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
GenAI	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) [33] 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 the action research.

Part I

THESIS

1 INTRODUCTION

1.1 MOTIVATION AND THESIS QUESTION

In the field of 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.

1.2 RESEARCH METHOD

Action Research was applied in order to gain scientific value out of the developed prototype. The advantage of this research method is that it is very supportive of the development process for information systems. According to Baskerville (1999), 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. The development cycle is conducted iteratively and collaboratively with different stakeholders. More details on this are described in chapter

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 how the Action Research method was applied and how the development iterations were conducted. Building on this, chapter 10 explains the final prototype with all of its components. Chapter 11 describes how the final prototype was tested and evaluated. Chapter 12 discusses the implementation, what went well, and what could be improved. Finally, chapter 13 presents the conclusion and possibilities for future work.

This thesis presupposes an understanding of general terms in the realm of computer science 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. [20, 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. [20, 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. [20, 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. [20, pg. 152-154]

2.1.3 *Architecture Diagrams and Tools*

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. [20, 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. [37]

Various tools exist in order to maintain such architecture diagrams. The modeling languages ArchiMate is a common practice for graphical representation of an enterprise's architecture. The software tool Archi builds upon the ArchiMate modeling language and offers a UI for modeling enterprise architectures. [5]

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 [16].

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

[43] 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. [22, 50]

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. [22] 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. [50]

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. [36, 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 [36, 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. [42, pgs. 34-35]

A popular and open source implementation for graph databases is Neo4j [32]. It is a native graph database, meaning it directly stores direct references to adjacent nodes [36, pgs. 149-150] This allows Neo4j to execute queries in

milliseconds [36]. Another main advantage of Neo4j is its features to visually interact with the graph via its Neo4j Desktop application or web based tool. [42] 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. [28, 31]

Within this prototype, Neo4j is implemented as a standalone server and is accessed by the prototype via Cypher queries [36, 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. [36, pg. 27][42, 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. [44] 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, 18]

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 [18].

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. [18] 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 [18]. 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. [29]

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. [7, 45]

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. [35]

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 [33]. 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 [8], 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.

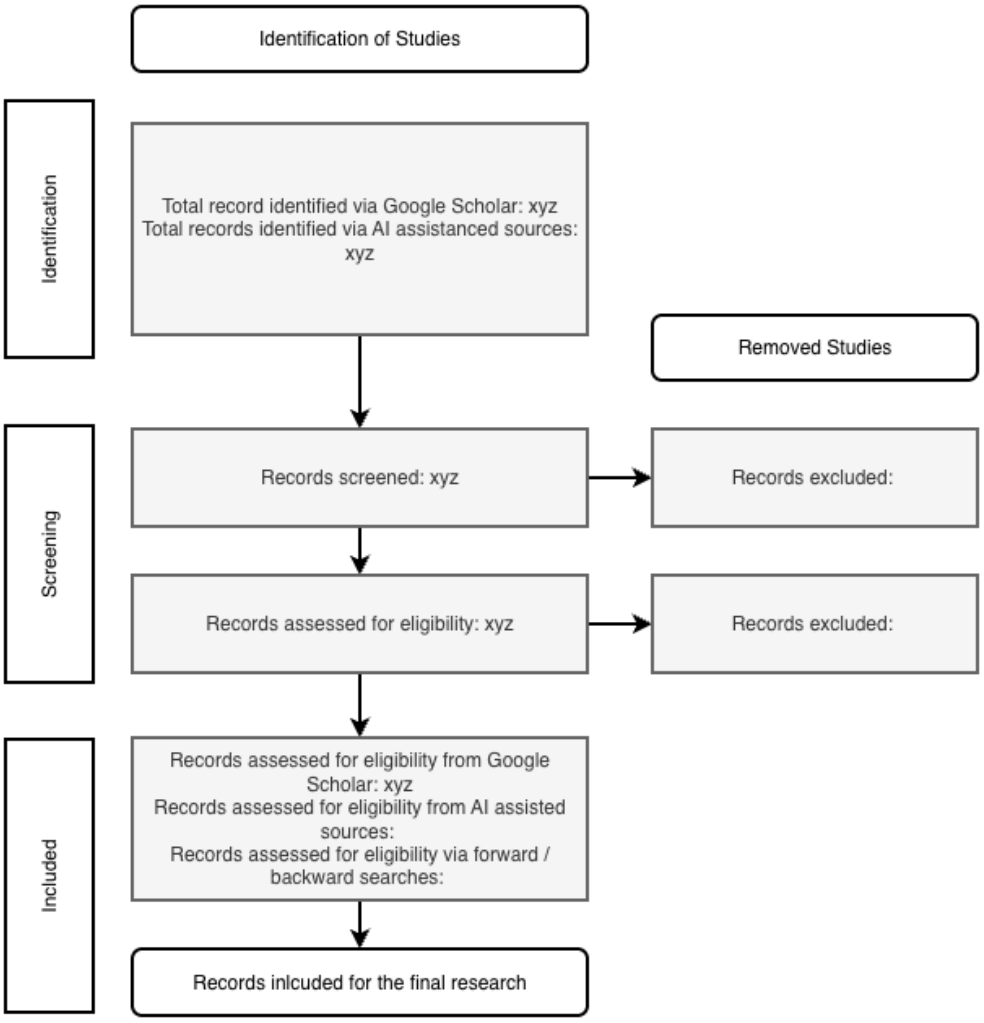


Figure 3.1: A flowchart of the applied PRISMA framework depicting how many sources were initially found with each search method and how many were left for the final current state of the art analysis. The framework is applied as seen in [26].

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 [20] 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) [19] 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. [17]

Zhao et al. (2024) [51] 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 4 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) [43] 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) [10] 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) [21] 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 9.3. 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) [40] 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 9.3 and 12.

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 [15]

This paper [46] 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: [23].

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[13]. The accompanying paper is here: [14]

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

This chapter describes 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 process through which the prototype was created. This gives the reader the understanding of how the research and implementation was conducted before discussing the implementation details in chapter 10.

4.1 ACTION RESEARCH DESIGN

Action research is a research method which is highly applicable when developing an information system such as the one presented in this paper [4]. The advantage of Action Research is that a large focus can be laid on the development of a system while still achieving an academic benefit. This allows for a very explorative approach to developing an information system.

Todo: describe why action research applies well to software development

Todo: add these challenges to the cycles:

- Chunking methoden beim Einlesen des Textbuches 21.10.25
- Die Datenstruktur des eingelesenen Textbuches 21.10.25
- Das abfragen der GraphDB zusammen mit dem LLM - zB wurden unendliche Rekursionen mit eingebaut am Anfang, weil Nodes auf sich im Kreis zeigen (schätze ich mal - nicht so wirklich verifiziert) 21.10.25

Cyclical phases are central to the concept of Action Research. Baskerville [4] describes Action Research as an iterative process consisting of five steps within a single cycle. Other sources, such as Cornish et al. [12], 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 [4, 12].

The paper at hand applied a cycle using the following five steps according to Baskerville [4]: 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..

Each cycle lasted between three and four weeks. The sources used 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 discuss 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 client 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. [4] From here on out, the term Participatory Action Research will be used interchangeably with Action Research.

The following subsections explain the research design of the applied Action Research. For the exhaustive Action Research protocol, refer to chapter A of the Appendix.

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, as well as the relationships between these two 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.

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 acted as an expert practitioner. The co-advisor acted as 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.

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.

4.3 ACTION RESEARCH CYLCES

The above mentioned vague details of the implementation became clear during the development cycles, leading to a final architecture with a clear structure. Each cycle began with a meeting between all three stakeholders. The end of one cycle was the incubator for the next cycle and was conducted in the same meeting. Table 9.1 summarizes these cycles.

Todo: note in the cycles where i attemtped implementation x, which didn't work out because of reason y. any path taken should be noted. also, note at what state the system as a whole was at the end of each cycle. you have this in the learnings for cycle 4 that it was able to answer all 3 categories of questions. do something similar for each cycle.

4.3.1 *Cycle 1*

Diagnosis: The initial problem statement lacked concrete technical formulation and the feasibility of natural language interaction with enterprise architecture data was unclear.

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 propsed of an AI-based black-box system capable of ingesting architecture data and deriving its own internal representations. 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. 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. This transparency aligns well with the mentioned advantages of a RAG-based LLM system in section 3.2 [51].

The key distinction between the black-box and white-box approaches is the transparency. While the black-box approach autonomously creates an

internal representation of the information, the white-box approach requires the manual design and implementation of an explicit technical architecture.

These discussions were necessary in order to scope the solution space. Early visions of an end goals included a chatbot that would support enterprise architects in exploring and improving application landscapes, for example by identifying inconsistencies or incomplete application landscapes.

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.

Action Planning: The first cycle aimed to develop a proof of concept that enables conversational access to a knowledge graph consisting of enterprise architecture knowledge grounded in textbook-based domain information. It was decided that a single-agent architecture will be used, as multi-agent architectures were considered unnecessarily complex for an initial proof of concept.

Action Taken: An initial knowledge graph was constructed based on content from the textbook *Masterclass Enterprise Architecture Management* [20]. The textual content was iteratively preprocessed and transformed into graph representations, with successive refinements applied to improve the mapping of domain concepts into nodes and relationships. This mapping was implemented using an LLM to parse the file and add it to the knowledge graph, as described by authors Laurenzi et al. (2024) [21] in section 3.2. The parser was individually fit to each file. After integrating the full textbook into the knowledge graph, additional prototypical data was incorporated in the form of an application landscape and business capability map to enable querying the knowledge graph of concrete architectural data. Each of these files also received a customized LLM-based parser.

The querying method was implemented via hard-coded cyphers in the frontend. The user's input prompt was then run through an LLM which attempted to fit the input into the predefined cyphers.

The LLM of choice during this stage was Qwen2.5-7B-Instruct [41] as it was free to use and readily available via the Hugging Pace platform.

Evaluation: The state of the proof of concept after the first cycle was demonstrated to both advisors and evaluated qualitatively through open discussions. 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.

Learning: Compared to the beginning of the cycle, in which the feasibility of a centralized knowledge graph was uncertain, the first cycle demonstrated that an explicit, white-box approach represents a practical path forward. The cycle also revealed key challenges in three critical layers of the application. These layers relate to importing heterogeneous data and transforming it into graph structures, the effective querying of such representations, and the slow response generation by the LLM. On top of this, the risk of relying too heavily on an LLM arose by giving full control of critical points of the

application to the LLM and thus being dependent on the capabilities of the language model.

Finally, it became evident that further iterations would be required to systematically address these challenges with an exploratory development process.

4.3.2 *Cycle 2*

Diagnosis: Following the initial feasibility assessment, the next identified challenge concerned extending the knowledge graph with more information and positioning the prototype as a useful tool within the realm of enterprise architecture management.

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 to enable direct relationships between conceptual textbook knowledge and architectural data, with the expectation of improved contextual reasoning.

The challenge imposed by lack of real-world data to test the system was also diagnosed. Potential test-datasets for further development and evaluation were discussed in order to bridge the gap before real-world data would be ready. The real-world data would require more time to be prepared because the co-advisor's company policy constraints meant the data cannot be used directly and has to be sanitzed first. Instead, it was agreed upon to use data from a completed university assignment, consisting of an application landscape, business capability map, business capability support matrix, business object model, and cross-application data-flow diagram for a fictitious company named SpeedParcel.

Action Planning: The objective of the second cycle was to extend the knowledge graph with additional data from the SpeedParcel dataset. This included integrating the business capability support matrix on top of the already existent application landscape and business capability map. A second goal of this cycle was to improve the database querying method as soon as more data is available to test with.

Action Taken: The capability support matrix of the SpeedParcel dataset was integrated into the knowledge graph, requiring adjustments to existing query mechanisms. A large challenge that hindered advancements in development came up. The hard-coded cyphers that were being used to query the database were reaching their limit. A viable solution was not found to hard-code cyphers in such a way that the changing dataset can be dynamically queried.

The idea of a Model Context Protocol (MCP) server came up during development. This brought the advantage of a standardized interface between the application and the knowlede-graph. Implementing the MCP server helped achieve a higher quality in the retrieved answers, as the prompt was being transformed into custom cyphers during runtime. This allowed the cyphers being used to query the database to no longer be hard-coded in the frontend,

but allowed them to be generated dynamically based on the user input. This text-to-cypher generation is achieved by using an LLM to transform the user's prompt directly into cyphers, as apposed to having predefined cyphers that get populated with the user's input by the LLM. A prerequisite for this to work, however, was that the schema of the knowledge graph was known. At this stage, the schema was hard-coded into the frontend.

Evaluation: The extended prototype was again evaluated qualitatively through a demonstration by the researcher and open discussions between all three stakeholders. Both advisors expressed strong interest in the approach and assured that the chatboat was being developed in the correct direction in order to achieve the end goal of allowing enterprise architects to interact with the knowledge graph via natural language.

Learning: As the prototype is starting to mature, more learnings are being pulled from each phase. Particularly, the hard-coded query method that was previously implemented was proved to be insufficient for flexible interaction with the evolving graph structure. The MCP implementation allows the entire system to be more dynamic, independent of the data in the knowledge graph.

Furthermore, three key categories of user questions were identified, similar to the ones described in section 3.2 [51]. The first category being conceptual questions related to enterprise architecture principles, concepts, and best practices found in textbooks. The second category being descriptive questions targeting concrete architectural elements and relationships. The third category being integrative questions combining conceptual knowledge with specific architectural contexts. Examples of these categories can be found in section **todo**.

Beyond the scope of this thesis, a broader vision by the advisors emerged in which the prototype could support project planning by assessing impacts on application interfaces, systems, and stakeholders. The co-advisor noted that the developments achieved thus far provide a starting point for exploring such directions in future work.

4.3.3 *Cycle 3*

Diagnosis: During this phase it became apparent that Action Research will be the most appropriate methodological framework for the project. The development process had evolved into an exploratory and iterative approach. This has been characterized by continuous development, reflection between all three stakeholders, and decision-making regarding which direction to take the prototype in. Consequently, the previously considered expert interview was no longer the methodology of choice for this thesis.

In parallel, the co-advisor began modelling real-world enterprise architecture data using the Archi tool. It was agreed upon that the real-world data would later be exported from Archi and imported into the knowledge graph in XML format. This meant that the format of the real-world data is finalized

and can be prepared for in the prototype, allowing the integration of more realistic data into the finished prototype.

Additionally, it was decided that a final review will be conducted at the end of the fourth cycle. This review would require that the prototype be finished and be executable in a local environment. This review will also mark the end of the development phase of the prototype.

Action Planning: The objective of the third cycle was to further expand the information saved in the knowledge graph by creating additional datasets from the SpeedParcel dataset. The datasets existed in a third party architecture modelling tool and had to be replicated in Archi in order to simulate the real-world data which will later also be exported from Archi. Modelling the data in Archi allows it to be exported into XML format and imported into the knowledge graph via an XML parser. This step aimed to prepare the system for handling the more complex and structured architectural models expected later.

Action Taken: Both the business object model and the cross-application data-flow diagram were recreated in Archi, in order to test how exported XML files will behave when parsing them into the knowledge graph. The parsing at this step was still being conducted via a custom, per-file LLM-based parser. This cycle mostly dealt with expanding the knowledge graph and fine tuning the system.

Evaluation: The progress of the third cycle was reviewed with both advisors during open discussions. The current state of the prototype and the extended data integration were assessed positively, and the overall research trajectory was again confirmed as appropriate. The end goal is in sight and is being worked towards in an appropriate manner.

The system had trouble generating qualitative answers for the newly imported dataset. This is due to the new data having a different schema than the other datasets.

Learning: The third cycle revealed scaling issues. While the individual per-file parsers have worked well so far, this cycle showed that its limits have been reached. Parsing the more complex files during this cycle showed that LLMs are no longer the appropriate method moving forward. On top of this, this custom, per-file parsing approach would also not allow for parsing files without manual implementation specific for the file at hand; a problem that has to be solved before the final review where files will have to be parsed on site and on the fly.

Another scalability issue revealed was a discussion about the real-world data. While SpeedParcel's data contained hundreds of nodes and edges, the real-world data will potentially have thousands of nodes and edges. This gap revealed that a better parsing solution and a better querying solution will be necessary before the prototype is ready for the final on site review. This will have to be tested before the on site review to ensure that this does not cause performance issues within the system.

4.3.4 *Cycle 4*

Diagnosis: The fourth and last cycle diagnoses that the system in its current state relies too heavily on LLMs for core precessing tasks. Importing the XML data into the knowledge graph is a critical point of the application and relying on an LLM to handle this parsing is not a good choice because XML is a standardized format and many parsers exist to handle XML. It was agreed upon that a generalized parser should be used to parse the incoming XML files into the knowledge graph, replacing the LLM at this critical point within the system's architecture.

A second critical part of the system that was diagnosed to be insufficient was the querying method to retrieve the information from the knowledge graph. Although the cyphers are being LLM-generated custom to the user's input, it is having trouble dealing with the changing schema of the knowledge graph because a hard-coded schema is written into the frontend which it uses as a reference for what to query. With an evolving knowledge graph, this approach is no longer viable because it is guessing at the graph's structure and not able to retrieve the data this way.

Action Planning: The goal of this cycle was to finish the prototype and have it be ready for the on site review at the end of the cycle where all three stakeholders will be testing and discussing the system together. Achieving this goal means that the XML data must be parsed via a generalized parser, replacing the LLM implementation. On top of that, the strategy of how to handle the schema must be refactored. It will be required to dynamically understand the data, rather than have one schema hard-coded. Lastly, it also meant setting up a containerized local environment to allow running the prototype on the advisor's devices during the on site review.

Action Taken: The LLM-based data parser was replaced by an out of the box solution from Awesome Procedures On Cypher (APOC), which contains functions to parse file types and also how to better query a knowledge graph. Here, APOC takes the input XML files and parses them directly into cyphers. This allows any type of Archi-exported XML file to be saved into the knowledge graph.

The schema-handling strategy was updated from the hard-coded variant. Before passing the user-prompt and system context to the LLM, a call to the knowledge graph was made via neo4j's built-in call `db.schema.visualization()` [30] in order to get the graph's schema information. While this did improve the generated results, it still proved to be insufficient, as the LLM was not able to handle this schema information well enough to generate reliable responses.

This was also solved via APOC. Rather than using neo4j's built-in schema call, the schema retrieval was changed to a predefined APOC function `apoc.meta.schema()` [31]. 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 is set up via Docker containers and can be run on any machine. It comes equipped with a frontend, backend, and two databases. The first database contains all data pertaining to SpeedParcel. The second database is a playground database and is only pre-filled with textbook information. This ensures that the on site review will allow the application to work.

Lastly, cosmetic changes to the UI were added as well as new features to support the on site review. For example, the user may now upload XML files via the web application as well as reset the playground database, and view the knowledge graph in an interactive way.

Evaluation: The refactored prototype was reviewed by both advisors via internal testing and demonstrations using the example data. The advisors assessed whether the system was executable, stable, and suitable for the final review. The replacement of the LLM-based XML parser with a generalized APOC-based parser and the updated schema-handling strategy were evaluated positively, as they enabled deterministic data imports and more reliable cypher generation. At the end of the fourth cycle, Masutā was deemed functionally complete and capable of answering all three categories of questions, thereby fulfilling the prerequisites for the final review.

Learning: Many learnings can be taken from this last cycle. The first being that a generalized parser to read XML files into the knowledge graph improves the quality of the import. Another added benefit of this is that the import is now deterministic, while before the LLM-based parser behaved in a non-deterministic way.

The second main learning from this cycle arose from refactoring the way that the queries get generated. Previously, the cyphers were straightforward to generate, as the schema of the SpeedParcel data was entirely known. However, during this cycle, new data was imported into the knowledge graph, which had an unknown structure, meaning the knowldge-graph's schema was unknown. The transformation of text to query has been improved by making a preliminary call to the database to fetch the current database schema. This schema information is then passed as context to the LLM in order to write more accurate cyphers. The results with this new querying strategy meant that for the first time the system was able to answer all three categories of questions.

With the conclusion of this cycle, Masutā was a finished prototype and ready to be tested during the final review.

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 be 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.

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.		

4.4 FINAL REVIEW

After the fourth and final Action Research cycle, a hands-on workshop was conducted as the final review of the prototype. The purpose of this review was to test Masutā using real-world EAM questions in order to observe and identify its behavior and capabilities 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 focusses exclusively on describing the setup and procedure of the final review. The evaluation of the results and implications can be found in chapter 12 and section 13.2.

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. Each participant interacted with the systems in an explorative manner.

For contextual comparison, the chatbot Rovo [3], which is integrated in Atlassian's knowledge management software Confluence, was used in parallel on the same set of data. This setup allowed the domain experts to contrast the behavior and outputs of both systems under identical conditions.

A new set of data was introduced specifically for the workshop. Rather than relying on the SpeedParcel data or synthetic data, the enterprise ar-

chitecture of the examination office at the Frankfurt University of Applied Sciences was used. This enabled evaluation in a realistic but controlled real-world scenario.

The workshop was purposefully conducted in an explorative nature, consistent with the applied Action Research approach. One domain expert interacted with Masutā, while the other interacted with Rovo. The researcher documented observations and supported the workshop with technical assistance for Masutā.

The four hour workshop was structured in five phases, each with a defined scope. After a short status overview, in which each participant aligned on the current state of the prototype's development and thesis, the first round of experiments began. The first phase focused on assessing the basic system functionality using simple domain questions. These questions were asked to both systems in parallel. 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. 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. The workshop concluded with a closing phase in which the domain experts summarized the findings, advantages of Masutā, and potential improvements for future development.

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.

As described in this chapter, 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 being clear and goal oriented. Each phase helped to examine what was possible from a technical standpoint as well as how to move forward. This supported the explorative nature of the project. The final review of the prototype proved that the 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.

5

CYCLE 1

This chapter describes 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 process through which the prototype was created. This gives the reader the understanding of how the research and implementation was conducted before discussing the implementation details in chapter 10.

5.1 ACTION RESEARCH DESIGN

Action research is a research method which is highly applicable when developing an information system such as the one presented in this paper [4]. The advantage of Action Research is that a large focus can be laid on the development of a system while still achieving an academic benefit. This allows for a very explorative approach to developing an information system.

Todo: describe why action research applies well to software development

Todo: add these challenges to the cycles:

- Chunking methoden beim Einlesen des Textbuches 21.10.25
- Die Datenstruktur des eingelesenen Textbuches 21.10.25
- Das abfragen der GraphDB zusammen mit dem LLM - zB wurden unendliche Rekursionen mit eingebaut am Anfang, weil Nodes auf sich im Kreis zeigen (schätze ich mal - nicht so wirklich verifiziert) 21.10.25

Cyclical phases are central to the concept of Action Research. Baskerville [4] describes Action Research as an iterative process consisting of five steps within a single cycle. Other sources, such as Cornish et al. [12], 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 [4, 12].

The paper at hand applied a cycle using the following five steps according to Baskerville [4]: 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..

Each cycle lasted between three and four weeks. The sources used 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 discuss 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 client 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. [4] From here on out, the term Participatory Action Research will be used interchangeably with Action Research.

The following subsections explain the research design of the applied Action Research. For the exhaustive Action Research protocol, refer to chapter A of the Appendix.

5.2 ACTION RESEARCH SETUP

The domain of EAM was focused on within this research. In particular, this study addressed mostly application landscapes, business capability maps, as well as the relationships between these two 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.

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 acted as an expert practitioner. The co-advisor acted as 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.

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.

5.3 ACTION RESEARCH CYLCES

The above mentioned vague details of the implementation became clear during the development cycles, leading to a final architecture with a clear structure. Each cycle began with a meeting between all three stakeholders. The end of one cycle was the incubator for the next cycle and was conducted in the same meeting. Table 9.1 summarizes these cycles.

Todo: note in the cycles where i attemtped implementation x, which didn't work out because of reason y. any path taken should be noted. also, note at what state the system as a whole was at the end of each cycle. you have this in the learnings for cycle 4 that it was able to answer all 3 categories of questions. do something similar for each cycle.

5.3.1 *Cycle 1*

Diagnosis: The initial problem statement lacked concrete technical formulation and the feasibility of natural language interaction with enterprise architecture data was unclear.

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 propsed of an AI-based black-box system capable of ingesting architecture data and deriving its own internal representations. 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. 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. This transparency aligns well with the mentioned advantages of a RAG-based LLM system in section 3.2 [51].

The key distinction between the black-box and white-box approaches is the transparency. While the black-box approach autonomously creates an

internal representation of the information, the white-box approach requires the manual design and implementation of an explicit technical architecture.

These discussions were necessary in order to scope the solution space. Early visions of an end goals included a chatbot that would support enterprise architects in exploring and improving application landscapes, for example by identifying inconsistencies or incomplete application landscapes.

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.

Action Planning: The first cycle aimed to develop a proof of concept that enables conversational access to a knowledge graph consisting of enterprise architecture knowledge grounded in textbook-based domain information. It was decided that a single-agent architecture will be used, as multi-agent architectures were considered unnecessarily complex for an initial proof of concept.

Action Taken: An initial knowledge graph was constructed based on content from the textbook *Masterclass Enterprise Architecture Management* [20]. The textual content was iteratively preprocessed and transformed into graph representations, with successive refinements applied to improve the mapping of domain concepts into nodes and relationships. This mapping was implemented using an LLM to parse the file and add it to the knowledge graph, as described by authors Laurenzi et al. (2024) [21] in section 3.2. The parser was individually fit to each file. After integrating the full textbook into the knowledge graph, additional prototypical data was incorporated in the form of an application landscape and business capability map to enable querying the knowledge graph of concrete architectural data. Each of these files also received a customized LLM-based parser.

The querying method was implemented via hard-coded cyphers in the frontend. The user's input prompt was then run through an LLM which attempted to fit the input into the predefined cyphers.

The LLM of choice during this stage was Qwen2.5-7B-Instruct [41] as it was free to use and readily available via the Hugging Pace platform.

Evaluation: The state of the proof of concept after the first cycle was demonstrated to both advisors and evaluated qualitatively through open discussions. 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.

Learning: Compared to the beginning of the cycle, in which the feasibility of a centralized knowledge graph was uncertain, the first cycle demonstrated that an explicit, white-box approach represents a practical path forward. The cycle also revealed key challenges in three critical layers of the application. These layers relate to importing heterogeneous data and transforming it into graph structures, the effective querying of such representations, and the slow response generation by the LLM. On top of this, the risk of relying too heavily on an LLM arose by giving full control of critical points of the

application to the LLM and thus being dependent on the capabilities of the language model.

Finally, it became evident that further iterations would be required to systematically address these challenges with an exploratory development process.

5.3.2 *Cycle 2*

Diagnosis: Following the initial feasibility assessment, the next identified challenge concerned extending the knowledge graph with more information and positioning the prototype as a useful tool within the realm of enterprise architecture management.

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 to enable direct relationships between conceptual textbook knowledge and architectural data, with the expectation of improved contextual reasoning.

The challenge imposed by lack of real-world data to test the system was also diagnosed. Potential test-datasets for further development and evaluation were discussed in order to bridge the gap before real-world data would be ready. The real-world data would require more time to be prepared because the co-advisor's company policy constraints meant the data cannot be used directly and has to be sanitzed first. Instead, it was agreed upon to use data from a completed university assignment, consisting of an application landscape, business capability map, business capability support matrix, business object model, and cross-application data-flow diagram for a fictitious company named SpeedParcel.

Action Planning: The objective of the second cycle was to extend the knowledge graph with additional data from the SpeedParcel dataset. This included integrating the business capability support matrix on top of the already existent application landscape and business capability map. A second goal of this cycle was to improve the database querying method as soon as more data is available to test with.

Action Taken: The capability support matrix of the SpeedParcel dataset was integrated into the knowledge graph, requiring adjustments to existing query mechanisms. A large challenge that hindered advancements in development came up. The hard-coded cyphers that were being used to query the database were reaching their limit. A viable solution was not found to hard-code cyphers in such a way that the changing dataset can be dynamically queried.

The idea of a Model Context Protocol (MCP) server came up during development. This brought the advantage of a standardized interface between the application and the knowlede-graph. Implementing the MCP server helped achieve a higher quality in the retrieved answers, as the prompt was being transformed into custom cyphers during runtime. This allowed the cyphers being used to query the database to no longer be hard-coded in the frontend,

but allowed them to be generated dynamically based on the user input. This text-to-cypher generation is achieved by using an LLM to transform the user's prompt directly into cyphers, as apposed to having predefined cyphers that get populated with the user's input by the LLM. A prerequisite for this to work, however, was that the schema of the knowledge graph was known. At this stage, the schema was hard-coded into the frontend.

Evaluation: The extended prototype was again evaluated qualitatively through a demonstration by the researcher and open discussions between all three stakeholders. Both advisors expressed strong interest in the approach and assured that the chatboat was being developed in the correct direction in order to achieve the end goal of allowing enterprise architects to interact with the knowledge graph via natural language.

Learning: As the prototype is starting to mature, more learnings are being pulled from each phase. Particularly, the hard-coded query method that was previously implemented was proved to be insufficient for flexible interaction with the evolving graph structure. The MCP implementation allows the entire system to be more dynamic, independent of the data in the knowledge graph.

Furthermore, three key categories of user questions were identified, similar to the ones described in section 3.2 [51]. The first category being conceptual questions related to enterprise architecture principles, concepts, and best practices found in textbooks. The second category being descriptive questions targeting concrete architectural elements and relationships. The third category being integrative questions combining conceptual knowledge with specific architectural contexts. Examples of these categories can be found in section **todo**.

Beyond the scope of this thesis, a broader vision by the advisors emerged in which the prototype could support project planning by assessing impacts on application interfaces, systems, and stakeholders. The co-advisor noted that the developments achieved thus far provide a starting point for exploring such directions in future work.

5.3.3 *Cycle 3*

Diagnosis: During this phase it became apparent that Action Research will be the most appropriate methodological framework for the project. The development process had evolved into an exploratory and iterative approach. This has been characterized by continuous development, reflection between all three stakeholders, and decision-making regarding which direction to take the prototype in. Consequently, the previously considered expert interview was no longer the methodology of choice for this thesis.

In parallel, the co-advisor began modelling real-world enterprise architecture data using the Archi tool. It was agreed upon that the real-world data would later be exported from Archi and imported into the knowledge graph in XML format. This meant that the format of the real-world data is finalized

and can be prepared for in the prototype, allowing the integration of more realistic data into the finished prototype.

Additionally, it was decided that a final review will be conducted at the end of the fourth cycle. This review would require that the prototype be finished and be executable in a local environment. This review will also mark the end of the development phase of the prototype.

Action Planning: The objective of the third cycle was to further expand the information saved in the knowledge graph by creating additional datasets from the SpeedParcel dataset. The datasets existed in a third party architecture modelling tool and had to be replicated in Archi in order to simulate the real-world data which will later also be exported from Archi. Modelling the data in Archi allows it to be exported into XML format and imported into the knowledge graph via an XML parser. This step aimed to prepare the system for handling the more complex and structured architectural models expected later.

Action Taken: Both the business object model and the cross-application data-flow diagram were recreated in Archi, in order to test how exported XML files will behave when parsing them into the knowledge graph. The parsing at this step was still being conducted via a custom, per-file LLM-based parser. This cycle mostly dealt with expanding the knowledge graph and fine tuning the system.

Evaluation: The progress of the third cycle was reviewed with both advisors during open discussions. The current state of the prototype and the extended data integration were assessed positively, and the overall research trajectory was again confirmed as appropriate. The end goal is in sight and is being worked towards in an appropriate manner.

The system had trouble generating qualitative answers for the newly imported dataset. This is due to the new data having a different schema than the other datasets.

Learning: The third cycle revealed scaling issues. While the individual per-file parsers have worked well so far, this cycle showed that its limits have been reached. Parsing the more complex files during this cycle showed that LLMs are no longer the appropriate method moving forward. On top of this, this custom, per-file parsing approach would also not allow for parsing files without manual implementation specific for the file at hand; a problem that has to be solved before the final review where files will have to be parsed on site and on the fly.

Another scalability issue revealed was a discussion about the real-world data. While SpeedParcel's data contained hundreds of nodes and edges, the real-world data will potentially have thousands of nodes and edges. This gap revealed that a better parsing solution and a better querying solution will be necessary before the prototype is ready for the final on site review. This will have to be tested before the on site review to ensure that this does not cause performance issues within the system.

5.3.4 *Cycle 4*

Diagnosis: The fourth and last cycle diagnoses that the system in its current state relies too heavily on LLMs for core precessing tasks. Importing the XML data into the knowledge graph is a critical point of the application and relying on an LLM to handle this parsing is not a good choice because XML is a standardized format and many parsers exist to handle XML. It was agreed upon that a generalized parser should be used to parse the incoming XML files into the knowledge graph, replacing the LLM at this critical point within the system's architecture.

A second critical part of the system that was diagnosed to be insufficient was the querying method to retrieve the information from the knowledge graph. Although the cyphers are being LLM-generated custom to the user's input, it is having trouble dealing with the changing schema of the knowledge graph because a hard-coded schema is written into the frontend which it uses as a reference for what to query. With an evolving knowledge graph, this approach is no longer viable because it is guessing at the graph's structure and not able to retrieve the data this way.

Action Planning: The goal of this cycle was to finish the prototype and have it be ready for the on site review at the end of the cycle where all three stakeholders will be testing and discussing the system together. Achieving this goal means that the XML data must be parsed via a generalized parser, replacing the LLM implementation. On top of that, the strategy of how to handle the schema must be refactored. It will be required to dynamically understand the data, rather than have one schema hard-coded. Lastly, it also meant setting up a containerized local environment to allow running the prototype on the advisor's devices during the on site review.

Action Taken: The LLM-based data parser was replaced by an out of the box solution from Awesome Procedures On Cypher (APOC), which contains functions to parse file types and also how to better query a knowledge graph. Here, APOC takes the input XML files and parses them directly into cyphers. This allows any type of Archi-exported XML file to be saved into the knowledge graph.

The schema-handling strategy was updated from the hard-coded variant. Before passing the user-prompt and system context to the LLM, a call to the knowledge graph was made via neo4j's built-in call `db.schema.visualization()` [30] in order to get the graph's schema information. While this did improve the generated results, it still proved to be insufficient, as the LLM was not able to handle this schema information well enough to generate reliable responses.

This was also solved via APOC. Rather than using neo4j's built-in schema call, the schema retrieval was changed to a predefined APOC function `apoc.meta.schema()` [31]. 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 is set up via Docker containers and can be run on any machine. It comes equipped with a frontend, backend, and two databases. The first database contains all data pertaining to SpeedParcel. The second database is a playground database and is only pre-filled with textbook information. This ensures that the on site review will allow the application to work.

Lastly, cosmetic changes to the UI were added as well as new features to support the on site review. For example, the user may now upload XML files via the web application as well as reset the playground database, and view the knowledge graph in an interactive way.

Evaluation: The refactored prototype was reviewed by both advisors via internal testing and demonstrations using the example data. The advisors assessed whether the system was executable, stable, and suitable for the final review. The replacement of the LLM-based XML parser with a generalized APOC-based parser and the updated schema-handling strategy were evaluated positively, as they enabled deterministic data imports and more reliable cypher generation. At the end of the fourth cycle, Masutā was deemed functionally complete and capable of answering all three categories of questions, thereby fulfilling the prerequisites for the final review.

Learning: Many learnings can be taken from this last cycle. The first being that a generalized parser to read XML files into the knowledge graph improves the quality of the import. Another added benefit of this is that the import is now deterministic, while before the LLM-based parser behaved in a non-deterministic way.

The second main learning from this cycle arose from refactoring the way that the queries get generated. Previously, the cyphers were straightforward to generate, as the schema of the SpeedParcel data was entirely known. However, during this cycle, new data was imported into the knowledge graph, which had an unknown structure, meaning the knowldge-graph's schema was unknown. The transformation of text to query has been improved by making a preliminary call to the database to fetch the current database schema. This schema information is then passed as context to the LLM in order to write more accurate cyphers. The results with this new querying strategy meant that for the first time the system was able to answer all three categories of questions.

With the conclusion of this cycle, Masutā was a finished prototype and ready to be tested during the final review.

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 be 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.

Table 5.1: 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.		

5.4 FINAL REVIEW

After the fourth and final Action Research cycle, a hands-on workshop was conducted as the final review of the prototype. The purpose of this review was to test Masutā using real-world EAM questions in order to observe and identify its behavior and capabilities 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 focusses exclusively on describing the setup and procedure of the final review. The evaluation of the results and implications can be found in chapter 12 and section 13.2.

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. Each participant interacted with the systems in an explorative manner.

For contextual comparison, the chatbot Rovo [3], which is integrated in Atlassian's knowledge management software Confluence, was used in parallel on the same set of data. This setup allowed the domain experts to contrast the behavior and outputs of both systems under identical conditions.

A new set of data was introduced specifically for the workshop. Rather than relying on the SpeedParcel data or synthetic data, the enterprise ar-

chitecture of the examination office at the Frankfurt University of Applied Sciences was used. This enabled evaluation in a realistic but controlled real-world scenario.

The workshop was purposefully conducted in an explorative nature, consistent with the applied Action Research approach. One domain expert interacted with Masutā, while the other interacted with Rovo. The researcher documented observations and supported the workshop with technical assistance for Masutā.

The four hour workshop was structured in five phases, each with a defined scope. After a short status overview, in which each participant aligned on the current state of the prototype's development and thesis, the first round of experiments began. The first phase focused on assessing the basic system functionality using simple domain questions. These questions were asked to both systems in parallel. 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. 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. The workshop concluded with a closing phase in which the domain experts summarized the findings, advantages of Masutā, and potential improvements for future development.

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.

As described in this chapter, 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 being clear and goal oriented. Each phase helped to examine what was possible from a technical standpoint as well as how to move forward. This supported the explorative nature of the project. The final review of the prototype proved that the 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.

6 CYCLE 2

This chapter describes 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 process through which the prototype was created. This gives the reader the understanding of how the research and implementation was conducted before discussing the implementation details in chapter 10.

6.1 ACTION RESEARCH DESIGN

Action research is a research method which is highly applicable when developing an information system such as the one presented in this paper [4]. The advantage of Action Research is that a large focus can be laid on the development of a system while still achieving an academic benefit. This allows for a very explorative approach to developing an information system.

Todo: describe why action research applies well to software development

Todo: add these challenges to the cycles:

- Chunking methoden beim Einlesen des Textbuches 21.10.25
- Die Datenstruktur des eingelesenen Textbuches 21.10.25
- Das abfragen der GraphDB zusammen mit dem LLM - zB wurden unendliche Rekursionen mit eingebaut am Anfang, weil Nodes auf sich im Kreis zeigen (schätze ich mal - nicht so wirklich verifiziert) 21.10.25

Cyclical phases are central to the concept of Action Research. Baskerville [4] describes Action Research as an iterative process consisting of five steps within a single cycle. Other sources, such as Cornish et al. [12], 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 [4, 12].

The paper at hand applied a cycle using the following five steps according to Baskerville [4]: 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..

Each cycle lasted between three and four weeks. The sources used 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 discuss 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 client 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. [4] From here on out, the term Participatory Action Research will be used interchangeably with Action Research.

The following subsections explain the research design of the applied Action Research. For the exhaustive Action Research protocol, refer to chapter A of the Appendix.

6.2 ACTION RESEARCH SETUP

The domain of EAM was focused on within this research. In particular, this study addressed mostly application landscapes, business capability maps, as well as the relationships between these two 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.

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 acted as an expert practitioner. The co-advisor acted as 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.

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.

6.3 ACTION RESEARCH CYLCES

The above mentioned vague details of the implementation became clear during the development cycles, leading to a final architecture with a clear structure. Each cycle began with a meeting between all three stakeholders. The end of one cycle was the incubator for the next cycle and was conducted in the same meeting. Table 9.1 summarizes these cycles.

Todo: note in the cycles where i attemtped implementation x, which didn't work out because of reason y. any path taken should be noted. also, note at what state the system as a whole was at the end of each cycle. you have this in the learnings for cycle 4 that it was able to answer all 3 categories of questions. do something similar for each cycle.

6.3.1 *Cycle 1*

Diagnosis: The initial problem statement lacked concrete technical formulation and the feasibility of natural language interaction with enterprise architecture data was unclear.

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 propsed of an AI-based black-box system capable of ingesting architecture data and deriving its own internal representations. 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. 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. This transparency aligns well with the mentioned advantages of a RAG-based LLM system in section 3.2 [51].

The key distinction between the black-box and white-box approaches is the transparency. While the black-box approach autonomously creates an

internal representation of the information, the white-box approach requires the manual design and implementation of an explicit technical architecture.

These discussions were necessary in order to scope the solution space. Early visions of an end goals included a chatbot that would support enterprise architects in exploring and improving application landscapes, for example by identifying inconsistencies or incomplete application landscapes.

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.

Action Planning: The first cycle aimed to develop a proof of concept that enables conversational access to a knowledge graph consisting of enterprise architecture knowledge grounded in textbook-based domain information. It was decided that a single-agent architecture will be used, as multi-agent architectures were considered unnecessarily complex for an initial proof of concept.

Action Taken: An initial knowledge graph was constructed based on content from the textbook *Masterclass Enterprise Architecture Management* [20]. The textual content was iteratively preprocessed and transformed into graph representations, with successive refinements applied to improve the mapping of domain concepts into nodes and relationships. This mapping was implemented using an LLM to parse the file and add it to the knowledge graph, as described by authors Laurenzi et al. (2024) [21] in section 3.2. The parser was individually fit to each file. After integrating the full textbook into the knowledge graph, additional prototypical data was incorporated in the form of an application landscape and business capability map to enable querying the knowledge graph of concrete architectural data. Each of these files also received a customized LLM-based parser.

The querying method was implemented via hard-coded cyphers in the frontend. The user's input prompt was then run through an LLM which attempted to fit the input into the predefined cyphers.

The LLM of choice during this stage was Qwen2.5-7B-Instruct [41] as it was free to use and readily available via the Hugging Pace platform.

Evaluation: The state of the proof of concept after the first cycle was demonstrated to both advisors and evaluated qualitatively through open discussions. 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.

Learning: Compared to the beginning of the cycle, in which the feasibility of a centralized knowledge graph was uncertain, the first cycle demonstrated that an explicit, white-box approach represents a practical path forward. The cycle also revealed key challenges in three critical layers of the application. These layers relate to importing heterogeneous data and transforming it into graph structures, the effective querying of such representations, and the slow response generation by the LLM. On top of this, the risk of relying too heavily on an LLM arose by giving full control of critical points of the

application to the LLM and thus being dependent on the capabilities of the language model.

Finally, it became evident that further iterations would be required to systematically address these challenges with an exploratory development process.

6.3.2 *Cycle 2*

Diagnosis: Following the initial feasibility assessment, the next identified challenge concerned extending the knowledge graph with more information and positioning the prototype as a useful tool within the realm of enterprise architecture management.

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 to enable direct relationships between conceptual textbook knowledge and architectural data, with the expectation of improved contextual reasoning.

The challenge imposed by lack of real-world data to test the system was also diagnosed. Potential test-datasets for further development and evaluation were discussed in order to bridge the gap before real-world data would be ready. The real-world data would require more time to be prepared because the co-advisor's company policy constraints meant the data cannot be used directly and has to be sanitzed first. Instead, it was agreed upon to use data from a completed university assignment, consisting of an application landscape, business capability map, business capability support matrix, business object model, and cross-application data-flow diagram for a fictitious company named SpeedParcel.

Action Planning: The objective of the second cycle was to extend the knowledge graph with additional data from the SpeedParcel dataset. This included integrating the business capability support matrix on top of the already existent application landscape and business capability map. A second goal of this cycle was to improve the database querying method as soon as more data is available to test with.

Action Taken: The capability support matrix of the SpeedParcel dataset was integrated into the knowledge graph, requiring adjustments to existing query mechanisms. A large challenge that hindered advancements in development came up. The hard-coded cyphers that were being used to query the database were reaching their limit. A viable solution was not found to hard-code cyphers in such a way that the changing dataset can be dynamically queried.

The idea of a Model Context Protocol (MCP) server came up during development. This brought the advantage of a standardized interface between the application and the knowlede-graph. Implementing the MCP server helped achieve a higher quality in the retrieved answers, as the prompt was being transformed into custom cyphers during runtime. This allowed the cyphers being used to query the database to no longer be hard-coded in the frontend,

but allowed them to be generated dynamically based on the user input. This text-to-cypher generation is achieved by using an LLM to transform the user's prompt directly into cyphers, as apposed to having predefined cyphers that get populated with the user's input by the LLM. A prerequisite for this to work, however, was that the schema of the knowledge graph was known. At this stage, the schema was hard-coded into the frontend.

Evaluation: The extended prototype was again evaluated qualitatively through a demonstration by the researcher and open discussions between all three stakeholders. Both advisors expressed strong interest in the approach and assured that the chatboat was being developed in the correct direction in order to achieve the end goal of allowing enterprise architects to interact with the knowledge graph via natural language.

Learning: As the prototype is starting to mature, more learnings are being pulled from each phase. Particularly, the hard-coded query method that was previously implemented was proved to be insufficient for flexible interaction with the evolving graph structure. The MCP implementation allows the entire system to be more dynamic, independent of the data in the knowledge graph.

Furthermore, three key categories of user questions were identified, similar to the ones described in section 3.2 [51]. The first category being conceptual questions related to enterprise architecture principles, concepts, and best practices found in textbooks. The second category being descriptive questions targeting concrete architectural elements and relationships. The third category being integrative questions combining conceptual knowledge with specific architectural contexts. Examples of these categories can be found in section **todo**.

Beyond the scope of this thesis, a broader vision by the advisors emerged in which the prototype could support project planning by assessing impacts on application interfaces, systems, and stakeholders. The co-advisor noted that the developments achieved thus far provide a starting point for exploring such directions in future work.

6.3.3 *Cycle 3*

Diagnosis: During this phase it became apparent that Action Research will be the most appropriate methodological framework for the project. The development process had evolved into an exploratory and iterative approach. This has been characterized by continuous development, reflection between all three stakeholders, and decision-making regarding which direction to take the prototype in. Consequently, the previously considered expert interview was no longer the methodology of choice for this thesis.

In parallel, the co-advisor began modelling real-world enterprise architecture data using the Archi tool. It was agreed upon that the real-world data would later be exported from Archi and imported into the knowledge graph in XML format. This meant that the format of the real-world data is finalized

and can be prepared for in the prototype, allowing the integration of more realistic data into the finished prototype.

Additionally, it was decided that a final review will be conducted at the end of the fourth cycle. This review would require that the prototype be finished and be executable in a local environment. This review will also mark the end of the development phase of the prototype.

Action Planning: The objective of the third cycle was to further expand the information saved in the knowledge graph by creating additional datasets from the SpeedParcel dataset. The datasets existed in a third party architecture modelling tool and had to be replicated in Archi in order to simulate the real-world data which will later also be exported from Archi. Modelling the data in Archi allows it to be exported into XML format and imported into the knowledge graph via an XML parser. This step aimed to prepare the system for handling the more complex and structured architectural models expected later.

Action Taken: Both the business object model and the cross-application data-flow diagram were recreated in Archi, in order to test how exported XML files will behave when parsing them into the knowledge graph. The parsing at this step was still being conducted via a custom, per-file LLM-based parser. This cycle mostly dealt with expanding the knowledge graph and fine tuning the system.

Evaluation: The progress of the third cycle was reviewed with both advisors during open discussions. The current state of the prototype and the extended data integration were assessed positively, and the overall research trajectory was again confirmed as appropriate. The end goal is in sight and is being worked towards in an appropriate manner.

The system had trouble generating qualitative answers for the newly imported dataset. This is due to the new data having a different schema than the other datasets.

Learning: The third cycle revealed scaling issues. While the individual per-file parsers have worked well so far, this cycle showed that its limits have been reached. Parsing the more complex files during this cycle showed that LLMs are no longer the appropriate method moving forward. On top of this, this custom, per-file parsing approach would also not allow for parsing files without manual implementation specific for the file at hand; a problem that has to be solved before the final review where files will have to be parsed on site and on the fly.

Another scalability issue revealed was a discussion about the real-world data. While SpeedParcel's data contained hundreds of nodes and edges, the real-world data will potentially have thousands of nodes and edges. This gap revealed that a better parsing solution and a better querying solution will be necessary before the prototype is ready for the final on site review. This will have to be tested before the on site review to ensure that this does not cause performance issues within the system.

6.3.4 *Cycle 4*

Diagnosis: The fourth and last cycle diagnoses that the system in its current state relies too heavily on LLMs for core precessing tasks. Importing the XML data into the knowledge graph is a critical point of the application and relying on an LLM to handle this parsing is not a good choice because XML is a standardized format and many parsers exist to handle XML. It was agreed upon that a generalized parser should be used to parse the incoming XML files into the knowledge graph, replacing the LLM at this critical point within the system's architecture.

A second critical part of the system that was diagnosed to be insufficient was the querying method to retrieve the information from the knowledge graph. Although the cyphers are being LLM-generated custom to the user's input, it is having trouble dealing with the changing schema of the knowledge graph because a hard-coded schema is written into the frontend which it uses as a reference for what to query. With an evolving knowledge graph, this approach is no longer viable because it is guessing at the graph's structure and not able to retrieve the data this way.

Action Planning: The goal of this cycle was to finish the prototype and have it be ready for the on site review at the end of the cycle where all three stakeholders will be testing and discussing the system together. Achieving this goal means that the XML data must be parsed via a generalized parser, replacing the LLM implementation. On top of that, the strategy of how to handle the schema must be refactored. It will be required to dynamically understand the data, rather than have one schema hard-coded. Lastly, it also meant setting up a containerized local environment to allow running the prototype on the advisor's devices during the on site review.

Action Taken: The LLM-based data parser was replaced by an out of the box solution from Awesome Procedures On Cypher (APOC), which contains functions to parse file types and also how to better query a knowledge graph. Here, APOC takes the input XML files and parses them directly into cyphers. This allows any type of Archi-exported XML file to be saved into the knowledge graph.

The schema-handling strategy was updated from the hard-coded variant. Before passing the user-prompt and system context to the LLM, a call to the knowledge graph was made via neo4j's built-in call `db.schema.visualization()` [30] in order to get the graph's schema information. While this did improve the generated results, it still proved to be insufficient, as the LLM was not able to handle this schema information well enough to generate reliable responses.

This was also solved via APOC. Rather than using neo4j's built-in schema call, the schema retrieval was changed to a predefined APOC function `apoc.meta.schema()` [31]. 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 is set up via Docker containers and can be run on any machine. It comes equipped with a frontend, backend, and two databases. The first database contains all data pertaining to SpeedParcel. The second database is a playground database and is only pre-filled with textbook information. This ensures that the on site review will allow the application to work.

Lastly, cosmetic changes to the UI were added as well as new features to support the on site review. For example, the user may now upload XML files via the web application as well as reset the playground database, and view the knowledge graph in an interactive way.

Evaluation: The refactored prototype was reviewed by both advisors via internal testing and demonstrations using the example data. The advisors assessed whether the system was executable, stable, and suitable for the final review. The replacement of the LLM-based XML parser with a generalized APOC-based parser and the updated schema-handling strategy were evaluated positively, as they enabled deterministic data imports and more reliable cypher generation. At the end of the fourth cycle, Masutā was deemed functionally complete and capable of answering all three categories of questions, thereby fulfilling the prerequisites for the final review.

Learning: Many learnings can be taken from this last cycle. The first being that a generalized parser to read XML files into the knowledge graph improves the quality of the import. Another added benefit of this is that the import is now deterministic, while before the LLM-based parser behaved in a non-deterministic way.

The second main learning from this cycle arose from refactoring the way that the queries get generated. Previously, the cyphers were straightforward to generate, as the schema of the SpeedParcel data was entirely known. However, during this cycle, new data was imported into the knowledge graph, which had an unknown structure, meaning the knowldge-graph's schema was unknown. The transformation of text to query has been improved by making a preliminary call to the database to fetch the current database schema. This schema information is then passed as context to the LLM in order to write more accurate cyphers. The results with this new querying strategy meant that for the first time the system was able to answer all three categories of questions.

With the conclusion of this cycle, Masutā was a finished prototype and ready to be tested during the final review.

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 be 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.

Table 6.1: 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.		

6.4 FINAL REVIEW

After the fourth and final Action Research cycle, a hands-on workshop was conducted as the final review of the prototype. The purpose of this review was to test Masutā using real-world EAM questions in order to observe and identify its behavior and capabilities 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 focusses exclusively on describing the setup and procedure of the final review. The evaluation of the results and implications can be found in chapter 12 and section 13.2.

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. Each participant interacted with the systems in an explorative manner.

For contextual comparison, the chatbot Rovo [3], which is integrated in Atlassian's knowledge management software Confluence, was used in parallel on the same set of data. This setup allowed the domain experts to contrast the behavior and outputs of both systems under identical conditions.

A new set of data was introduced specifically for the workshop. Rather than relying on the SpeedParcel data or synthetic data, the enterprise ar-

chitecture of the examination office at the Frankfurt University of Applied Sciences was used. This enabled evaluation in a realistic but controlled real-world scenario.

The workshop was purposefully conducted in an explorative nature, consistent with the applied Action Research approach. One domain expert interacted with Masutā, while the other interacted with Rovo. The researcher documented observations and supported the workshop with technical assistance for Masutā.

The four hour workshop was structured in five phases, each with a defined scope. After a short status overview, in which each participant aligned on the current state of the prototype's development and thesis, the first round of experiments began. The first phase focused on assessing the basic system functionality using simple domain questions. These questions were asked to both systems in parallel. 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. 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. The workshop concluded with a closing phase in which the domain experts summarized the findings, advantages of Masutā, and potential improvements for future development.

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.

As described in this chapter, 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 being clear and goal oriented. Each phase helped to examine what was possible from a technical standpoint as well as how to move forward. This supported the explorative nature of the project. The final review of the prototype proved that the 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.

7 CYCLE 3

This chapter describes 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 process through which the prototype was created. This gives the reader the understanding of how the research and implementation was conducted before discussing the implementation details in chapter 10.

7.1 ACTION RESEARCH DESIGN

Action research is a research method which is highly applicable when developing an information system such as the one presented in this paper [4]. The advantage of Action Research is that a large focus can be laid on the development of a system while still achieving an academic benefit. This allows for a very explorative approach to developing an information system.

Todo: describe why action research applies well to software development

Todo: add these challenges to the cycles:

- Chunking methoden beim Einlesen des Textbuches 21.10.25
- Die Datenstruktur des eingelesenen Textbuches 21.10.25
- Das abfragen der GraphDB zusammen mit dem LLM - zB wurden unendliche Rekursionen mit eingebaut am Anfang, weil Nodes auf sich im Kreis zeigen (schätze ich mal - nicht so wirklich verifiziert) 21.10.25

Cyclical phases are central to the concept of Action Research. Baskerville [4] describes Action Research as an iterative process consisting of five steps within a single cycle. Other sources, such as Cornish et al. [12], 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 [4, 12].

The paper at hand applied a cycle using the following five steps according to Baskerville [4]: 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..

Each cycle lasted between three and four weeks. The sources used 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 discuss 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 client 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. [4] From here on out, the term Participatory Action Research will be used interchangeably with Action Research.

The following subsections explain the research design of the applied Action Research. For the exhaustive Action Research protocol, refer to chapter A of the Appendix.

7.2 ACTION RESEARCH SETUP

The domain of EAM was focused on within this research. In particular, this study addressed mostly application landscapes, business capability maps, as well as the relationships between these two 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.

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 acted as an expert practitioner. The co-advisor acted as 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.

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.

7.3 ACTION RESEARCH CYLCES

The above mentioned vague details of the implementation became clear during the development cycles, leading to a final architecture with a clear structure. Each cycle began with a meeting between all three stakeholders. The end of one cycle was the incubator for the next cycle and was conducted in the same meeting. Table 9.1 summarizes these cycles.

Todo: note in the cycles where i attemtped implementation x, which didn't work out because of reason y. any path taken should be noted. also, note at what state the system as a whole was at the end of each cycle. you have this in the learnings for cycle 4 that it was able to answer all 3 categories of questions. do something similar for each cycle.

7.3.1 *Cycle 1*

Diagnosis: The initial problem statement lacked concrete technical formulation and the feasibility of natural language interaction with enterprise architecture data was unclear.

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 propsed of an AI-based black-box system capable of ingesting architecture data and deriving its own internal representations. 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. 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. This transparency aligns well with the mentioned advantages of a RAG-based LLM system in section 3.2 [51].

The key distinction between the black-box and white-box approaches is the transparency. While the black-box approach autonomously creates an

internal representation of the information, the white-box approach requires the manual design and implementation of an explicit technical architecture.

These discussions were necessary in order to scope the solution space. Early visions of an end goals included a chatbot that would support enterprise architects in exploring and improving application landscapes, for example by identifying inconsistencies or incomplete application landscapes.

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.

Action Planning: The first cycle aimed to develop a proof of concept that enables conversational access to a knowledge graph consisting of enterprise architecture knowledge grounded in textbook-based domain information. It was decided that a single-agent architecture will be used, as multi-agent architectures were considered unnecessarily complex for an initial proof of concept.

Action Taken: An initial knowledge graph was constructed based on content from the textbook *Masterclass Enterprise Architecture Management* [20]. The textual content was iteratively preprocessed and transformed into graph representations, with successive refinements applied to improve the mapping of domain concepts into nodes and relationships. This mapping was implemented using an LLM to parse the file and add it to the knowledge graph, as described by authors Laurenzi et al. (2024) [21] in section 3.2. The parser was individually fit to each file. After integrating the full textbook into the knowledge graph, additional prototypical data was incorporated in the form of an application landscape and business capability map to enable querying the knowledge graph of concrete architectural data. Each of these files also received a customized LLM-based parser.

The querying method was implemented via hard-coded cyphers in the frontend. The user's input prompt was then run through an LLM which attempted to fit the input into the predefined cyphers.

The LLM of choice during this stage was Qwen2.5-7B-Instruct [41] as it was free to use and readily available via the Hugging Pace platform.

Evaluation: The state of the proof of concept after the first cycle was demonstrated to both advisors and evaluated qualitatively through open discussions. 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.

Learning: Compared to the beginning of the cycle, in which the feasibility of a centralized knowledge graph was uncertain, the first cycle demonstrated that an explicit, white-box approach represents a practical path forward. The cycle also revealed key challenges in three critical layers of the application. These layers relate to importing heterogeneous data and transforming it into graph structures, the effective querying of such representations, and the slow response generation by the LLM. On top of this, the risk of relying too heavily on an LLM arose by giving full control of critical points of the

application to the LLM and thus being dependent on the capabilities of the language model.

Finally, it became evident that further iterations would be required to systematically address these challenges with an exploratory development process.

7.3.2 *Cycle 2*

Diagnosis: Following the initial feasibility assessment, the next identified challenge concerned extending the knowledge graph with more information and positioning the prototype as a useful tool within the realm of enterprise architecture management.

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 to enable direct relationships between conceptual textbook knowledge and architectural data, with the expectation of improved contextual reasoning.

The challenge imposed by lack of real-world data to test the system was also diagnosed. Potential test-datasets for further development and evaluation were discussed in order to bridge the gap before real-world data would be ready. The real-world data would require more time to be prepared because the co-advisor's company policy constraints meant the data cannot be used directly and has to be sanitzed first. Instead, it was agreed upon to use data from a completed university assignment, consisting of an application landscape, business capability map, business capability support matrix, business object model, and cross-application data-flow diagram for a fictitious company named SpeedParcel.

Action Planning: The objective of the second cycle was to extend the knowledge graph with additional data from the SpeedParcel dataset. This included integrating the business capability support matrix on top of the already existent application landscape and business capability map. A second goal of this cycle was to improve the database querying method as soon as more data is available to test with.

Action Taken: The capability support matrix of the SpeedParcel dataset was integrated into the knowledge graph, requiring adjustments to existing query mechanisms. A large challenge that hindered advancements in development came up. The hard-coded cyphers that were being used to query the database were reaching their limit. A viable solution was not found to hard-code cyphers in such a way that the changing dataset can be dynamically queried.

The idea of a Model Context Protocol (MCP) server came up during development. This brought the advantage of a standardized interface between the application and the knowlede-graph. Implementing the MCP server helped achieve a higher quality in the retrieved answers, as the prompt was being transformed into custom cyphers during runtime. This allowed the cyphers being used to query the database to no longer be hard-coded in the frontend,

but allowed them to be generated dynamically based on the user input. This text-to-cypher generation is achieved by using an LLM to transform the user's prompt directly into cyphers, as apposed to having predefined cyphers that get populated with the user's input by the LLM. A prerequisite for this to work, however, was that the schema of the knowledge graph was known. At this stage, the schema was hard-coded into the frontend.

Evaluation: The extended prototype was again evaluated qualitatively through a demonstration by the researcher and open discussions between all three stakeholders. Both advisors expressed strong interest in the approach and assured that the chatboat was being developed in the correct direction in order to achieve the end goal of allowing enterprise architects to interact with the knowledge graph via natural language.

Learning: As the prototype is starting to mature, more learnings are being pulled from each phase. Particularly, the hard-coded query method that was previously implemented was proved to be insufficient for flexible interaction with the evolving graph structure. The MCP implementation allows the entire system to be more dynamic, independent of the data in the knowledge graph.

Furthermore, three key categories of user questions were identified, similar to the ones described in section 3.2 [51]. The first category being conceptual questions related to enterprise architecture principles, concepts, and best practices found in textbooks. The second category being descriptive questions targeting concrete architectural elements and relationships. The third category being integrative questions combining conceptual knowledge with specific architectural contexts. Examples of these categories can be found in section **todo**.

Beyond the scope of this thesis, a broader vision by the advisors emerged in which the prototype could support project planning by assessing impacts on application interfaces, systems, and stakeholders. The co-advisor noted that the developments achieved thus far provide a starting point for exploring such directions in future work.

7.3.3 *Cycle 3*

Diagnosis: During this phase it became apparent that Action Research will be the most appropriate methodological framework for the project. The development process had evolved into an exploratory and iterative approach. This has been characterized by continuous development, reflection between all three stakeholders, and decision-making regarding which direction to take the prototype in. Consequently, the previously considered expert interview was no longer the methodology of choice for this thesis.

In parallel, the co-advisor began modelling real-world enterprise architecture data using the Archi tool. It was agreed upon that the real-world data would later be exported from Archi and imported into the knowledge graph in XML format. This meant that the format of the real-world data is finalized

and can be prepared for in the prototype, allowing the integration of more realistic data into the finished prototype.

Additionally, it was decided that a final review will be conducted at the end of the fourth cycle. This review would require that the prototype be finished and be executable in a local environment. This review will also mark the end of the development phase of the prototype.

Action Planning: The objective of the third cycle was to further expand the information saved in the knowledge graph by creating additional datasets from the SpeedParcel dataset. The datasets existed in a third party architecture modelling tool and had to be replicated in Archi in order to simulate the real-world data which will later also be exported from Archi. Modelling the data in Archi allows it to be exported into XML format and imported into the knowledge graph via an XML parser. This step aimed to prepare the system for handling the more complex and structured architectural models expected later.

Action Taken: Both the business object model and the cross-application data-flow diagram were recreated in Archi, in order to test how exported XML files will behave when parsing them into the knowledge graph. The parsing at this step was still being conducted via a custom, per-file LLM-based parser. This cycle mostly dealt with expanding the knowledge graph and fine tuning the system.

Evaluation: The progress of the third cycle was reviewed with both advisors during open discussions. The current state of the prototype and the extended data integration were assessed positively, and the overall research trajectory was again confirmed as appropriate. The end goal is in sight and is being worked towards in an appropriate manner.

The system had trouble generating qualitative answers for the newly imported dataset. This is due to the new data having a different schema than the other datasets.

Learning: The third cycle revealed scaling issues. While the individual per-file parsers have worked well so far, this cycle showed that its limits have been reached. Parsing the more complex files during this cycle showed that LLMs are no longer the appropriate method moving forward. On top of this, this custom, per-file parsing approach would also not allow for parsing files without manual implementation specific for the file at hand; a problem that has to be solved before the final review where files will have to be parsed on site and on the fly.

Another scalability issue revealed was a discussion about the real-world data. While SpeedParcel's data contained hundreds of nodes and edges, the real-world data will potentially have thousands of nodes and edges. This gap revealed that a better parsing solution and a better querying solution will be necessary before the prototype is ready for the final on site review. This will have to be tested before the on site review to ensure that this does not cause performance issues within the system.

7.3.4 *Cycle 4*

Diagnosis: The fourth and last cycle diagnoses that the system in its current state relies too heavily on LLMs for core precessing tasks. Importing the XML data into the knowledge graph is a critical point of the application and relying on an LLM to handle this parsing is not a good choice because XML is a standardized format and many parsers exist to handle XML. It was agreed upon that a generalized parser should be used to parse the incoming XML files into the knowledge graph, replacing the LLM at this critical point within the system's architecture.

A second critical part of the system that was diagnosed to be insufficient was the querying method to retrieve the information from the knowledge graph. Although the cyphers are being LLM-generated custom to the user's input, it is having trouble dealing with the changing schema of the knowledge graph because a hard-coded schema is written into the frontend which it uses as a reference for what to query. With an evolving knowledge graph, this approach is no longer viable because it is guessing at the graph's structure and not able to retrieve the data this way.

Action Planning: The goal of this cycle was to finish the prototype and have it be ready for the on site review at the end of the cycle where all three stakeholders will be testing and discussing the system together. Achieving this goal means that the XML data must be parsed via a generalized parser, replacing the LLM implementation. On top of that, the strategy of how to handle the schema must be refactored. It will be required to dynamically understand the data, rather than have one schema hard-coded. Lastly, it also meant setting up a containerized local environment to allow running the prototype on the advisor's devices during the on site review.

Action Taken: The LLM-based data parser was replaced by an out of the box solution from Awesome Procedures On Cypher (APOC), which contains functions to parse file types and also how to better query a knowledge graph. Here, APOC takes the input XML files and parses them directly into cyphers. This allows any type of Archi-exported XML file to be saved into the knowledge graph.

The schema-handling strategy was updated from the hard-coded variant. Before passing the user-prompt and system context to the LLM, a call to the knowledge graph was made via neo4j's built-in call `db.schema.visualization()` [30] in order to get the graph's schema information. While this did improve the generated results, it still proved to be insufficient, as the LLM was not able to handle this schema information well enough to generate reliable responses.

This was also solved via APOC. Rather than using neo4j's built-in schema call, the schema retrieval was changed to a predefined APOC function `apoc.meta.schema()` [31]. 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 is set up via Docker containers and can be run on any machine. It comes equipped with a frontend, backend, and two databases. The first database contains all data pertaining to SpeedParcel. The second database is a playground database and is only pre-filled with textbook information. This ensures that the on site review will allow the application to work.

Lastly, cosmetic changes to the UI were added as well as new features to support the on site review. For example, the user may now upload XML files via the web application as well as reset the playground database, and view the knowledge graph in an interactive way.

Evaluation: The refactored prototype was reviewed by both advisors via internal testing and demonstrations using the example data. The advisors assessed whether the system was executable, stable, and suitable for the final review. The replacement of the LLM-based XML parser with a generalized APOC-based parser and the updated schema-handling strategy were evaluated positively, as they enabled deterministic data imports and more reliable cypher generation. At the end of the fourth cycle, Masutā was deemed functionally complete and capable of answering all three categories of questions, thereby fulfilling the prerequisites for the final review.

Learning: Many learnings can be taken from this last cycle. The first being that a generalized parser to read XML files into the knowledge graph improves the quality of the import. Another added benefit of this is that the import is now deterministic, while before the LLM-based parser behaved in a non-deterministic way.

The second main learning from this cycle arose from refactoring the way that the queries get generated. Previously, the cyphers were straightforward to generate, as the schema of the SpeedParcel data was entirely known. However, during this cycle, new data was imported into the knowledge graph, which had an unknown structure, meaning the knowldge-graph's schema was unknown. The transformation of text to query has been improved by making a preliminary call to the database to fetch the current database schema. This schema information is then passed as context to the LLM in order to write more accurate cyphers. The results with this new querying strategy meant that for the first time the system was able to answer all three categories of questions.

With the conclusion of this cycle, Masutā was a finished prototype and ready to be tested during the final review.

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 be 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.

Table 7.1: 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.		

7.4 FINAL REVIEW

After the fourth and final Action Research cycle, a hands-on workshop was conducted as the final review of the prototype. The purpose of this review was to test Masutā using real-world EAM questions in order to observe and identify its behavior and capabilities 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 focusses exclusively on describing the setup and procedure of the final review. The evaluation of the results and implications can be found in chapter 12 and section 13.2.

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. Each participant interacted with the systems in an explorative manner.

For contextual comparison, the chatbot Rovo [3], which is integrated in Atlassian's knowledge management software Confluence, was used in parallel on the same set of data. This setup allowed the domain experts to contrast the behavior and outputs of both systems under identical conditions.

A new set of data was introduced specifically for the workshop. Rather than relying on the SpeedParcel data or synthetic data, the enterprise ar-

chitecture of the examination office at the Frankfurt University of Applied Sciences was used. This enabled evaluation in a realistic but controlled real-world scenario.

The workshop was purposefully conducted in an explorative nature, consistent with the applied Action Research approach. One domain expert interacted with Masutā, while the other interacted with Rovo. The researcher documented observations and supported the workshop with technical assistance for Masutā.

The four hour workshop was structured in five phases, each with a defined scope. After a short status overview, in which each participant aligned on the current state of the prototype's development and thesis, the first round of experiments began. The first phase focused on assessing the basic system functionality using simple domain questions. These questions were asked to both systems in parallel. 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. 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. The workshop concluded with a closing phase in which the domain experts summarized the findings, advantages of Masutā, and potential improvements for future development.

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.

As described in this chapter, 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 being clear and goal oriented. Each phase helped to examine what was possible from a technical standpoint as well as how to move forward. This supported the explorative nature of the project. The final review of the prototype proved that the 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.

8

CYCLE 4

This chapter describes 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 process through which the prototype was created. This gives the reader the understanding of how the research and implementation was conducted before discussing the implementation details in chapter 10.

8.1 ACTION RESEARCH DESIGN

Action research is a research method which is highly applicable when developing an information system such as the one presented in this paper [4]. The advantage of Action Research is that a large focus can be laid on the development of a system while still achieving an academic benefit. This allows for a very explorative approach to developing an information system.

Todo: describe why action research applies well to software development

Todo: add these challenges to the cycles:

- Chunking methoden beim Einlesen des Textbuches 21.10.25
- Die Datenstruktur des eingelesenen Textbuches 21.10.25
- Das abfragen der GraphDB zusammen mit dem LLM - zB wurden unendliche Rekursionen mit eingebaut am Anfang, weil Nodes auf sich im Kreis zeigen (schätze ich mal - nicht so wirklich verifiziert) 21.10.25

Cyclical phases are central to the concept of Action Research. Baskerville [4] describes Action Research as an iterative process consisting of five steps within a single cycle. Other sources, such as Cornish et al. [12], 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 [4, 12].

The paper at hand applied a cycle using the following five steps according to Baskerville [4]: 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..

Each cycle lasted between three and four weeks. The sources used 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 discuss 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 client 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. [4] From here on out, the term Participatory Action Research will be used interchangeably with Action Research.

The following subsections explain the research design of the applied Action Research. For the exhaustive Action Research protocol, refer to chapter A of the Appendix.

8.2 ACTION RESEARCH SETUP

The domain of EAM was focused on within this research. In particular, this study addressed mostly application landscapes, business capability maps, as well as the relationships between these two 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.

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 acted as an expert practitioner. The co-advisor acted as 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.

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.

8.3 ACTION RESEARCH CYLCES

The above mentioned vague details of the implementation became clear during the development cycles, leading to a final architecture with a clear structure. Each cycle began with a meeting between all three stakeholders. The end of one cycle was the incubator for the next cycle and was conducted in the same meeting. Table 9.1 summarizes these cycles.

Todo: note in the cycles where i attemtped implementation x, which didn't work out because of reason y. any path taken should be noted. also, note at what state the system as a whole was at the end of each cycle. you have this in the learnings for cycle 4 that it was able to answer all 3 categories of questions. do something similar for each cycle.

8.3.1 *Cycle 1*

Diagnosis: The initial problem statement lacked concrete technical formulation and the feasibility of natural language interaction with enterprise architecture data was unclear.

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 propsed of an AI-based black-box system capable of ingesting architecture data and deriving its own internal representations. 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. 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. This transparency aligns well with the mentioned advantages of a RAG-based LLM system in section 3.2 [51].

The key distinction between the black-box and white-box approaches is the transparency. While the black-box approach autonomously creates an

internal representation of the information, the white-box approach requires the manual design and implementation of an explicit technical architecture.

These discussions were necessary in order to scope the solution space. Early visions of an end goals included a chatbot that would support enterprise architects in exploring and improving application landscapes, for example by identifying inconsistencies or incomplete application landscapes.

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.

Action Planning: The first cycle aimed to develop a proof of concept that enables conversational access to a knowledge graph consisting of enterprise architecture knowledge grounded in textbook-based domain information. It was decided that a single-agent architecture will be used, as multi-agent architectures were considered unnecessarily complex for an initial proof of concept.

Action Taken: An initial knowledge graph was constructed based on content from the textbook *Masterclass Enterprise Architecture Management* [20]. The textual content was iteratively preprocessed and transformed into graph representations, with successive refinements applied to improve the mapping of domain concepts into nodes and relationships. This mapping was implemented using an LLM to parse the file and add it to the knowledge graph, as described by authors Laurenzi et al. (2024) [21] in section 3.2. The parser was individually fit to each file. After integrating the full textbook into the knowledge graph, additional prototypical data was incorporated in the form of an application landscape and business capability map to enable querying the knowledge graph of concrete architectural data. Each of these files also received a customized LLM-based parser.

The querying method was implemented via hard-coded cyphers in the frontend. The user's input prompt was then run through an LLM which attempted to fit the input into the predefined cyphers.

The LLM of choice during this stage was Qwen2.5-7B-Instruct [41] as it was free to use and readily available via the Hugging Pace platform.

Evaluation: The state of the proof of concept after the first cycle was demonstrated to both advisors and evaluated qualitatively through open discussions. 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.

Learning: Compared to the beginning of the cycle, in which the feasibility of a centralized knowledge graph was uncertain, the first cycle demonstrated that an explicit, white-box approach represents a practical path forward. The cycle also revealed key challenges in three critical layers of the application. These layers relate to importing heterogeneous data and transforming it into graph structures, the effective querying of such representations, and the slow response generation by the LLM. On top of this, the risk of relying too heavily on an LLM arose by giving full control of critical points of the

application to the LLM and thus being dependent on the capabilities of the language model.

Finally, it became evident that further iterations would be required to systematically address these challenges with an exploratory development process.

8.3.2 *Cycle 2*

Diagnosis: Following the initial feasibility assessment, the next identified challenge concerned extending the knowledge graph with more information and positioning the prototype as a useful tool within the realm of enterprise architecture management.

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 to enable direct relationships between conceptual textbook knowledge and architectural data, with the expectation of improved contextual reasoning.

The challenge imposed by lack of real-world data to test the system was also diagnosed. Potential test-datasets for further development and evaluation were discussed in order to bridge the gap before real-world data would be ready. The real-world data would require more time to be prepared because the co-advisor's company policy constraints meant the data cannot be used directly and has to be sanitzed first. Instead, it was agreed upon to use data from a completed university assignment, consisting of an application landscape, business capability map, business capability support matrix, business object model, and cross-application data-flow diagram for a fictitious company named SpeedParcel.

Action Planning: The objective of the second cycle was to extend the knowledge graph with additional data from the SpeedParcel dataset. This included integrating the business capability support matrix on top of the already existent application landscape and business capability map. A second goal of this cycle was to improve the database querying method as soon as more data is available to test with.

Action Taken: The capability support matrix of the SpeedParcel dataset was integrated into the knowledge graph, requiring adjustments to existing query mechanisms. A large challenge that hindered advancements in development came up. The hard-coded cyphers that were being used to query the database were reaching their limit. A viable solution was not found to hard-code cyphers in such a way that the changing dataset can be dynamically queried.

The idea of a Model Context Protocol (MCP) server came up during development. This brought the advantage of a standardized interface between the application and the knowlede-graph. Implementing the MCP server helped achieve a higher quality in the retrieved answers, as the prompt was being transformed into custom cyphers during runtime. This allowed the cyphers being used to query the database to no longer be hard-coded in the frontend,

but allowed them to be generated dynamically based on the user input. This text-to-cypher generation is achieved by using an LLM to transform the user's prompt directly into cyphers, as apposed to having predefined cyphers that get populated with the user's input by the LLM. A prerequisite for this to work, however, was that the schema of the knowledge graph was known. At this stage, the schema was hard-coded into the frontend.

Evaluation: The extended prototype was again evaluated qualitatively through a demonstration by the researcher and open discussions between all three stakeholders. Both advisors expressed strong interest in the approach and assured that the chatboat was being developed in the correct direction in order to achieve the end goal of allowing enterprise architects to interact with the knowledge graph via natural language.

Learning: As the prototype is starting to mature, more learnings are being pulled from each phase. Particularly, the hard-coded query method that was previously implemented was proved to be insufficient for flexible interaction with the evolving graph structure. The MCP implementation allows the entire system to be more dynamic, independent of the data in the knowledge graph.

Furthermore, three key categories of user questions were identified, similar to the ones described in section 3.2 [51]. The first category being conceptual questions related to enterprise architecture principles, concepts, and best practices found in textbooks. The second category being descriptive questions targeting concrete architectural elements and relationships. The third category being integrative questions combining conceptual knowledge with specific architectural contexts. Examples of these categories can be found in section **todo**.

Beyond the scope of this thesis, a broader vision by the advisors emerged in which the prototype could support project planning by assessing impacts on application interfaces, systems, and stakeholders. The co-advisor noted that the developments achieved thus far provide a starting point for exploring such directions in future work.

8.3.3 *Cycle 3*

Diagnosis: During this phase it became apparent that Action Research will be the most appropriate methodological framework for the project. The development process had evolved into an exploratory and iterative approach. This has been characterized by continuous development, reflection between all three stakeholders, and decision-making regarding which direction to take the prototype in. Consequently, the previously considered expert interview was no longer the methodology of choice for this thesis.

In parallel, the co-advisor began modelling real-world enterprise architecture data using the Archi tool. It was agreed upon that the real-world data would later be exported from Archi and imported into the knowledge graph in XML format. This meant that the format of the real-world data is finalized

and can be prepared for in the prototype, allowing the integration of more realistic data into the finished prototype.

Additionally, it was decided that a final review will be conducted at the end of the fourth cycle. This review would require that the prototype be finished and be executable in a local environment. This review will also mark the end of the development phase of the prototype.

Action Planning: The objective of the third cycle was to further expand the information saved in the knowledge graph by creating additional datasets from the SpeedParcel dataset. The datasets existed in a third party architecture modelling tool and had to be replicated in Archi in order to simulate the real-world data which will later also be exported from Archi. Modelling the data in Archi allows it to be exported into XML format and imported into the knowledge graph via an XML parser. This step aimed to prepare the system for handling the more complex and structured architectural models expected later.

Action Taken: Both the business object model and the cross-application data-flow diagram were recreated in Archi, in order to test how exported XML files will behave when parsing them into the knowledge graph. The parsing at this step was still being conducted via a custom, per-file LLM-based parser. This cycle mostly dealt with expanding the knowledge graph and fine tuning the system.

Evaluation: The progress of the third cycle was reviewed with both advisors during open discussions. The current state of the prototype and the extended data integration were assessed positively, and the overall research trajectory was again confirmed as appropriate. The end goal is in sight and is being worked towards in an appropriate manner.

The system had trouble generating qualitative answers for the newly imported dataset. This is due to the new data having a different schema than the other datasets.

Learning: The third cycle revealed scaling issues. While the individual per-file parsers have worked well so far, this cycle showed that its limits have been reached. Parsing the more complex files during this cycle showed that LLMs are no longer the appropriate method moving forward. On top of this, this custom, per-file parsing approach would also not allow for parsing files without manual implementation specific for the file at hand; a problem that has to be solved before the final review where files will have to be parsed on site and on the fly.

Another scalability issue revealed was a discussion about the real-world data. While SpeedParcel's data contained hundreds of nodes and edges, the real-world data will potentially have thousands of nodes and edges. This gap revealed that a better parsing solution and a better querying solution will be necessary before the prototype is ready for the final on site review. This will have to be tested before the on site review to ensure that this does not cause performance issues within the system.

8.3.4 *Cycle 4*

Diagnosis: The fourth and last cycle diagnoses that the system in its current state relies too heavily on LLMs for core precessing tasks. Importing the XML data into the knowledge graph is a critical point of the application and relying on an LLM to handle this parsing is not a good choice because XML is a standardized format and many parsers exist to handle XML. It was agreed upon that a generalized parser should be used to parse the incoming XML files into the knowledge graph, replacing the LLM at this critical point within the system's architecture.

A second critical part of the system that was diagnosed to be insufficient was the querying method to retrieve the information from the knowledge graph. Although the cyphers are being LLM-generated custom to the user's input, it is having trouble dealing with the changing schema of the knowledge graph because a hard-coded schema is written into the frontend which it uses as a reference for what to query. With an evolving knowledge graph, this approach is no longer viable because it is guessing at the graph's structure and not able to retrieve the data this way.

Action Planning: The goal of this cycle was to finish the prototype and have it be ready for the on site review at the end of the cycle where all three stakeholders will be testing and discussing the system together. Achieving this goal means that the XML data must be parsed via a generalized parser, replacing the LLM implementation. On top of that, the strategy of how to handle the schema must be refactored. It will be required to dynamically understand the data, rather than have one schema hard-coded. Lastly, it also meant setting up a containerized local environment to allow running the prototype on the advisor's devices during the on site review.

Action Taken: The LLM-based data parser was replaced by an out of the box solution from Awesome Procedures On Cypher (APOC), which contains functions to parse file types and also how to better query a knowledge graph. Here, APOC takes the input XML files and parses them directly into cyphers. This allows any type of Archi-exported XML file to be saved into the knowledge graph.

The schema-handling strategy was updated from the hard-coded variant. Before passing the user-prompt and system context to the LLM, a call to the knowledge graph was made via neo4j's built-in call `db.schema.visualization()` [30] in order to get the graph's schema information. While this did improve the generated results, it still proved to be insufficient, as the LLM was not able to handle this schema information well enough to generate reliable responses.

This was also solved via APOC. Rather than using neo4j's built-in schema call, the schema retrieval was changed to a predefined APOC function `apoc.meta.schema()` [31]. 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 is set up via Docker containers and can be run on any machine. It comes equipped with a frontend, backend, and two databases. The first database contains all data pertaining to SpeedParcel. The second database is a playground database and is only pre-filled with textbook information. This ensures that the on site review will allow the application to work.

Lastly, cosmetic changes to the UI were added as well as new features to support the on site review. For example, the user may now upload XML files via the web application as well as reset the playground database, and view the knowledge graph in an interactive way.

Evaluation: The refactored prototype was reviewed by both advisors via internal testing and demonstrations using the example data. The advisors assessed whether the system was executable, stable, and suitable for the final review. The replacement of the LLM-based XML parser with a generalized APOC-based parser and the updated schema-handling strategy were evaluated positively, as they enabled deterministic data imports and more reliable cypher generation. At the end of the fourth cycle, Masutā was deemed functionally complete and capable of answering all three categories of questions, thereby fulfilling the prerequisites for the final review.

Learning: Many learnings can be taken from this last cycle. The first being that a generalized parser to read XML files into the knowledge graph improves the quality of the import. Another added benefit of this is that the import is now deterministic, while before the LLM-based parser behaved in a non-deterministic way.

The second main learning from this cycle arose from refactoring the way that the queries get generated. Previously, the cyphers were straightforward to generate, as the schema of the SpeedParcel data was entirely known. However, during this cycle, new data was imported into the knowledge graph, which had an unknown structure, meaning the knowldge-graph's schema was unknown. The transformation of text to query has been improved by making a preliminary call to the database to fetch the current database schema. This schema information is then passed as context to the LLM in order to write more accurate cyphers. The results with this new querying strategy meant that for the first time the system was able to answer all three categories of questions.

With the conclusion of this cycle, Masutā was a finished prototype and ready to be tested during the final review.

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 be 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.

Table 8.1: 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.		

8.4 FINAL REVIEW

After the fourth and final Action Research cycle, a hands-on workshop was conducted as the final review of the prototype. The purpose of this review was to test Masutā using real-world EAM questions in order to observe and identify its behavior and capabilities 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 focusses exclusively on describing the setup and procedure of the final review. The evaluation of the results and implications can be found in chapter 12 and section 13.2.

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. Each participant interacted with the systems in an explorative manner.

For contextual comparison, the chatbot Rovo [3], which is integrated in Atlassian's knowledge management software Confluence, was used in parallel on the same set of data. This setup allowed the domain experts to contrast the behavior and outputs of both systems under identical conditions.

A new set of data was introduced specifically for the workshop. Rather than relying on the SpeedParcel data or synthetic data, the enterprise ar-

chitecture of the examination office at the Frankfurt University of Applied Sciences was used. This enabled evaluation in a realistic but controlled real-world scenario.

The workshop was purposefully conducted in an explorative nature, consistent with the applied Action Research approach. One domain expert interacted with Masutā, while the other interacted with Rovo. The researcher documented observations and supported the workshop with technical assistance for Masutā.

The four hour workshop was structured in five phases, each with a defined scope. After a short status overview, in which each participant aligned on the current state of the prototype's development and thesis, the first round of experiments began. The first phase focused on assessing the basic system functionality using simple domain questions. These questions were asked to both systems in parallel. 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. 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. The workshop concluded with a closing phase in which the domain experts summarized the findings, advantages of Masutā, and potential improvements for future development.

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.

As described in this chapter, 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 being clear and goal oriented. Each phase helped to examine what was possible from a technical standpoint as well as how to move forward. This supported the explorative nature of the project. The final review of the prototype proved that the 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.

9 FINAL REVIEW

This chapter describes 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 process through which the prototype was created. This gives the reader the understanding of how the research and implementation was conducted before discussing the implementation details in chapter 10.

9.1 ACTION RESEARCH DESIGN

Action research is a research method which is highly applicable when developing an information system such as the one presented in this paper [4]. The advantage of Action Research is that a large focus can be laid on the development of a system while still achieving an academic benefit. This allows for a very explorative approach to developing an information system.

Todo: describe why action research applies well to software development

Todo: add these challenges to the cycles:

- Chunking methoden beim Einlesen des Textbuches 21.10.25
- Die Datenstruktur des eingelesenen Textbuches 21.10.25
- Das abfragen der GraphDB zusammen mit dem LLM - zB wurden unendliche Rekursionen mit eingebaut am Anfang, weil Nodes auf sich im Kreis zeigen (schätze ich mal - nicht so wirklich verifiziert) 21.10.25

Cyclical phases are central to the concept of Action Research. Baskerville [4] describes Action Research as an iterative process consisting of five steps within a single cycle. Other sources, such as Cornish et al. [12], 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 [4, 12].

The paper at hand applied a cycle using the following five steps according to Baskerville [4]: 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..

Each cycle lasted between three and four weeks. The sources used 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 discuss 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 client 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. [4] From here on out, the term Participatory Action Research will be used interchangeably with Action Research.

The following subsections explain the research design of the applied Action Research. For the exhaustive Action Research protocol, refer to chapter A of the Appendix.

9.2 ACTION RESEARCH SETUP

The domain of EAM was focused on within this research. In particular, this study addressed mostly application landscapes, business capability maps, as well as the relationships between these two 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.

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 acted as an expert practitioner. The co-advisor acted as 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.

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.

9.3 ACTION RESEARCH CYLCES

The above mentioned vague details of the implementation became clear during the development cycles, leading to a final architecture with a clear structure. Each cycle began with a meeting between all three stakeholders. The end of one cycle was the incubator for the next cycle and was conducted in the same meeting. Table 9.1 summarizes these cycles.

Todo: note in the cycles where i attemtped implementation x, which didn't work out because of reason y. any path taken should be noted. also, note at what state the system as a whole was at the end of each cycle. you have this in the learnings for cycle 4 that it was able to answer all 3 categories of questions. do something similar for each cycle.

9.3.1 *Cycle 1*

Diagnosis: The initial problem statement lacked concrete technical formulation and the feasibility of natural language interaction with enterprise architecture data was unclear.

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 propsed of an AI-based black-box system capable of ingesting architecture data and deriving its own internal representations. 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. 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. This transparency aligns well with the mentioned advantages of a RAG-based LLM system in section 3.2 [51].

The key distinction between the black-box and white-box approaches is the transparency. While the black-box approach autonomously creates an

internal representation of the information, the white-box approach requires the manual design and implementation of an explicit technical architecture.

These discussions were necessary in order to scope the solution space. Early visions of an end goals included a chatbot that would support enterprise architects in exploring and improving application landscapes, for example by identifying inconsistencies or incomplete application landscapes.

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.

Action Planning: The first cycle aimed to develop a proof of concept that enables conversational access to a knowledge graph consisting of enterprise architecture knowledge grounded in textbook-based domain information. It was decided that a single-agent architecture will be used, as multi-agent architectures were considered unnecessarily complex for an initial proof of concept.

Action Taken: An initial knowledge graph was constructed based on content from the textbook *Masterclass Enterprise Architecture Management* [20]. The textual content was iteratively preprocessed and transformed into graph representations, with successive refinements applied to improve the mapping of domain concepts into nodes and relationships. This mapping was implemented using an LLM to parse the file and add it to the knowledge graph, as described by authors Laurenzi et al. (2024) [21] in section 3.2. The parser was individually fit to each file. After integrating the full textbook into the knowledge graph, additional prototypical data was incorporated in the form of an application landscape and business capability map to enable querying the knowledge graph of concrete architectural data. Each of these files also received a customized LLM-based parser.

The querying method was implemented via hard-coded cyphers in the frontend. The user's input prompt was then run through an LLM which attempted to fit the input into the predefined cyphers.

The LLM of choice during this stage was Qwen2.5-7B-Instruct [41] as it was free to use and readily available via the Hugging Pace platform.

Evaluation: The state of the proof of concept after the first cycle was demonstrated to both advisors and evaluated qualitatively through open discussions. 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.

Learning: Compared to the beginning of the cycle, in which the feasibility of a centralized knowledge graph was uncertain, the first cycle demonstrated that an explicit, white-box approach represents a practical path forward. The cycle also revealed key challenges in three critical layers of the application. These layers relate to importing heterogeneous data and transforming it into graph structures, the effective querying of such representations, and the slow response generation by the LLM. On top of this, the risk of relying too heavily on an LLM arose by giving full control of critical points of the

application to the LLM and thus being dependent on the capabilities of the language model.

Finally, it became evident that further iterations would be required to systematically address these challenges with an exploratory development process.

9.3.2 *Cycle 2*

Diagnosis: Following the initial feasibility assessment, the next identified challenge concerned extending the knowledge graph with more information and positioning the prototype as a useful tool within the realm of enterprise architecture management.

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 to enable direct relationships between conceptual textbook knowledge and architectural data, with the expectation of improved contextual reasoning.

The challenge imposed by lack of real-world data to test the system was also diagnosed. Potential test-datasets for further development and evaluation were discussed in order to bridge the gap before real-world data would be ready. The real-world data would require more time to be prepared because the co-advisor's company policy constraints meant the data cannot be used directly and has to be sanitzed first. Instead, it was agreed upon to use data from a completed university assignment, consisting of an application landscape, business capability map, business capability support matrix, business object model, and cross-application data-flow diagram for a fictitious company named SpeedParcel.

Action Planning: The objective of the second cycle was to extend the knowledge graph with additional data from the SpeedParcel dataset. This included integrating the business capability support matrix on top of the already existent application landscape and business capability map. A second goal of this cycle was to improve the database querying method as soon as more data is available to test with.

Action Taken: The capability support matrix of the SpeedParcel dataset was integrated into the knowledge graph, requiring adjustments to existing query mechanisms. A large challenge that hindered advancements in development came up. The hard-coded cyphers that were being used to query the database were reaching their limit. A viable solution was not found to hard-code cyphers in such a way that the changing dataset can be dynamically queried.

The idea of a Model Context Protocol (MCP) server came up during development. This brought the advantage of a standardized interface between the application and the knowlede-graph. Implementing the MCP server helped achieve a higher quality in the retrieved answers, as the prompt was being transformed into custom cyphers during runtime. This allowed the cyphers being used to query the database to no longer be hard-coded in the frontend,

but allowed them to be generated dynamically based on the user input. This text-to-cypher generation is achieved by using an LLM to transform the user's prompt directly into cyphers, as apposed to having predefined cyphers that get populated with the user's input by the LLM. A prerequisite for this to work, however, was that the schema of the knowledge graph was known. At this stage, the schema was hard-coded into the frontend.

Evaluation: The extended prototype was again evaluated qualitatively through a demonstration by the researcher and open discussions between all three stakeholders. Both advisors expressed strong interest in the approach and assured that the chatboat was being developed in the correct direction in order to achieve the end goal of allowing enterprise architects to interact with the knowledge graph via natural language.

Learning: As the prototype is starting to mature, more learnings are being pulled from each phase. Particularly, the hard-coded query method that was previously implemented was proved to be insufficient for flexible interaction with the evolving graph structure. The MCP implementation allows the entire system to be more dynamic, independent of the data in the knowledge graph.

Furthermore, three key categories of user questions were identified, similar to the ones described in section 3.2 [51]. The first category being conceptual questions related to enterprise architecture principles, concepts, and best practices found in textbooks. The second category being descriptive questions targeting concrete architectural elements and relationships. The third category being integrative questions combining conceptual knowledge with specific architectural contexts. Examples of these categories can be found in section **todo**.

Beyond the scope of this thesis, a broader vision by the advisors emerged in which the prototype could support project planning by assessing impacts on application interfaces, systems, and stakeholders. The co-advisor noted that the developments achieved thus far provide a starting point for exploring such directions in future work.

9.3.3 *Cycle 3*

Diagnosis: During this phase it became apparent that Action Research will be the most appropriate methodological framework for the project. The development process had evolved into an exploratory and iterative approach. This has been characterized by continuous development, reflection between all three stakeholders, and decision-making regarding which direction to take the prototype in. Consequently, the previously considered expert interview was no longer the methodology of choice for this thesis.

In parallel, the co-advisor began modelling real-world enterprise architecture data using the Archi tool. It was agreed upon that the real-world data would later be exported from Archi and imported into the knowledge graph in XML format. This meant that the format of the real-world data is finalized

and can be prepared for in the prototype, allowing the integration of more realistic data into the finished prototype.

Additionally, it was decided that a final review will be conducted at the end of the fourth cycle. This review would require that the prototype be finished and be executable in a local environment. This review will also mark the end of the development phase of the prototype.

Action Planning: The objective of the third cycle was to further expand the information saved in the knowledge graph by creating additional datasets from the SpeedParcel dataset. The datasets existed in a third party architecture modelling tool and had to be replicated in Archi in order to simulate the real-world data which will later also be exported from Archi. Modelling the data in Archi allows it to be exported into XML format and imported into the knowledge graph via an XML parser. This step aimed to prepare the system for handling the more complex and structured architectural models expected later.

Action Taken: Both the business object model and the cross-application data-flow diagram were recreated in Archi, in order to test how exported XML files will behave when parsing them into the knowledge graph. The parsing at this step was still being conducted via a custom, per-file LLM-based parser. This cycle mostly dealt with expanding the knowledge graph and fine tuning the system.

Evaluation: The progress of the third cycle was reviewed with both advisors during open discussions. The current state of the prototype and the extended data integration were assessed positively, and the overall research trajectory was again confirmed as appropriate. The end goal is in sight and is being worked towards in an appropriate manner.

The system had trouble generating qualitative answers for the newly imported dataset. This is due to the new data having a different schema than the other datasets.

Learning: The third cycle revealed scaling issues. While the individual per-file parsers have worked well so far, this cycle showed that its limits have been reached. Parsing the more complex files during this cycle showed that LLMs are no longer the appropriate method moving forward. On top of this, this custom, per-file parsing approach would also not allow for parsing files without manual implementation specific for the file at hand; a problem that has to be solved before the final review where files will have to be parsed on site and on the fly.

Another scalability issue revealed was a discussion about the real-world data. While SpeedParcel's data contained hundreds of nodes and edges, the real-world data will potentially have thousands of nodes and edges. This gap revealed that a better parsing solution and a better querying solution will be necessary before the prototype is ready for the final on site review. This will have to be tested before the on site review to ensure that this does not cause performance issues within the system.

9.3.4 *Cycle 4*

Diagnosis: The fourth and last cycle diagnoses that the system in its current state relies too heavily on LLMs for core precessing tasks. Importing the XML data into the knowledge graph is a critical point of the application and relying on an LLM to handle this parsing is not a good choice because XML is a standardized format and many parsers exist to handle XML. It was agreed upon that a generalized parser should be used to parse the incoming XML files into the knowledge graph, replacing the LLM at this critical point within the system's architecture.

A second critical part of the system that was diagnosed to be insufficient was the querying method to retrieve the information from the knowledge graph. Although the cyphers are being LLM-generated custom to the user's input, it is having trouble dealing with the changing schema of the knowledge graph because a hard-coded schema is written into the frontend which it uses as a reference for what to query. With an evolving knowledge graph, this approach is no longer viable because it is guessing at the graph's structure and not able to retrieve the data this way.

Action Planning: The goal of this cycle was to finish the prototype and have it be ready for the on site review at the end of the cycle where all three stakeholders will be testing and discussing the system together. Achieving this goal means that the XML data must be parsed via a generalized parser, replacing the LLM implementation. On top of that, the strategy of how to handle the schema must be refactored. It will be required to dynamically understand the data, rather than have one schema hard-coded. Lastly, it also meant setting up a containerized local environment to allow running the prototype on the advisor's devices during the on site review.

Action Taken: The LLM-based data parser was replaced by an out of the box solution from Awesome Procedures On Cypher (APOC), which contains functions to parse file types and also how to better query a knowledge graph. Here, APOC takes the input XML files and parses them directly into cyphers. This allows any type of Archi-exported XML file to be saved into the knowledge graph.

The schema-handling strategy was updated from the hard-coded variant. Before passing the user-prompt and system context to the LLM, a call to the knowledge graph was made via neo4j's built-in call `db.schema.visualization()` [30] in order to get the graph's schema information. While this did improve the generated results, it still proved to be insufficient, as the LLM was not able to handle this schema information well enough to generate reliable responses.

This was also solved via APOC. Rather than using neo4j's built-in schema call, the schema retrieval was changed to a predefined APOC function `apoc.meta.schema()` [31]. 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 is set up via Docker containers and can be run on any machine. It comes equipped with a frontend, backend, and two databases. The first database contains all data pertaining to SpeedParcel. The second database is a playground database and is only pre-filled with textbook information. This ensures that the on site review will allow the application to work.

Lastly, cosmetic changes to the UI were added as well as new features to support the on site review. For example, the user may now upload XML files via the web application as well as reset the playground database, and view the knowledge graph in an interactive way.

Evaluation: The refactored prototype was reviewed by both advisors via internal testing and demonstrations using the example data. The advisors assessed whether the system was executable, stable, and suitable for the final review. The replacement of the LLM-based XML parser with a generalized APOC-based parser and the updated schema-handling strategy were evaluated positively, as they enabled deterministic data imports and more reliable cypher generation. At the end of the fourth cycle, Masutā was deemed functionally complete and capable of answering all three categories of questions, thereby fulfilling the prerequisites for the final review.

Learning: Many learnings can be taken from this last cycle. The first being that a generalized parser to read XML files into the knowledge graph improves the quality of the import. Another added benefit of this is that the import is now deterministic, while before the LLM-based parser behaved in a non-deterministic way.

The second main learning from this cycle arose from refactoring the way that the queries get generated. Previously, the cyphers were straightforward to generate, as the schema of the SpeedParcel data was entirely known. However, during this cycle, new data was imported into the knowledge graph, which had an unknown structure, meaning the knowldge-graph's schema was unknown. The transformation of text to query has been improved by making a preliminary call to the database to fetch the current database schema. This schema information is then passed as context to the LLM in order to write more accurate cyphers. The results with this new querying strategy meant that for the first time the system was able to answer all three categories of questions.

With the conclusion of this cycle, Masutā was a finished prototype and ready to be tested during the final review.

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 be 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.

Table 9.1: 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.		

9.4 FINAL REVIEW

After the fourth and final Action Research cycle, a hands-on workshop was conducted as the final review of the prototype. The purpose of this review was to test Masutā using real-world EAM questions in order to observe and identify its behavior and capabilities 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 focusses exclusively on describing the setup and procedure of the final review. The evaluation of the results and implications can be found in chapter 12 and section 13.2.

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. Each participant interacted with the systems in an explorative manner.

For contextual comparison, the chatbot Rovo [3], which is integrated in Atlassian's knowledge management software Confluence, was used in parallel on the same set of data. This setup allowed the domain experts to contrast the behavior and outputs of both systems under identical conditions.

A new set of data was introduced specifically for the workshop. Rather than relying on the SpeedParcel data or synthetic data, the enterprise ar-

chitecture of the examination office at the Frankfurt University of Applied Sciences was used. This enabled evaluation in a realistic but controlled real-world scenario.

The workshop was purposefully conducted in an explorative nature, consistent with the applied Action Research approach. One domain expert interacted with Masutā, while the other interacted with Rovo. The researcher documented observations and supported the workshop with technical assistance for Masutā.

The four hour workshop was structured in five phases, each with a defined scope. After a short status overview, in which each participant aligned on the current state of the prototype's development and thesis, the first round of experiments began. The first phase focused on assessing the basic system functionality using simple domain questions. These questions were asked to both systems in parallel. 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. 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. The workshop concluded with a closing phase in which the domain experts summarized the findings, advantages of Masutā, and potential improvements for future development.

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.

As described in this chapter, 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 being clear and goal oriented. Each phase helped to examine what was possible from a technical standpoint as well as how to move forward. This supported the explorative nature of the project. The final review of the prototype proved that the 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.

10 IMPLEMENTATION

The methodology chapter answers the question of how things were iteratively built and why. 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.

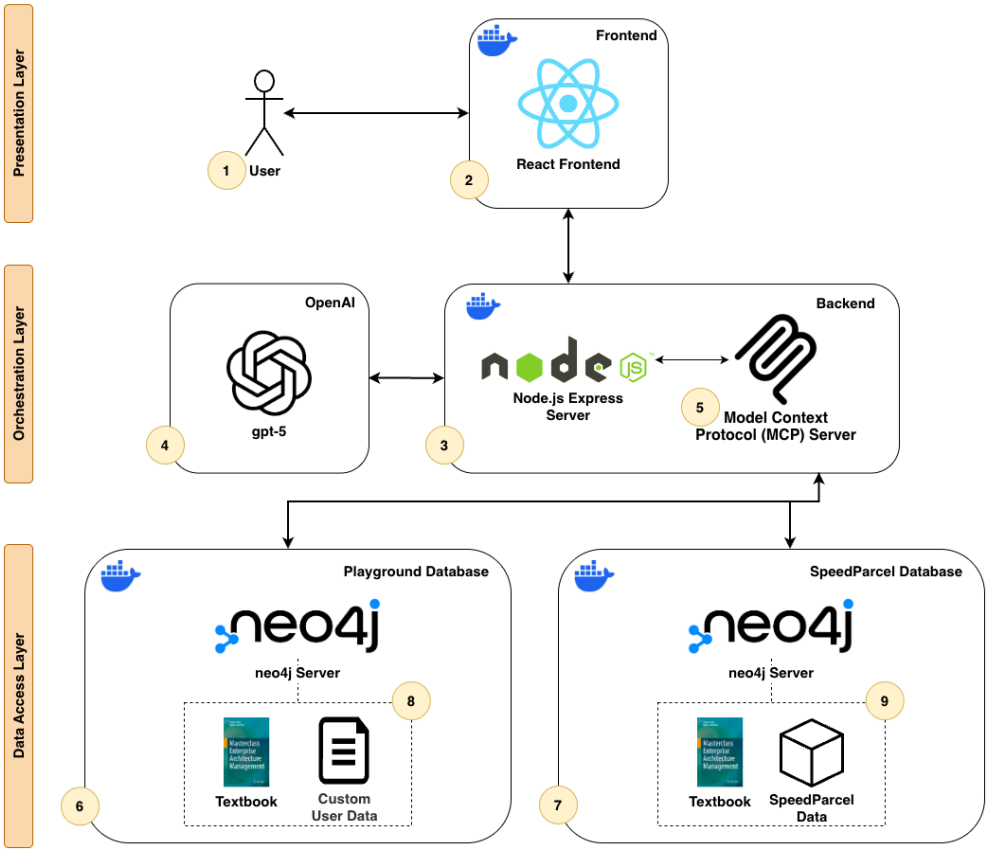


Figure 10.1: An architecture diagram showing each component and which components interact with one another. The Docker icon also indicates if the component is containerized.

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 [6]. 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 [9]. 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 B.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 [33].

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 [47]. 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. [47] 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. [43, 47]

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()` [31]. 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 [27].

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 C.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 9.3. 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 [49]. 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 [49].

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 [43].

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 [34]. 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, 18]

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. [29] 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 9.3. 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 [20]. 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 B.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 B.

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 9.3. 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. [36, 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 [24]. 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 [48]. 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.

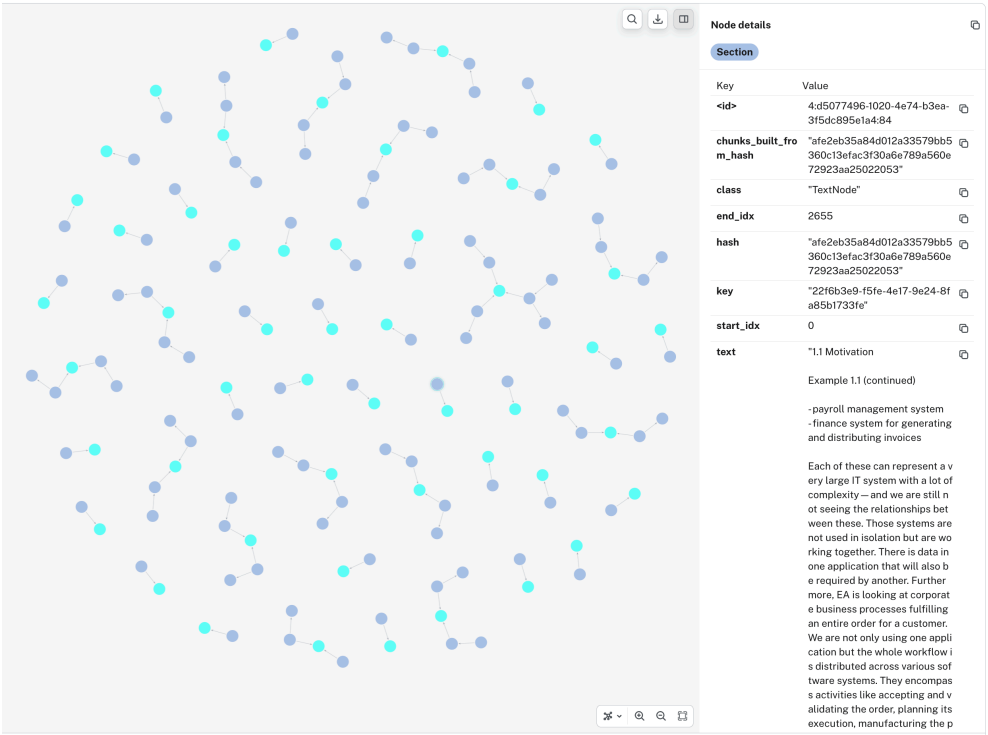


Figure 10.2: A screenshot of a subsection of the textbook's knowledge graph. It shows how different node types (e.g. documents and sections) are connected within their own sub-graphs. On the right hand side is the meta-data for a selected node. Notice the "text" label within the meta-data where it is 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. [11] XPath is used within APOC's predefined procedure `apoc.load.xml()` to allow XML to be parsed and exposed for cypher processing [28].

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. [28] 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. [28]

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.

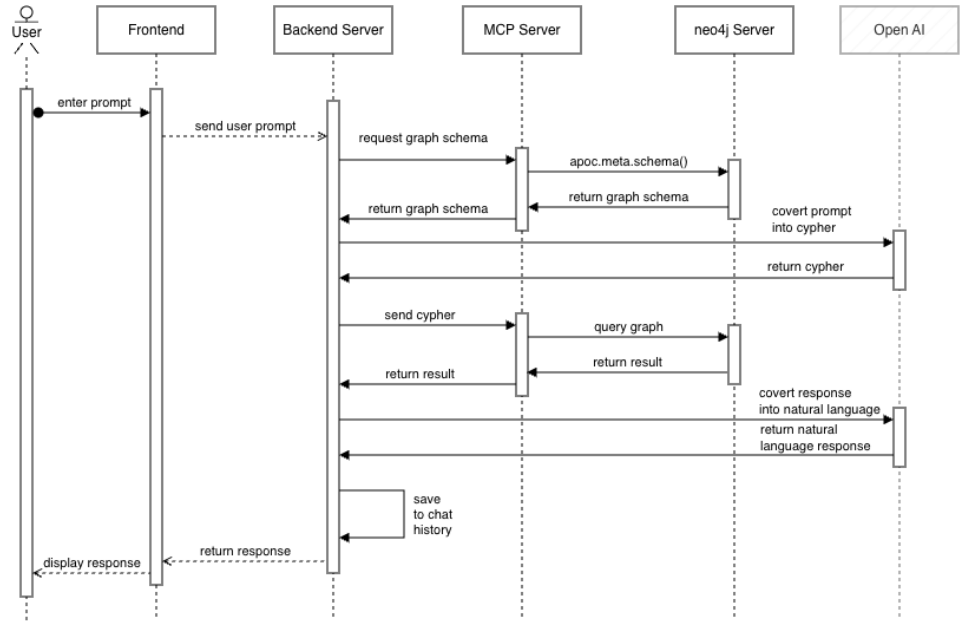


Figure 10.3: A UML sequence diagram showing the main flow of data when a user prompts the system.

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 11 elaborates case studies to justify the prototype.

11 REVIEW WORKSHOP WITH EXPERTS

With an understanding of the prototype's architecture from chapter 10, this chapter dives into the test cases that Masutā was put through during the final review from section 9.4. This chapter describes the context of the case study, how the system was prepared, the tests executed, and how everything was evaluated.

11.1 CASE CONTEXT

Masutā was put to the test by two domain experts within the realm of enterprise architecture management. This ensured that the questions being asked of the system mirrored a real world setting. The goal of the conducted case study was to evaluate the final system end-to-end and compare its results to a similar, mainstream system. 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 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 strengths and weaknesses within Masutā.

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.

11.2 SYSTEM SETUP AND DATA

The finished prototype system as described in chapter 10 was used throughout. Masutā was installed on both practitioners computers and came out of the box with the SpeedParcel example dataset and the empty playground database which only contained 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, the enterprise architecture for the examination office of Frankfurt University of Applied Science was used. This architecture included several diagrams within the Archi modeling tool, 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 appendix section B.4.4. Frankfurt University of Applied Science's architecture is described in German. The imported data from the examination office's architecture diagrams contained a total of 92 nodes and 152 edges.

Parallel to Masutā, Rovo was being used with the same ingested architecture data loaded into Confluence. This allowed the practitioners to test the same dataset with a second system. It included the same documentation and architecture data as found in Archi.

11.3 CASE EXECUTION

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

11.4 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."

11.5 SUMMARY OF CASE OBSERVATIONS

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."

What we did with the finished prototype and how we tested it. Prompts we used, cases we built, edge cases, etc.

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.

Idea: create a list of questions over each category with expected answers and see how it performs. also run x amount of experiments and see how

many errors came up (e.g. cypher errors) and count the percent of answers that were perfect, good but could be improved, and wrong / faulty.

12 RESULTS AND DISCUSSION

Now that it is clear how the study was conducted and

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 [21] and [40].

possible improvements:

- The schema call before every prompt is costly for large schemas (i assume as of 02.01.26). This source from wan [43] 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: [25]
- 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 [51] 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: [38]
- 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: [51] 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. [10]
- 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 Archi-Type in den Metadaten stehen. das muss verbessert werden!

13 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.

13.1 CONCLUSION

main learnings include

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

13.2 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] Atlassian. *Meet Rovo, AI that knows your business*. https://www.atlassian.com/software/rovo. Accessed: 2026-01-27. 2026.

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

[5] Phillip Beauvoir and Jean-Baptiste Sarrodie. *Archi: Open Source ArchiMate Modelling Tool*. Accessed: 2026-02-08. 2026. URL: https://www.archimatetool.com/.

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

[7] 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).

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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/.

[14] 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.

[15] 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).

[16] 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.

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

[18] 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).

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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).

[28] 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).

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

[30] Neo4j, Inc. *Procedures*. Neo4j Operations Manual documentation including the `db.schema.visualization` procedure. 2025. URL: https://neo4j.com/docs/operations-manual/current/procedures/#procedure_db_schema_visualization (visited on 01/01/2026).

[31] 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).

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

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

[34] 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).

[35] 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).

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

[37] 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.

[38] 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).

[39] 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.

[40] 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.

[41] Qwen Team. *Qwen2.5: A Party of Foundation Models*. 2024. URL: https://qwenlm.github.io/blog/qwen2.5/.

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

[43] 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.

[44] 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.

[45] 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.

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

[47] 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).

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

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

[50] 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.

[51] 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

ACTION RESEARCH PROTOCOL

This chapter contains the full Action Research protocol.

A.1 PRECONDITIONS

Test: This reference should have a lowercase, small caps A if the option

A.2 CYCLE 1

Start: October 17th, 2025. End: October 31st, 2025

Diagnosis: The initial problem statement lacked concrete technical formulation and the feasibility of the natural language interaction with enterprise architecture data was unclear. The co-advisor described in detail what his daily work looks like and what tools are used at his company to document and interact with enterprise architecture data. He also showcased the existing solution Dragon1 (described in more detail in section todo).

The co-advisor had the idea of creating an AI, acting as a black-box system, that ingests the specific enterprise architecture data and creates its own internal knowledge graph. All as a black-box. The feasibility as well as academic applicability of this was questioned intensely because of the difficulty of testing such a black-box system when nothing is known about the innern mechanisms.

The academic supervisor recommended an approach using a knowledge graph and supplementing a language model with the domain specific information. He also recommended tools such as Microsoft's Copilot Studio. The academic supervisor also allowed the use of his textbook "Masterclass Enterprise Architecture Management" [20] to feed the knowledge graph with information specific to the role of an enterprise architect.

The difference between the black-box and white-box (knowledge graph) versions is that the black-box system would build its inner mechanisms itself in order to generate domain specific answers. In the white-box version the technical architecture would have to be built manually.

These steps were required in order to scope the potential solution and to think about what possible data sources could be.

First rough ideas were a chatbot with which an enterprise architect could make changes to the application landscape. This included being able to complete unfinished or incomplete application landscapes. For example, being able to interact with the chatbot to identify inconsistencies and weak spots in the landscape.

Action Planning: The goal of the first phase was clear: create a proof of concept. This meant setting up a basic LLM, feeding it with general enter-

prise architecture data and context so that it is trained to see the interaction through the eyes of an enterprise architect. On top of this a small UI to be able to interact with the proof of concept chatbot. The end-goal of this proof of concept chatbot was to be able to make recommendations on how to edit an application landscape while still covering all business capabilities. The goal until the next cycle would be to be able to ask a chatbot general questions about enterprise architecture management and receive answers based off of the textbook information fed into a knowledge graph.

It was agreed upon, that a single-agent architecture will be used, as multi-agent architectures come with more complexity and are not ideal for this use case.

At this stage, it was not clear that Action Research will be applied during development. At this stage it was assumed that the finished prototype will be passed through an expert interview.

Action Taken: Setup a neo4j database, created a graph representation of the first chapter of "Masterclass EAM". Created a React.js frontend which was directly connected to the graph database. Setup the first prompt context to query the database. Confirmed that the supplemented answers used the knowledge graph by comparing the textbook to the generated answers. Improved embeddings into the graph database (better nodes and relationships). Transforming the textbook into graph representations was done using a python parser which was created with the help of ChatGPT. Improvements on parsing the textbook into a graph representation. Read in the rest of the chapters from the textbook. Created a capability support matrix in excel and parsed it into the database and adapted the queries for this.

Evaluation: Both advisors were ecstatic to see the progress so far after giving a demo of the current state. The co-advisor agreed that building a knowledge graph is the correct path forward after having seen the proof of concept in action.

Learning:

A.3 CYCLE 2

Start: October 31st, 2025. End: November 28th, 2025 **Diagnosis**: a next idea would be to feed in more textbooks. idea would be to compare my results to an existing copilot such as the Microsoft Copilot Studio. An idea came up that it might make sense to have 1 database for the textbook information and 1 database for the enterprise architecture data. we agreed to use one database in hopes of achieving better results as nodes between the textbook and the enterprise data may be connected together in the single database, achieving better results. This is also the phase, where we started discussing potential data sets. the problem here being that the co-advisor is not allowed to simply export his company data and give it to me. we also started discussing how to evaluate my system. we discussed potential experiments and evaluation methods - including testing the knowledge graph directly. we agreed that i would use the data from the assignment of our uni's course "Enterprise

Architecture Management" which i completed in the previous semester. this dataset contains an application landscape and a business capability map for a fictitious company named "SpeedParcel".

Action Planning: we agreed that in this cycle, i would continue implementing more data from speedparcel - new data includes a business object model and a cross-application data-flow diagram. the goal until the end of the cycle would be to improve the querying and have more data in the knowledge graph to query. the "real world data" will come at a later point and only speedparcel data will be used in the next phase.

Action Taken: development went like this: read in the capability support matrix for speedparcel and fit the queries to work with this. i was faced with the challenge that it was impossible to query the data correctly because the schema was getting complex and i needed a better solution than having hard-coded cyphers. after discussing this with a friend, he recommended an MCP server to me. i implemented this and also created a custom node.js backend. UI changes were also done. with the MCP server, responses were quicker and more precise because the cyphers were now being built via LLM (text-to-cypher method) instead of being hardcoded.

Evaluation: Both advisors were very satisfied with the MCP server and the improvements made. both were glowing with ideas on how to continue and how this can be applied in the real world. the co-advisor confirmed that being able to interact with a chatbot is a great way to gain information about the architectural landscape of an enterprise, especially as the data gets more and more complex.

Learning: we identified that there are three categories of questions that the chatbot may be faced with. category 1 being questions specifically in the realm of textbook data, e.g. "how does an enterprise architect do x, y, or z?". Category 2 being simple questions directed towards the architecture, e.g. "how many capabilities does application x support?". Category 3 being a cross section between 1 and 2, e.g. "what do we have to look out for when replacing application x?"

possible use cases for my application came up during discussions about the daily work of an enterprise architect. for example: if an application gets removed and replaced, then there isn't just the technical overhead, but also the application owners who are being replaced. meaning, enterprise architecture also has a lot to do with people management. a seemingly simple change can lead to a complex situation because of stakeholders.

my co-advisor has a large-scale vision for my prototype (this goes beyond my thesis) that it can help during project planning. for example, when planning to remove or add an application, which interfaces need to be touched, which stakeholders will be affected, etc.

A.4 CYCLE 3

Start: November 28th, 2025. End: December 17th, 2025

A.5 CYCLE 4

Start: December 17th, 2025.End: January 23rd, 2026

A.6 FINAL REVIEW

Meeting date: January 23rd, 2026. Location: Campus of the Frankfurt University of Applied Sciences.

B

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).

B.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.

B.2 FIRST TIME SETUP

B.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.

B.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.

B.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.

B.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.

B.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.

B.4 FEATURE OVERVIEW

This section details the features within Masutā.

B.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.

B.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.

B.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.

B.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 B.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.

B.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.

B.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.

B.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 B.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.

B.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 B.4.4.

B.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.

B.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.

C

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.

C.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.

C.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(...).
  "
};
```

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

todo

C.1.3 *Visualizing Knowledge Graphs in Neo4j*

todo: show some nice blooms of the knowledge graphs.

D

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.

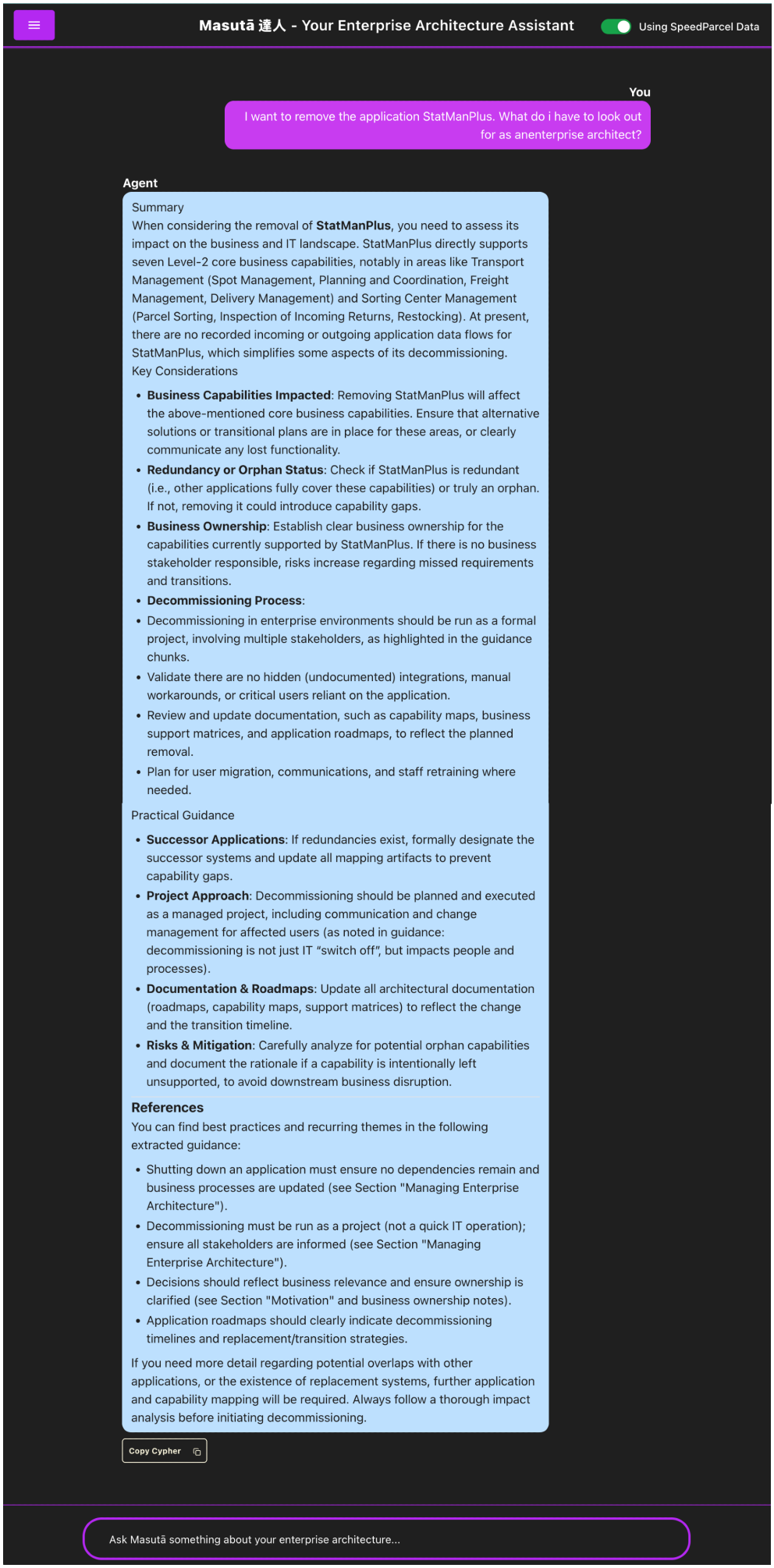
D.1 QUESTION XYZ

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,
```

```
38        app_chunks AS chunks_referencing_application,
          guidance_chunks AS chunks_with_decommission_guidance
```

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