chunks. Metadata were also attached to each chunk and included the source filename and page number, which enabled the chatbot to provide precise citations for the retrieved information.
- **Images** Image processing required an intermediate step to convert visual data into semantic text. This was achieved using image captioning, where an LLM was tasked with generating a detailed textual description of the image content within a limit of 1000 characters. This limit was set to ensure that the description of the image would be stored in one text segment. This method was chosen because it converts the data within the image to text that can be processed in the same way as PDF documents. The possibility of having a unified method for processing images and PDF reduces the complexity of the experiment and the architecture. The description was then segmented and vectorized identically to the PDF text. The metadata for the images included a unique database ID, which is required to display the image file to the user upon successful match.

Retrieval of Data

The retrieval mechanism is consistent for both PDF and images, the difference lies in how the retrieved data was utilized. The process began by generating a vector embedding for the user's query. Within this framework, the LLM performed an automated sanitation step by transforming the raw user input into a structured tool call input. This produced a refined search query, which was then embedded and used for a cosine similarity search. Results were ranked on a scale of -1 to 1, with 1 representing an exact match.

Retrieval was conducted using a new MCP server that contains tools exclusively for the Pinecone database. The reason for the new MCP server was to isolate this experiment and therefore improve the quality of test data from the experiment. Two distinct retrieval tools were implemented: one for PDF documents and one for images. Additionally, a tool was introduced to display retrieved images directly in the browser. The output of these tools were handled as follows:

- **For PDFs**, the text chunk and metadata were fed directly into the LLM context window to generate an answer, the reason for that is that they are the most likely text chunks to be related to the question and therefore will give context to the LLM that assists in answering the question.
- **For Images**, the system uses the retrieved ID to open the image in a new browser tab, allowing the user to view the visual information directly. The reason for this was that the stored text chunk was generated by an LLM and therefore it got a limited importance to the user, and for requesting an image the location in the system and the visual

was the expected use case.

Evaluation Methodology

The evaluation of this experiment was performed with the test framework. A new test suite with ten test cases was produced. Five for PDF documents and five for images. The reason for this smaller test suite was that there is only one tool for each of the document types, which does not require a large number of test cases to evaluate. Since this experiment examines a new use case, there will be no relevant data to compare it with, and the accuracy of the responses will be the most important aspect for evaluation. The entire test suite can be found in Appendix A.3.

Results and Conclusions

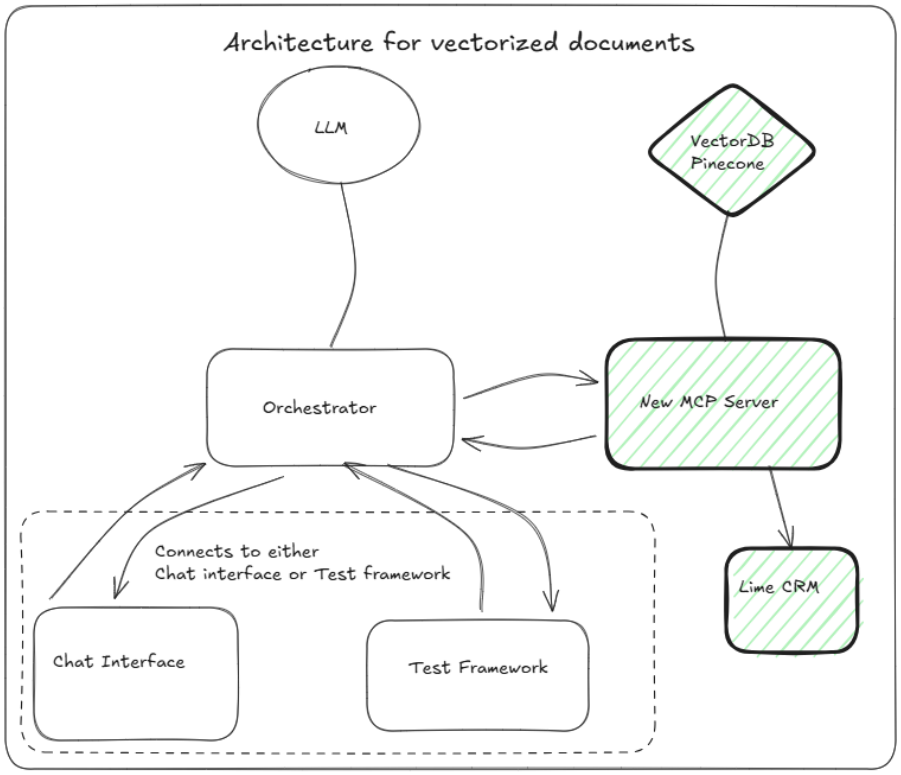


Figure 4.5: Overview of the architecture for Iteration 3.2

Figure 4.5 illustrate the changes compared to Iteration 2. As displayed in the figure the Orchestrator and test framework are intact, but the MCP server was new and only included the needed tools, also a new Pinecone database was connected to that server. Lastly, the connection to Lime CRM is introduced to be able to open the images retrieved.

As shown in Table 4.4 the evaluation was conducted five times, this was performed to lower the variance. As shown in the results, the variance for this experiment was low and had a consistent result.

For accuracy, from the evaluation it was consistent at 90%, the test suite included ten test cases, so one failed each test iteration. From analyses of the test data, it was concluded that the

Table 4.4: Evaluation results from experiment with unstructured data

Metric	1	2	3	4	5	Min	Max	Mean
Accuracy (%)	90.0	90.0	90.0	90.0	90.0	90.0	90.0	90.0
Latency (s)	145.88	134.65	129.64	129.98	113.77	113.77	145.88	130.78
Token Usage	56,410	57,521	51,066	53,166	43,929	43,929	57,521	52,418
Tools Called	19	19	18	18	16	16	19	18.0

same test case failed for all iterations. That test case was **id: D_4** from further investigation it was found that the reason for this failure is that the fixed sized chunking, created a short text segment that included the information required for the test. The semantics of that segment did not have enough information for the embedding to match with the question embedding, and therefore, the data were not retrieved correctly.

The latency displayed in Table 4.4 had a mean time of 131 s for the entire test suite. The test suite included ten test cases, resulting in the mean time for each test case being 13 s. That is an improvement compared to the experiment in Iteration 2. That had around 20 s per test case, but there is not enough to draw any conclusions from this because multiple components changed between these experiments.

The token usage found in the evaluation shows that the experiment was efficient in token usage with an average of 52,418 for ten test cases. This was expected due to the low number of tool calls for retrieval.

The total tool call varied between 16-19 for the five test iterations. This discrepancy was due to the LLM producing the query that was matched with the text segments and that an LLM is not deterministic. Depending on the exact wording, one or more attempts might be needed to retrieve the text segments that contain the required context. Also, to open the images in the browser for the test suite, four tool calls were used. That means that 12-15 tool calls were used for data retrieval.

The failed test case indicates that there might be other segmentation techniques that would further improve retrieval. Despite these potential improvements, this experiment demonstrated that document retrieval effectively increases the chatbot's knowledge base by enabling retrieval from unstructured data that was not retrievable before.

However, despite these promising initial results, questions about scalability remain. As the volume of documents in the vector database increases, the risk of retrieving irrelevant matches may increase, potentially challenging the system's ability to retrieve the correct chunk among similar candidates. We expect that this experiments accuracy would decrease if the database size increased, the reason for this is that the embedding size remains constant and that spectrum will be filled with more text segments, therefore the retrieval has a higher risk of wrong retrievals.

4.4 Iteration 4 - Final Experiment

While previous iterations successfully provided insights and data into the performance of specific techniques in isolation, they failed to capture the synergy of a multi-source system functions when these components operate simultaneously. This was because we first wanted to validate the feasibility of the different experiments before merging them. Without this integrated evaluation, it is impossible to assess how the combined complexity of the retrieval from multiple databases affects performance according to the evaluation parameters from Phase 1.

Consequently, we integrate all previous iterations into a single, unified experiment. We then explore and analyze how performance evolves when multiple database techniques are combined and draw brief conclusions.

4.4.1 Merging the Experiments

The framework design allowed us to easily add more sources for gathering information, which was performed with a new MCP server for this experiment that combined all previous tools. It included all the tools from the experiment in Iteration 2, Iteration 3, and the Elasticsearch tool from the starting point. Some minor changes were made to the tool descriptions to align them with the new structure. This experiment had no new features because the focus was on how a multi-sourced system affects the retrieval.

4.4.2 Evaluation Methodology

The evaluation was the largest part of this iteration because only components from previous experiments were combined into one larger system. This iteration was evaluated on three test suites and compared with the experiment in Iteration 2, and both experiments from Iteration 3. By conducting this evaluation we were able to draw a well-founded conclusion about the impact of the combined MCP server.

4.4.3 Results and Conclusions

Figure 4.6 illustrate that the only new component in this experiment was the MCP server, which was a combination of previous experiments.

Table 4.5: Evaluation of Iteration 4 on test suite from Iteration 2

Metric	1	2	3	4	5	Min	Max	Mean
Accuracy (%)	100.0	93.3	100.0	100.0	100.0	93.3	100.0	98.7
Latency (s)	227.94	252.60	262.21	234.34	252.57	227.94	262.21	245.93
Token Usage	307,432	350,558	338,729	304,646	33,451	33,451	350,558	266,963
Tools Called	34	39	38	35	39	34	39	37.0

In Table 4.5 the data from the test compared to Table 4.1 for the graphDB experiment show that all the evaluation parameters have improved. From our analyzes the improvement in

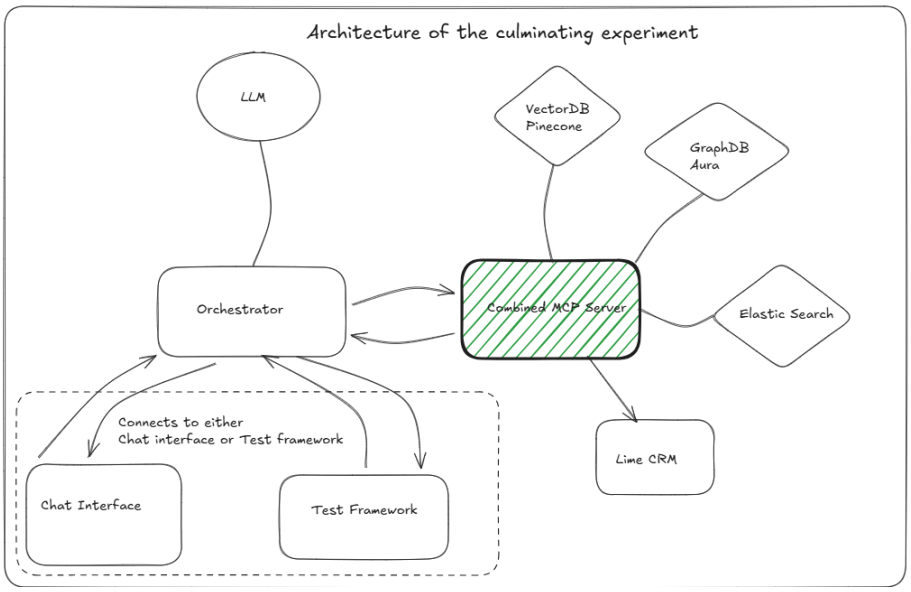


Figure 4.6: Overview of the architecture for Iteration 4

latency and token usage was due to how Elasticsearch can answer direct look-ups with only one tool call. This was also largely the reason for the decrease in the number of tool calls. The improved accuracy came from the multiple sources to retrieve data, it could confirm and search the data with more methods.

Table 4.6: Evaluation of the final experiment with the same test suite as node vectorization Table 4.2

Metric	1	2	3	4	5	Min	Max	Mean
Accuracy (%)	60.0	100.0	100.0	60.0	80.0	60.0	100.0	80.0
Latency (s)	60.89	63.97	65.15	65.55	62.51	60.89	65.55	63.61
Token Usage	40,169	40,311	40,450	40,612	40,346	40,169	40,612	40,378
Tools Called	5	5	5	5	5	5	5	5.0

When Table 4.6 was compared to Table 4.2 which was the result for the vectorization experiment. The data show that both experiments perform equally in all parameters except token usage. The increase in token usage for the final experiment was around 7000 tokens on average for the five test cases compared to the vectorization experiment. This may be due to the increase in the number of tools in the final experiment that also included more and larger descriptions that had to be included in each request to the LLM when it selects tools.

Table 4.7 was compared to Table 4.4 which represents the vectorization of documents experiment. From this comparison the conclusion can be drawn that no improvement was shown, rather the token usage has greatly degenerated. The reason for the increased amount of token usage was explained by the addition of all other tools. This was due to the descriptions of all the tools that were sent to the LLM for each tool call. The impact of the description size becomes apparent when comparing with the token usage of the vectorization of document experiment.

Table 4.7: Evaluation of Iteration 4 on same test suite as Table 4.4

Metric	1	2	3	4	5	Min	Max	Mean
Accuracy (%)	90.0	90.0	100.0	90.0	90.0	90.0	100.0	92.0
Latency (s)	144.83	160.99	132.23	142.11	157.46	132.23	160.99	147.52
Token Usage	125,602	150,935	118,344	127,062	138,297	118,344	150,935	132,048
Tools Called	18	21	17	18	20	17	21	18.8

These results show that a larger MCP server has drawbacks according to our evaluation parameters. The difference in token degradation in vectorization of nodes and vectorization of documents can be contributed to the fact that vectorization of nodes had all the tools from Iteration 2 already included, while the vectorization of documents only had three tools in its MCP server.

The impact of the increased amount of description size was not shown in the comparison with the experiment in Iteration 2, which showed that the addition of tools can also reduce token usage. Also, in the comparison it showed that the number of tool calls for final experiment is on average 20 lower than it was for graph database experiment. That is according to our earlier findings the primary contributor for the improvements.

4.5 Summary

This chapter presented the outcome of four iterations that established a framework and conducted experiments. They provided insights into different areas of interest and how they compared with respect to the evaluation parameters. It can be difficult to navigate large amounts of results. Therefore, we present the key findings from the iterations and the experiments within them in a summarized form:

- **Iteration 1 - Initial Framework**

  The establishment of the initial framework demonstrated the feasibility of connecting a graphDB to the Lime CRM chatbot as an external source of knowledge. It also presented several tools that the MCP server could use to guide the LLM through the data in the graphDB. Although not an objective at this stage, some initial tests were conducted to gain further insight into the functionality of the chatbot.

- **Iteration 2 - GraphDB Experiment**

  The graphDB experiment gave deeper insights of the graphDB capabilities and the MCP tool was extended and improved as performance metrics were analyzed more thoroughly. Another notable step for this iteration was the replacement of the previous graphDB with a new graphDB that contained larger amounts of test data to produce more realistic working conditions for the chatbot. A finding from this iteration was the perceived correlation between tool calls and overall performance.

- **Iteration 3 - Vectorization of Nodes**

  The vectorization of nodes experiment provided insight into the feasibility of vector-

ization that enable similarity search. The results concluded that it provided a better performance for certain types of questions, but generalization still remained uncertain. The number of tool calls reduced drastically, which in turn improved performance and further strengthened our finding between tool calls and performance.

- **Iteration 3 - Vectorization of Documents**

  The vectorization of documents experiment introduced the use case of retrieving documents from the solution. This was possible with the integration of a vector database in which the context of each document was stored. New tools were also introduced that enabled this functionality. The results demonstrated the feasibility of document retrieval, a use case that performed well given the scale of the database.

- **Iteration 4 - Final Experiment**

  The final experiment consisted of merging all iterations into a large architecture in order to investigate the impact this had on performance. The results concluded that the final experiment was faster or as fast as previous iterations for the same combined test prompts, except for the document tests that performed worse. This was due to the additional token usage for descriptions that the combined architecture introduced.

Chapter 5

Discussion and Related Work

Although the preceding chapters detailed the methodology and the results of the experiments, these results alone do not provide a complete understanding of the reliability and limitations of this thesis. Without critical reflection on potential methodological shortcomings, the threats to the validity, generalization, and scalability of the findings remain uncertain. Consequently, by evaluating the findings in relation to the chosen methodology, the experiment results, and contextualizing the findings we can place them in a broader scientific landscape.

In this chapter, we first reflect on our chosen methodology and put it in the context of our research questions. Secondly, we discuss different aspects of our results. Furthermore, we discuss related work and place our findings within the broader scientific landscape. Finally, we propose avenues for future work based on the insights and limitations identified in this study.

5.1 Reflection on Methodology

During our thesis, several methodological decisions were made that impacted the results either positively or negatively. Therefore, by looking back, reflecting and learning from these decisions we hope to contribute insights, as well as propose alternative methods. This allows us to ground our research better in academia and continue improving future work processes.

This chapter will start by discussing the overall reflections of the methodology. That is followed by more specific reflections on our methodology approach for Phase 1 and Phase 2.

5.1.1 Overall Reflections

Our thesis consisted of two research questions that received a drastically different amount of time. Defining the system requirements was a short process that did not allow a particularly deep analysis of the system requirements. However, its purpose of providing valuable input for us to experiment on in Phase 2 was successful. We were able to gain insight from our interviews and analyze them to conclude use cases. We were also able to discuss the influence of the CRM structure on the system requirements with the technical stakeholders in a sufficient way. In hindsight, this limited approach was valuable as it gave us a fixed set of requirements upon which we could ground the experiments. If we had instead decided to iteratively reshape the system requirements throughout our work, we think it would have made the evaluation more difficult. In particular, comparing the experiments to each other would no longer yield the same insight since they adhered to different requirements.

The second phase of our research was structured without clear time frames or predetermined areas for experimentation. We believe that this proved advantageous, since the iterative nature of our experiments allowed us to pivot towards new areas when we encountered obstacles. Also, not deciding which areas to research beforehand, allowed us to decide on areas later in the thesis when we had more domain knowledge and results from earlier experiments to guide the process.

5.1.2 Phase 1

Phase 1 strived to determine the system requirements for the chatbot based on use case scenarios concluded from our interviews. The interviews gave us several insights into how employees at Lime viewed the chatbot from their roles at the company. We interviewed a wide range of stakeholders within the company, but did not ground these findings with real customers of Lime. By not investigating this, we potentially missed out on finding highly requested use cases that might need technical exploration. Although, since we interviewed stakeholders that continuously conducted a lot of customer probing and meetings in regards to Lime's AI development, a sufficient congested and filtered version of customer thoughts was conveyed to us. These ideas proposed by the stakeholders laid the foundation for the areas of interest that we later experimented on. Therefore, we likely covered the most requested ideas from customers but may have missed more niche ideas. Further research could therefore have been beneficial, but would likely provide only marginal improvements to our areas of interest and was therefore not investigated further in this thesis.

5.1.3 Phase 2

Our iterative process for Phase 2 was successful, since we could conduct experiments that were built on each other. Our choice to perform a broad exploration with multiple iterations was well suited for this thesis since the domain knowledge at the start was low, the risk of going into depth on a technique that would prove to not be suitable was substantial. Therefore, utilizing iterations that were built on knowledge from previous experiments allowed us to decide when sufficient knowledge was acquired.

The method used to create new databases was time-consuming, meaning that a significant

portion of the iterations was spent setting up the required environments. While the exact design of these databases is not the focus of this thesis, the time required to migrate the data was initially underestimated. When we decided not to attempt to create a fully automatic process for the migration, we expected the diligence required to ensure accurate migration for manual migration to be a lesser task, than it proved to be. We did not analyze methods for automatic migration, Lime's modular design provides an additional challenge because solutions can have different database structures. This is something that has to be further investigated by Lime to understand the possibilities of a production ready infrastructure.

In hindsight, it would be interesting to conduct a further investigation of how the retriever for the SQL database could be improved, which would create a more fair comparison with the current database structure of the solution. We expected the SQL database to underperform against the graph database, but without an experiment optimizing the retrieval mechanism for the SQL database, a definitive conclusion cannot be drawn.

The method used to evaluate our experiments included human evaluation of accuracy. This raises the question whether it would have been a better method to use LLM-as-a-judge instead to reduce the manual time needed for evaluation which would give more time for further experimentation. Existing studies suggest benefits of LLM-as-a-judge, especially in regards to scalability, adaptability and cost-effectiveness [6]. However, the importance of accuracy for our experiments may dissuade this use, since we speculate that our human evaluation is prone to less accuracy errors than an LLM-as-a-judge method. If human evaluation becomes unsustainable, due to more rigorous test, LLM-as-a-judge would be increasingly suitable.

5.2 Discussion of Results

There are several factors that affected the results of this thesis that may have been overlooked in the thesis. Therefore, to ensure that there is transparency with what may have contributed to any bias, we will critically discuss the results.

In this section, we start by discussing potential threats to the validity of the results. This is followed by an examination of their generalizability. Lastly, we analyze the scalability of the findings, specifically addressing how they would be impacted by larger, more realistic datasets.

5.2.1 Threats to Validity

The datasets used for this project were not from production databases, but datasets created to demonstrate functions and features. That could result in that the data structures we conducted our evaluation on are not representative of production databases. Data sanitation of the database was performed manually and should have increased its similarity to a production database by removing clear test data, but data sanitation also risks removing real connections and data resulting in decreased complexity of the database.

The accuracy evaluation of the experiments was conducted solely with human evaluation. That ensured that it was consistent and fair evaluation, but in less coverage of the database was tested since the time consuming process of evaluating correct results. This leads to the

fact that some parts of the database that were not tested could have performed worse or better, but was never evaluated by our test suite.

There is a potential issue with the questions in the test suite that they include a hidden bias towards questions that the system could answer. This was something that was known when designing the test suites, but no alternative methods were found. To be able to create the test questions required knowledge about the data in the database, therefore we manually created the test suites.

The test questions also risk not including the scenarios that real users would ask from the system. From the interviews we analyzed what type of questions stakeholders expected users to prompt, but that information might not cover all question types. There are many ways to ask a question, and our test questions might have a tone or semantic structure that users do not use with real questions. All of the reasons stated are factors that might affect the results of our evaluation.

5.2.2 Generalizability

The results of this thesis are not directly affected by the environment in which the study was conducted. Our designed framework creates a layer of independence from the Lime solution, the only part that was Lime specific was the exact database structure. That means that the same results should be achieved in another context.

The component that is needed for this technique is an SQL database with a large number of relations. Without the high number of relations, the benefit of graph database is reduced. For Lime, the modular design of the solutions introduces additional challenges with a specific migration required for each solution. This is something that other systems can automate in a more efficient procedure.

5.2.3 Scalability

In this thesis, we were unable to experiment and investigate the effects of scale as initially planned in the research questions. The reason for that was the lack of possible databases that provided the required scale. Our largest database had 5,500 nodes and 13,500 edges. There was no practical or time-efficient method for populating the database with synthetic data, as simply duplicating existing records or adding arbitrary filler data would have compromised the retrieval mechanisms rather than accurately testing the database at scale.

The research question about scalability can be divided into two sub meanings that we will discuss in this section. Firstly, we will discuss how the performance of database types are affected by size. After that we will discuss how scalable the databases are for Lime, if its feasible to introduce these techniques to all their customers.

Effect of Database Size

This section will discuss the effects of increasing the database size for the two database types used in this project. The discussion will be theoretical, since no experiments were conducted on a sufficient size to get results on how scale affects the performance.

All databases are affected by an increase in data but are affected differently. For an SQL database a costly operation is the join operation that is performed multiple times when retrieving complex data for a chatbot. That is a cheap operation for a graph database since they are built around relations [19].

For a vector database, it suffers from semantic crowding when more entries are added. This reduces the accuracy when the database size increases [26]. These examples show that architectures are affected by size in different ways.

Scaling of Customers

To scale the integration of the chatbot towards customers, there are two important factors to consider. First, there are the setup costs associated with integrating the chatbot with the modular design that Lime offers. Since every Lime customer has their own unique solution, requiring custom integrations and database migration of every solution could be costly and something that may not scale well in production.

Another aspect is the infrastructure that is needed for the traffic load of a large number of customers. When discussing with the Lime engineers there was a concern surrounding architectural constraints and whether graph or vector databases could lower these as opposed to traditional SQL databases is difficult to say. Our results point towards that our solutions could lower the number of tool calls as opposed to Lime existing SQL solution, but if it lowers them to a point where it would scale significantly better for customers have not been evaluated.

5.3 Related Work

To understand what insights this thesis contribute to the academic field it is important to place and compare it to other work that interconnect with ours. Without this foundation in academia, it is difficult to establish whether our work strengthens, weakens or completes related work. This section presents four papers from our research field that will be introduced and discussed in relation to our work.

5.3.1 Retrieval-Augmented Generation for Large Language Models: A Survey

This paper provides a comprehensive overview of the current landscape surrounding RAG [5]. Techniques discussed in the paper have been used in our study, though our investigation evaluates them in a real world scenario and, therefore provides further insight into the techniques.

Summary

The primary problem addressed in this paper is the overwhelming volume of distinct technologies and concepts that surround the new research context of retrieval for LLMs. Additionally, it seeks to address the lack of clarity regarding the way RAG systems are evaluated.

The purpose of this paper is to provide a necessary overview of the current RAG environment.

This paper provides a detailed description of techniques across the multiple steps of the retrieval process. The contributions can be categorized into three main areas. Firstly, it classifies RAG systems into three levels: Naive RAG (minimal/traditional setup), Advanced RAG (includes pre/post-retrieval strategies), and Modular RAG (supports dynamic patterns, loops, and query rewriting). Secondly, it provides a deep dive into the retrieval component, highlighting how data structure (structured vs. unstructured) affects retrieval and introducing query processing techniques to enhance LLM understanding. It also covers generation techniques, such as re-ranking to highlight pertinent documents and context compression to reduce noise. Third, the paper outlines evaluation techniques, specifically focusing on quality scores (context relevance, answer faithfulness) and required RAG abilities like noise robustness and negative rejection.

The paper conducts a comprehensive survey and an overview of the retrieval component responsible for collecting contextual information

The paper concludes that RAG systems have evolved beyond linear approaches to Modular RAG, where the model can generate context before searching or loop between steps. Regarding generation, it highlights that noise reduction is critical because, like humans, models may forget information in the middle of long texts. Ultimately, the paper serves as a strong starting point for technical professionals new to this specific area.

Discussion

The authors of this paper have provided an overview on how techniques can be used in isolation and in terms of accuracy. This study evaluates one of our evaluation parameters on multiple methods for the same strategy. The lack of information surrounding other factors, such as latency, is something our study further investigates. According to our interviews, an accurate system that is too slow will not be sufficient for many use cases.

5.3.2 Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions.

Since our work utilizes MCP servers as a contemporary solution for integrating tools with an LLM, one naturally reflects on whether such technology will continue to be an alternative in the future. One has to reflect on the security aspects of MCP when considering deploying their solution to customers in the future. This article [9] provides perspective on how these aspects may affect our solution and Lime as a future MCP integrator.

Summary

The Model Context Protocol (MCP) is a relatively new open source tool for AI integration. Prior to MCP, AI applications relied on manual API wiring that required custom authentication and error handling for each API. This has led to a large interest in MCP servers from

AI stakeholders. Prior to this paper there were little overviews given to MCP servers and industry stakeholders lacked knowledge in the key phases of MCP. There were also no papers on its future development and sustainability in enterprise software that prompted the authors to write their paper.

This paper makes four significant contributions. It provides a comprehensive analysis of MCP's architecture, detailing three core components: MCP host, client, and server. Secondly, it defines the MCP server lifecycle across creation, operation, and update phases, identifying security vulnerabilities for each phase including name collision, tool conflicts, and privilege persistence. Third, it examines MCP's adoption landscape, documenting integration by OpenAI, Anthropic, Cloudflare, and the community ecosystem. Fourth, it proposes mitigation strategies and outlines future research directions for scalability and governance.

The authors used architectural analysis, lifecycle-based threat modeling, and ecosystem surveys. They investigated MCP components and map communication workflows through the transport layer. The security analysis systematically examines vulnerabilities at each life cycle phase (creation risks, operation risks and update risks). Ecosystem analysis involved data collection as of March 2025, surveying platforms such as MCP.so and Glama while also documenting industry adoption patterns.

The paper concludes that the field of unifying LLMs and KGs is a field that has attracted a lot of attention and that their research might provide guidelines that can advance future research. They envision multiple milestones on the road map of unification and present the three stages in which they view increased research.

Discussion

The authors identify security vulnerabilities in MCP implementations, particularly regarding information disclosure through lengthy tool descriptions. Our development experience at Lime provides some support for these concerns. During implementation, we observed instances where the chatbot exposed detailed information about database patterns and system architecture, potentially creating attack surfaces for malicious actors. However, we acknowledge that these observations were informal and did not rely on systematic security evaluation.

The novelty of MCP as a protocol means that security practices remain underdeveloped compared to mature alternatives. While our observations cannot quantify actual risk levels, they suggest that the security considerations raised by the authors merit attention from MCP users. Organizations integrating MCP-based systems, such as Lime, should conduct thorough security assessment before production deployment, as the protocol's may be vulnerable.

5.3.3 Unifying Large Language Models and Knowledge Graphs: A Roadmap

This article details the benefits of unifying LLMs and knowledge graphs to reduce limitations of LLMs such as hallucinations and lack of domain-specific knowledge [15]. Despite utilizing knowledge graphs as opposed to graph databases, this article tackles a similar problem to our, which is to make a LLM context aware with the help of more semantically rich graphs.

Summary

Large Language Models (LLMs) show strong performance in many areas. However, they are difficult to interpret for humans due to their "black box models" and their reasoning is not always easy to grasp. This has partly been fixed with the introduction of chain of thoughts that explain the steps an LLM takes, but this reasoning still suffers from hallucination issues. The authors therefore argues that LLMs are unsuitable in their current state for high stakes scenarios which demand correct answers. To address these issues the authors propose a potential solution of introducing knowledge graphs (KGs) into LLMs.

The paper makes four main contributions. First a roadmap covering the integration of LLMs and KGs that include three general frameworks for their unification. Secondly, they present detailed categorizations of integration strategies for each framework of the roadmap. Thirdly, they present emerging techniques for both LLMs and KGs. Lastly, a summary of challenges and future work is presented for further research possibilities.

The paper is primarily structured as a literature review. The authors analyze recent research on LLM–KG integration and organize their findings. The roadmap is developed by examining architectural designs, integration strategies, and application domains across many studies.The methodology is analytical and qualitative with the focus on providing insights and direction to the field of LLM and KG integration.

The paper concludes that the field of unifying LLMs and KGs is a field that has attracted a lot of attention and that their research might provide guidelines that can advance future research. They envision there being multiple milestones in the roadmap of the unification. And present the three stages they view increased research in.

Discussion

The difference in database structures limits the extent to which findings that the authors found to also be applicable on our work. However, there are several findings of the authors that strengthen our work. For example, our graph database is used for knowledge injection at inference-time. The authors claim that this is preferable when you have knowledge that is subject to change, which is certainly the case with CRM data.

The authors also present future work that strengthens our suspicions of scalability issues for large graphs. They claim that they foresee challenges with scaling knowledge graphs while maintaining real-time streaming efficiency and overall performance. This is something that we suspects as an important issue for Lime to consider if they decide to migrate towards graph or knowledge graphs.

5.3.4 Comparing RAG and GraphRAG for Page-Level Retrieval Question Answering on Math Textbook

This paper has conducted an experiment on how graphRAG and RAG compare when creating a QA bot [3]. The experiment conducted uses a similar database to ours, but in another

domain and use case. This makes it interesting to compare the findings to understand how generalizable they are.

Summary

With the development in LLM technologies there is a question of how it can be utilized in modern learning and as an AI tutor. This study specifically analyses how AI enhanced with RAG would perform as a QA application connected to a math textbook at undergraduate level. With both question answering and reference providing the correct page in the book.

This paper's contribution is to answer the question of how traditional RAG compares to graphRAG for retrieval in this educational environment. It analyses the process to convert the book into retrievable data, how different embedding models and re-ranking affects RAG and how LLM models and embedding models affect the findings.

From the paper it is clear that RAG with re-ranking is the retrieval method that performs best. In all 3 types of score that is presented in the paper, RAG is the best option. For context size the graphRAG consumes 10x the tokens of a RAG system, and in both accuracy and precision Rag with re-ranking performs better. But the authors of the paper argue that even RAG does not perform at a sufficient level for usages reaching an accuracy level of 70% which they argue would be too low to be trusted as a tutor.

Discussion

The authors of this paper provide results on how RAG and graphRAG compares that aligns with other studies, that RAG outperform graphRAG for QA bots on unstructured data with single-hop and detailed answers [8]. Our thesis uses similar database types but structure it from structured data, and use another method for retrieval, graph traversal. In our experiment with vector embedding of nodes we experimented with a more similar method. This paper clarify, that the increased complexity graphs introduce does not result in performance improvements.

5.4 Future Work

During our thesis there was work that we did not have the time or means to investigate further, there were also ideas that arose from our experiments that could benefit from further work. To make sure that our research can be further improved and expanded we present some areas of interest which could be a basis for future work.

Large scale evaluation with a production database

Since we did not have the possibility to conduct our experiments on a larger database or a production database, there is no conclusive answer to how the different database types would perform on a larger scale for a CRM system. As stated in the related work [15], there is also a general need for further evaluation of scalability when graph or knowledge databases are used with LLMs. The performance metrics proposed in this thesis could serve as benchmarks for this research and an analysis of architectural constraints that appear as demand grow could also be of interest.

Rights management system

The security of the chatbot is something that this thesis did not investigate. All users had access to all information through the chatbot, but in an enterprise environment that is not the case. Future work could therefore be to analyze how a secure and efficient solution for this could be designed and implemented. This could also involve securing the MCP servers as brought up in the related work [9].

Streaming of data between databases

For the experiments, we created new databases from snapshots, but in production, databases constantly update and change. In the case of chatbots that can alter data in the database, the streaming has to be bidirectional between the databases. This was something that was outside of the scope of this thesis, but needs further investigation for effective methods to stream data between databases.

How LLM choice affects the result

This thesis did not focus on how different LLMs impact the retrieval logic for the MCP server as we used the same model throughout our work. We suspect that the model can impact the evaluation parameters in multiple ways. Our strongest suspicion is that LLMs with weaker logical reasoning might struggle with correct tool usage, since tool descriptions can be difficult to interpret correctly. The exact differences between models and their impact must be further analyzed in future research to determine best practices.

Chapter 6

Conclusion

This thesis explored the use cases for a chatbot within a CRM solution and identified system requirements and critical evaluation parameters for this context. Based on these findings, we conducted an experimental phase to evaluate how different database types performed given our context.

User interviews highlighted primary needs, with the focus on retrieving data from both structured databases and unstructured documents. The interviews also identified accuracy and latency as the most important evaluation parameters, while token usage emerged as an economic constraint.

The experimental results demonstrate that while graph databases provide high accuracy for data traversal, they suffer from latency and token usage issues due to multiple tool calls for each retrieval. However, later experiments suggest that incorporating vectorization in the graph can reduce latency and token costs. Furthermore, the inclusion of a vector database for unstructured documents proved to be highly efficient, requiring only a single tool call and achieving low latency and token usage while maintaining high accuracy.

This study contributes to the field by evaluating alternative database types for chatbots for the CRM field. We conclude that a system combining a graph database for structured relationships and a vector database for unstructured content provides a reliable and effective architecture. These conclusions provide guidance and will assist future decisions on which database and belonging techniques a chatbot should utilize. Future work should focus on implementing this approach in large-scale production environments to further validate these findings.

References

[1] Introducing meta: A social technology company. https://about.fb.com/news/2021/10/facebook-company-is-now-meta/. Accessed: 2025-10-01.

[2] Anthropic. Introducing claude 4. https://www.anthropic.com/news/claude-4, 2026. Accessed: 2026-02-23.

[3] Eason Chen, Chuangji Li, Shizhuo Li, Zimo Xiao, Jionghao Lin, and Kenneth R. Koedinger. Comparing RAG and GraphRAG for Page-Level Retrieval Question Answering on Math Textbook, September 2025. arXiv:2509.16780 [cs].

[4] Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, and David Owen. The rising costs of training frontier ai models. https://arxiv.org/abs/2405.21015, 2024.

[5] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-Augmented Generation for Large Language Models: A Survey, March 2024. arXiv:2312.10997 [cs].

[6] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Zhouchi Lin, Bowen Zhang, Lionel Ni, Wen Gao, Yuanzhuo Wang, and Jian Guo. A survey on LLM-as-a-Judge. *The Innovation*, January 2026.

[7] Shailja Gupta, Rajesh Ranjan, and Surya Narayan Singh. A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions, October 2024. arXiv:2410.12837 [cs].

[8] Haoyu Han, Li Ma, Harry Shomer, Yu Wang, Yongjia Lei, Kai Guo, Zhigang Hua, Bo Long, Hui Liu, Charu C. Aggarwal, and Jiliang Tang. RAG vs. GraphRAG: A Systematic Evaluation and Key Insights, October 2025. arXiv:2502.11371 [cs].

[9] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions, April 2025. arXiv:2503.23278 [cs].

[10] Hugging Face. all-minilm-l6-v2 model card. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, 2021. Accessed: 2026-02-10.

[11] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*. 3rd edition, 2025. Online manuscript released August 24, 2025.

[12] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models, January 2020. arXiv:2001.08361 [cs].

[13] Hugo Larochelle and Neural Information Processing Systems Foundation, editors. *Retrieval-augmented generation for knowledge-intensive NLP tasks*. Number 33 in Advances in neural information processing systems. Curran Associates, Inc, Red Hook, NY, 2020. Meeting Name: NeurIPS.

[14] Inc. Neo4j. What is a graph database. https://neo4j.com/docs/getting-started/graph-database/, 2025. Accessed: 2025-11-10.

[15] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599, July 2024. arXiv:2306.08302 [cs].

[16] Model Context Protocol. Getting started: Introduction. https://modelcontextprotocol.io/docs/getting-started/intro, 2024. Accessed: 2025-09-26.

[17] Salvatore Iuculand Raieli, Gabriele. *Building AI Agents with LLMs, RAG, and Knowledge Graphs: a practical guide to autonomous and... modern ai agents*. Packt Publishing Limited, S.l., 2025. OCLC: 1516983470.

[18] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019.

[19] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases [new opportunities for connected data]*. O'Reilly, Sebastopol, CA, 2nd ed edition, 2015. OCLC: 1028626678.

[20] Palak Sharma, Ayushman Singh Pundir, Gagan Dev Singh, and Ruchi Gupta. Integration of vector databases with large language models (llms). In *2025 International Conference on Next Generation Information System Engineering (NGISE)*, volume 1, pages 1–5, 2025.

[21] Toni Taipalus. Vector databases and language models: Synergies and challenges. In Panos K. Chrysanthis, Kjetil Nørvåg, Kostas Stefanidis, Zheying Zhang, Elisa Quintarelli, and Ester Zumpano, editors, *New Trends in Database and Information Systems*, pages 192–202, Cham, 2026. Springer Nature Switzerland.

[22] Lime technologies. Lime query. https://platform.docs.lime-crm.com/en/latest/development/lime-query/, 2026. Accessed: 2026-02-10.

[23] Aditya Tiwari, Neha Sengar, and Vrinda Yadav. Next word prediction using deep learning. In *2022 IEEE Global Conference on Computing, Power and Communication Technologies (GlobConPT)*, pages 1–6, 2022.

[24] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[25] Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data. https://arxiv.org/abs/2211.04325, 2024.

[26] Oliver Zahn, Matt Beton, and Simran Chana. Attention Is Not Retention: The Orthogonality Constraint in Infinite-Context Architectures, February 2026. arXiv:2601.15313 [q-bio].

Appendices

Appendix A

Test Data

A.1 Iteration 2

A.1.1 Test Suite

Test cases configuration for automatic testing of the MCP server.

Each test case can include:

- **id**: Unique identifier for the test
- **prompt**: The message to send to the AI
- **expected_output**: Description of expected result

```
TEST_CASES = [
    # --- DIRECT LOOKUPS ---
    {
        "id": "DIRECT_LOOKUP_1",
        "prompt": "Show me the details for the account Malmö FF",
        "expected_output": "Active customer, Martin Fredlund (
    responsible), Erik Perssons väg 30, support@malmoff.se, etc.",
    },
    {
        "id": "DIRECT_LOOKUP_2",
        "prompt": "List the contact methods for Bosse Larsson",
        "expected_output": "+460564959, bosse.larsson@guden.se",
    },
    {
        "id": "DIRECT_LOOKUP_3",
        "prompt": "How much value is associated with the deal Lime
    CRM Full Digitalization Project at panduro hobby AB?",
```

```
        "expected_output": "394 000 SEK",
    },
    {
        "id": "DIRECT_LOOKUP_4",
        "prompt": "Which department does Armand Duplantis work at?",
        "expected_output": "Sales Department",
    },

    # --- GRAPH TRAVERSAL ---
    {
        "id": "GRAPH_TRAVERSAL_1",
        "prompt": "Which people work at Lime Technologies AB in Lund?",
        "expected_output": "Isaac Kiese Thelin, Oscar Bjelvert, Tommas Davoust",
    },
    {
        "id": "GRAPH_TRAVERSAL_2",
        "prompt": "Show me the deals associated with Martin Fredlund",
        "expected_output": "Lime CRM Unified Data Migration Bundle, Lime CRM Automated Notifications Suite",
    },

    # --- FILTERED QUERIES ---
    {
        "id": "FILTERED_QUERIES_1",
        "prompt": "How many of Magnus Fagerlunds deals are valued over 100 000?",
        "expected_output": "5 deals",
    },
    {
        "id": "FILTERED_QUERIES_2",
        "prompt": "Which company-names do there exist two or more of?",
        "expected_output": "9 companies have exact duplicates: Engros Specialisten A/S, IKEA AS, Lime Technologies AB, Sepa OY, T-BYG, Vandelay Industries AB, Veidekke ASA, Välinge Innovation, NTA Ejendomme",
    },
    {
        "id": "FILTERED_QUERIES_3",
        "prompt": "How many companies start with the letter N excluding duplicates?",
        "expected_output": "9 unique companies (10 om dubletter inte utesluts)",
    },

    # --- AGGREGATION & ANALYTICS ---
    {
        "id": "AGGREGATION_ANALYTICS_2",
        "prompt": "Which person is responsible for the most amount of companies?",
        "expected_output": "Fredrik Eriksson, 40",
    },
```

```
    {
        "id": "AGGREGATION_ANALYTICS_3",
        "prompt": "Rank the top three deals won by salespersons excluding Magnus Fagerlund",
        "expected_output": "Processkonsult till chokladfabriken (Tomas Eketorp), Lime CRM Full Digitilazation Project (Fredrik Ekman), Lime CRM Opportunity Pipeline Redesign (Johan Dahlin)",
    },

    # --- ERROR HANDLING + AMBIGUITY MANAGEMENT ---
    {
        "id": "ERROR_HANDLING_AMBIGUITY_1",
        "prompt": "Get data for company SAAB AB",
        "expected_output": "No such company exists",
    },
    {
        "id": "ERROR_HANDLING_AMBIGUITY_2",
        "prompt": "Show me information on Viktor Eriksson",
        "expected_output": "There are two Viktor Eriksson. Should spot and declare that",
    },
    {
        "id": "ERROR_HANDLING_AMBIGUITY_3",
        "prompt": "List deals for company Malmö FF",
        "expected_output": "No deals exist",
    },
    {
        "id": "ERROR_HANDLING_AMBIGUITY_4",
        "prompt": "Is there a coworker named Sergei Bubka?",
        "expected_output": "No but there is one named Sergey Bubka",
    },
]
```

Listing A.1: Test Suite iteration2

A.2 Iteration 3

A.2.1 Vectorization of Nodes

Test Suite

```
TEST_CASES = [

    # --- EMBEDDING TESTS FOR LEADS ---

    # HIGH-VALUE SEMANTIC TESTS
    {
        "id": "HIGH_VALUE_SEMANTIC_1",
        "prompt": "Find leads who requests immediate purchases",
        "expected_output": "Gubriella Fleed (Fleed Food Services), Magnus Fagerlund (Lime Technologies), Carlito Corleone (Corleone Cafe)",
```

```
    },

    # SYNONYM AND VARIATION TESTS
    {
        "id": "SYNONYM_VARIATION_1",
        "prompt": "Find leads that are looking for sponsors",
        "expected_output": "Johan Andersson, Erik Botheim, Torsten
    Nyman",
    },
    {
        "id": "SYNONYM_VARIATION_2",
        "prompt": "Find leads looking for a couple dozen coffee
    makers",
        "expected_output": "Torsten Flinckman (Flinck Foundations)"
    ,
    },

    # INTENT-BASED QUERIES
    {
        "id": "INTENT_BASED_1",
        "prompt": "Find spam or low-quality leads that bring up
    celebrities",
        "expected_output": "Erik Botheim (Haalands polare)",
    },

    # SPECIFICITY TESTS
    {
        "id": "SPECIFICITY_1",
        "prompt": "Find leads interested in Robusta coffee beans",
        "expected_output": "Corleone cafe and special offers",
    },
]
```

Listing A.2: Test Suite Vectorization of Nodes

A.2.2 Vectorization of Documents

Test Suite

```
TEST_CASES = [
    #Document retrival (PDF)
    {
        "id": "D_1",
        "prompt": "What observations were made about maria
    magdalena kyrkan from the inspection?",
        "expected_output": "OBSERVATIONS: "
        "1. Exterior Facade: - Condition: Good. - Notes: No visible
     cracks. Stone cleaning recommended for the north side. "
        "2. HVAC & Climate Control: - Condition: Action Required. -
     Notes: The heating unit in the main hall is operating at 70%
    efficiency. Filters need replacement immediately. "
        "3. Fire Safety: - Condition: Compliant. - Notes: Fire
    extinguishers were last serviced in Jan 2025. Next service due
    Jan 2026.",
```

```
    },
    {
        "id": "D_2",
        "prompt": "Can you summarize the RFP response for Lego A/S?
    ",
        "expected_output": "A detailed bid for a global CRM rollout
    . Has lots of info.",
    },
    {
        "id": "D_3",
        "prompt": "Who wrote the final signature from the client
    side when Indiana Jones signed off the PAC document?",
        "expected_output": "Dr. Henry Jones jr.",
    },
    {
        "id": "D_4",
        "prompt": "Who sent the DSAR in September 2025, requesting
    all their personal data?",
        "expected_output": "Tove Jansson Turku, Finland Email:
    tove@moominvalley.fi",
    },
    {
        "id": "D_5",
        "prompt": "How long was and when did the meeting of the
    internal risk committee take place?",
        "expected_output": "
    Transcript_Internal_Risk_Meeting_Nov2025, Start 14:00 for 45 min
    , 24 Nov",
    },
    #Images
    {
        "id": "I_1",
        "prompt": "How many images of flowers are there in the
    system?",
        "expected_output": "6 from Lego",
    },
    {
        "id": "I_2",
        "prompt": "Can you show me images of klarnas marketing?",
        "expected_output": "Should list 3. Marketing, Marketing2,
    Cybermonday",
    },
    {
        "id": "I_3",
        "prompt": "Can you show me an image of a lego set that has
    the dark version?",
        "expected_output": "Opens image of Dark Flacon",
    },
    {
        "id": "I_4",
        "prompt": "Are there any images special edition objects?",
        "expected_output": "Yes, found atleast 2",
    },
    {
        "id": "I_5",
```

```
        "prompt": "Are there any image that is video game related?"
    ,
        "expected_output": "Lego Star Wars: The Complete Saga",
    },
]
```

Listing A.3: Test Suite Vectorization of documents

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2026-03-17

EXAMENSARBETE Evaluating Database Types for AI Chatbot Data Retrieval
STUDENTER Martin Fredlund, Magnus Herstedt
HANDLEDARE Lars Bendix (LTH)
EXAMINATOR Per Andersson (LTH)

Vad döljer sig bakom en smart chattbot? Vikten av att välja rätt databas

POPULÄRVETENSKAPLIG SAMMANFATTNING **Martin Fredlund, Magnus Herstedt**

När AI-lösningar blir allt vanligare finns det flera övervägningar företag behöver göra i sina arkitektur-val. En av dessa är vilken databastyp man bör använda för att fullt ut nyttja kraften av AI. Ogenomtänkta databas-val riskerar att begränsa prestandan av kopplade AI-lösningar gällande snabbhet, korrekthet och kostnad. Vi har därför utvärderat olika databastyper och tillhörande tekniker specifikt ur en CRM-kontext.

Företag använder i allt större utsträckning AI-chattbottar i sina CRM-system. Men att få chattbottarna att snabbt hitta rätt i djungeln av kunddata är komplicerat och hur en databas är internt strukturerad är helt avgörande för hur smart chattboten upplevs.

Det här arbetet undersöker vilken databastyp som fungerar bäst när en AI-chattbot ska söka fram information i ett kundsystem. För att ta reda på detta behövde vi först förstå vilka krav som faktiskt ställs på en chattbot i den här miljön, för att sedan kunna jämföra hur olika databastyper och tekniker mäter sig med varandra.

Från intervjuer till praktiska experiment För att få en tydlig bild av behoven började vi med att intervjua personer som arbetar med de här systemen. Deras insikter blev sedan startskottet för en rad experiment. Genom att bygga, testa, utvärdera och skruva på tekniken i fyra olika experiment kunde vi steg för steg analysera vad som fungerade bäst i praktiken.

Grafer mot vektorer: Vem vinner? I våra tester ställde vi olika tekniker mot varandra och resultaten var tydliga. Grafdatabaser, som strukturerar information ungefär som en stor tankekarta där allt hänger ihop, gav bäst prestanda och högst träffsäkerhet.

Men utmanaren, vektordatabaser, var vassare i specifika situationer. När informationen kom från mindre strukturerade källor som PDF dokument eller när semantiskt otydliga frågor behövde tolkas var vektordatabasen bättre.

Nyckeln till en snabb och smart chattbot Trots utmanarens styrkor visar vår studie att grafdatabasen är det mest lämpade valet som grund för den här typen av kundsystem.

Under utvärderingen upptäckte vi en avgörande detalj för att bygga en snabb chattbot. För att systemet inte ska bli trögt gäller det att minimera hur ofta AI:n behöver fråga databasen efter mer information. Varje gång ett sådant anrop görs måste stora mängder bakgrundsinformation skickas fram och tillbaka. Om chattboten måste göra det för ofta blir den seg och väntetiden för användaren ökar avsevärt. Den allra viktigaste insikten vi såg för att bygga en snabb och effektiv chattbot är därför att designa den så att den behöver göra så få databasförfrågningar som möjligt.