Frankfurt University of Applied Sciences

Faculty of Computer Science and Engineering

Implementation and Evaluation of an Enterprise Architect Chatbot

Using a RAG-Based Approach

Thesis to Obtain the Academic Degree

Master of Science (M.Sc.)

Submitted by

Hendrik Gruber

Matriculation Number: 1458240

Advisor : Prof. Dr. Jürgen Jung

Co-Advisor : Dr. Rainer Schlör

Hendrik Gruber: *Implementation and Evaluation of an Enterprise Architect Chatbot-Using a RAG-Based Approach,* 16. April, 2026

DECLARATION (ERKLÄRUNG)

I hereby assure that I wrote the present work independently and that I did not use any other sources than those given in the bibliography.

All passages that are taken verbatim or correspondingly from published or not yet published sources are marked as such.

The drawings or images in this work were created by myself or provided with a corresponding source reference.

This work has not been submitted to any other examination authority in the same or a similar form.

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Frankfurt a.M., 16. April, 2026

Hendrik Gruber

ABSTRACT

Lorem ipsum ...

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AI	Artificial Intelligence
APOC	Awesome Procedures on Cypher
AR	Action Research
EA	Enterprise Architect / Architecture
EAM	Enterprise Architecture Management
Gen AI	Generative Artificial Intelligence
HTTP	Hypertext Transfer Protocol
IT	Information Technology
LLM	Large Language Model
MCP	Model Context Protocol
PAR	Participatory Action Research
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
RAG	Retrieval-Augmented Generation
REST	Representational State Transfer
STDIO	Standard Inuput / Output
UI	User Interface
UML	Unified Modeling Language
XML	Extensible Markup Language

USAGE OF GENERATIVE AI

OpenAI GPT-5 and GPT-5.2 (OpenAI, 2025) [30] was used in order to aid in the creation of this thesis as described below.

- In the literature review to help expand search terms and improve search quality by considering additional / similar keywords leading to better search result in Google Scholar and other search engines as well as by leveraging the "Deep Research" functionality which conducts a multi-step research on the internet.
- In order to assist in the writing process. Textual information was not generated with the assistance of AI. Rather, it was used to proofread, give feedback, and support in giving the paper an academic writing style.
- In order to summarize meeting notes, ideas, and protocols while organizing and writing about the Action Research.

Part I

THESIS

1 INTRODUCTION

In the field of xyz. Define that we are building a RAG system which will be referred to as a chatbot. it is not an agent, as it is not executing anything.

Goal of the thesis: this does not have to be a central question, but rather a concrete goal (as discussed on 10.02): definier einfach ein ziel, dass für einen konkreten anwendungsbereich ein AI chatbot einem Junior Berater unterstützt mit der speicherung von fachspezifischen wissen um später explorativ zu evaluieren.

This thesis is structured as follows. Chapter 2 defines the key concepts used throughout the research. Chapter 3 gives an overview of current research, implementations within the domain, and how this research fits into the existing approaches. Chapter 4 describes what the Action Research method is, while the subsequent chapters 5, 6, 7, and 8 describe the details of each iteration of the development. The Action Research is concluded in the final review found in chapter 9. Building on the action research, chapter 10 explains the final prototype with all of its components and technical descriptions. Finally, chapter 12 presents the conclusion and possibilities for future work.

This thesis presupposes an understanding of general terms in the realm of application development and enterprise architecture management. Understanding at an intermediate level is sufficient, as all of the key concepts are explained in detail in the next chapter.

2 TERMINOLOGY AND TECHNOLOGY

This chapter goes in detail on the terminology and technology that will be relevant for the reader to have a foundational understanding of the rest of this thesis. Later chapters will build upon these concepts and pieces of technology.

2.1 ENTERPRISE ARCHITECTURE TERMINOLOGY

2.1.1 *Enterprise Architecture Management*

Enterprise Architecture Management (EAM) can be summarized as being the bridge between the business and IT departments of an enterprise. The goal is to implement information technology that is aligned with the business needs of the company. This is in contrast to the IT department implementing information technology for the sake of implementing information technology, which people in IT are often fond of doing [1]. An unwanted situation would then be when the IT department falls into a siloed way of thinking where they are decoupled from the rest of the company. EAM helps to ensure that the implemented information technology is achieving the right things, namely supporting the business capabilities and processes. [18, pg.s 2-3]

Ensuring that the architecture aligns with business needs requires that the business capabilities of the organization are understood. Business capabilities are defined as what the organization is doing, in contrast to a business process which describes how an organization does something. A business capability thus supports achieving the business strategy by enabling the organization in what it does. [18, pgs. 46-47]

Core concepts of EAM include maintaining application landscapes, ensuring business capabilities are being enabled through the enterprise's applications, and aligning IT systems with business objectives. The goal is to align IT and business by transforming the as-is architecture into an improved to-be architecture. [18, pgs. 14-25, 46-50]

2.1.2 *Enterprise Architect Role*

The role of an enterprise architect is thus complex, because they are faced with the challenge of preparing, planning, deciding, and supporting the changes necessary to improve the application architecture and business. A core task of the enterprise architect is creating the necessary artifacts by gathering the necessary information and presenting it in the relevant types of documentation. Applying changes to artifacts in a landscape comes with

the responsibility of making the changes public and accessible for the respective stakeholders. The enterprise architect is also responsible for maintaining compliance regulations, as not everyone in an enterprise should be able to view this information. Because architecture landscapes are ever changing, the enterprise architect is also faced with the task of maintaining the existing artifacts. This also includes establishing processes and mechanisms to stay on top of the changing organization and stakeholder requirements. Lastly, the enterprise architect is faced with organizing this process. They are responsible for selecting the right tooling to manage the architecture, organize EAM team, and embed enterprise architecture in governance mechanisms. [18, pg. 152-154]

2.1.3 *Architecture Diagrams*

Enterprise architects deal with various artifacts, many of which are architecture diagrams. Common architecture diagrams include business capability maps, application landscapes, business support matrixes, and business object models, to name a few. Each of these diagrams serves its own purpose in documenting an existing as-is architecture or an improved to-be architecture. For example, a business capability map gives an overview of the capabilities that an enterprise fulfills. This map breaks capabilities down from a level 1 capability, which is more high level, down to more detailed capabilities of level n. Application landscapes are used in order to give an overview of which applications are being used throughout the enterprise, often grouped in to functional categories. [18, pg. 53, 76] Combining the two results in a business support matrix, where the cross section between an application and capability indicates whether or not the application enables the capability. This type of matrix allows enterprise architects to maintain an overview of which capabilities are being covered by which applications, if there are redundancies, applications which are being overly relied on, or even uncovered capabilities. [34]

The in this section described concepts will be relevant throughout the remainder of this research.

2.2 TECHNOLOGY

The following descriptions of technologies will help the reader to later understand the implementation details. They serve as a high-level, but sufficient, description of each.

2.2.1 *Large Language Models (LLM)*

A Large Language Models (LLM)

LLMs are capable of supporting in language-related tasks where text needs to be generated, translated, summarized, analysed, or questions answered [14].

[36] describes what llms are and why they are not good with domain specific information and how that causes them to hallucinate.

[39] describes what hallucinating is.

2.2.2 *Retrieval-Augmented Generation (RAG)*

Retrieval-augmented generation (RAG) aims to reduce hallucinations by retrieving relevant information from an external database, such as a knowledge graph, and incorporating that information into the response generation process. [20, 46]

Although LLMs contain basic knowledge within the realm of enterprise architecture management, it may not be sufficiently broad enough or in depth to answer domain- or organization-specific questions. Furthermore, generally trained LLMs do not have access to proprietary information about an enterprise's architecture. As a result, prompting a detailed question related to enterprise architecture or organization-specific data can lead to hallucinations. [20] This is a known challenge that can be overcome by implementing RAG.

RAG offers increased flexibility, as the underlying database can be updated with more information without requiring the language model to be retrained. [46]

2.2.3 *Graph Databases, Neo4j, Cypher, and Knowledge Graphs*

A graph database is a special type of database which represents its data via nodes and edges. Via this structure, it is able to represent entities and the relationships between them. [33, pg. 1] For example, a single business capability may be represented as one node, an application as another node, and a relationship between the two with the information of how the application supports this capability. This structure enables graph databases to build real world models that closely resemble the real world and map closely to the domain [33, pg. 6 and 38].

A key feature of graph databases is the possibility to traverse the graph. As opposed to relation databases, where an index lookup would be necessary, graph databases allow neighboring nodes to be discovered. This is an advantage when attempting to find information related to a looked up entity. [38, pgs. 34-35]

A popular and open source implementation for graph databases is Neo4j [29]. It is a native graph database, meaning it directly stores direct references to adjacent nodes [33, pgs. 149-150] This allows Neo4j to execute queries in milliseconds [33]. Another main advantage of Neo4j is its features to visually interact with the graph via its Neo4j Desktop application or web based tool. [38] This gives the user the option to display the nodes and edges within the graph. Finally, there are a variety of Neo4j libraries that extend its features. One such library is Awesome Procedure on Cypher (APOC), which extends

Neo4j to allow specific procedures and functions. Relevant features include retrieving schema information and converting data. [26, 28]

Within this prototype, Neo4j is implemented as a standalone server and is accessed by the prototype via Cypher queries [33, pgs. 27 and 77]. Cypher enables access to the knowledge graph through specific patterns that match the data. Data can be added, read, and deleted via Cypher. Queries may match data exactly or specify a matching pattern to find similar or related nodes. This is done by declaring the pattern of how the graph should be traversed and letting the database decide how to go about the retrieval. [33, pg. 27][38, pg. 49]

A simple example of Cypher can be seen in listing 2.1. This Cypher retrieves all capabilities and the corresponding applications that support them. This exemplifies the structure of how two nodes (application and capability) are connected via a relationship (supports).

```
MATCH (a:Application)-[:SUPPORTS]->(c:Capability)
RETURN c.name AS capability, c.id AS capabilityId
ORDER BY capability
```

Listing 2.1: A simple Cypher example to retrieve the capabilities supported by applications.

A knowledge graph is thus a graph database composed of various relationships to represent a real world structure or network, as found in enterprise architecture diagrams. [40] Building such a knowledge graph and inferring information from it is a central concept within this research.

2.2.4 *Model Context Protocol (MCP)*

Modern AI applications are typically designed to be connected with further external tools and services, such as a Neo4j database, in order to allow the AI to xyz to do. However, this brings the challenge with it that the developers of the proprietary application must manually define interfaces in order to connect their AI application to these external services. If the AI application is to be connected with n external services, then it needs n custom-built connections; one for each service. This comes with a lot of overhead, makes interoperability difficult to achieve, and can hinder long-term maintainability of a system. [2, 16]

This is where the Model Context Protocol (MCP), developed by Anthropic and released in 2024, comes into play. It standardizes the interface between an AI application and its connected external services by defining a structured protocol for tool discovery, invocation, and result handling. This allows a secure, two-way connection between the AI application and external tools. [2] With an MCP server sitting between the MCP client (the AI application) and the external service, the MCP client only needs to communicate with the standardized MCP interface [16].

Imagine an MCP client which leverages several external services such as an e-mail provider and calendar application. If a user prompt is sent to the

MCP client with the request to create a calendar entry and send the calendar invite per email, then it must decide which of the external tools to invoke in order to fulfil this request. Based on the user's prompt, the MCP client decides which tool exposed by the MCP server should be invoked in order to fulfill the request. The MCP server is then able to invoke the API calls to these external services in order to fulfil the request, i.e. it is able to call the calendar application's API to create the calendar entry and then call the e-mail provider's API to send out the calendar invite. The results are then sent back to the MCP client. [16] Without the MCP server, the AI application would have to have manually written interfaces to communicate with and handle the e-mail provider's API and the calendar application's API.

The MCP server is able to offer this agnostic connection to external services, such as an e-mail provider or calendar application, by requiring the tool's providers to explicitly implement their services as MCP-compatible tool interfaces [16]. There are many pre-built MCP servers offered that allow standardized communication to specific external tools. For example, Neo4j has an official, open source MCP Server. As described later in section 10.2.4, this is the MCP server implemented for the prototype within this research. [27]

3 CURRENT STATE OF THE ART

In order to understand how this research fits into the current state of the art, it is beneficial to go over current research and implementations. This helps to scientifically position the research at hand and justifies its implementation methods. This chapter describes how research on the current state of the art was conducted and summarizes the most relevant papers and projects found.

3.1 PRISMA FRAMEWORK

The importance of documenting the research process in order to demonstrate the exhaustiveness of the research conducted has been emphasized in prior work. A robust literature research comprises querying databases for relevant literature using keywords as well as backward and forward searches based on these findings. Because of the plethora of literature on the subject matter, systematically including and excluding existing research has to be made as transparent as possible. Describing which existing research was included and excluded is vital for the credibility of a literature analysis. [5, 41]

To support in this, the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) framework was applied. The goal of PRISMA is to show why the review was done, how studies were identified and considered, and what was found. [32]

Literature was primarily identified using the Google Scholar search engine. The search process was conducted by defining relevant keywords and searching for these in the search engine. Forward and backward citation tracking was also applied to the extracted sources. Search terms were iteratively improved and clustered around the main themes of this thesis. Search terms included EAM, LLMs, RAG, and Conversational Agents. In addition to manual search methods, a small number of relevant sources were found through prompted searching via ChatGPT [30]. A flowchart of the applied PRISMA framework can be seen in figure 3.1.

Setting up criteria to filter out excessive literature is necessary in order to only be left with the relevant pieces. Academic, peer-reviewed sources were the main filtering criteria. This was to ensure the academic integrity of the research process. The research's focus was then laid on the categories defined by the keyword search terms. The sources found were required to provide value to this thesis. Because this work is very practically oriented, more design and implementation oriented approaches were analyzed, as opposed to meta analyses. A further crucial filtering criteria applied was the publishing date. This is especially relevant for research on LLMs and con-

versational agents, as LLMs such as ChatGPT only adopted widespread use in late 2022 [6], meaning research before then is less likely to be applicable to this research. Meta data of a published paper such as the amount of times it has been cited by further papers may also be an indication of how well grounded the source within its domain. For example, a paper cited by thousands of further papers is likely to be a key paper within its domain. A paper cited by little to no further papers is to be critically considered. Internet sources were kept to a bare minimum.

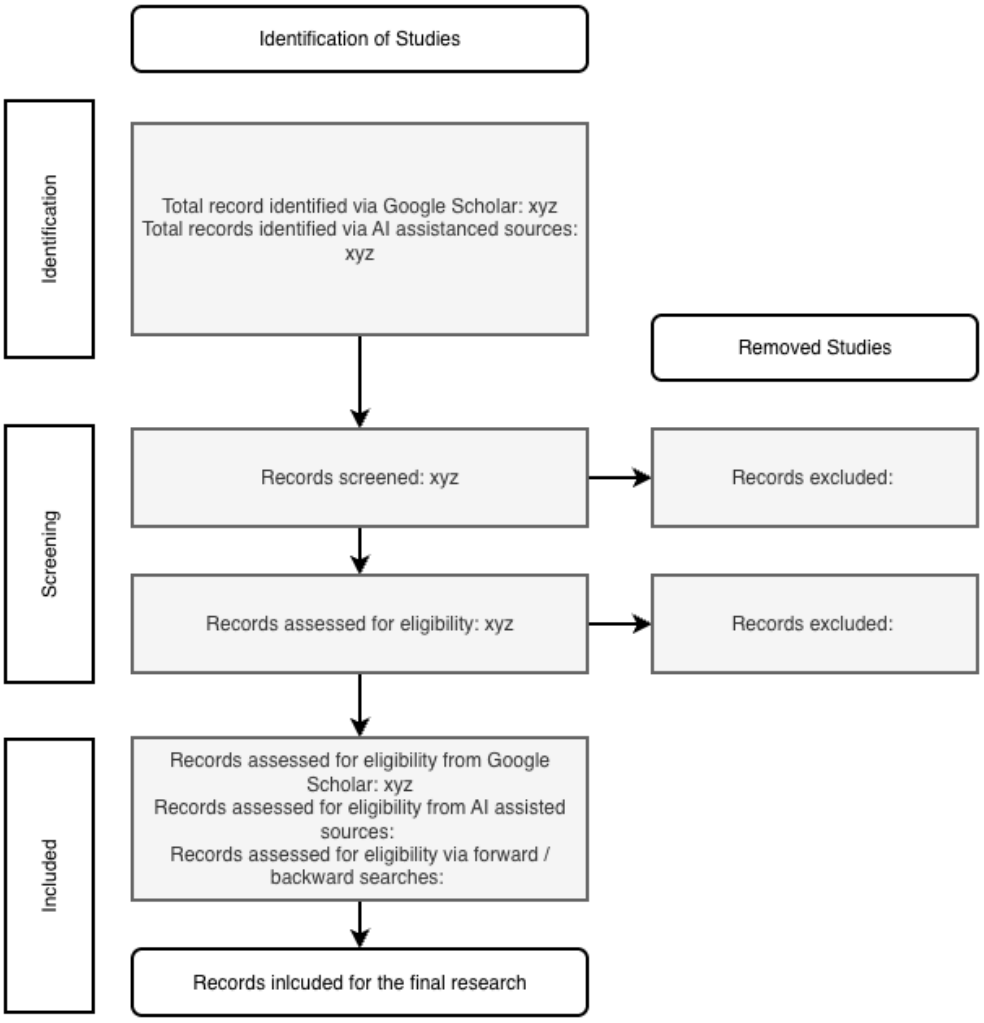


Figure 3.1: A flowchart of the applied PRISMA framework depicting how many sources were initially found with each search method and how many were left for the final current state of the art analysis. The framework is applied as seen in [24].

Applying the above filters, papers found were initially skimmed in order to assess if it may be relevant to this research. This meant going over the papers' abstracts and key points, but not reading every detail yet. This allows papers that do not support the goal of this thesis to be filtered out quickly. Afterwards, the remaining papers were read more thoroughly.

By applying the PRISMA framework as mentioned above, it was possible to reduce the amount of sources from an initial count of *todo* sources down to y sources after the first step and down to the final z sources after the final step.

Todo 10.02.26: was vielleicht ganz gut wäre, wäre eine konzeptmatrix darzustellen zu allen papern, die ich in meiner prisma suche gefunden habe. prisma kann man auch ergänzen um eine unstrukturierte literaturstudie. welche paper beschäftigen sich mit LLMs, welche mit fragentypen, etc. Manchmal hilft auch strukturierung. wenn ich zB eine tabelle einführe mit projekt + kurz charakterisierung, dann brauche ich wenig text und muss nur noch die tabelle beschreiben. deswegen auch eine konzeptmatrix - kreuzprodukt zwischen thema und quelle. das kann ich genauso machen. zeilen quellen, spalten begriffe die relevant sind.

3.2 RELATED WORK AND RESEARCH LANDSCAPE

This section synthesizes and discusses the relevant literature and projects found through the previously described PRISMA-based research process. The selected works are semantically grouped and allow the reader to understand how the research at hands fits within the current research landscape. The purpose of this section is to identify what relevant research and approaches have been conducted to support the relevance and positioning of the RAG-based system of this thesis.

Authors Jung and Fraunholz 2021 [18] lay foundational work for many concepts within the realm of EAM and identify recurring challenges associated with understanding, documenting, and managing enterprise architectures and application landscapes in practice.

One common challenge addressed throughout the textbook is that large, complex application landscapes are difficult to keep an overview over. In extreme cases, this can even lead to an enterprise lacking a complete overview of the applications in use, which severely limits transparency and hinders decision making based off of the data at hand. Another common hurdle arises through uncontrolled growth of the landscape, for example through a merger or acquisition. In such scenarios, previously independent landscapes are fused into one, often leading to redundant systems, overlapping functionality, and unnecessary costs. Even in cases where the application landscape may not be overly complex, it may still suffer from poor documentation. This lack of reliable documentation hinders the ability of enterprise architects to understand application purposes, dependencies, and capabilities covered.

Author Jung (2019) [17] also conducted eight interviews in which enterprise architects, consultants, and decision makers in the German logistics industry to identify concerns that enterprise architecture management is expected to address in practice. The results highlight that stakeholders deal with recurring topics such as providing information about applications, optimizing application landscapes, and reducing the number of applications

to save costs. These concerns form a practical starting point for examining how enterprise architecture data can be structured, queried, and analyzed in order to support questions about existing landscapes.

theories, digital twin efforts, EA tool landscapeStandards or frameworks (e.g., TOGAF, ArchiMate, IATA ONE Record, LeanIX). Theoretical foundations (auch auf prozessmanagment eingehen, wie der aktuelle Prozess aussieht, wenn die Landschaft geändert werden soll) Current tools and methods

The above challenges can be summarized as a lack of transparency within the application landscape and supports the thought that these challenges can be addressed with the assistance of AI. Recurring tasks such as retrieving simple information about an application landscape may be challenging if done manually, due to the complex, heterogeneous nature of large landscapes. Such hurdles may be especially difficult for junior enterprise architects, as they are not able to navigate the landscape as well yet. The challenges addressed by the previous two sources make it seem that they are predestined to be supported by AI-assisted tooling. This supports the case for LLM-assisted enterprise architecture management.

However, LLMs are notorious for hallucinating, meaning that incorrect information is spewed while looking as though it could be factually correct. This is a major downside when interacting with an LLM, especially if expert knowledge in a specific domain is required. An LLM is trained on a large corpus of data and generates its answers through probability calculations. It is thus not concerned with whether or not the returned information is factually correct, but rather only if the text is human-like. [15]

Zhao et al. (2024) [47] investigate how to overcome this issue by supplementing an LLM with a RAG-based solution. Augmenting the context given to an LLM comes with the advantage that the data is up to date, aligned with domain expert knowledge, reduces hallucination, and the data supplied in the context explains the answers generated.

A key challenge mentioned by the authors when developing an RAG-based LLM system is enabling the LLM to comprehend varied forms of data necessary within the applied domain. In the case of the research at hand, this means that a central challenge when developing the RAG-based LLM system will be to structure and query the data in such a way that the LLM is given the right context to respond to the user with factual correctness. It is also mentioned that making a system transparent helps in increasing trust and reliability in the system.

To tackle this challenge, the authors identify four levels of complexity when it comes to information retrieval. The levels are as follows and range from level 1, being less complex queries, to level 4, which require complex information retrieval operations.

Level 1 Explicit Facts: Require no reasoning and are facts found directly in the data.

Level 2 Implicit Facts: May require common reasoning or logical deductions based on the data.

Level 3 Interpretable Rationales: Require a grasp of the factual content as well as domain-specific knowledge which is integral to the data's context.

Level 4 Hidden Rationales: Require that more information than that which is being asked is leveraged to generate the answer.

While queries in level 1 and level 2 are rather straightforward and only require the system to retrieve the relevant facts, queries within level 3 and level 4 require that the system has the capacity to learn and apply the rationales that go above and beyond the data itself. The challenge of query tasks which require rationale can be overcome through in-context learning. The authors describe this as the ability of an LLM to infer hidden rationales by being given relevant examples within the context of the prompt.

These categories will be relevant again later in chapter **??** as these are also applicable within the domain of enterprise architecture management. Each question posed to the system within this research can be broken down into one of these four categories.

However, this does not answer the question of how information from a knowledge graph can be retrieved. This is especially challenging as each question by a user contains a certain level of complexity which needs to be covered by the generated query. How can this challenge be overcome?

Wan et al. (2025) [39] investigate how LLMs can effectively be employed in order to generate Cypher queries and retrieve domain-specific information from a knowledge graph in order to supplement LLM-generated answers. The authors propose a text-to-cypher approach that leverages a lightweight, semantic schema derived from the domain ontology. By injecting the relevant parts of this schema at query runtime, the LLM can be guided toward generating syntactically correct and semantically aligned Cypher queries without requiring model retraining and without overloading the LLM's context. The injected schema context enables the LLM to generate domain-aligned answers grounded in the structure of the knowledge-graph.

For this thesis, this illustrates that a text-to-Cypher approach offers key advantages, such as aligning an LLM with domain-specific information and enabling domain-aligned reasoning without retraining. Additionally, this allows an LLM to remain up to date with a changing knowledge graph by dynamically incorporating the updated schema information at runtime. By exposing the knowledge graph through natural language inputs, the proposed approach lowers the barrier for non-technical users to interact with and extract information from the knowledge graph, allowing for a democratization of the information within it. The research by Wan et al. further reinforces the suitability of a graph-based solution for the research at hand, as enterprise architecture involves highly interconnected data and requires reasoning across various entities and their relationships.

Connecting the research conducted by Zhao et al. (2024) and Wan et al. (2025), it becomes clear that a RAG-based LLM system must be able to categorize the complexity of a user's request and build the queries accordingly. While Zhao et al. (2024) provide a high-level framework for understanding the complexity of the task given by the user, their work remains agnostic to concrete query-generation techniques required to interact with the knowledge graph. This gap is addressed by Wan et al. (2025) through their text-to-cypher approach.

This now begs the question of how such a RAG-based LLM system is able to retrieve the relevant information from the knowledge graph in order to answer the user's prompt with factual correctness.

Chen et al. (2025) [8] propose a graph-retrieval approach that is tightly integrated with an LLM by allowing it to iteratively retrieve and refine relevant subgraphs from a knowledge graph for downstream reasoning. This works by identifying seed entities within the graph based on the user's prompt. A graph retriever then iteratively expands from these seed entities along promising relations while pruning irrelevant nodes and edges. The extracted relevant subgraphs are then given to the LLM as context for reasoning. This multi-hop retrieval strategy allows the system to retrieve complex relational information that is necessary to answer more complex user queries.

In contrast to this multi-hop approach, simpler approaches rely on the LLM-as-retriever paradigm. In such approaches, the LLM performs a one-time extraction based on what the LLM thinks is relevant within the graph. Chen et al. explicitly criticize this LLM-as-retriever strategy. They argue that this does not allow the subgraphs and its intricacies to be fully exploited. Moreover, it requires that multiple LLM calls be made to explore different parts of the graph, leading to complex queries and scalability issues when applied to large knowledge graphs.

So far it has only been discussed how to retrieve information from a knowledge graph. However, it has not yet been discussed how to persist information within it. Preparing and storing data in a graph representation comes with challenges of its own. The applicable method to break data down into nodes and edges depends on the source data. For example, converting a textbook, which is semi-structured, will require a different approach compared to converting an existing application landscape, which is structured data. How can the challenge of creating a knowledge graph be overcome?

Laurenzi et al. (2024) [19] address this challenge by proposing a six-step process for creating a knowledge graph with the assistance of LLMs. The six-step cycle is broken down as follows:

1. Understand what types of questions the knowledge graph will have to be able to answer, as defined in the previous research by Zhao et al. (2024).

2. Construct the knowledge graph's schema.

3. Extract the knowledge from the data source and map it to the ontology of the knowledge graph.

4. Validate the mapping between the data source and the knowledge graph.
5. Add the data to the knowledge graph.
6. Update and maintain the knowledge graph.

Their research takes an LLM-based approach to each of the six steps. For example, in step 1 an LLM was used in order to generate competency questions in natural language. In step 5 an LLM was used in order to generate queries based on natural language input. Within the research, ChatGPT achieved the best results. Sticking to the example of step 1, the LLM was able to generate well tailored competency questions for the knowledge graph. For step 5 it was also able to generate queries, with only minor corrections necessary for successful knowledge extraction. However, the authors note that this step prerequisites a certain awareness of the LLM and the underlying ontology schema, enabling the user to understand what information is being queried and how this may be improved. Query results can then be validated against the actual knowledge graph instance. This awareness is beneficial when working with LLMs and knowledge graphs.

The results of this study are beneficial for the research at hand as it details how to set up a knowledge graph with the assistance of LLMs. This will be relevant again later in section **??**. However, while the work by Laurenzi et al. (2024) demonstrates how LLMs can assist in the creation of a knowledge graph, this process requires a human in the loop that is aware of the ontology. Moreover, this approach may be flexible, but it is not scalable. In many cases when dealing with enterprise architectures, the data is already structured in architecture diagrams and can be exported from the respective modeling tool into structured formats such as XML. This raises the question of how an automated approach can be taken in cases where the artifacts exist and only require systematic parsing in order to be added to the knowledge graph, without the assistance of an LLM.

Smajevic and Bork (2021) [37] introduce a generic, end-to-end platform that enables automated transformation of conceptual models into knowledge graphs. The authors define a cloud-based platform in which predefined models, exported in XML from Archi, can be uploaded. The uploaded model is then transformed and added to their Neo4j graph database. However, the authors do not reveal in detail how this transformation is conducted. For the research at hand, the results show that an automated approach for transforming common exported formats is a feasible solution when generating a knowledge graph via a preexisting conceptual model.

The above two researches by Laurenzi et al. (2024) and Smajevic et al. (2021) both exemplify two different approaches to creating knowledge graphs. While Laurenzi et al. (2024) proposes an LLM-based, hands-on approach which offers flexibility in creating a knowledge graph from the ground up, Smajevic et al. (2021) offer a generic solution to transform an existing model into a knowledge graph in an automated fashion. Both approaches come

with advantages and disadvantages and will later be relevant in sections **??** and 11.

Taking a step back from the technical details of a RAG-based implementation, it is fair to ask why it is even necessary to digitalize the work that can be done by expert enterprise architects.

This paper covers how ai tools are more scalable than manual expertise analysis of things. The source is highly relevant. Look at the summary in notebookLM. 05.10.25 [13]

This paper [42] gives a standardized method and framework for evaluating conversational AI agents.

The next section takes a step away from academic papers and delves into similar projects and prototypes.

3.3 RELATED PROJECTS

Because the research at hand deals with the development of a prototypical RAG-based LLM chatbot, it is worthwhile to have a look at similar projects on the market. This section also takes a look at common challenges faced by such chatbots.

This paper gives an overview on how to control the dialog sequence and also notes 4 types of dialog options for chatbots in the related works section: [21].

Proof-of-concepts, research prototypes, industry whitepapers, GitHub projects.

Tools like ChatEA, LeanIX AI features, or Microsoft Copilot integrations in architecture/governance.

A prototypical graph-based RAG approach for text-summarization has been created by Microsoft: y[11]. The accompanying paper is here: [12]

Dragon1 needs to be detailed here!

The reviewed literature demonstrates that recent advances in LLM-based text-to-cypher approaches xyz... However, existing approaches xyz (what do they lack?). This thesis builds upon these findings by proposing a graph-based, schema-aware approach tailored to enterprise architects and enterprise architecture data, aiming to combine the flexibility of LLMs with the required domain knowledge of enterprise architecture.

4 ACTION RESEARCH

The next six chapters describe how the chatbot “Masutā” (pronounced “mah-soo-taa” - a phonetic adaptation of the English term "master" into Japanese katakana) was developed. It provides further insight into the applied research method and the undergone iterations through which the prototype was created. This gives the reader the necessary understanding of how the research and implementation was conducted before discussing the implementation details afterwards in chapter 10.

4.1 ACTION RESEARCH DESIGN

Action Research was applied in order to gain scientific value out of the development of the prototype. The advantage of this research method is that it is very supportive of the development process for information systems. According to Baskerville (1999) [3], all types of Action Research have the following four characteristics in common: An orientation towards developing, a focus on a specific problem, an iterative process, and a collaboration amongst participants. This is applicable to the work at hand because a prototype is being developed for a specific problem and in an explorative manner with the support of industry experts.

Cyclical phases are central to the concept of Action Research, which contains an iterative process consisting of five steps within a single cycle. Other sources, such as from Cornish et al. (2023) [10], propose variations with fewer (usually three) phases within a cycle; however these models also boil down to the same concepts. Across the literature, Action Research cycles follow the same structure: planning what should be done in the new cycle, taking action, and evaluating the outcome of the completed cycle before moving on to the next one [3, 10].

The paper at hand applied a cycle using the following five steps according to Baskerville (1999) [3]: diagnosing, action planning, action taking, evaluating, and specifying learning. The reason this research applies five cycles instead of three is the benefit of describing the development process in more detail. This five-step-cycle including the preliminary and subsequent steps built the basis of the conducted Action Research.

The following list describes the goal of each of the five phases [3]:

1. Diagnosing: Deals with diagnosing the primary problems that require the organization to adapt.
2. Action planning: Researchers and practitioners collaborate on what the next steps are to relieve the diagnosed pain points.
3. Action taking: The actual implementation of the planned actions.

4. Evaluating: The collaborative evaluation of the outcomes of the action taking phase. Evaluation includes determining if the theoretical effects of the action were realized and whether or not this relieved the problems. The output of this phase is generally the input for the next cycle.

5. Specifying learning: The learnings of the cycle must be documented and applied to the next cycle.

Within this research, each cycle lasted between three and four weeks. The applied sources do not mention how long a single cycle should last. However, a timeframe of two to three weeks for each cycle was deemed as a reasonable for the development of Masutā because status updates were held at the end of each cycle and with the given amount of time for the cycle, there was enough progress to evaluate in these status update meetings.

Action Research typically leaves the main theorizing up to the researcher. However, an extended form of Action Research named Participatory Action Research goes a step further in creating a more collaborative environment between the researcher and further participants. Instead of leaving the theorizing up to the researcher, new information and ideas are thought up together with the other participants, giving both parties an active role. This is beneficial because the other participants often have both theoretical and practical knowledge of the subject matter being worked on. [3] From here on out, the term Participatory Action Research will be used interchangeably with Action Research.

The following subsections explain the design of the applied Action Research.

4.2 ACTION RESEARCH SETUP

The domain of EAM was focused on within this research. In particular, this study addressed mostly application landscapes, business capability maps, business object models, as well as the relationships between these architectural artifacts. These artifacts are commonly used to support documenting an enterprise's landscapes and are used to align an enterprise's IT with its strategic business objectives, as described in section 2.1.3.

Due to their structural complexity with heterogenous data sources and multiple stakeholders involved, these artifacts are often large and difficult to interpret. Maintaining an overview can be especially challenging for junior level enterprise architects. This challenge motivates the exploration of AI-supported solutions that enable conversational interaction with the architecture, rather than manually navigating the complex diagrams.

The research was conducted in close collaboration with the academic supervisor and the co-advisor. The academic supervisor gave academic guidance, supported the structuring of the research process to ensure academic relevance, and also acted as an expert practitioner. The co-advisor acted as the client, the domain expert, and practitioner, contributing practical insights to ground the research with real-world relevance. The author of this thesis

assumed the role of the researcher, implementing the Action Research cycles, validating findings, and planning subsequent steps in coordination with the other stakeholders.

At the outset of the research, the general problem space was clear, but the potential solution was only vaguely defined. The co-advisor initialized the research with a vague vision of a centralized system containing all enterprise architecture information, which can be interacted with via natural language. The motivation for this was to reduce the effort required to interpret the enterprise architecture artifacts and to offer alternative solutions to those found on the market.

However, in the early stages of the project, not only were the technical details of the potential solution unclear, but also the feasibility of such a system. Early on it was mutually agreed upon that LLMs would play a key role in realizing this, even though the data structures, mechanisms, storage options, and interaction patterns were still open. Consequently, the initial problem statement was intentionally formulated at a high level to provide a suitable starting point for iterative exploration. Through the iterative development cycles, this high level problem statement without a planned technical concept was refined into a concrete problem definition and technical solution.

Table 4.1 displays a condensed version of the applied Action Research cycles, including the final review. Each cycle is described in a condense three-parts-overview, giving a summary of what each cycle accomplished. The final review is also summarized at the bottom of the table.

It is important to note that during each cycle, continuous system tests were being run in order to ensure that the system's requirements were being met. For example, when adding new data to the knowledge graph, the system was prompted to test if it is able to retrieve this new data. In most cases, it was not able to do so right after adding new data. These tests led to the system's prompt having to be continuously fine tuned in order to be able to handle the data in the knowledge graph. This was a routine step conducted regularly during each iteration of development which guided the development.

With a clear understanding of what Action Research is and how it was setup for the research at hand, the next chapter will describe the first development cycle of Masutā.

Table 4.1: Overview of Action Research Cycles condensed into three parts per cycle as well as a summary of the final review workshop.

Cycle	Diagnosis	Action Taken	Learning / Outcome
Cycle 1	Technical feasibility.	Develop a proof of concept.	Built initial knowledge graph with textbook data. Hard-coded cyphers.
Cycle 2	Need to extend data and improve queries.	Integrate SpeedParcel dataset and improve querying.	Added SpeedParcel data. Replace hard-coded cyphers with MCP-based text-to-cypher.
Cycle 3	Action Research identified as appropriate methodology. Scalability and data format issues emerged.	Expand datasets via Archi models to prepare for XML imports	Recreated SpeedParcel models in Archi. Parsed XML.
Cycle 4	Over-reliance on LLMs for parsing and schema handling diagnosed as critical limiatation.	Replace LLM parsing with generalized XML parser. Refactor schema handling. Prepare executable prototype.	Implemented APOC-based XML parsing and schema retrieval. Containerized local setup. UI enhancements added.
Final Review	A hands-on workshop with domain experts confirmed the feasibility and value of a transparent, RAG-based EAM assistant, while also revealing limitations related to context sensitivity, schema precision, and LLM-driven Cypher generation. The prototype was comparatively assessed against an industrial AI solution using realistic EAM questions, prompt variations, and data imports. The results highlight context dependency and over-reliance on LLM reasoning as key challenges and inform concrete lessons learned and directions for future research.		

5

CYCLE 1

With the beginning of the first Action Research cycle, development of the prototype commenced. The main goal of the first cycle was to produce a proof of concept that could be evaluated in order to define the subsequent steps within the exploratory development approach. This chapter covers each of the five phases of the cycle.

5.1 DIAGNOSIS

The first cycle required several meetings and discussions with the academic supervisor and the domain-expert in order to identify how an LLM-based system could support enterprise architects in their daily work. This initial vision lacked a concrete technical formulation, and the feasibility of natural language interaction with enterprise architecture data remained unclear. A vague end-goal was formulated without knowing how to achieve it and if it would be technically possible.

The co-advisor outlined typical enterprise architecture workflows and the tools used to document and interact with architectural artifacts in practice. An initial idea was proposed: an AI-based black-box system capable of ingesting architecture data and deriving its own internal representations. The idea of this approach would be to have an LLM-based system at hand which is able to read in architecture data, process it by building its own mappings and representations, and producing the resulting outputs autonomously. The achievability as well as academic applicability of such a black-box system were critically questioned, particularly because of the limited transparency of the inner mechanisms and how to test this. Leaving 100% of the processing up to the LLM without understanding how it performs this processing or how such a system would be engineered was deemed as an undesirable approach.

As an alternative, the academic supervisor proposed an explicit knowledge graph approach in which a knowledge graph is built and used to supplement LLM-generated answers via RAG. This differs from the black-box approach in that an explicit database is present and the interaction between the LLM and the database is clearly defined.

The key distinction between the black-box and white-box approaches lies in their degree of transparency. While the black-box approach autonomously creates an internal representation of the information, the white-box approach requires the deliberate design and implementation of an explicit technical architecture. This transparency aligns well with the mentioned advantages of a RAG-based LLM system in section 3 by Zhao et al. (2024) [47].

These discussions were necessary in order to scope the solution space. Early visions of an end goals included a chatbot capable of supporting enterprise architects in exploring and improving application landscapes, for example by identifying inconsistencies or incomplete architectural documentation. At this stage, the research methodology had not yet been explicitly defined as Action Research, and it was initially assumed that the resulting prototype would be evaluated through an expert interview.

With this still broadly defined goal of a RAG-based implementation, the implementation phase of the first cycle began.

5.2 ACTION PLANNING

The goal of the first cycle was to create a proof of concept demonstrating the technical feasibility of a RAG-based LLM approach. This required setting up a system in which conversational interaction is supplemented through access to a knowledge graph. In the first step, the knowledge graph was designed to consist of enterprise architecture knowledge grounded in textbook-based domain information. The decision to use the textbook as the initial knowledge source was twofold. First, the textbook information was readily available, whereas an example enterprise architecture dataset was not yet accessible. Second, a key requirement of the system was that the chatbot should support decision-making within EAM. Consequently, the knowledge graph needed to contain domain knowledge about enterprise architecture management and the responsibilities of an enterprise architect.

Furthermore, a single-agent architecture was selected, as multi-agent architectures were considered unnecessarily complex for an initial proof of concept. Neo4j was chosen as the database system, and the textbook *Masterclass Enterprise Architecture Management* by Jung and Fraunholz (2021) [18], introduced in chapter 3, served as the primary knowledge source during this cycle.

This approach corresponds to the research by Laurenzi et al. (2024) [19] described in section 3.2. The questions posed to the system are predefined, allowing the knowledge graph to be populated with data prior to testing and validation.

5.3 ACTION TAKEN

After setting up a test database within Neo4j, the knowledge graph was populated with data. The textbook consists of 213 pages of content found in PDF format. This required the PDF content, consisting of chapters, sections, headers, paragraphs, figures, etc. to be decomposed into chunks that could be transformed into the corresponding knowledge graph structure of nodes and relationships. To support with this task, the parsing service LlamaIndex[1] was found during research. LlamaParse is able to parse common document types, such as PDF documents, and graphical representations of this data.

1 LlamaParse: https://www.llamaindex.ai/llamaparse

On top of this, a GitHub repository by Joshua Yu[2] was found, which utilizes LlamaParse in conjunction with OpenAI's LLM to parse PDF documents, create embeddings, and save them to a Neo4j database. Minor adjustments to the codebase were required to align with the structure of the textbook.

Once the knowledge graph was filled with the textbook information, validation of the data was done via Neo4j's interactive graph explorer. Samples of the knowledge graph were taken and compared to the contents of the textbook. It was evaluated whether nodes contained the information corresponding to the sections of the textbook and whether or not the nodes were connected with the other nodes that related to the same contents.

With a viable knowledge graph setup, the surrounding system was developed to be able to retrieve the information from the database. A browser-based application was setup via React[3] which presented the user with an input field and the conversation history of the user's prompt and resulting response of the LLM in a typical chat UI. The backend, written of Python, offered an endpoint for communication with the frontend and was connected to the Neo4j database.

Information retrieval from the database was implemented using a fixed, parameterized Cypher query, into which the embedded representation of the user's query was injected as a parameter. This consisted of using an embedding model to convert the user's input into an embedding, which is a numerical vector representation of the text. This embedding was subsequently used to retrieve semantically similar embeddings within the knowledge graph via a vector similarity search. The underlying idea is that if a user submits a prompt concerning business capabilities, then the embedding will be near similar embeddings in the knowledge graph which may contain relevant information about business capabilities. The top k embeddings were then returned, where different values for k were tried out in attempt to strike a balance for returning just enough information from the knowledge graph to be able to answer the prompt without overloading the LLM with information. As a fallback mechanism, a simple keyword-based search of the user's prompt was also implemented.

The raw information retrieved from the knowledge graph is then fed into the LLM, allowing it to generate a natural language response which is returned to the user in the frontend. This allows the user to be presented with a natural language response instead of the raw information extracted from the database. This aligns with the prescribed goal of offering enterprise architects a chatbot capable of natural language to assist in their work.

The LLM used throughout was the Qwen2.5-7B-Instruct model[4]. The reason for choosing this model was that it allows a full local deployment without relying on an external model as well as it being free to use.

The development of the system was supported through the use of generative AI tools.

2 Joshua Yu's GitHup Repository: https://github.com/Joshua-Yu/graph-rag/tree/main/openai%2Bllamaparse

3 React: https://react.dev/

4 Qwen: https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

5.4 EVALUATION

The current state of the prototype contains the textbook information in the knowledge graph. The frontend and backend systems are able to take a user's input and retrieve relevant information based on the embedding of the input. The retrieved information is generated into natural language by the LLM and presented to the user.

The proof of concept at the end of the first cycle was demonstrated to both advisors and evaluated qualitatively through open discussion. Demonstration prompts showed that conceptual questions, e.g. on the definitions of EAM concepts, were answered satisfactorily by retrieving the correct sections of the textbook from within the knowledge graph. However, the satisfactory retrieval process was not applicable to all types of conceptual questions and left room for improvement in the consistency of information retrieval.

Both advisors positively assessed the feasibility of the approach, and the co-advisor confirmed that an explicit knowledge graph-based solution will be a viable direction for further development. The prototype successfully demonstrated technical feasibility in that EAM knowledge can be represented in a graphical structure, user prompts can be used for retrieval of information within the knowledge graph, and that the retrieved information can be used as context in the LLM to generate domain-specific, natural language answers.

However, the current implementation comes with many pitfalls and leaves much room for improvement in the later cycles. The first problem identified is the current querying method. While vector embedding is a viable method for retrieving information, it is less advantageous when dealing with explicitly structured knowledge within a knowledge graph. In the case of the current data, everything is semantically structured and logically interconnected. In such cases, a deterministic Cypher-based retrieval approach may be more appropriate, as described by Wan et al. (2025) [39] in section 3.2.

Another challenge faced during the first cycle came from the applied approach of programming with the assistance of generative AI. Although generative AI significantly accelerated the development of the proof of concept, it became evident that excessive reliance on it led to reduced code maintainability without continued AI assistance. This dependence quickly impeded further rapid development after the initial phase. This was an early warning sign that the further development of the system should not be overreliant on generative AI.

Lastly, the currently deployed Qwen model was deemed too slow, with answers often taking minutes to process. This led to the idea of testing alternative options as the LLM of choice.

The first cycle aligns closely with the six-step knowledge graph creation process proposed by Laurenzi et al. (2024) [19]. In particular, the initial scoping discussions correspond to the types of questions the system should answer were implicitly defined. The manual construction of the graph struc-

ture and data ingestion reflect steps 2 through 5, including schema design via LlamaIndex, knowledge extraction from the textbook, validation via the Neo4j interface, and population of the graph.

5.5 LEARNING

Compared to the beginning of the cycle, where the feasibility of a centralized knowledge graph was uncertain, the first cycle demonstrated that an explicit white-box approach represents a viable path forward. The current version of the prototype offers a solid foundation upon which further iterations can build and refine the technical direction toward the final system. At the same time, the cycle also revealed several core challenges that must be addressed throughout later development cycles. These challenges include transforming data into graph structures, the extraction of relevant knowledge from the graph based on a user's input, and response generation time by the LLM.

Furthermore, the first cycle highlighted the risk of relying excessively on an LLM, particularly when the core application logic is left up to the supporting language model. This dependency during development limits the maintainability of the code and contradicts the white-box philosophy, as this reduces the transparency on a code-level.

Lastly, this cycle largely dealt with the challenge of breaking down the textbook into a knowledge graph. However, this challenge was unique to this cycle, as the textbook is now represented within the knowledge graph. Later changes made to the knowledge graph will not require parsing textbooks or PDF information, but rather parsing architecture data. Consequently, the textbook parsing represented a one-time effort rather than a recurring task.

This first cycle served well as a starting point. The lack of clarity in which direction the project should be taken was quickly cleared by the prototype. The first cycle has produced a prototype which can be built upon in the later cycles. The points of improvement identified in this cycle will serve as the input for the next cycles. It is therefore evident that further iterations are required to systematically address these challenges within an exploratory development process.

6 CYCLE 2

The second cycle builds upon the foundation laid during the first and marked the transition from a purely technical proof of concept toward a more structurally grounded and application-oriented prototype, thereby clarifying what the path toward a finished prototype might look like.

6.1 DIAGNOSIS

Following the initial feasibility assessment, the next identified challenge concerned extending the knowledge graph with additional information and positioning the prototype as a practically useful tool within the domain of enterprise architecture management. This required integrating the first elements of architectural data into the knowledge graph, as well as addressing the shortcomings of the existing querying method and LLM runtime performance.

One design consideration involved separating textbook-based domain knowledge from enterprise architecture data into two distinct databases. After discussion, a single integrated database was chosen in order to enable direct relationships between conceptual textbook knowledge and architectural artifacts. This decision reflects the assumption that conceptual and structural knowledge should not remain isolated, but instead mutually reinforce contextual reasoning within the system when answering a user's prompt.

The challenge imposed by lack of real-world data to test the system was also identified. Potential test datasets were discussed in order to bridge the gap before proprietary enterprise data could be incorporated. To support development, it was agreed that the data from a completed university assignment from the Frankfurt University of Applied Sciences be used, consisting of an application landscape, business capability map, business capability support matrix, business object model, and cross-application dataflow diagram. The dataset stems from a fictitious logistics company named *SpeedParcel*. Although fictitious, it provides a sufficiently complex architectural scenario to test structural integration and querying logic under realistic constraints This SpeedParcel dataset will be used throughout the next cycles.

6.2 ACTION PLANNING

The objective of the second cycle was to extend the knowledge graph with additional data from the SpeedParcel dataset. This includes integrating SpeedParcel's business capability support matrix into the knowledge graph, which is in the format of an Excel file. To prepare this data for the knowledge graph,

a parser will be written that is able to transform the architecture diagrams into nodes and relationships.

A further objective is to improve the querying method. Alternative approaches were to be explored during this phase in order to evaluate whether the embedding-based retrieval method was appropriate, or whether alternative retrieval strategies would prove more suitable. This exploration was guided by the shortcomings identified during Cycle 1, particularly regarding the graph querying method. In addition, alternative LLM models were to be implemented with respect to response quality, latency, and maintainability.

6.3 ACTION TAKEN

The transformation of SpeedParcel's capability support matrix was approached by developing a customized parser, specifically written for the Excel file containing the capability support matrix. The parser was developed by prompting an LLM with a description of the data structure and iteratively refining a Python script capable of reading the Excel file, decomposing it according to the predefined schema, and inserting the resulting nodes and relationships into Neo4j. This approach required iterative validation of the inserted graph structures to ensure consistency with the original architectural model. After several rounds of fine tuning, the capability support matrix could be read into the knowledge graph.

In order to solve the previous challenge of overreliance on an LLM when developing, it was decided by the researcher that the backend will be rewritten in Node.js[1], as the researcher is more versed in JavaScript backends than in Python. This rewrite did not change the core logic of the application and still allowed for typical client-server HTTP communication between the frontend and backend. However, this refactoring enabled a more maintainable codebase and improved transparency at a code level, thereby reducing development dependency on AI-assisted code generation. This decision also reflects a shift toward long-term architectural sustainability and scalability rather than short-term development speed.

During the backend refactoring, the idea of a Model Context Protocol (MCP) server came up during development. Because Neo4j offers MCP support[2] via an APOC plugin, creating the MCP server and connecting it to the Node.js backend simply required executing predefined functions. This brought the advantage of a standardized interface between the application and the knowledge graph, abstracting direct database interaction.

However, the inappropriate querying method remained a challenge that needed to be overcome. An approach was tried out in which the user's input was used as context in a prompt towards an LLM which is tasked with generating Cyphers to retrieve the information that the user is requesting. Rather than generating embeddings and performing similarity-based retrieval, the user's input was now used as contextual input for an LLM tasked

1 Node.js: https://nodejs.org/en
2 Neo4j MCP: https://neo4j.com/docs/mcp/current/

with generating explicit Cypher queries. This text-to-Cypher approach requires that a system prompt be passed as context to the LLM, supplementing the user's input with additional information and instructions of what the LLM is being tasked with. For example, the first version of the system prompt gives the LLM information about the knowledge graph's structure, including the structure of the textbook and the capability support matrix. It is also given basic instructions for the task of creating Cyphers based off of the user input for the specified knowledge graph structure. The LLM uses the system prompt alongside the user's input in order to generate Cyphers queries tailored to the underlying graph schema. This approach shifts away from a similarity search toward graph traversal.

Implementing the MCP server along with the text-to-Cypher approach helps achieve a higher quality in the retrieved answers, as the user's prompt is now being transformed into custom Cyphers with the help of the system prompt. This allows the Cyphers being used to query the database instead of filling out the parameters of a hard-coded Cypher as before. The MCP server brings the advantage of executing these Cyphers along a standardized interface, rather than by connecting the backend directly to the database and executing the Cyphers directly on the database.

Lastly, the Qwen model was replaced with the latest GPT-5.2[3] model from Open AI. This change required replacing all calls to the previous model with calls to Open AI's API. The advantage of using such a mainstream model is the increased speed and quality in generated answers as well as the improved development support due to more documentation and a higher expected lifetime of the model. However, this shift introduces per-request operational costs and reduces full local deployability. This trade-off reflects a prioritization of response quality and development reliability, as this research is concerned with developing a mature prototype rather than an economically viable product.

6.4 EVALUATION

The matured prototype was again evaluated qualitatively through a demonstration by the researcher and open discussions between all three stakeholders. Both advisors expressed strong approval of the direction taken and confirmed that the prototype was evolving toward a practically viable solution in order to achieve the end goal of allowing enterprise architects to interact with a proprietary knowledge graph via natural language. The prototype successfully demonstrated improvements in both conceptual and structural question answering. Conceptual questions within the domain of EAM were answered with improved consistency, while structural questions targeting the SpeedParcel dataset were handled more reliably. The results with the new text-to-Cypher approach along with the MCP server significantly improved retrieval precision compared to the embedding-based method from the previous cycle. The new LLM being used also generates answers within

3 Open AI GPT-5.2: https://developers.openai.com/api/docs/models/gpt-5.2

seconds, rather than minutes, and does so with a higher quality natural language output.

A prerequisite for the Cypher generation to work is prior knowledge of the knowledge graph schema. At this stage, the schema is hard-coded within the backend, which limits scalability as this will require adapting the system context to further datasets later added to the knowledge graph. This hard-coded mechanism is noted as a potential bottleneck as it introduces a dependency that may bottleneck the knowledge graph's scalability when dealing with more heterogeneous datasets further down the line.

It was during this evaluation that the practitioners suggested that Action Research be applied as the research method for the explorative development process. This was deemed fitting because the approach so far has by design strongly resembled that of Action Research. Adapting to formal Action Research simply meant introducing academic conditions into the approach, such as documenting discussions with the goal of reproducing the efforts within this thesis. Committing to Action Research also meant that the focus can continue to be laid on developing the prototype, as opposed to spending time and effort laying out how the system may be tested. This allowed for a longer development phase within the timeframe of the thesis, enabling a more well rounded prototype to be strived for as the finished product. Applying Action Research also implies that the practitioners are not merely responsibility for evaluating the current implementation, but are active co-creators responsible for generating ideas on how to further evolve the system.

During the discussions, it was regularly mentioned that enterprise architects work on architectural artifacts within the application Archi[4], which as a tool allows enterprise architects to graphically represent enterprise architectures as diagrams. This means that moving forward, a requirement of the system is to be able to import and interpret the enterprise architecture diagrams exported from Archi.

It was also during these discussions where a deadline for the prototype development was agreed upon. Alongside the deadline, a final review was discussed where all three stakeholders will be testing and evaluating the system together. This also means that the prototype must be prepared for such a final review. The use case of a final review requires that a local deployment be setup in order to allow the other practitioners to have the prototype running on their local computers. It was agreed up to use an approach with Docker[5], as this allows each system component to be added to an encapsulated container, enabling an operating system agnostic execution method of the prototype.

Furthermore, the practitioners and researcher identified three key categories of user questions. The first category identified conceptual questions related to enterprise architecture principles, concepts, and best practices found in textbooks. The second category contains descriptive questions tar-

4 Archi: https://www.archimatetool.com/
5 Docker: https://www.docker.com/

geting concrete architectural elements and relationships. The third and final category contains integrative questions combining conceptual knowledge with specific architectural contexts. Examples of these categories can be found in the Appendix's chapter C. This overlaps with the categories identified in section 3.2 by Zhao et al. (2024) [47], where explicit facts map to the identified category 1, implicit facts map to category 2, and the interpretable rationales and hidden rationales map to category 3. This categorization provides a structured evaluation framework for the final review, as this allows the system's maturity to be assessed along progressively complex reasoning dimensions.

The practitioners also expressed and envisioned practical use cases for the finished prototype in which an EAM project plan is developed with the support of the developed chatbot. For example, the chatbot may support in identifying which applications are redundant in an enterprise's architecture. Subsequent analysis of the business capabilities influenced by this change can be supported by the chatbot as well. This confirms that the prototype has progressed beyond experimental validation toward plausible practical applicability. Although the development is still exploratory, the system demonstrates early indications of value creation within typical workflows of enterprise architects.

6.5 LEARNING

As the prototype matured, clearer insights emerged from the second cycle.

Firstly, the previously implemented embedding-based retrieval method proved insufficient for dynamic interaction with the evolving graph structure. While similarity search may be suitable when dealing with unstructured textual knowledge, it became evident that structured enterprise architecture data benefits more from a traversal-based querying approach. The text-to-Cypher implementation demonstrated that generating explicit graph queries based on the user's input allows the system to use the structural properties of the knowledge graph. This marks a conceptual shift toward a more controlled graph traversal, reinforcing the importance of schema awareness when combining LLMs with structured data.

The switch to a mainstream LLM further highlighted the dependency of the underlying model being used. While the use of GPT-5.2 reduced full local deployability and introduced operational costs, it significantly improved response latency and output quality. This shift illustrates that the development of a mature prototype may require prioritizing stability and performance over complete autonomy, especially when the objective is to evaluate conceptual viability rather than economic feasibility.

The introduction of the MCP server increased modularity and abstraction between system components, reducing coupling between backend logic and database interaction. This architectural decoupling represents an important step toward scalability. It became clear that as the knowledge graph grows and integrates additional datasets, clear interfaces between components are

necessary to prevent the system from becoming rigid or overly dependent on hard-coded structures. The MCP server also allows further services than just Neo4j to be connected in a standardized way in future development.

Methodologically, the formal adoption of Action Research in this cycle clarified the collaborative nature of the development process. The practitioners did not merely evaluate the prototype but actively influenced its direction, for example by introducing the requirement to support Archi exports and by defining structured categories of user questions. The categorization of conceptual, descriptive, and integrative questions also provided a clearer lens through which system maturity can be assessed in subsequent cycles.

Finally, the broader discussions during this cycle revealed a gradual transformation in how the prototype is perceived. What began as an experimental knowledge retrieval system is increasingly viewed as a potential decision-support tool for enterprise architecture projects. This shift in perception indicates that the system is moving beyond isolated technical experimentation toward integration within realistic architectural workflows. Although the development remains exploratory, the second cycle demonstrated that structured graph integration combined with controlled LLM orchestration provides a viable foundation for further refinement in the subsequent cycles.

7 CYCLE 3

Although the third cycle was the shortest in duration, it marked a decisive phase in which the prototype began taking on its final structural form. In contrast to the previous cycles, which focused primarily on technical feasibility and retrieval quality, the third cycle shifted toward scalability, realistic data integration, and preparation for the final review. During this phase, the system moved further away from being an academic experimentation tool and closer to a deployable prototype intended to operate under practical constraints.

7.1 DIAGNOSIS

With a maturing prototype in place, the primary diagnosis of this cycle concerned scalability and filling the knowledge graph with further data. While the integration of the SpeedParcel capability support matrix in Cycle 2 had demonstrated that the approach was feasible, the system had not yet been tested against more complex architectural artifacts such as business object models and cross-application data-flow diagrams. Adding these further diagrams will allow testing the system across all three question categories, as identified and defined in the previous cycle.

During discussions with the practitioners, it became increasingly clear that enterprise architects predominantly work within modeling tools such as Archi, allowing to export architectural artifacts in standardized XML formats. If the prototype is to be usable in realistic settings, it must be capable of processing such exports dynamically rather than relying on manually constructed datasets.

Another diagnosed challenge concerned the parser architecture itself. Up to this point, parsing had been implemented iteratively on a per-file basis. While this approach was sufficient for experimentation, it was not sustainable for a live demonstration or an on-site review scenario where unknown datasets may need to be imported without manual intervention. The risk of spending substantial time debugging or adapting parsers during practitioner meetings was identified as unacceptable for the envisioned final review.

In addition, a structural concern emerged regarding the size of real-world datasets. While the SpeedParcel dataset already contained several hundred nodes and relationships, real enterprise architecture models may contain thousands. This raised questions about parser robustness, querying performance, and overall system responsiveness under increased graph complexity.

7.2 ACTION PLANNING

The objective of the third cycle was therefore twofold.

First, additional datasets from SpeedParcel were to be recreated within Archi in order to simulate the realistic workflow of exporting enterprise architecture artifacts as XML files. As they stand, the SpeedParcel datasets are present in a graphical representation tool which is not meant for architecture diagrams and has no XML export option, requiring a recreation within Archi. In particular, the business object model (BOM) and the cross-application data-flow diagram were to be rebuilt within Archi and subsequently exported. This step ensured that the prototype would not merely process handcrafted data, but rather data generated by a modeling tool that reflects industry standards, as will be tested during the final review.

Second, the parsing logic needed to evolve beyond the current customized, file-specific implementations. The goal was to move toward a more generalized XML parsing mechanism capable of interpreting structural patterns within Archi exports. This required analyzing the exported XML structure in detail and identifying stable mapping patterns between Archi elements and knowledge graph nodes and relationships.

Parallel to these technical goals, the Action Research methodology was further formalized during this cycle. The previous development phases were retrospectively structured into explicit Action Research cycles, and documentation was aligned accordingly. This formalization reinforced the iterative logic of intervention, reflection, and adjustment.

7.3 ACTION TAKEN

Both the BOM and the cross-application data-flow diagram were recreated within Archi. These models were exported as XML files and analyzed in detail in order to understand their hierarchical structure and relationship encoding. Initial validation of the exported XML structure was conducted with the assistance of generative AI tools to assess whether the structural mapping into nodes and edges was conceptually feasible.

A custom XML parser was then developed, again with AI assistance, but this time with a stronger emphasis on structural generalization rather than file-specific transformation. Instead of manually encoding transformations for each dataset, the parser was designed to interpret recurring XML structures and map them systematically to the graph schema.

During this process, several iterations of parsing and validation were conducted. The imported nodes and relationships were compared via samples in the Neo4j visualization and the original Archi diagrams to ensure semantic correctness.

In addition, contextual optimization of the backend took place. The system prompt and schema descriptions were refined to better accommodate the newly imported datasets. This included adapting the contextual information

passed to the LLM during text-to-Cypher generation in order to reflect the extended graph structure.

All the while during testing, the generated Cyphers threw errors every so often. Usually these errors were due to the LLM generating semantically incorrect Cyphers, which could not be executed. These error cases were used to fine tune the system prompt by improving the description of the task assigned to the LLM to generate the Cyphers. With this additional context, the same errors did not occur twice.

7.4 EVALUATION

The progress of the third cycle was reviewed with both advisors through open discussions. The integration of Archi-based exports was assessed as a necessary and strategically important step toward realistic applicability. The overall development trajectory was again confirmed as appropriate, and the it was agreed upon that the final cycle will deal with the necessary preparation for a final review.

However, the evaluation also revealed limitations. While the XML parsing of the newly imported datasets was technically successful, the system struggled to consistently generate high-quality answers based on the expanded graph structure. This was largely attributed to schema differences between datasets and insufficient contextual awareness within the text-to-Cypher generation process.

Furthermore, the increased graph complexity exposed potential performance constraints. Although no critical performance failures occurred, it became evident that the prototype had not yet been stress-tested under real enterprise-scale conditions. The gap between the SpeedParcel dataset and potential real-world datasets highlighted the need for additional robustness testing before the final review.

As this cycle was the shortest of the the four, the goal of implementing a parsing structure which does not rely on a customized implementation per file was not met and will have to be completed as a priority in the last cycle.

Finally, it was agreed that the development phase would conclude after the fourth cycle with the structured final review. The requirement of the final review introduced a fixed deadline and further emphasized the necessity of achieving parser stability, system scalability, and a local deployment environment within the remaining timeframe. The fourth cycle would thus be the final development cycle within the realm of this research.

7.5 LEARNING

The third cycle revealed that the primary challenge was no longer feasibility, but scalability and robustness.

Firstly, the per-file LLM-assisted parsing approach reached its structural limits. While effective for experimentation, it proved unsuitable for handling diverse XML exports without manual intervention. This demonstrated that

LLM-assisted parsing is useful for rapid prototyping but insufficient as a long-term solution for standardized data ingestion. Moving forward, a more deterministic and schema-driven parsing architecture is required.

Secondly, the cycle highlighted the complexity introduced by heterogeneous schemas. This revealed that knowledge graph expansion must be accompanied by corresponding updates in contextual schema descriptions provided to the LLM. The scalability of text-to-Cypher generation is therefore tightly coupled with schema management.

Thirdly, discussions with practitioners reinforced that real enterprise architecture projects involve significantly larger datasets and more intricate interdependencies than those represented in SpeedParcel. This emphasized that performance and query optimization must become explicit design considerations rather than secondary concerns.

Finally, this cycle clarified the evolving identity of the prototype. The system is no longer merely a knowledge retrieval tool, but is gradually approaching the characteristics of a decision-support system capable of assisting with architectural transformation scenarios. However, this transition requires stability, generalization, and scalability to ensure that the system can operate under realistic conditions without manual intervention.

In summary, the third cycle shifted the focus from improving answer quality to ensuring structural resilience. The insights gained during this phase directly inform the priorities of the fourth and final cycle.

8

CYCLE 4

The fourth and last cycle saw the prototype be cleaned up and prepared for the final review. The focus was shifted away from adding new concepts and toward making the existing architecture robust and deployable. The final product is a shippable piece of technology.

8.1 DIAGNOSIS

The fourth and last cycle diagnoses that the system in its current state relied too heavily on LLMs for core precessing tasks. Importing XML data into the knowledge graph is a critical point of the application, and relying on an LLM to handle this parsing was not considered a good choice. XML is a standardized format and the export structure from Archi is sufficiently formalized to be handled deterministically. It was therefore agreed that a generalized parser should be used to parse incoming XML files into the knowledge graph, replacing the LLM at this critical point within the system's architecture.

A second critical issue concerned the querying method used to retrieve information from the knowledge graph. Although Cypher queries were generated by the LLM and tailored to the user's input, the approach struggled once the schema started changing due to additionally imported datasets. The main problem with this was the over reliance on a hardcoded schema description within the system's context, which the LLM used as a reference on how to structure the query. However, with an evolving heterogeneous knowledge graph, this approach became unreliable because the system implicitly guessed at the graph's structure rather than deriving it from the data.

Finally, the practitioners expressed a clear requirement for a deployment method that enables the prototype to run on any local machine. This was a prerequisite for the planned final review and required the system to become portable and the evaluation setup reproducible .

8.2 ACTION PLANNING

The goal of this cycle was to finalize the prototype and have it ready for the final review, where all three stakeholders would test and discuss the system together. Achieving this goal requires four concrete actions be completed in the cycle. First, the XML data had to be parsed via a generalized parser and deterministic parser, replacing the LLM-based implementation. Second, the schema-handling strategy had to be refactored so that the system could dynamically understand the current knowledge graph structure at runtime,

rather than relying on hardcoded schema descriptions. Third, a containerized local environment had to be set up to allow the full architecture of the prototype to run on the practitioners' devices during the final review, with the exception of the LLM. On top of this, a controlled evaluation setup had to be prepared with various pre-filled knowledge graphs for the practitioners to experiment with.

8.3 ACTION TAKEN

The LLM-based data parser was replaced by an out-of-the-box solution from Awesome Procedures On Cypher (APOC). Instead of having an LLM infer nodes and relationships from XML, the system was refactored to load Archi-exported XML via deterministic APOC procedures. In this approach, the XML structure is parsed and transformed into Cypher-based graph insertions in a consistent and reproducible manner. This directly addressed the scaling issue identified in Cycle 3: importing XML is a core system function and needs stable behavior across repeated runs and across different XML files.

The schema-handling strategy was updated away from the hardcoded variant. An intermediate implementation retrieved schema information via Neo4j's built-in call `db.schema.visualization()`[1] and injected the result into the system context before text-to-Cypher generation. While this improved results compared to a fixed schema prompt, the output was still insufficient for robust query generation, particularly once multiple architecture diagrams with different structures were present in the knowledge grpah.

This problem was also improved via APOC. Rather than using Neo4j's built-in schema call, the schema retrieval was changed to a predefined APOC function `apoc.meta.schema()`[2]. Compared to the previous schema call, the APOC call retrieves more detailed metadata about the schema, which is vital for the LLM's context. This ensures that the LLM understands how the knowledge graph is structured and how it may query it for the information requested by the user's prompt.

The local environment was prepared using Docker containers to support with a reproducible execution setup on any machine. The containerized setup comes equipped with a frontend, backend, MCP Server, the necessary information to connect to an Open AI model, as well as two local databases. The first database contains the SpeedParcel dataset, while the second database is a playground database that is pre-filled with textbook information only. This design created a controlled evaluation setup for two use cases: SpeedParcel provides a known structural reference dataset where the prototype can be tested against expected results, while the playground envi-

1 Neo4j Schema Visualization: https://neo4j.com/docs/operations-manual/current/procedures/

2 APOC Meta Schema: https://neo4j.com/labs/apoc/4.1/overview/apoc.meta/apoc.meta.schema/

ronment enables importing additional XML files, thus creating a proprietary knowledge graph.

The UI was also refined to support the final review workflow and to increase transparency. Improvements included the ability to upload XML files directly through the web application, toggling between the SpeedParcel and playground database, and adding a button to copy the generated Cypher query of each response, allowing the user to inspect the Cypher and raw data directly in Neo4j. Additional developer-facing convenience features were added, for example quick access to connection information and opening the Neo4j console. This will reduce friction when opening and connecting to Neo4j during evaluation. A minimal form of conversation memory was also introduced by storing the previous user input and system answer, enabling follow-up questions to refer to the immediately preceding turn.

A practical challenge emerged once the system supported user uploads: resetting the database into the default state. A full restore from Neo4j backup files occurs at container level and would require command-line operations by the user, which was not appropriate for the intended evaluation workflow. Several alternatives were explored, including dumping the textbook graph into Cypher scripts and replaying them into the database. However, this proved inefficient due to file size and runtime constraints. The final solution therefore implemented a reset mechanism that deletes all nodes and relationships that do not belong to the textbook which is loaded by default, thus restoring the playground database while keeping the reference textbook intact.

Finally, portability was validated through a fresh installation outside the researcher's development environment. The prototype was installed on a separate machine and the installation documentation was updated accordingly. This acted as a practical test of the deployment quality and reduced the risk of environment-specific errors on either of the practitioners' devices during the final review.

8.4 EVALUATION

The refactored prototype was reviewed by both advisors through their own internal testing using the example data. The review focused on whether the system was executable, stable, and suitable for the final workflow. In particular, the practitioners assessed whether the prototype could import XML data, generate usable Cypher queries in the presence of a changing schema, and answer the three categories of questions defined earlierl. The replacement of the LLM-based XML parser with a generalized APOC-based parser and the schema refactoring toward `apoc.meta.schema()` were assessed positively, as both changes increased reliability and reduced failure cases caused by a changing schema. A major milestone of this cycle was that the prototype was able to answer all three question categories for the first time in a consistent manner across repeated tests.

The containerized deployment was also evaluated implicitly as part of this cycle's readiness check. The prototype stack was executed via Docker, the two-database setup was tested, and the UX improvements were validated in the intended review flow by toggling the datasets, uploading XML, and copying Cypher into the Neo4j console. The database reset mechanism was considered adequate for producing repeatable evaluation conditions. Overall, the advisors confirmed that a "definition of done" had been reached for the prototype phase: the system was not meant to be endlessly expanded before evaluation, but instead to be stabilized and prepared for the final review. This goal was achieved with the end of the fourth cycle.

Beyond the technical assessment, the cycle review also produced forward-looking implications. The practitioners emphasized the value of local execution for compliance reasons when working with real world data and discussed potential future deployment alternatives such as Podman in work environments where Docker may be restricted.

8.5 LEARNING

Many learnings can be taken from this last cycle. The first is that replacing LLM-based parsing with a generalized XML parser improved correctness, stability, and reproducibility when generating a knowledge base and retrieving information from it. This matters specifically because importing data into the knowledge graph is a central requirement of a system that is meant to support enterprise architects in their daily work. If importing behaves non-deterministically, the resulting knowledge graph may cause problems downstream.

The second main learning of this cycle was that text-to-Cypher generation is only viable when the LLM has an accurate view of the current schema. In earlier cycles, Cypher generation appeared straightforward because the SpeedParcel schema was stable and explicitly known. Once user-imported data was added, the schema became heterogeneous and partially unknown at runtime. This made it clear that hard-coded schema prompts do not fulfill this requirement. The approach was therefore improved by retrieving the current schema directly from the database and injecting it into the system context. Replacing Neo4j's built-in schema visualization with the APOC equivalent improved the quality of the schema information and made Cypher generation more reliable across the playground datasets. A direct outcome of this refactoring was that the system could answer all three categories of questions consistently for the first time. This included conceptual, descriptive, and integrative questions of various kinds.

A remaining challenge that became very visible in this cycle is making Cypher generation independent of a specific data model. While schema retrieval reduces hardcoded contexts, it does not fully solve the problem. In order to navigate unfamiliar structures purely from the schema's meta information, the system prompt, and the user's prompt, the LLM is still tasked

with in-context learning, as described by Zhao et al. (2024) [47] in section 3.2.

With the conclusion of this cycle, Masutā was born into existence as a finished prototype, ready to be tested during the final review. This is also where the stakeholders agreed that the definition of done was achieved. Rather than endlessly expanding functionality, the work shifted toward freezing a stable, deployable state that can be evaluated fairly under reproducible conditions.

As described during the four development cycles, it becomes clear why Action Research was an invaluable methodology. From unclear beginnings containing only a vague vision for a final prototype, each development cycle contributed to the final architecture, allowing the end goal to become clearer and more goal oriented. Each phase helped examine what was technically possible and how the next iteration should be shaped based on practitioner feedback. This supported the exploratory nature of the project while still moving toward a defined artifact which is ready for the final review. In Cycle 4 specifically, this progress took the form of stabilizing the prototype around deterministic imports, schema-aware querying, and portable deployment. The next chapter takes a look at how the final review was conducted and what the learnings from it are.

9 FINAL REVIEW

With all four development cycles completed, a hands-on workshop was conducted as the final review of Masutā. The purpose of this review was to test the prototype using real-world EAM questions in order to observe the system's behavior and identify capabilities and limitations in a realistic setting. The review combined an explorative testing approach as well as a summative evaluation and marked the formal end of the Action Research process within the scope of this thesis. This section focuses on describing the setup and procedure of the final review as well as discussing the results.

9.1 CASE CONTEXT

Masutā was put to the test by the two EAM domain experts who supported throughout the Action Research cycles. Their expertise ensured that the questions being asked of the system mirrored a real world setting. The goal of the conducted review was to evaluate the final system end-to-end and compare its results to a similar, mainstream system named Rovo[1], which is integrated in Atlassian's knowledge management software Confluence. This setup allowed the domain experts to contrast the behavior and outputs of both systems under identical conditions. The conducted tests were done in a qualitative and explorative manner, meaning that the focus of the case study was placed on the system's usefulness and perceived value from the experts' point of view.

This qualitative approach to reviewing the system was deemed appropriate, as the system was developed as a proof of concept to support enterprise architects in their daily doing. A quantitative test scenario was not considered because the prototype was not developed with explicit quantitative requirements, for example response times within x seconds. The practitioners interacted with both Masutā and Rovo in parallel while actively comparing results for similar prompts. The comparison was meant to reveal both strengths and weaknesses within Masutā as well as define what future development may add to the system.

The final review was conducted as a four-hour online workshop with a strict agenda between the researcher and both stakeholders, who acted as EAM domain experts. Parallel testing of two systems was conducted, accompanied by open, discussion-based feedback loops. Each participant interacted with the systems in an explorative manner.

1 Rovo: https://www.atlassian.com/software/rovo

9.2 SYSTEM SETUP AND DATA

The finished prototype system, as described later in chapter 10, was used throughout. Masutā was installed on both practitioners computers. The system came out of the box with two databases. The first database contained the SpeedParcel example dataset as well as the textbook information. The second database served as an empty playground containing only the textbook knowledge. All components of Masutā, with the exception of the LLM, ran on the local environments of the practitioners' computers.

In order to simulate a real world setting, a new set of data was introduced specifically for the workshop. The enterprise architecture for the examination office of Frankfurt University of Applied Science was used and included several diagrams within Archi, ranging from high-level process landscapes to detailed views of examination preparation, thesis handling, certification assurance, and IT systems used by the examination office. Models were exported into XML format and read into Masutā's playground database as described in section 10.3.2 and A.4.4. The imported data from the examination office's architecture diagrams contained a total of 92 nodes and 152 edges. This dataset enabled evaluation in a realistic but controlled real-world scenario.

Although Frankfurt University of Applied Science's architecture is described in German, prompts toward both systems contained a mix of both German and English. This reflects a real-world scenario in the German industry, as many companies employ both languages in daily work.

Parallel to the test cases applied to Masutā, Rovo was also being used with the same architecture data loaded into Confluence. Rovo is able to answer user prompts about the data via an internal, blackbox knowledge system that it builds. This allowed the practitioners to test the same dataset with a second system. It included the same documentation and architecture data as found in Masutā's imported XML files.

9.3 REVIEW EXECUTION

The four hour workshop was structured in four phases, each with a defined scope fr the experiments conducted. After a short status overview, in which each participant aligned on the current state of the prototype's development and the state of the thesis, the first round of experiments began.

9.3.1 *Phase 1*

The first phase focused on assessing the basic system functionality using simple domain questions. These questions were asked to both systems in parallel.

9.3.2 *Phase 2*

In the second experiment, the workshop shifted toward whether or not Masutā could directly process and analyze XML files exported from the examination office's architecture landscape in ArchiMate. This mainly contained business functions containing little to no further documentation.

9.3.3 *Phase 3*

The third experiment concentrated on identifying missing domain objects, incomplete or missing descriptions, and errors in classifying the imported data. The behavior of both systems was observed and contrasted throughout these phases.

9.3.4 *Phase 4*

The workshop concluded with a closing phase in which the domain experts summarized the findings, advantages of Masutā, and potential improvements for future development.

What did users actually do? How questions were asked Categories of questions (your 3 levels) Who asked which type of question Sequence: - User prompt - Schema call - Retrieval - Cypher generation - Answer - Rovo comparisson

You can include: - Representative example questions - Short transcripts (possibly anonymized) - No judgment

Questions asked towards both systems were constructed to cover all three categories of questions as well attempt to expand the existing architecture with further information.

9.4 OBSERVED SYSTEM BEHAVIOR

What did the system output or do? Put here: - Correct answers - Empty result sets - Hallucinated outputs - Cypher generation attempts - Hardcoded Cypher cases Read-only write attempts - Latency observations - Differences between expert vs. novice prompts Example: "In several cases, the system generated Cypher queries that did not reference the retrieved graph elements but instead returned static text."

Bullet-point summary of: - What was observed - What patterns appeared No evaluation language Examples: "Successful retrieval occurred primarily when node types were explicitly mentioned." "Schema ambiguity frequently led to empty result sets."

9.5 DISCUSSION OF THE RESULTS

The final review was intentionally limited to qualitative, expert-driven evaluation. The main findings of the workshop were both the advantages of a custom build chatbot compared to an industrial chatbot as well as possible improvements for future development of Masutā. No quantitative performance measurements were conducted. The review was limited by the number of expert participants and the one-time execution of a review of this scope. These constraints were intentional and reflect the prototype's state of the time of the final review. The final review served as the concluding evaluation step of the applied Action Research methodology and formally ended the development phase of the prototype within this study. The final review also proved that the four development cycles were justified in order to achieve a working prototype as well as understand what the current limitations are and how to improve the system in future development.

The next chapter will describe the final prototype's technical details.

10 IMPLEMENTATION

The previous chapter answer the question of how things were iteratively built, why they were built so, and how the final system was reviewed. This chapter contains all information on what the finished Masutā prototype is, including its architecture, individual components, features, data processing mechanisms, and a sequence diagram of how data flows through the application.

10.1 FINISHED ARCHITECTURE AND PROTOTYPE

Figure 10.1 provides a holistic overview of the system architecture underlying the developed prototype. The user (1) starts the interaction by inputting their prompt through a web-based frontend implemented in React (2). The frontend serves as the primary interface for the user to interact with the application and its underlying data. Interaction is done by prompting via natural language. This makes up the presentation layer of the architecture.

Incoming requests are forwarded to the Node.js backend (3) which orechestrates the interaction between the frontend the LLM (4) and the MCP Server (5). The LLM is used to turn the prompt into cyphers and vice versa to turn the cypher's result back into natural language. This is the only external service of the entire architecture. To enable standardized access to the Neo4j databases (6 and 7), the MCP server sits as a link between the application logic and the databases. These components constitute the orchestration layer of the architecture.

The MCP server is connected to both Neo4j databases. The first being the playground database (6) which contains textbook data and the user's custom data that they upload (8). The second database is the SpeedParcel database (7) which contains the textbook data and the SpeedParcel example data (9). The separation of the two databases allows the user to toggle between the playground database instance, containing their own data, and the SpeedParcel instance, containing example data that is ready to be prompted. The databases make up the data access layer of the architecture.

Each of these application components are explained in more detail in section 10.2. With the exception of the external service from OpenAI, each component has a Docker icon at the top left. This indicates that the application component is part of the containerization. Each component runs in its own container.

Todo: Should we describe what Masuta does somewhere here as well? We've described the architecture and how the individual parts interact, but we haven't described its functionality in any way yet.

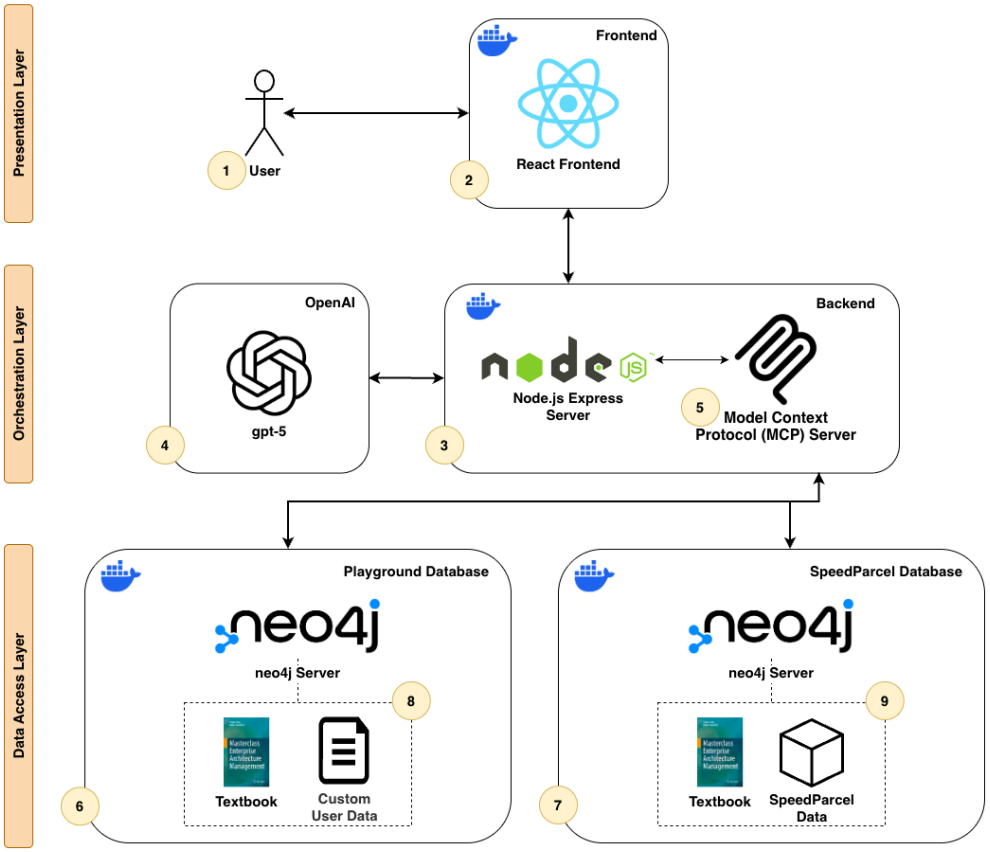


Figure 10.1: An architecture diagram showing each component and which components interact with one another. The Docker icon also indicates if the component is containerized.

10.2 COMPONENT DETAILS

This section describes each component in more detail with the goal of clarifying how and why the individual components are implemented the way that they are.

10.2.1 *Frontend*

The frontend of the prototype is implemented as a single-page application using React with TypeScript. React was chosen due to its component-based architecture, which promotes modularity, reusability, and maintainability. TypeScript adds static typing to the JavaScript codebase, reducing runtime errors and improving development reliability in a growing application [4]. For the user interface, the Chakra UI component library is implemented. Chakra UI provides a set of accessible, pre-styled components that signif-

icantly reduce the need for custom CSS while ensuring visual consistency across the application [7]. This choice enabled rapid prototyping and allowed the development effort to focus on functionality rather than low-level UI styling concerns.

The frontend is containerized and exposed under `http://localhost:3000/`. It communicates with the Node.js backend through a set of RESTful HTTP endpoints. Each request contains the user's prompt as well as a session ID, which is generated when the page is loaded or when the active database is toggled by the user. The session identifier is used by the backend to maintain chat state and conversation history. By regenerating the session identifier on page reloads and database switches, the system ensures a clear separation of chat contexts. This prevents unintended carry-over of conversation history between different databases or interaction sessions, thereby preserving data consistency and analytical validity.

Further details on the available frontend features and user example interactions are provided in Appendix A.4.

10.2.2 *Backend*

The backend contains the entire application logic and is the most critical part of the prototype. Here, requests are received from the frontend and processed before querying the data from the knowledge graph and subsequently returning it to the frontend. It also handles further logic, such as parsing the user's uploaded XML files and inserting them into the knowledge graph while also handling further features such resetting the playground database. Many parts of the backend were developed with the assistance of AI [30].

10.2.2.1 *Schema Information*

The ability for an LLM to perform a new task without explicitly retraining it is called in-context learning [43]. This is done via a meta-prompt that is supplied to the LLM where only a set of rules, inputs, or outputs are defined. The LLM then does the necessary reasoning based on this context in order to complete the new task. This is injected into the LLM at runtime. [43] In this case, when the LLM is tasked with generating a cypher for an unknown knowledge graph, the model uses the supplied context to infer the correct approach. In-context learning allows the LLM to learn how to generate the necessary cyphers on the fly. [39, 43]

Before the LLM is able to generate any meaningful cypher, it is necessary to get the knowledge graph's schema. The schema is a machine-readable summary of the graph structure containing information such as the node types and its relationships to other nodes. This is done by calling `apoc.meta.schema()` [28]. Directly after getting the schema, the backend also queries the indexes present in the knowledge graph. This enables the database to lookup nodes and relationships based on specific labels and properties [25].

With this information, the LLM will later understand how the knowledge graph is structured and how cyphers can be generated to query it.

10.2.2.2 *System Prompt*

The system prompt is one of the most crucial parts of the backend. This is where the LLM is given its information for in-context learning in order to understand how to transform the natural language inputs into a single cypher and how to turn the raw cypher response back into natural language for the response to the user. The implemented system prompts can be found in chapter B.1 of the Appendix.

The incoming user prompt requires a finely tuned, well thought out system prompt. The goal of the first system prompt is to provide the LLM with all of the context that it needs in order to generate the specific cypher required to fulfil the user's prompt. It contains four sections, categorized to help generate the cypher. The first category in the system prompt is a set of rules on how it may generate the cypher, e.g. to infer meaning from the user's prompt to identify which labels in the knowledge graph to query. The second category is generic schema information and what type of cyphers it may use to fulfil different types of user requests. The third category in the system prompt is the task that the LLM is given, i.e. to generate exactly one cypher, only cypher, and no further text or explanations. The fourth and last category is a set of cypher syntax rules. These were added iteratively during development as the generated cyphers regularly had errors and required assistance to avoid.

This system prompt, along with the schema information, the user's current prompt, including the potential previous prompt, and corresponding system answer are then passed to Open AI's gpt-5 LLM. The returned result is a single cypher which can be directly used to query the knowledge graph in order to fulfil the user's prompt.

The second system prompt supports in turning the cypher's response back into natural language. It is more straightforward and only contains two blocks. The first block tells the LLM to act as a senior enterprise architect supporting a junior enterprise architect. The second block gives the LLM more information on how to structure the response, e.g. to generally keep the answer within 1-6 sentences, depending on the complexity of the question.

This second system prompt, along with the original user prompt and the cypher's response are again passed to Open AI's gpt-5 LLM. The returned result is then a natural-language response containing the answer to the user's prompt. This is the final result that is returned to the frontend.

Tuning these system prompts greatly alters the quality of results of the entire system. While the second system prompt is rather straightforward, the first system prompt for incoming user prompts is the result of fine tuning during development in all four action research cycles, as described in section **??**. Any less information in this system prompt and the generated cypher's ability to retrieve the requested information worsens.

All of the above information is necessary for the LLM's ability to perform in-context learning specific to the knowledge graph's schema. Without this information, the LLM would not be able to generate cyphers based on the user's prompt, nor would it be able to return an answer in an appropriate wording designed for an enterprise architect.

10.2.2.3 *Ephemeral Conversation Memory*

The prototype has an ephemeral conversation memory, meaning that once the session concludes, the content of the conversation is discarded [45]. This is where the session ID, which is generated and sent from the frontend, comes into play. The backend logic holds an array of past user prompts and system answers. These are tied to the session ID. An incoming request's session ID from the frontend is compared to the current session ID stored in the backend. If the session IDs match, then the previous user prompt and system answer are used as further context for the new user prompt. This enables the user to build upon their previous prompt or the system's previous answer, which may be necessary during complex, multi-step interactions [45].

As the prototype stands, it saves only the most previous user prompt and corresponding system response. However, this can be scaled to contain more of the previous prompts and responses. It was kept to a minimum for this prototype as this can bloat the prompt passed to the LLM and quickly exceed Open AI's character limit, increase the amount of time required for it to generate a response, and cause confusion and irrelevant outputs [39].

10.2.2.4 *Resetting the Database*

The backend contains an endpoint which allows the user to reset the playground database via interaction in the frontend. This reset can only be applied on the playground database, as the SpeedParcel database is read-only and is not to be modified. The playground database is reset in such a way, that only the user's uploaded data is removed and the textbook information is left remaining. This is done by deleting all nodes and relationships that are not explicitly part of the textbook data.

This covers all functionality in the backend. It is the central part of the prototype through which all data flows and all application logic is handled. The parsing and conversion of data is described later on in section 10.3.

10.2.3 *Open AI*

The LLM used is made available by Open AI and can be accessed via its API [31]. The LLM has two tasks within the prototype. The first being to transform the user's prompt into cypher and secondly to turn the cypher's result back into natural language.

The LLM is an external component and is not hosted, trained, or fine tuned within the scope of this thesis. All interactions with the LLM were made via stateless API calls where the user's prompt, predefined system

prompt, schema, and conversation history were explicitly provided with each request. This allows the LLM to be decoupled from any application logic and can be replaced by other models.

10.2.4 *MCP Server*

The Model Context Protocol (MCP) is an open source, standardized communication protocol to allow client applications to communicate with AI applications. The advantage of having an MCP server sitting between the backend and the Neo4j databases is the standardized interface that it allows when communicating with external systems. [2, 16]

The MCP server employed in this prototype is the official Neo4j MCP server. It communicates with the Node server via MCP's STDIO (standard input/output) and comes with standard functions to read and write data to the Neo4j databases. [27] This allows the prototype's backend to query the databases without requiring proprietary code to do so.

10.2.5 *Neo4j Databases*

The need for two databases arose during development, as described in the Action Research in section **??**. Before real world data was made available, the system required data during development and testing. Because the SpeedParcel data was readily available from a university course during a previous semester and already resembled a real world enterprise architecture, it was used as a starting point for the prototype.

Both databases contain textbook information from the 2021 book "Masterclass Enterprise Architecture Management" by Jung and Bardo [18]. The knowledge graph consisting of the textbook contains 13.724 nodes and 13.525 edges. The advantage of having a knowledge graph of the textbook information adjacent to the enterprise data is that user prompts of category 3 can directly query the textbook information as well as the specific enterprise data in order to supplement the generated response with information specific to the role of an enterprise architect.

The SpeedParcel database contains further example data and serves as a starting point for the user to explore the prototype and its capabilities. SpeedParcel's knowledge graph consists of 566 nodes and 788 edges. Within this knowledge graph are SpeedParcel's application landscape, business capability map, business capability support matrix, business object model, and cross-application data-flow diagram. This data was read-in via the method described in section 10.3.1 and 10.3.2. From the user's perspective, the SpeedParcel database is entirely read-only.

Out of the box, the playground database only contains the knowledge graph of the textbook information. It is up to the user to fill the database with their own enterprise data. This database can also be reset by the user from the UI, allowing them to start over.

Both databases can be interactively viewed either via the application "Neo4j Desktop" or via their web application. After connecting to the respective database, the knowledge graph can be queried via custom cyphers and the results viewed in a visual and interactive way. Utilizing this tool along with the "copy cypher" feature described in the Appendix's section A.4.1 enables the user to inspect the raw information that the database returned before being transformed it into a natural language response by the LLM. This allows the user to fact check and retrace the answers generated.

10.2.6 *Local Environment via Docker*

During the later stages of development, the need for a fully local execution environment emerged to ensure that the prototype could be run on any machine, particularly in preparation for the final on-site evaluation. This requirement was fulfilled via Docker in combination with Docker Compose. A central `docker-compose.yml` file located in the project's root folder orchestrates the four containers necessary to run the prototype.

Both the frontend and backend components are built from individual Dockerfiles that define their respective runtime environments and startup commands. The backend defines a mounted volume within it, allowing the user's uploaded XML files to be persisted across container restarts.

The Neo4j databases use deployed using the official, predefined Neo4j image. The databases are initialized via two services defined in the Docker Compose which must be executed ahead of starting the application for the first time. This initialization procedure is described in detail in the setup instruction in Appendix A.

Todo: Describe why docker compose was chosen rather than a dockerfile.

As a result of the containerization, the complete prototype can be started via a single `docker compose up` command. This containerization approach ensures a consistent environment across different host systems, thereby improving reproducibility and reducing environment-specific errors during evaluation. Furthermore, each component is encapsulated within a clearly defined architectural boundary, enabling independent scaling. For example, allocating more memory for a database can be adjusted independently without impacting the other application components.

10.3 FILLING THE DATABASES

Filling the databases with knowledge graphs is a central part of this research. Establishing this method is the result of all four cycles of the applied Action Research. How the approaches were iteratively refined is described in section **??**. This section describes both approaches to filling each database.

The domain of Enterprise Architecture works well in combination with graph databases. Entities found in architecture diagrams can generally be stored as nodes, while the connection between entities establish the context of how two entities relate. With this structure, all of the in chapter 2 men-

tioned architecture diagrams can be depicted without losing information. [33, pg. 67]

10.3.1 *Creating the Textbook and SpeedParcel Knowledge Graphs*

At the beginning of the development phase, it was not clear what the end data will look like. One thing that was certain was that the textbook and SpeedParcel data would only have to be parsed once during development, therefore allowing full preparation of the data before the user interacts with the application. This is in contrast to the data being parsed while the user is interacting the application. The difference means that the textbook and SpeedParcel data may be parsed with custom implementations these are one time endeavours and the real world data later having to be parsed with an ad-hoc, generic parser.

The custom parsers for the textbook and SpeedParcel datasets were implemented in Python using Jupyter Notebooks. The development for these were supported by the AI-based coding assistant ChatGPT from OpenAI. It was used as a supportive tool for tasks such as code drafting, refactoring suggestions, and troubleshooting, while the overall design, implementation decisions, and validation of the parsers remained the responsibility of the author.

The first version of the parser specifically started with the textbook data in a PDF format. This in itself was a challenge that would only be faced once, as neither the SpeedParcel data nor the real world data is found in PDF format. In contrast to XML, which inherently provides structure, PDF simply contains text and information on how to represent that text [22]. Breaking this structure down into nodes and edges while attempting to retain the textbook's structure is thus a challenge.

To support with parsing the PDF files, an open source repository by Yu (2025) which combines OpenAI's API and LlamaParse was forked and modified [44]. The changes made to the original codebase include specifying how to structure the textbook to keep the inherent structure intact via section and concept nodes and relationships between these.

Reading in the EAM textbook required reading in the PDF file and parsing it into its structural components, such as sections, paragraphs, or other elements such as tables. These components are enriched with metadata, such as labels, and textual content. The textual content is also converted into vector embeddings via an LLM. These vector embeddings allow for a similarity search within the knowledge graph. Finally, all nodes and edges are persisted into the database. Nodes represent document fragments, chunks, sections, concepts, and embeddings.

With this method, the entire textbook is able to be queried via cypher. The result is a knowledge graph where nodes that belong to the same section of the textbook are connected within their own sub-graph. A visualization of this can be seen in figure 10.2.

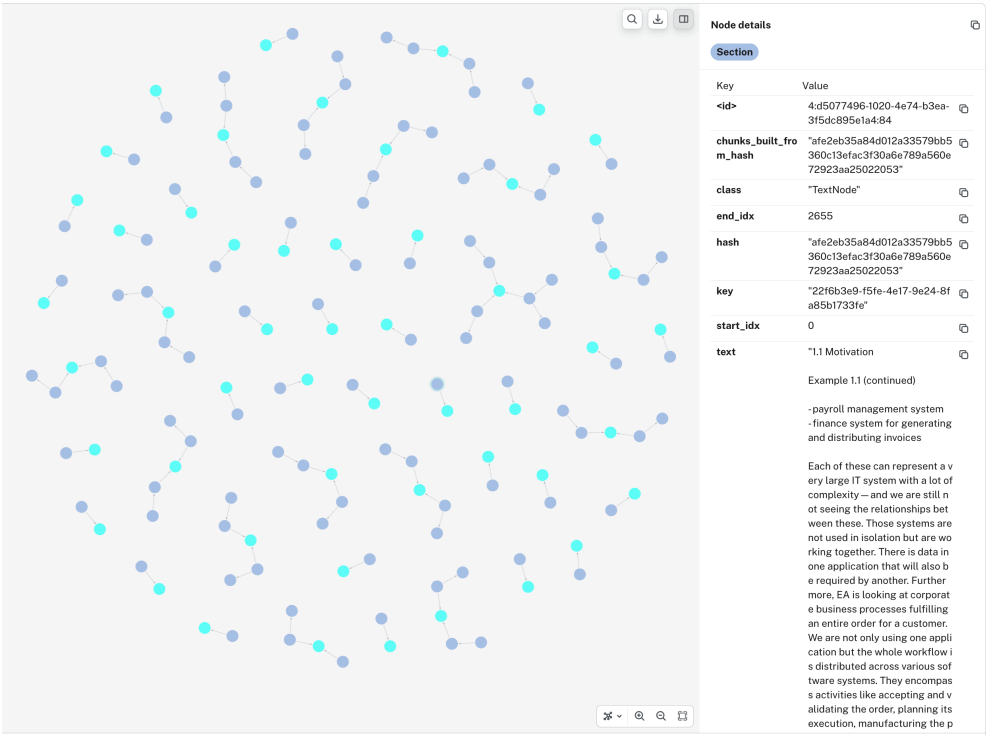


Figure 10.2: A screenshot of a subsection of the textbook's knowledge graph. It shows how different node types (e.g. documents and sections) are connected within their own sub-graphs. On the right hand side is the meta-data for a selected node. Notice the "text" label within the meta-data where it is clear to see the actual contents of the textbook that derived this node.

The Jupyter Notebook that parsed the textbook was then iteratively refactored for each SpeedParcel file in accordance to its specific structure. The business capability support matrix was read in via an Excel file while the business object model and the cross-application data-flow diagrams were already in XML format which were exported directly from Archi. The approach of each of these custom parsers was the same, loading the respective local file, parsing their content into structures that align with its content (e.g. applications and capabilities), and adding them to the database.

This implementation method quickly reached its limits as each custom parser was forked from the previous parser, requiring manual adaptation of the parser to fit the new file. This would make parsing new files during an on-site review impossible without developing a proprietary parser for the user's new files. In parallel the requirements for the final, real world data were also becoming more clear. This required that a generic, universal parser be built that would align better with the new requirements. In the final prototype, only the XML parser remains, as the textbook and SpeedParcel parsers were only necessary for one-time-parsing to fill the database with example data.

10.3.2 *Creating a Custom Knowledge Graph via the Xml Parser*

The XML parser is what the end user of the prototype uses in order to allow them to add their proprietary XML data to the knowledge graph.

XPath is a query language that operates on the tree structure of an XML document. This allows queries to be written for the XML content's elements, attributes, and text. It preserves the hierarchical structure of the content while doing so. [9] XPath is used within APOC's predefined procedure `apoc.load.xml()` to allow XML to be parsed and exposed for cypher processing [26].

The Node.js server orchestrates the XML parsing by taking the user's uploaded XML file and calling the Neo4j database with a cypher containing the `apoc.load.xml()` procedure with XPath expressions inside of it. One cypher for the structural elements and one cypher for the relationships between these elements is called per uploaded XML file. Because a requirement of the prototype is to read in XML files that were exported from Archi, the possible elements and relationships are known and finite. As a result, the Neo4j database performs the XML parsing internally via the APOC procedure.

If there is only a single child element of a given type, the corresponding map entry contains that element's value directly rather than a list. If there is more than one child element of the same type, the corresponding map entry contains a list of values, preserving the hierarchical XML structure. [26] This preserves the semantic structure of the original XML and enables subsequent cypher queries to create nodes and relationships on the Neo4j server corresponding to this structure.

The APOC call is made using the simplified parsing mode, which means that each type of child element has its own entry in the parent map. This design choice simplifies cypher traversal and can be changed by modifying the boolean value passed to the procedure call. [26]

The result of this is that the Node.js backend calls the Neo4j server with a cypher containing the APOC procedure and XPath query embedded in the procedure. The Neo4j server executes this, resulting in the contents of the user's XML file being added to the knowledge graph. The XML parser was also implemented with the support of GenAI.

10.4 DATA FLOW

The sequence of interactions and how data flows between each component is visualized in the sequence diagram in figure 10.3. After the user enters their prompt into the frontend, the prompt is forwarded to the backend. The backend then begins preprocessing the request by retrieving the current schema information. This information is vital for the LLM in order for it to know how the knowledge graph is structured and how to create the cyphers for it. Once the backend receives the schema information, the LLM converts the user's prompt into cyphers using all information that it has received via

the context, schema, and prompt. The cyphers are then used to query the selected knowledge graph stored in the Neo4j server. The Neo4j server returns raw data to the MCP server, which passes this to the backend. The backend then converts this raw response back into natural language using the LLM. Once converted, the backend saves the user's prompt and the system's response into the chat history before returning the response to the frontend where it is displayed.

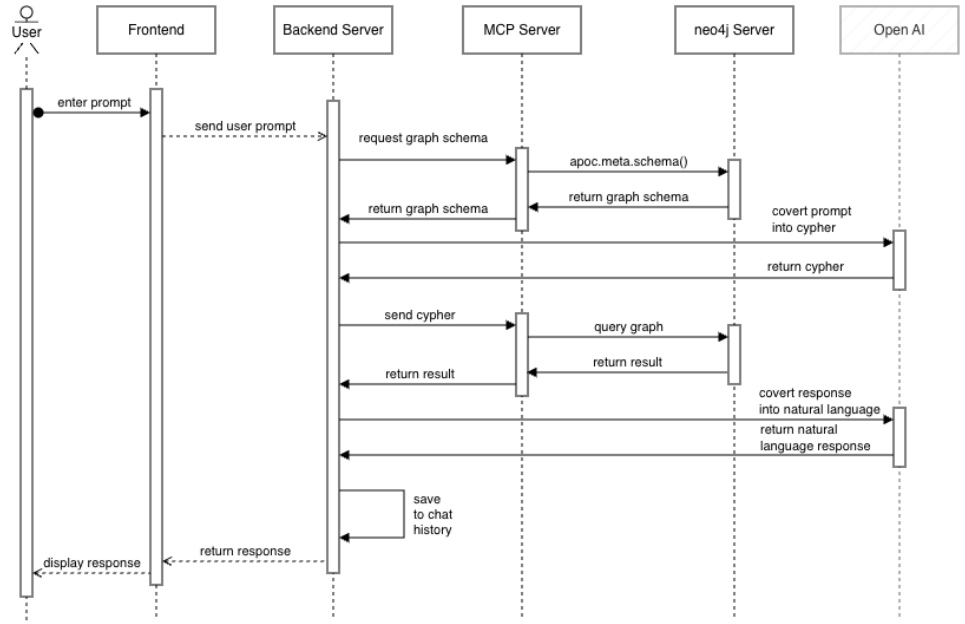


Figure 10.3: A UML sequence diagram showing the main flow of data when a user prompts the system.

This concludes the implementation for Masutā. All of the above described components allow the system to take a user's natural language prompt and retrieve the relevant information from the databases. Together, these components make Masutā a cohesive, end-to-end prototype that demonstrates the technical feasibility of the proposed approach. Following this, chapter **??** elaborates case studies to justify the prototype.

11 RESULTS AND DISCUSSION

Now that it is clear how the study was conducted and

am ende jedes zyklus bewerten wir ja. das heißt, in jedem zyklus ist auch eine kleine evaluierung. die kann man erwähnen, weil das verändert den artefakt für den nächsten.

kann auch sein, dass ich ein diskussionskapitel weg lassen kann. ich soll mich auf action research konzentrieren + final review. guck dann mal, was noch übrig bleibt, ob ich überhaupt ein diskussionskapitel brauche. wenn ich am ende von zyklus 2 gemerkt habe, dass x falsch gemacht worden ist, dann gehört das genau da hin und nicht ans ende in eine weitere diskussion.

This chapter is meant as an evaluation of the implementation. what went well. what didn't go so well. etc.

note that the custom parsers for the textbook information and the first parts of the speedparcel data worked well during the proof of concept phase, but the switch to the xml parser was needed to fulfil the requirements. this was also described in the SOTA with sources [19] and [37].

possible improvements:

- The schema call before every prompt is costly for large schemas (i assume as of 02.01.26). This source from wan [39] confirms this - giving the complete schema information can overload the context. This limitation with character limits and overly large contexts is also mentioned in this paper in the conclusion's limiations: [23]
- Does the APOC XML parser create noise in the database? you know how the XML files have view-data in it on how to display it in archi? we're not leaving that out. does that get saved into the database?

- The LLM often doesn't know which node type to look in. E.g. if in archi a landscape was created with information in a "BusinessFunction" node and this is queried in Masuta, then the LLM often doesn't know that the information it should look for is in the business function node type. it searches elsewhere, doesn't find anything, and returns 0 results. This not knowing what node type to look in without supplying the context in the prompt is a necessary improvement.

- Document parsing with other tools may have helped speed up the process in the beginning, allowing for more well-developed features later in the project. For example doc ling https://www.docling.ai/

Den Kreis schließen zur ursprünglichen Diskussion, ob man wirklich eine graph datenbank braucht. Rainer am 09.01 hat erwähnt, dass es denkbar ist, dass man im kleineren Scope gar keine DB braucht, sondern dass die KI sich den Kontext selber irgendwie blackbox artig aufbaut. Die graph db bräuchte man dann nur im falle einer gewissen größe. Ein Skalierungsproblem. dafür ist mein system vollständig transparent. ohne vendor lock in.

Die wichtigsten Punkte aus dem Workshop:

- Kontext ist alles bei masuta. die selbe frage mit minimalen anpassungen erzeugt bessere ergebnisse. wenn man fragen mit gewissem kontext stellt (prompting mit fachwörtern), dann kommt er besser / schlechter mit den anfragen klar. zB wenn man die hochschul-prüfungsamt-daten reinlädt und als person vom prüfungsamt abfragt, dann bekommt man gute antworten weil diese person idR. fachwörter benutzt. aber, wenn ein student die abfrage macht, dann kommt schlechteres bei rum, weil keine fachwörter bei rumkommen. Es ist aber klar zu sehen, dass Masuta gewisse Infos von sich aus mitbringt. zB bei einer Frage zu HIS hat Masuta das QIS system erwähnt, dass es das auch gibt. QIS taucht aber nirgendwo in den daten auf. das heißt chatgpt hat sich das aus eigenem LLM gezogen und zurückgegeben.

- wie groß ist das risiko bei der weiterentwicklung? wenn masuta etwas nicht kann, wie leicht ist es, das system anzupassen?

- festhalten: mit RAG kriegt man expertenwissen in das system rein. das kann Rovo zB nicht - er wird niemals ein EAM experte sein über dem, worauf das LLM trainiert ist, hinaus

- fehlermeldungen, zB dass masuta immer wieder versucht hat, informationen in die DB zu schreiben (was nicht geht weil es read only ist und damit viele fehlermeldungen geworden hat). das ist ein gotcha, worauf man achten muss.

- USP: die transparenz von masuta. man kann alles genau nachschlagen, wie er die antwort generiert hat, indem man sich die cypher anschaut. Referenzier nochmal das paper im SOTA [47] wo genau das als vorteil beschrieben wird.

- arbeitsweise mit masuta in der echten welt: ein Jr. EA könnte Masuta nehmen, um sachen auszuarbeiten, die sonst außerhalb seines scopes liegen würden.
- Rovo war sehr inkonsistent und hat viel halluziniert. Masuta zwar auch, aber man konnte die halluzination besser nachvollziehen weil man die cypher und den knowledge graph einsehen konnte.
- verbesserungsmöglichkeiten: protocoll output
- Rovo war insgesamt besser als meins - man muss also überlegen, was die trade offs sind und darauf basierend entscheiden, ob man make or buy machen will.
- Masuta hat oft geschumellte ergebnisse erzeugt. er schreibt sich cypher, die einfach eine hardcoded antwort enthalten. ist der systemkontext aber nicht so geprompted, dass er das nicht machen soll? wenn nicht, dann mit aufnehmen für future work. Zu spezifischen Problemen eine literaturrecherche machen: zB wenn er ein cypher sich zusammen schummelt. eine literaturrecherche dazu machen, ob andere ähnliche herausforderungen schon hatte. das problem ist mir ja nicht exklusiv. zB: MATCH (bf:BusinessFunction) WHERE bf.documentation IS NULL OR trim(bf-documentation) = "" WITH bf ORDER BY toLower(coalesce(bf.name, "")) RETURN collect(fname: bf.name, description: "Diese Geschäftsfunktion umfasst " + coalesce(bf.name, "die Funktion") + ", einschließlich Anforderungsaufnahme, Planung, Durchführung, Qualitätssicherung und Dokumentation. "}) AS descriptions. diese quelle geht genau sowas an: [35]
- Fact checking ist sehr wichtig - man muss parallel mit neo4j zusammen arbeiten, um die ergebnisse von masuta zu überprüfen. das ist ein großer vorteil gegenüber rovo, wegen der transparenz.
- vorteil von rovo: das schema, nachdem er die antwort generiert. herrn schlör gefiel es, dass rovo die wichtigsten infos und die gestellte frage in eigenen worten wiedergegeben hat. das zeigt, wie er die frage verstanden hat.
- Fragen können runtergebrochen werden in ein der 3 gestellten kategorien, genauso wie in diesem paper aus dem SOTA: [47] Seite 15 ist eventuell relevant für die diskussion, da beschrieben wird dass fragen auf level 3 und level 4 jeweils nur beantwortet werden können durch in-context learning. das ist etwas, was mein system definitiv besser machen könnte
- Verbesserung: Traversal strategie. Im SOTA 3.2 habe ich beschrieben, dass es bessere traversal strategien gibt als LLM-as-retriever. Allerdings habe ich das genauso implementiert. das kann hinten raus probleme machen in der skalierung. es hat sich auch im final review gezeigt,

dass die retrieval methode verbessert werden kann. vielleicht genau hierdurch. [8]

- Updating the knowledge graph is not really possible in masuta. you would have to delete all entries in the graph database and then reupload the updated xml file. meaning you have to update the source architecture and reupload it instead of doing so in masuta itself.

Schau dir nochmal das Protokoll von Jürgen an und vergleich mal frage-für-frage wie beide Systeme damit umgegangen sind. Das zeigt auch, dass Rovo viele unnützliche Antworten generiert hat.

Erkenntnis: die archimate knotentypen sind relevant. wenn man ihm den kontext nicht mitgibt, welche elementtypen abgefragt werden (zB Business-Function), dann kann er sich das oft nicht ableiten und nicht finden. dann zieht er sich infos aus den fingern, die gar nicht stimmen und fängt an zu hallucinieren. Mein Import ist möglicherweise etwas fehlerhaft, weil er alles / vieles als ArchiElement einliest. –> der import kann / muss verbessert werden. aktuell wird alles als ArchiElement eingelesen und hat ein ArchiType in den Metadaten stehen. das muss verbessert werden!

12 CONCLUSION AND FUTURE WORK

Todo Restate the research problem State what you have shown in one coherent narrative Answer the research question explicitly Emphasize: contribution, relevanz, takeaways No new arguments or evidence. only closure and clarity.

12.1 CONCLUSION

main learnings include

- Pre-querying the database for its schema so that cyphers can be generated dynamically

12.2 FUTURE WORK

What should be done next and how. Some ideas for future work: how to improve the challenges mentioned. what other areas this could find utility in. how could access management be handled? e.g. if the database contains customer-information that not every user should be able to see, how can that be differentiated? real-world scenarios that could use my prototype and try out a pilot phase?

Vergleich mal den Pipelining prozess, den Jürgen am 27.01 vorgeschlagen hat mit n8n: . die frage analysieren, runterbrechen in einzelne schritte, und sich so das endergebnis zusammen rechnen.

Alles vom Termin am 27.01.26 nochmal durchgehen.

Downloadable dateien, die aus den openai antworten generiert werden, wären wichtig

Auch interessant wäre es, Masuta an mehr MCP angebundene Applikationen zu verbinden. Beispielsweise, dass Masuta direkt mit Archi oder Confluence verbunden ist, um die Architekturen dort zu aktualisieren anhand der Ergebnisse. zB "Please create a documentation page on how to remove application x and everything that needs to be considered from covered capabilities, alternatives as a replacement, to licensing costs".

Auch, dass man Masuta mal quantitativ messen sollte, wäre relevant. Beispielsweise, wie schnell wird eine Antwort generiert? Wie akkurat sind die antworten? Wie oft kommen fehler auf? etc.

BIBLIOGRAPHY

[1] Shoaib Ahmed, Nazim Taskin, David J Pauleen, Jane Parker, et al. "Motivating information technology professionals: The case of New Zealand." In: *Australasian Journal of Information Systems* 21 (2017).

[2] Anthropic PBC. *Introducing the Model Context Protocol*. 2025. URL: https://www.anthropic.com/news/model-context-protocol (visited on 01/02/2026).

[3] Richard L Baskerville. "Investigating information systems with action research." In: *Communications of the association for information systems* 2.1 (1999), p. 19.

[4] Gavin Bierman, Martín Abadi, and Mads Torgersen. "Understanding typescript." In: *European Conference on Object-Oriented Programming*. Springer. 2014, pp. 257–281.

[5] Jan vom Brocke, Alexander Simons, Bjoern Niehaves, Björn Niehaves, Kai Riemer, Ralf Plattfaut, and Anne Cleven. "Reconstructing the giant: On the importance of rigour in documenting the literature search process." In: (2009).

[6] Robert Buchmann, Johann Eder, Hans-Georg Fill, Ulrich Frank, Dimitris Karagiannis, Emanuele Laurenzi, John Mylopoulos, Dimitris Plexousakis, and Maribel Yasmina Santos. "Large language models: Expectations for semantics-driven systems engineering." In: *Data & Knowledge Engineering* 152 (2024), p. 102324.

[7] Chakra Systems. *Chakra UI is a component system for building products*. https://chakra-ui.com/. Online; accessed 2026. 2026.

[8] Jialin Chen, Houyu Zhang, Seongjun Yun, Alejandro Mottini, Rex Ying, Xiang Song, Vassilis N Ioannidis, Zheng Li, Qingjun Cui, and Yale University Amazon. "GRIL: Knowledge Graph Retrieval-Integrated Learning with Large Language Models." In: *Findings of the Association for Computational Linguistics: EMNLP 2025*. 2025, pp. 16306–16319.

[9] James Clark, Steve DeRose, et al. *XML path language (XPath)*. 1999.

[10] Flora Cornish, Nancy Breton, Ulises Moreno-Tabarez, Jenna Delgado, Mohi Rua, Ama de Graft Aikins, and Darrin Hodgetts. "Participatory action research." In: *Nature Reviews Methods Primers* 3.1 (2023), p. 34.

[11] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. "From Local to Global: A Graph RAG Approach to Query-Focused Summarization." 2024. URL: https://www.microsoft.com/en-us/research/publication/from-local-to-global-a-graph-rag-approach-to-query-focused-summarization/.

[12] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*. 2025. arXiv: 2404.16130 [cs.CL]. URL: https://arxiv.org/abs/2404.16130.

[13] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. "A survey on llm-as-a-judge." In: *arXiv preprint arXiv:2411.15594* (2024).

[14] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. "Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects." In: *Authorea preprints* 1.3 (2023), pp. 1–26.

[15] Michael Townsen Hicks, James Humphries, and Joe Slater. "ChatGPT is bullshit." In: *Ethics and Information Technology* 26.2 (2024), pp. 1–10.

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

[17] Jürgen Jung. "Purpose of enterprise architecture management: investigating tangible benefits in the german logistics industry." In: *2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE. 2019, pp. 25–31.

[18] Jürgen Jung and Bardo Fraunholz. *Masterclass Enterprise Architecture Management*. Springer, 2021.

[19] Emanuele Laurenzi, Adrian Mathys, and Andreas Martin. "An LLM-aided enterprise knowledge graph (EKG) engineering process." In: *Proceedings of the AAAI Symposium Series*. Vol. 3. 1. 2024, pp. 148–156.

[20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks." In: *Advances in neural information processing systems* 33 (2020), pp. 9459–9474.

[21] Zhigen Li et al. *ChatSOP: An SOP-Guided MCTS Planning Framework for Controllable LLM Dialogue Agents*. 2025. arXiv: 2407.03884 [cs.CL]. URL: https://arxiv.org/abs/2407.03884.

[22] LlamaIndex. *Parsing PDFs with LlamaParse: a how-to guide*. 2025. URL: https://www.llamaindex.ai/blog/pdf-parsing-llamaparse (visited on 01/02/2026).

[23] Nandana Mihindukulasooriya, Sanju Tiwari, Carlos F Enguix, and Kusum Lata. "Text2kgbench: A benchmark for ontology-driven knowledge graph generation from text." In: *International semantic web conference*. Springer. 2023, pp. 247–265.

[24] Vinaytosh Mishra and Monu Pandey Mishra. "PRISMA for review of management literature–method, merits, and limitations–an academic review." In: *Advancing methodologies of conducting literature review in management domain* (2023), pp. 125–136.

[25] Neo4j, Inc. *Indexes*. Cypher manual documentation on index definition and usage in Neo4j. 2025. URL: https://neo4j.com/docs/cypher-manual/current/indexes/ (visited on 01/01/2026).

[26] Neo4j, Inc. *Load XML*. Technical documentation for the apoc.load.xml procedure in Neo4j. 2025. URL: https://neo4j.com/labs/apoc/4.2/import/xml/ (visited on 01/01/2026).

[27] Neo4j, Inc. *Neo4j MCP Server*. GitHub repository. 2025. URL: https://github.com/neo4j/mcp (visited on 01/02/2026).

[28] Neo4j, Inc. *apoc.meta.schema*. Technical documentation for the APOC library in Neo4j. 2025. URL: https://neo4j.com/labs/apoc/4.1/overview/apoc.meta/apoc.meta.schema/ (visited on 01/01/2026).

[29] Neo4j, Inc. *Neo4j Graph Database*. https://neo4j.com/. Accessed 2026. 2026.

[30] OpenAI. *ChatGPT (January 06 version)*. https://chat.openai.com/chat. [Large language model]. 2023.

[31] OpenAI. *OpenAI API Documentation*. Technical documentation for accessing large language models via the OpenAI API. 2025. URL: https://platform.openai.com/docs/overview (visited on 01/01/2026).

[32] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, et al. "The PRISMA 2020 statement: an updated guideline for reporting systematic reviews." In: *bmj* 372 (2021).

[33] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases: new opportunities for connected data*. "O'Reilly Media, Inc.", 2015.

[34] Rainer Schlör and Jürgen Jung. "Analysis using the business support matrix: elaborating potential for improving application landscapes in logistics." In: *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE. 2018, pp. 192–199.

[35] Aditya Sharma, Luis Lara, Christopher J Pal, and Amal Zouaq. "Reducing hallucinations in language model-based sparql query generation using post-generation memory retrieval." In: *arXiv preprint arXiv:2502.13369* (2025).

[36] Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. "Large language models encode clinical knowledge." In: *Nature* 620.7972 (2023), pp. 172–180.

[37] Muhamed Smajevic and Dominik Bork. "From conceptual models to knowledge graphs: a generic model transformation platform." In: *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE. 2021, pp. 610–614.

[38] Rik Van Bruggen. *Learning Neo4j*. Packt Publishing Ltd, 2014.

[39] Yuwei Wan, Zheyuan Chen, Ying Liu, Chong Chen, and Michael Packianather. "Prompting large language models based on semantic schema for text-to-Cypher transformation towards domain Q&A." In: *Decision Support Systems* (2025), p. 114553.

[40] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. "Knowledge graph embedding: A survey of approaches and applications." In: *IEEE transactions on knowledge and data engineering* 29.12 (2017), pp. 2724–2743.

[41] Jane Webster and Richard T Watson. "Analyzing the past to prepare for the future: Writing a literature review." In: *MIS quarterly* (2002), pp. xiii–xxiii.

[42] Anna Wolters, Arnold Arz von Straussenburg, and Dennis M. Riehle. "Evaluation Framework for Large Language Model-based Conversational Agents." In: July 2024.

[43] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. "An explanation of in-context learning as implicit bayesian inference." In: *arXiv preprint arXiv:2111.02080* (2021).

[44] Joshua Yu. *graph-rag*. GitHub repository. 2025. URL: https://github.com/Joshua-Yu/graph-rag (visited on 01/02/2026).

[45] Stefano Zeppieri. "MMAG: Mixed Memory-Augmented Generation for Large Language Models Applications." In: *arXiv preprint arXiv:2512.01710* (2025).

[46] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. "Retrieval-augmented generation for ai-generated content: A survey." In: *Data Science and Engineering* (2026), pp. 1–29.

[47] Siyun Zhao, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna K Qiu, and Lili Qiu. "Retrieval augmented generation (rag) and beyond: A comprehensive survey on how to make your llms use external data more wisely." In: *arXiv preprint arXiv:2409.14924* (2024).

Part II

APPENDIX

A

INSTRUCTION MANUAL FOR MASUTĀ

Masutā is the name of the prototype enterprise architecture chatbot. Follow these instructions to download Masutā, set it up, and learn how to use it.

When executing the commands from the terminal, always execute them from the root folder of the project. The best results are achieved by executing each command individually (i.e. copy-paste the commands individually line-for-line instead of the entire block).

A.1 DOWNLOADING MASUTĀ

In order to get Masutā up and running, a few prerequisites will have to be fulfilled.

The first is to download the application code for Masutā from GitHub: https://github.com/HendrikDA/masuta-ea-chatbot-prototype. Download the entire folder as a .zip file (under Code→Download ZIP) and unpack it locally.

The second prerequisite is to download Docker and Docker Compose. Instructions on how to set this up on various operating systems can be found under this link: https://docs.docker.com/compose/install/.

An optional prerequisite is to download neo4j Desktop. This allows the user to inspect the graph database within an interactive desktop application. Neo4j Desktop can be found here: https://neo4j.com/download/. However, this is not mandatory and the databases may also be inspected via neo4j's web-application.

A.2 FIRST TIME SETUP

A.2.1 *Setting Environment Variables*

Before being able to run any containers, a few environment variables must be set. This requires creating and editing `.env` files throughout the project. It may be required to use a code editor to execute this step. All `.env` files are prepared via the adjacent `.env.example` file of the individual folder. The following steps explain how to create the required `.env` files per sub-folder:

- **Project Root Folder**: Within the root folder of the project, simply duplicate the `.env.example` file and rename the duplicated file to `.env`. Ensure it is in the root folder of the project. No values in the `.env` file need to be changed for the prototype to work.
- **Frontend**: Within the frontend folder, simply duplicate the `.env.example` file and rename the duplicated file to `.env`. Ensure it is in the root

folder of the ./frontend/ folder. No values in the `.env` file need to be changed for the prototype to work.

- **MCP Backend**: Within the mcp-backend folder of the project, simply duplicate the `.env.example` file and rename the duplicated file to `.env`. Ensure it is in the root folder of the ./mcp-backend/ folder. Within the `.env` file, the value for **OPENAI_API_KEY** must be set. It may require that an API key is generated within OpenAI's API platform. All other values may remain as they are for the prototype.

Once these environment variables are set, the databases may be set up.

A.2.2 *Preparing the Databases*

Before being able to set up the databases with the example data, the backup files need to be downloaded and placed into the correct folders. They can be found under the following links:

- Textbook Data: neo4j-2025-11-16T12-53-54-fde218db.backup
- SpeedParcel Data: neo4j-2025-12-10T21-44-58-fde218db.backup

If the above files are not accessible, please reach out to the author or advisors of this thesis.

Once both files are downloaded they are ready to be placed within the project's folders. The Textbok Data backup file must be placed within the folder `./textbook-data/`. The SpeedParcel Data backup file must be placed under `./speedparcel-data/`. This ensures that the backup files are present and in the correct folders when running the Docker commands to set up the databases in the next step.

A.2.3 *Setting Up the Databases*

If Masutā is being run for the first time, then two containers need to be executed once so that they populate both databases with the default data. From within the root folder of the project, execute the following commands

```
docker compose --profile speedparcel-restore run --rm speedparcel-restore
docker compose --profile textbook-restore  run --rm textbook-restore
```

Windows: If the executing system runs on Windows, this step requires the use of WSL (Windows Subsystem for Linux) and to set the HOME variable via `SET HOME=\C:\Users\<username>`. It may be required to execute these commands from a normal terminal instead of the Windows PowerShell.

Once these commands have run through successfully, both databases are populated with the default data and are ready to be used.

A.2.4 *Hard-Resetting Everything*

If at any time a hard-reset for the entire application is required, run the following commands:

```
docker compose down
rm -rf $HOME/neo4j/data
rm -rf $HOME/neo4j_empty/data
```

After resetting the application, restart with the first time setup.

A.3 RUNNING THE APPLICATION

Running Masutā is as simple as running the following command:

```
docker compose up
```

When running this command for the first time, it may take a while as it will have to download dependencies.

With this command, four containers are started.

- The SpeedParcel neo4j database filled with example data and the textbook
- The playground neo4j database filled with only the textbook
- The MCP-Backend which is the backend node.js which by default runs under http://localhost:4000
- The frontend which by default runs under http://localhost:3000/

If Docker does not automatically open the frontend in the browser, navigate to it under http://localhost:3000/. Everything is now ready and Masutā may be used.

A.4 FEATURE OVERVIEW

This section details the features within Masutā.

A.4.1 *Input Field and Chat History*

At the bottom of the application is an input field where the user may prompt Masutā. Pressing enter sends the prompt. The user's input prompt then appears as a chat bubble bound to the right of the chat-area. The system then starts working on answering the prompt and displays a chat bubble bound to the left of the chat-area displaying "Thinking...". As soon as the application is done thinking, the response is displayed in the same chat bubble.

Masutā's response has a button appended to it which allows the user to copy the cypher which was used to generate the answer. The user may take this copied cypher and paste it into neo4j desktop to inspect the raw response that was used to generate the answer.

A.4.2 *Toggling the Database*

At the top right of the application is a toggle switch. This allows the user to toggle between the SpeedParcel database and the playground database.

A.4.3 *Resetting the Playground Database*

Under the menu at the top left, the user is presented with the option to reset the database. This option is only available if the playground database is selected via the toggle switch. After confirming the reset in the dialog that pops up, the playground is reset to its default state with only the textbook information. The SpeedParcel database cannot be reset via the UI.

A.4.4 *Importing Custom Data*

Under the menu at the top left, the user is presented with the option to import their own data. This option is only available if the playground database is selected via the toggle switch. Within the dialog that pops up, the user can add up to 10 files either by selecting them via the file system or by dragging-and-dropping them into the dialog. Only .xml files are accepted. Check section A.5 on how to export XML files from ArchiMate.

After clicking the import button, the files are uploaded. If the upload was successful, an alert is shown notifying the user of this. The uploaded data remains within the realms of the Docker containers and is not uploaded to any third-party sources. The data can then be queried if the database toggle is set to the playground database.

A.4.5 *Inspect Database*

Under the menu at the top left, the user is presented with the option to view the graph data in neo4j browser their own data. This option is available for both the SpeedParcel and playground database. Clicking this opens another tab under https://console-preview.neo4j.io/tools/query. Here, the user can connect to the selected database. The connection string is displayed in a dialog within Masutā. Here, the user can inspect the graph structure and paste the cyphers used for the generated responses.

In order to access the neo4j console, it may be required that the user has a neo4j account.

A.4.6 *Chat Context and Building on Previous Answers*

Masutā saves the chat history so that the user is able to build upon previous prompts. The chat history is set so that the single previous user prompt and corresponding response is saved. This means that a further question can be

asked about the previous prompt or its response. The chat history's context only goes back one prompt and response.

A.4.7 *Exported Chat Protocol*

In order to support the explorative nature of the prototype, a built-in function that runs in the background of Masutā logs all interactions between the user and the system. This protocol can be found in the project's directory under `./mcp-backend/protocols/chat_history.txt`. The log's schema is always the same and looks as follows:

```
--- New Chat Turn <current date as YYYY-MM-DDTHH:mm:ss.SSSZ> ---
Using DB Target: <SpeedParcel or Playground>
User: <prompt>
Agent: <response>
Executed Cypher: <cypher>
```

Listing A.1: Schema of the automatically exported protocol

Each turn between the user and the chatbot is appended to the end of the text file. The chat history is never reset and is thus persistent across multiple sessions. This protocol can be helpful in documenting results when interacting with Masutā.

A.5 EXPORTING XML FILES FROM ARCHIMATE

It is recommended to import XML files into Masutā that have been exported directly from Archi. To do so, within the Archi application, click on file → export → model to open exchange file. Save this to a location you can find later. Notice that the exported file is an XML file. This can then be imported into Masutā as described in section A.4.4.

A.6 USING A DIFFERENT LLM

Although Masutā was implemented with the intention of using an LLM from Open AI, it is technically possible to use a different LLM. To do so, the source code needs to be edited in x different locations as follows:

- ./mcp-backend/.env: Set the variable of the OPENAI_API_KEY to the API key of whatever new LLM you wish to use.
- ./mcp-backend/src/server.ts: The root server file needs to have all calls toward Open AI replaced with calls to the new LLM. These can be found in the functions `nlToCypher()` and `explainResult()`. The structure of the calls made and the way they are subsequently handled may have to be refactored as well. It must also be ensured that the structure of the response to the frontend remains the same, otherwise a refactoring on the frontend will also be necessary.

A.7 EXAMPLE QUESTIONS

Questions can be broken down into three categories:

1. Category 1: Textbook questions about enterprise architecture management
2. Category 2: Basic information about the enterprise architecture data
3. Category 3: Questions that span both category 1 and 2 and are more complex than questions in category 2. These require more in-depth information about the data as well as how to deal with it as an enterprise architect.

The following list of questions serve as examples that Masutā can handle. These simply serve as ideas to test the system - the user may input their own thought up questions.

1. Questions in Category 1
    - What are the schools of enterprise architecture management?
    - How does enterprise architecture relate to town planning?
2. Questions in Category 2
    - How many capabilities are supported?
    - How many applications are in use?
    - Which application supports the most capabilities?
    - If I remove application x, which capabilities will be affected?
3. Questions in Category 3
    - We plan on removing application x. What do we have to look out for when doing so?
    - We wish to implement a new application x which covers the capabilities x, y, and z. Which existing applications may become redundant?
    - Which capabilities currently rely on single-point-of-failure applications? How should we deal with this?
    - Which capabilities are over-supported by multiple applications?
    - What interfaces does application x have?
    - How big of a task is it to migrate from application x to application y?

This concludes the instruction manual for Masutā. Beyond the example questions, users are encouraged to prompt Masutā to explore the example data provided in SpeedParcel or to analyze their own proprietary data after importing it, in order to gain insights into their enterprise architecture.

B

MASUTĀ CODE SNIPPETS

This chapter showcases code snippets which are of special interest as they show how key functionalities of Masutā work. The code snippets do not showcase the full application and may not complete.

B.1 SYSTEM PROMPTS

The following code snippets showcase what system prompts are passed to the LLM in order to generate the cyphers via the natural language text input and how the cypher results are transformed back into natural language.

B.1.1 *System Prompt to Generate Cyphers from Natural Language Prompt*

```
async function nlToCypher(nlPrompt: string, schema: string) {
  const systemPrompt = "
    You are working with a Neo4j graph whose structure is described in the
        schema summary.

    Important rules:
    - Do not assume fixed labels like :Application or :Chunk unless they appear
        in the schema.
    - Infer meaning from label names (e.g. ApplicationComponent     application)
        .
    - If no text-centric nodes exist, answer using structural relationships.

    Index usage based on the schema information passed to you:
    - If the schema summary lists a VECTOR index relevant to the task, start
        with:
      CALL db.index.vector.queryNodes(<indexName>, $embedding, <k>) YIELD node,
          score
    - If the schema summary lists a FULLTEXT index relevant to the task, start
        with:
      CALL db.index.fulltext.queryNodes(<indexName>, $query, {limit: <k>}) YIELD
           node, score
      (Do NOT pass a bare integer as the 3rd argument.)
    - After any CALL ... YIELD, you MUST finish with a RETURN clause.
      Example: CALL ... YIELD node, score RETURN node, score
    - Otherwise use MATCH with WHERE + indexed properties.

    Your task:
    - Write a SINGLE valid Cypher query.
    - Output ONLY Cypher. No explanations.

    Cypher syntax rules (important):
    - Do NOT use exists(node.property).
    - Neo4j 5+ requires property existence checks to use:
        node.property IS NOT NULL
```

```
    - Always use IS NOT NULL instead of exists(...)
    - NEVER use EXPLAIN or PROFILE. Always output an executable query that ends
      with RETURN.
    - You MAY start with CALL db.index.fulltext.queryNodes(...) or CALL db.index
      .vector.queryNodes(...).
  "
};
```

B.1.2 *System Prompt to Natural Language Response from Cypher Results*

todo

B.1.3 *Visualizing Knowledge Graphs in Neo4j*

todo: show some nice blooms of the knowledge graphs.

C

MASUTĀ EXAMPLE QUESTIONS AND ANSWERS

This chapter contains screenshots of user prompts and the system's responses. It serves as a showcase of what the application looks like and how it responds to questions.

show some test cases here. break down an answer like "i want to remove the application StatManPlus. What do i have to look out for as an enterprise architect?". the returned answer goes into a lot of depth as this is a level 3 question. break down the result of the query and how the result pulls information about the application but also pulls information from the Lehrbuch database.

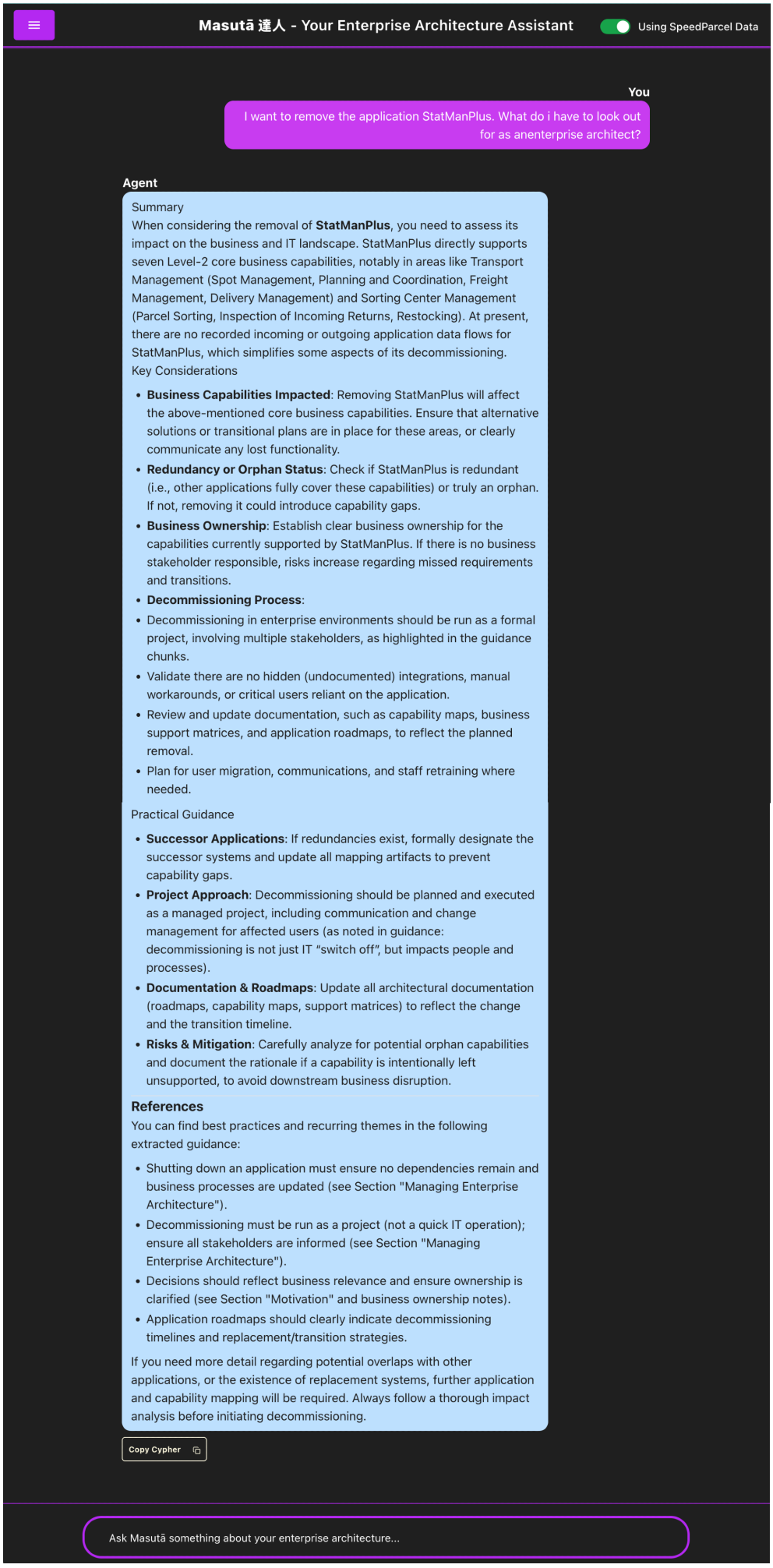
C.1 QUESTION XYZ

todo

```
MATCH (a:Application {name: 'StatManPlus'})
OPTIONAL MATCH (a)-[:SUPPORTS]->(cap:Capability)
WITH a, collect(DISTINCT cap) AS caps
UNWIND (CASE WHEN size(caps) > 0 THEN caps ELSE [NULL] END) AS cap
OPTIONAL MATCH (cap)-[:HAS_CHILD*0..]->(desc:Capability)
WITH a, caps, collect(DISTINCT desc) AS cap_impacted
OPTIONAL MATCH (a)-[:DATA_FLOW]->(outApp:Application)
WITH a, caps, cap_impacted, collect(DISTINCT outApp.name) AS outgoing_apps
OPTIONAL MATCH (inApp:Application)-[:DATA_FLOW]->(a)
WITH a, caps, cap_impacted, outgoing_apps, collect(DISTINCT inApp.name) AS
    incoming_apps
OPTIONAL MATCH (ch:Chunk)
WHERE toLower(coalesce(ch.text,'')) CONTAINS toLower(a.name)
   OR toLower(coalesce(ch.table_summary,'')) CONTAINS toLower(a.name)
   OR toLower(coalesce(ch.title,'')) CONTAINS toLower(a.name)
   OR toLower(coalesce(ch.definition,'')) CONTAINS toLower(a.name)
WITH a, caps, cap_impacted, outgoing_apps, incoming_apps,
     collect(DISTINCT {key: ch.key, title: ch.title, snippet: ch.
    table_summary, text: ch.text}) AS app_chunks
OPTIONAL MATCH (guid:Chunk)
WHERE toLower(coalesce(guid.text,'')) CONTAINS 'decommission'
   OR toLower(coalesce(guid.text,'')) CONTAINS 'retire'
   OR toLower(coalesce(guid.text,'')) CONTAINS 'sunset'
   OR toLower(coalesce(guid.title,'')) CONTAINS 'decommission'
   OR toLower(coalesce(guid.title,'')) CONTAINS 'retire'
   OR toLower(coalesce(guid.title,'')) CONTAINS 'sunset'
   OR toLower(coalesce(guid.definition,'')) CONTAINS 'decommission'
   OR toLower(coalesce(guid.definition,'')) CONTAINS 'retire'
   OR toLower(coalesce(guid.definition,'')) CONTAINS 'sunset'
   OR toLower(coalesce(guid.table_summary,'')) CONTAINS 'decommission'
   OR toLower(coalesce(guid.table_summary,'')) CONTAINS 'retire'
   OR toLower(coalesce(guid.table_summary,'')) CONTAINS 'sunset'
WITH a, caps, cap_impacted, outgoing_apps, incoming_apps, app_chunks,
```

```
     collect(DISTINCT {key: guid.key, title: guid.title, snippet: guid.
    table_summary, text: guid.text}) AS guidance_chunks
RETURN a.name AS application,
       [c IN caps WHERE c IS NOT NULL | c.id] AS supported_capability_ids,
       [g IN cap_impacted WHERE g IS NOT NULL | g.id] AS
    impacted_capability_ids,
       outgoing_apps AS outgoing_data_flows_to_apps,
       incoming_apps AS incoming_data_flows_from_apps,
       app_chunks AS chunks_referencing_application,
       guidance_chunks AS chunks_with_decommission_guidance
```

Listing C.1: Cypher query to retrieve application impact and decommission guidance


Masutā 達人 - Your Enterprise Architecture Assistant
Using SpeedParcel Data
You
I want to remove the application StatManPlus. What do i have to look out for as anenterprise architect?
Agent
Summary
When considering the removal of StatManPlus, you need to assess its impact on the business and IT landscape. StatManPlus directly supports seven Level-2 core business capabilities, notably in areas like Transport Management (Spot Management, Planning and Coordination, Freight Management, Delivery Management) and Sorting Center Management (Parcel Sorting, Inspection of Incoming Returns, Restocking). At present, there are no recorded incoming or outgoing application data flows for StatManPlus, which simplifies some aspects of its decommissioning.
Key Considerations
Business Capabilities Impacted: Removing StatManPlus will affect the above-mentioned core business capabilities. Ensure that alternative solutions or transitional plans are in place for these areas, or clearly communicate any lost functionality.
Redundancy or Orphan Status: Check if StatManPlus is redundant (i.e., other applications fully cover these capabilities) or truly an orphan. If not, removing it could introduce capability gaps.
Business Ownership: Establish clear business ownership for the capabilities currently supported by StatManPlus. If there is no business stakeholder responsible, risks increase regarding missed requirements and transitions.
Decommissioning Process:
Decommissioning in enterprise environments should be run as a formal project, involving multiple stakeholders, as highlighted in the guidance chunks.
Validate there are no hidden (undocumented) integrations, manual workarounds, or critical users reliant on the application.
Review and update documentation, such as capability maps, business support matrices, and application roadmaps, to reflect the planned removal.
Plan for user migration, communications, and staff retraining where needed.
Practical Guidance
Successor Applications: If redundancies exist, formally designate the successor systems and update all mapping artifacts to prevent capability gaps.
Project Approach: Decommissioning should be planned and executed as a managed project, including communication and change management for affected users (as noted in guidance: decommissioning is not just IT “switch off”, but impacts people and processes).
Documentation & Roadmaps: Update all architectural documentation (roadmaps, capability maps, support matrices) to reflect the change and the transition timeline.
Risks & Mitigation: Carefully analyze for potential orphan capabilities and document the rationale if a capability is intentionally left unsupported, to avoid downstream business disruption.
References
You can find best practices and recurring themes in the following extracted guidance:
Shutting down an application must ensure no dependencies remain and business processes are updated (see Section "Managing Enterprise Architecture").
Decommissioning must be run as a project (not a quick IT operation); ensure all stakeholders are informed (see Section "Managing Enterprise Architecture").
Decisions should reflect business relevance and ensure ownership is clarified (see Section "Motivation" and business ownership notes).
Application roadmaps should clearly indicate decommissioning timelines and replacement/transition strategies.
If you need more detail regarding potential overlaps with other applications, or the existence of replacement systems, further application and capability mapping will be required. Always follow a thorough impact analysis before initiating decommissioning.
Copy Cypher
Ask Masutā something about your enterprise architecture...

application	supported_capability_ids	impacted_capability_ids	outg	incon	chunks_referencing_application	chunks_with_decommission_guidance

"StatManPlus"	["cap_l2_51_spot_management", "cap_l2_52_planning_and_coordination", "cap_l2_53_freight_management", "cap_l2_54_delivery_management", "cap_l2_58_parcel_sorting", "cap_l2_59_inspection_of_incoming_returns", "cap_l2_60_restocking"]	["cap_l2_51_spot_management", "cap_l2_52_planning_and_coordination", "cap_l2_53_freight_management", "cap_l2_54_delivery_management", "cap_l2_58_parcel_sorting", "cap_l2_59_inspection_of_incoming_returns", "cap_l2_60_restocking"]	[]	[]	[{
					snippet:null,
					text:"Capability "Spot Management" (Level 2 in band "Core Capabilities", under L1 capability "Transport Management") is supported by: StatManPlus (flag "x").",
					title:null,
					key:"c7e0b12a9b94ca889c7e13d9e4cea9e504e5e0d2_sec_transport_management_chunk_spot_management"
					}, {
					snippet:null,
					text:"Capability "Planning and Coordination" (Level 2 in band "Core Capabilities", under L1 capability "Transport Management") is supported by: StatManPlus (flag "x"), TrackDB (flag "x").",
					title:null,
					key:"c7e0b12a9b94ca889c7e13d9e4cea9e504e5e0d2_sec_transport_management_chunk_planning_and_coordination"
					}, {
					snippet:null,
					text:"Capability "Freight Management" (Level 2 in band "Core Capabilities", under L1 capability "Transport Management") is supported by: StatManPlus (flag "x").",
					title:null,
					key:"c7e0b12a9b94ca889c7e13d9e4cea9e504e5e0d2_sec_transport_management_chunk_freight_management"
					}, {
						[{
						snippet:null,
						text:"This last difference is visible in a lot of companies today. There are software systems available and used by business people, but nobody on the business side feels responsible for them. Responsibility would include maintenance (i.e. conducting projects to perform changes), support and funding (e.g. for licenses). It, furthermore, refers to making decisions on the future of individual software applications, like its decommissioning if it does no longer provide relevant value. Making decisions based on business relevance implies an assessment from the business perspective. In fact, many systems are just handed over to IT so that they keep them running but there is no business ownership. IT people then need to drive changes driven by business requirements. Ideally, these should be driven by business stakeholders. We still have to convince people in business and IT to understand the relevance of each IT system and then make sure the IT system has business relevance. We further need clear ownership within the business to take the responsibility for decisions on this IT system.",
						title:null,
						key:"27be2852-75f0-44bc-a0d8-343ee3ed7ce9_10"

Figure C.2: Raw response returned from the Cypher in listing C.1. Notice that the response contains both information about the applications themselves as well as textbook information on enterprise architecture management (right-most column).