Hochschule Konstanz
Department of Computer Science

Submitted by

Sebastian Voigt
Student Number: 297637
sebastian.voigt@htwg-konstanz.de

Masterthesis

An LLM-Based Multi-Source Retrieval Agent for Enhanced Knowledge Management

Konstanz, 29.04.2025

Hochschule Konstanz
Department of Computer Science

Masterthesis

An LLM-Based Multi-Source Retrieval Agent for Enhanced Knowledge Management

by

Sebastian Voigt

in partial fulfillment of the requirements for the degree of

Master of Science

in Computer Science

at the HTWG Konstanz - University of Applied Sciences.

Hochschule Konstanz
Department of Computer Science

Student Number:	297637
Date of Start:	01.11.2024
Date of Submission:	29.04.2025
Supervisor:	Prof. Dr. Ing. Oliver Haase
Second Examiner:	Dr. Benjamin Heitmann

Acknowledgements

I am grateful to my supervisors Prof. Dr. Ing. Oliver Haase, Dr. Benjamin Heitmann, and Thomas Vamos for their invaluable insights and unwavering patience throughout this journey.
My deepest thanks go to my parents, Dorothee and Gunter Voigt, whose financial and emotional support enabled me to complete my studies. I am particularly grateful to my father, Gunter, for his unwavering support. I regret that he was not able to see this journey to its conclusion.
I would also like to thank my classmates and friends for their enduring patience and for listening to my endless discussions on topics that may not have interested everyone.

This research is conducted in collaboration with the SEITENBAU GmbH. The choice of data sources reflects practical constraints and opportunities inherent to this professional setting. I would like to express my gratitude to SEITENBAU for providing this opportunity and their invaluable support throughout the research process.

I acknowledge the use of large language models (LLMs), such as models by OpenAI and Mistral, in assisting with the refinement of grammar and vocabulary throughout this thesis. The model was employed as a proofreading tool to ensure the text is concise, clear, and adheres to academic standards. All outputs were critically reviewed and adapted to align with my own writing style and intent.

Abstract

In recent years, large language models (LLMs) have gained significant popularity, seamlessly integrating into our daily lives by assisting with a multitude of tasks - from composing emails to providing insightful recommendations. However, their effectiveness tends to decline in professional environments, where the need for precision and access to internal information is paramount.

Trained primarily on publicly available data, using an LLM to retrieve proprietary information can lead to falsehoods, inaccuracies, or fictitious details, thereby creating a trust deficit in business applications. Although solutions such as hosting private LLMs, fine-tuning, and retrieval augmented generation (RAG) approaches show promise, they often face limitations in adaptability and robustness.

In response to these challenges, we propose a novel architecture focused on LLM-based autonomous agents, that specializes in retrieval and aggregation of proprietary information. Capable of navigating complex problem-solving scenarios and leveraging diverse tools for data retrieval makes them the ideal choice for business-critical applications.

Contents

Glossary

***AI* – artificial intelligence** vii, 1, 14, 16, 18, 36, 37, 38, 39, 40, 41, 44, 48, 50

***ANN* – approximate nearest neighbor** 13

***API* – application programming interface** 11, 15, 21, 22, 23, 25, 26, 27, 32, 42, 51

***CI* – continuous integration** 36

***CoT* – chain-of-thought** 10, 31, 33, 34, 35

***CQL* – Confluence query language** 10, 22, 23, 24, 31, 32, 47

***IQR* – interquartile range** vii, 38, 39, 40, 44, 45, 46

***LLM* – large language model** i, ii, vii, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 18, 20, 21, 22, 23, 24, 25, 27, 30, 32, 33, 34, 35, 36, 37, 41, 42, 43, 45, 46, 47, 48, 49, 50, 51

***MVC* – model-view-controller** vii, 25, 26

***NLP* – natural language processing** 7, 12, 22

***NN* – neural network** 7

***OSS* – open source software** 17, 18

***PoC* – proof-of-concept** vii, viii, 21, 25, 27, 36, 38, 40, 42, 43, 44, 45, 46, 47, 49, 51, 64

***RAG* – retrieval augmented generation** ii, vii, 7, 13, 14, 18, 19, 20, 21, 22, 23, 32, 48, 49, 50, 51

***RCT* – randomized control trial** 36, 49

***SQL* – structured query language** 20

***UI* – user interface** vii, 26, 27, 37, 45, 48, 51

***UX* – user experience** 4, 36, 37, 38, 41, 48, 51

List of Figures

List of Tables

1 Introduction

Searching for information requires significant effort, which is why librarians serve as invaluable guides within the structured environment of libraries. They help users navigate complex collections and efficiently locate the resources they need.

However, modern workplaces rarely offer either a dedicated library or the expertise of a librarian. Instead, information resides in fragmented repositories: knowledge bases, ticketing systems, documents, shared drives, and code repositories. Gathering answers from these disconnected sources - especially when synthesis and cross-referencing are required - quickly becomes a daunting task. This fragmentation makes it difficult to form a comprehensive picture and increases the risk of overlooking crucial information, particularly when precision and completeness matter most.

As organizations accumulate knowledge authored by many contributors across time, the challenge grows. Subject matter experts may be unavailable, have left the organization, or forgotten critical details. This proliferation and scattering of information leads to wasted time, reduced productivity, and growing frustration among employees (Stack Overflow 2024).

1.1. Thesis Goal

Building on the librarian analogy, this thesis aims to create a digital librarian-an AI-powered assistant that reimagines information retrieval beyond traditional keyword search. Rather than forcing users to sift through results or manually aggregate data, the system leverages advanced large language models (LLMs) to deliver accurate, context-aware answers. Users can pose questions in natural language and receive concise, actionable responses, reducing reliance on manual searching or colleague input.

For example, a user might simply ask, "*How do I set up my development environment?*" The system interprets the intent, identifies relevant documents, and summarizes the required steps. To ensure transparency and trust, it also cites its sources, enabling users to verify or explore details further.

1.2. Research Questions

This thesis addresses key research questions that highlight current limitations in LLM-based organizational knowledge applications. These questions shaped our approach and evaluation strategy.

1. Can an LLM-based system provide relevant and accurate information based on internal document storage from a particular institution?
   i. Can we instruct an LLM to construct queries that reliably retrieve information from an internal knowledge base?
   ii. Can we instruct an LLM to merge and rank information from multiple sources?
   iii. How can we guarantee consistent access control with the institutions document storage?
   iv. Can the system supply this information in a timely manner?
2. Can an LLM-based system iteratively refine and control a complex workflow to progressively improve the accuracy and relevance of a reply?
   i. Can we instruct the system to retrieve necessary information autonomously and iteratively refine its queries based on feedback or new data?
   ii. How do the LLM-based system's results compare to state of the art keyword search?

1.3. Outline

We begin by describing the problem landscape and articulating the requirements that motivated this thesis. Next, we review the relevant literature and technical background, examining how others have addressed similar challenges and surveying the technologies at our disposal.

Building on this foundation, we present our proposed system, detail initial experiments, and analyze their outcomes. The implementation chapter delves into the architecture and engineering decisions behind our digital librarian.

In the evaluation chapter, we explain how we tested the implementation against our requirements and research questions, as outlined in Section 2.2 and Section 1.2. Finally, we summarize our key findings and discuss their implications in the concluding chapter.

2 Use Case and Requirements

This chapter presents a technically focused use case that demonstrates the relevance of this thesis, followed by a thorough definition of the requirements necessary to fulfill its objectives.

We begin with a concrete, domain-specific scenario, highlighting how such a system can effectively address challenges that would otherwise demand significant manual effort or remain unsolved. Subsequently, we outline the key requirements-organized into relevant categories-that serve as benchmarks for evaluating the thesis outcomes.

2.1. Use Case

Software engineers frequently encounter questions such as, "*Does software X provide feature Y, and if so, where is it located?*" or "*Have we already implemented a function that solves problem Z?*" These questions typically require searching through documentation and code repositories using keywords. When keyword searches fail to yield results, engineers often seek help from colleagues.

Answering these questions requires combining information from multiple sources - a process that can be time-consuming and frustrating. According to the 2024 Stack Overflow Developer Survey, 53% of developers agree with the statement, "*Waiting on answers to questions often causes interruptions and disrupts my workflow*" (Stack Overflow 2024).

This challenge extends beyond software engineering and affects knowledge workers in many fields. To address this issue, this thesis investigates solutions that streamline access to and integration of information across different sources.

By defining explicit requirements in Section 2.2, we systematically evaluate available technologies in Section 4 and introduce our proposed solution in Section 6. Ultimately, we aim to enhance productivity and workflow continuity for knowledge workers, minimizing interruptions caused by inefficient information retrieval.

2.2. Requirements

This section defines the formal requirements essential to this thesis. Each requirement serves as a concrete, measurable metric to evaluate potential solutions and the final implementation. These requirements are grouped into the following categories: general, data security, performance, and integration.

It is important to note that these requirements differ from the research questions outlined in Section 1.2. While the research questions provide broad, natural language formulations of the core challenges and goals addressed in this thesis, the requirements translate these questions into specific, operational criteria that we can objectively measure and assess. This distinction ensures that our evaluation, as presented in Section 7, remains both rigorous and directly aligned with the thesis objectives.

2.2.1. General Requirements

Inspired by the helpful librarian analogy from Section 1, we distill three general requirements essential for this thesis. These requirements draw from the desired user experience (UX) and are mostly binary in nature - each must be either present or absent.

Requirement	Description
Iterative Refinement	The system should progressively refine a query or goal, adjusting responses based on its observations.
Semantic Search	Enable semantic, meaning-based search beyond simple keyword matching.
German	Ensure that the LLM can understand and respond fluently in German.

Table 1: General Requirements

2.2.2. Data Security

Data security is paramount for LLM applications, especially those that interact with sensitive environments. In the proposed system, the agent accesses proprietary company data, requiring robust safeguards to prevent unauthorised access or data leakage. Strict adherence to data security requirements is therefore essential to maintain confidentiality and integrity at all times. These are also binary in nature.

Requirement	Description
Confidentiality	Prevent unauthorized access to sensitive company data.
Access Control	Enforce rules governing who can access specific data and permitted actions.
Vendor Independence	Prevent dependency on a single technology stack by supporting modular, switchable integrations. Some cloud-based providers lock in their LLM, which can complicate maintenance or migration-especially when upgrades affect output quality or performance.

Table 2: Data Security Requirements

2.2.3. Performance

As the agent interfaces directly with users, its performance directly impacts productivity and satisfaction. We define concrete performance requirements that our implementation must achieve. These aspects are further quantified and evaluated using the framework outlined in Section 7.

Requirement	Description
Query Response Time	Deliver query responses so that users perceive the exchange as quick.
Query Accuracy	Provide responses that consistently address users' information needs.
Ease	Ensure that the search process is intuitive and requires minimal effort from the user.
Economy	Maintain a positive cost-benefit ratio, balancing performance and operational expenses.

Table 3: Performance Requirements

2.2.4. Integration

A core function of the proposed agent is the retrieval of relevant information from diverse sources, making seamless integration into existing infrastructures essential. This section enumerates the requirements for connecting to internal (non-public) data sources and ensuring compatibility with available tools. Most of these requirements are binary, with measurable aspects assessed during evaluation.

Requirement	**Description**
Speed	Retrieve relevant documents quickly, contributing to prompt query response times.
Multi-Source	Concurrently access and integrate information from multiple sources.
Customizability	Allow seamless addition of new or custom data sources as requirements evolve.
LLM Infrastructure	Leverage the organization’s existing LLM provider while enabling future changes with minimal reconfiguration.

Table 4: Integration Requirements

3 Background

This chapter provides the relevant theoretical background, with an emphasis on the current research. It commences by introducing the foundation of modern natural language processing (NLP), specifically embeddings and LLMs. We then take a brief detour to introduce semantic search, a concept crucial for subsequent technologies. Following this introduction, we delve into retrieval augmented generation (RAG), a process that enables an LLM to dynamically incorporate information into its generation process. Finally, we discuss LLM-based autonomous agents, a recent development that allows an LLM to reason about its environment and execute complex actions.

3.1. Embedding

In mathematics, an embedding describes a function f that maps an $x \in X$ to a $y \in Y$, $f : x \to y$ while preserving certain topological or geometric properties. In the context of NLP and this thesis, we embed input text x into a vector space Y while preserving the semantic meaning of x (Bengio et al. 2003). Unlike mathematical embeddings, which preserve algebraic or geometric operations, NLP embeddings aim to capture semantic relationships between linguistic units such as words, phrases, or sentences (Bengio et al. 2003; Mikolov et al. 2013a).

Embedding models are typically preceded by tokenizers, which split input text into sub-word tokens to handle out-of-vocabulary words effectively and reduce sparsity in the representation (Chaofan Tao et al. 2024). A tokenizer ensures that each piece of text can be represented as a unique token while trying to keep the dimensionality of each token as low as possible (Mikolov et al. 2013a; Chaofan Tao et al. 2024). Modern embedding models are neural network (NN) trained on tokens, popularized by research on Word2Vec (Mikolov et al. 2013b; Z. Wang et al. 2024).

The embedding y is typically a dense vector, which reduces the dimensionality of x compared to its original representation, thereby enabling more efficient computation while retaining meaningful semantic information (Bengio et al. 2003). Preserving semantic meaning allows y to be used for efficient comparisons, i.e., to calculate distance in terms of meaning (Lewis et al. 2020; Gao, Galley, and Li 2019). This means that the angle between two embeddings indicates how “related” two pieces of text are; the closer their semantic meanings, the smaller the angle between their embeddings.

In practice, embedding models are often presented to users as sentence models. This means they produce an embedding y for an entire string x, instead of generating a list of embeddings for each token in x. Conceptually, the output of these models is akin to the hidden state of a LLM, particularly those based on the transformer architecture. Research has explored leveraging this similarity (Chongyang Tao et al. 2024), but sentence models typically generate outputs through pooling (Reimers and Gurevych 2019) or by training separate attention-based models (Li et al. 2020).

3.2. Large Language Models

Modeling human language has a long history in computer science, with the current state of the art being LLMs (Zhao et al. 2024; Z. Wang et al. 2024). The transformer architecture underpins modern LLMs, using multi-head attention to model complex token relationships. Initially introduced to enhance machine translation (Vaswani et al. 2023), LLMs have since improved by increasing training data size, model parameter counts (Z. Wang et al. 2024; Zhao et al. 2024) and specializations in terms of architecture (DeepSeek-AI et al. 2025; OpenAI et al. 2024).

The generation process of LLMs involves several key steps, introduced by (Vaswani et al. 2023). First, the input text is tokenized into smaller units, such as words or sub-words, and these tokens are embedded into dense vector representations that capture their semantic relationships, as described in Section 3.1. These vectors are then processed through transformer blocks, where attention mechanisms dynamically weight token relationships, allowing each token to focus on contextually relevant parts of the sequence. Simultaneously, feed-forward networks refine each token’s representation in a non-linear manner. At the final stage, the model produces a probability distribution over the entire vocabulary, with each token assigned a likelihood based on its contextual fit.

At the end of the generation process, a single token is selected using temperature-adjusted sampling, where lower temperatures yield more deterministic outputs and higher temperatures encourage greater diversity. The chosen token is appended to the input sequence, and the process repeats iteratively until a special stop token or a predefined length limit ends the generation. This architecture enables LLMs to generate text that balances contextual coherence with controlled randomness.

One of the big limitations of LLMs is, that they have a limited context window, which varies significantly between models (ranging from 2048 to 128000 tokens (Zhao et al. 2024)). This limitation means the model can only consider a finite amount of preceding text when generating new tokens, which can affect its ability to maintain long-term coherence in generated text. This is particularly problematic in tasks that require understanding and referencing information from earlier parts of the sequence.

3.2.1. Prompting

Prompting is "*the act of trying to make someone say something*" (Cambridge University Press and Assessment 2025). In the context of LLMs, prompting describes *the act of instructing the model on what it should produce* and a prompt is the instruction in this process (Liu and Chilton 2022; Reynolds and McDonell 2021; White et al. 2025).

Defining a prompt for a general-purpose model, trained on a vast corpus of public information, instead of training a dedicated model, is known as prompt-based learning (P. Liu et al. 2023; Reynolds and McDonell 2021). The process of designing prompts and searching for prompts that produce the desired outcome is called prompt-engineering or prompt-programming (Reynolds and McDonell 2021; Liu and Chilton 2022). This approach reduces the need for extensive training data and model-specific tuning, making it more efficient than traditional supervised learning methods in many tasks (P. Liu et al. 2023).

For example, in Listing 1, *input* is a word, phrase, or sentence that needs to be translated into another *language*. Demonstrating how prompt-based learning can be applied to various tasks without the need for task-specific model training.

```
Please translate '{input}' into {language}.
```

Listing 1: Simple translation prompt

We will use the notation, used by (P. Liu et al. 2023), to describe current prompting techniques. This means, that that we denote the original input as x, which is modified by a prompt (also referred to as a template) x' to construct the final string that is given to the LLM $\hat{x}$. The resulting output y is given by $P(y|\hat{x})$.

The most basic technique to instruct an LLM is called *zero-shot prompting*, which only provides a simple task without context or examples (Kojima et al. 2022). An example of this can be seen in Listing 1, where the prompt provides just the instruction to translate text into another language without additional context or examples. Zero-shot prompting is enough for many simple tasks, such as translation of small passages of text or retrieval of general information (Schulhoff et al. 2025; Petroni et al. 2019).

For complex tasks, one can leverage the emergent ability of large LLMs, known as in-context learning (Zhao et al. 2024). This technique involves adding examples to x', which helps the model solve complex tasks (Brown et al. 2020; Zhao et al. 2024; Z. Wang et al. 2024). Researchers refer to this method as few-shot prompting (Brown et al. 2020). For example, Listing 2 is a few-shot prompt that includes examples of how to construct a query in Confluence query language (CQL) from input questions.

```
Question: Did we implement feature bar in product foo?
Query: text ~ "foo" and text ~ "bar~"
Question: Is foo a benefit at my current employer?
Query: text ~ "benefit" or text ~ "foo"
Question: {x}
Query:
```

Listing 2: Example of a few-shot prompt that gives the model examples on how to construct a query in CQL. We go into more detail on CQL in Section 5.2.

Further improvements to the output quality of a LLM can be achieved by employing chain-of-thought (CoT) prompting, introduced by (Wei et al. 2023). This technique includes a step-by-step reasoning process (the CoT) in the prompt, which the LLM then uses to solve complex problems. As this requires more manual labour and prompt-engineering, zero-shot CoT, which adds phrases like "*Let's think step by step.*" to the prompt, can be used instead. This compels the model to generate intermediate steps in its output which improves performance in many tasks (Kojima et al. 2022; Schulhoff et al. 2025).

More recently, models like OpenAI's o1 and DeepSeek's R1 were published, which are specifically designed to generate reasoning traces by default. This means that they generate CoT reasoning without explicit prompts, allowing them to break down complex problems into logical steps. This capability is achieved through advanced training techniques such as reinforcement learning, which encourages the development of multi-step reasoning behaviors (OpenAI et al. 2024; DeepSeek-AI et al. 2025).

One important limitation to be aware of, when prompting LLMs, is the recency bias they exhibit (Zhao et al. 2021). This means their performance decreases when a large input is provided after the instructions and/or examples. This is because LLMs tend to prioritize recent text. To mitigate this issue and improve prompt reliability, it is good practice to place large inputs before the instruction section of the prompt (Zhao et al. 2021; Ma, Yang, and Kästner 2024).

3.2.2. Structured Output

There are several categories and constraints of structured output for an LLM (Liu et al. 2024). The most basic form involves a well-formed prompt that enforces a certain, basic format, such as "… *use markdown to format your reply.*". This, in combination with basic constraints like an approximate length or selecting an option from a list, defines low-level constraints (Liu et al. 2024).

High-level constraints of structured output include hallucination prevention, stylistic choices of text, and, most importantly for this thesis, semantic constraints (Liu et al. 2024). The semantic constraints we are interested in allow the definition of a grammar that restricts the output generated by an LLM to follow a specific code/structure. This is often used to generate valid JSON but can also define basic code structures for programming languages (Li et al. 2025).

Several methods restrict the output of an LLM to adhere to a certain structure. These methods include fine-tuning (Rozière et al. 2024; B. Liu et al. 2023), prompting/in-context learning (Wang et al. 2023), or algorithmic solutions (Koo, Liu, and He 2024). More commonly, and relevant to this thesis, the output of an LLM is restricted by a pre-defined grammar that masks the output distribution of an LLM to only allow tokens that can fulfill the grammar (Dong et al. 2024; Scholak, Schucher, and Bahdanau 2021).

This approach defines a grammar that guarantees a parsable output from the LLM and is often integrated into the application programming interface (API) hosting the LLM (vLLM Team 2025; OpenAI 2025). This means that a client can use this feature by providing a prompt, in combination with a description of the output format, often a JSON schema (vLLM Team 2025; OpenAI 2025).

While limiting the LLM to generate text under these restrictions can limit output quality (Tam et al. 2024), it also enables the parsing of the output and its use to trigger certain actions. Additionally, the requirement for structured output can enforce a certain way of thinking and consideration on the part of the LLM. By constraining the output format, users can guide the model's reasoning process, potentially leading to more focused and relevant responses. This approach can be particularly useful in scenarios requiring precise, actionable outputs, such as in decision support systems or automated workflow processes (Liu et al. 2024). One such use-case, where it is paramount to generate very specific tokens, is tool use, further explained in the next section.

3.2.3. Tool Use

The priciples established for structured ouput enable LLMs to generate parameters for functions, which they can subsequently invoke. This process, known as tool use, empowers LLMs to autonomously execute user-defined actions, expanding their functionality beyond text generation. For instance, an LLM can generate the necessary parameters to query a database (Sun et al. 2024; Xue et al. 2024; Hu et al. 2023) or call a specific API (Shen et al. 2023; L. Wang et al. 2024).

This capability allows the model to interact with external systems and retrieve or manipulate data as needed. Moreover, tool use can be employed to perform complex calculations, access up-to-date information, or even control external devices, thereby greatly enhancing the model's problem-solving capabilities (L. Wang et al. 2024).

3.3. Semantic Search

Semantic Search, a concept that has gained attention in information retrieval and NLP, lacks a universal definition and is interpreted differently across various domains of expertise. At its core, however, Semantic Search can be broadly understood as "search with meaning" (Bast, Buchhold, and Haussmann 2016, p. 120). This section explores two key interpretations of Semantic Search that are relevant to this thesis.

The first interpretation focuses on knowledge representation for meaningful retrieval. Defined as "[…] representing knowledge in a way suitable for meaningful retrieval" by (Bast, Buchhold, and Haussmann 2016, p. 119). This approach involves storing documents in a search-friendly manner, often utilizing advanced data structures. For instance, vector databases store documents in the form of embeddings as introduced in Section 3.1 (Lewis et al. 2020). These embeddings can be compared to a similarly embedded query to find documents that are semantically close to the query, thus enabling more nuanced and contextually relevant search results.

The second interpretation emphasizes query understanding, defined as "[…] understanding the query (instead of just finding matches of its components in the data)" by (Bast, Buchhold, and Haussmann 2016, p. 119). This approach transcends traditional keyword-based search by introducing a higher level of abstraction. The system attempts to discern the user's true intent behind the query and performs a search based on this interpretation. For example, if a user asks, "How do I set up my development environment?", the system might interpret this query to search for terms such as "development environment", "setup", "configuration", "tools", and "best practices".

3.4. RAG

LLMs excel at answering questions based on well-known public knowledge by storing vast amounts of data (Petroni et al. 2019). For example, they can provide useful responses to queries like "What are the most important sights in the capital of France?" by giving information on the Eiffel Tower, the Louvre, the Champs-Elysées, etc.. However, LLMs struggle with questions requiring specific, non-public information, such as "What is the policy for creating passwords in the company I'm working in?". In such cases, an LLM may, in the best case, acknowledge it lacks that specific information and suggest where a user might find it. In the worst case, it may generate inaccurate or hallucinated responses (Petroni et al. 2019).

While LLMs can be fine-tuned to incorporate non-public data, this process is often costly and impractical in rapidly evolving environments (Bommasani et al. 2021). As an alternative, RAG has emerged as a promising solution. It enables LLMs to access and utilize up-to-date, non-public data that was not part of their original training (Lewis et al. 2020). RAG aligns with the first interpretation of semantic search discussed in Section 3.3, leveraging embeddings for efficient information storage and retrieval.

As shown in Figure 1, the RAG framework consists of two primary components: the generator and the retriever (Lewis et al. 2020). The generator is essentially an LLM used for text generation, while the retriever implements a semantic search based on embeddings. Before the retrieval process begins, documents are preprocessed and stored in a vector database. This preprocessing stage involves converting documents into embeddings to capture semantic meaning.

During the retrieval process, the retriever employs the same embedding model to transform the user's query into a vector representation (Singhal 2001). This allows for efficient comparison between the query and stored documents using approximate nearest neighbor (ANN) algorithms. These algorithms identify a set of documents that are semantically 'similar' to the query based on the distances between their vector representations (Gao, Galley, and Li 2019). They allow for the efficient searching of large sets of vectors for similarity using techniques such as clustering (Dhillon, Fan, and Guan 2001). The retrieved documents are then used to augment the LLM's knowledge, enabling it to generate more accurate and contextually relevant responses.

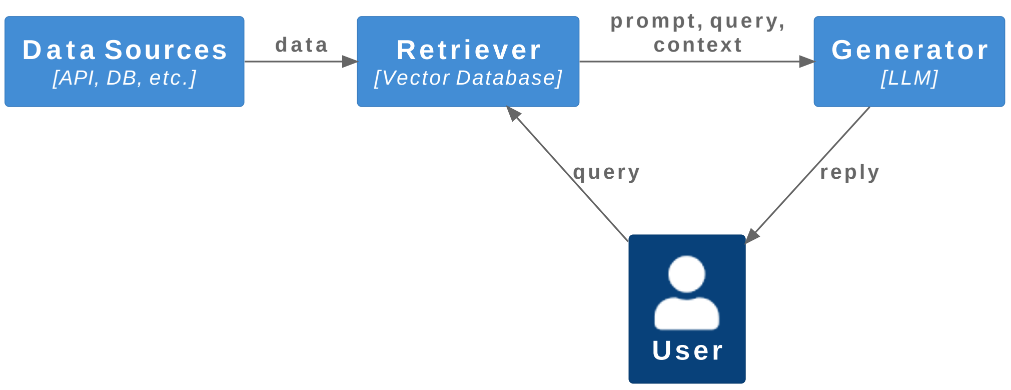


Figure 1: The diagram shows a RAG system, where a user queries a retriever, which interacts with a generator (LLM) to generate a reply. This approach allows to include arbitrary information into the generation process, leverage in-context learning. Relationship descriptions are shown on the right or on top of an arrow.

3.5. Agents

```
An autonomous agent is a system situated within and a part of an environment
that senses that environment and acts on it, over time, in pursuit of its own
agenda and so as to effect what it senses in the future.
```

Listing 3: Definition of autonomous agents (Franklin and Graesser 1996)

Agents are a fundamental concept with a long history in computer science and especially AI research (Russell and Norvig 2016, p. 59 – p. 61). This thesis expands the traditional definition, shown in Listing 3, to include recent advancements in LLMs, introducing the concept of LLM-based (autonomous) agents. For our purposes, an LLM-based autonomous agent, hereafter referred to as an “agent,” is an LLM-based system that leverages advanced cognitive processes such as planning and memory to execute actions, interacting with its environment, in pursue its goals based on gathered information. This definition aligns with the current research (L. Wang et al. 2024; Huang et al. 2024; Xi et al. 2025).

In this framework, the LLM serves as the primary decision-maker, determining which tools to use and how to employ them effectively. It can also engage in self-reflection, analyzing and refining its previous outputs. In the following sections, we will introduce the components an agents consists of and a simple paradigm, that can be used to design agents.

3.5.1. Components

Agents are often described by a set of components that define their functionality. This section explores a set of components commonly identified in the current literature as the basis for agents (Xi et al. 2025; L. Wang et al. 2024). The framework, which is better aligned with this thesis, captures the same concepts with fewer components, making it more applicable and better aligned with the paradigm described in Section 3.5.2 (Xi et al. 2025).

In (Xi et al. 2025), an agent is defined by three main components: the brain, the perception, and the action. Each component fulfills a specific task, together forming the foundation of a functional agent.

Brain The brain acts as the primary decision maker of the agent. It is primarily composed of an LLM and is responsible for planning, reasoning, decision-making, memory management, and generalization/transferability. Recent developments in LLMs enable these features to be handled by a single, general, pre-trained model. This component can be seen as a combination of the memory module and planning module in alternate research (L. Wang et al. 2024).

Perception Humans and other animals use sensory organs to perceive their surroundings. To enable an agent to do the same, we can leverage different formats and sources to inform its decisions and actions. Text is the most obvious input type for LLMs, as they are traditionally trained to work with it. Recent advancements in multi-modal LLMs allow some models to work with visual and auditory input, but this is not relevant to this thesis. This component corresponds to the environmental exploration part of the agent module in alternate research (L. Wang et al. 2024).

Action After perceiving the environment and integrating, analyzing, and reasoning on the data, the agent can execute an action. The most basic action of an agent is textual output, which is the fundamental function of an LLM. More interesting is tool use, described in Section 3.2, where an LLM generates parameters for a function, which it can then execute. This allows direct interaction with the environment, such as API calls, code execution, and environmental exploration. This is the job of the action module in alternate research (L. Wang et al. 2024).

3.5.2. ReAct

Inner speech, as the foundation for integrating task-oriented actions with verbal reasoning, has been extensively studied (Lev Semyonovich Vygotsky 1987; Alderson-Day and Fernyhough 2015). Building on this concept, "ReAct: Synergizing *Re*asoning + *Act*ing" by (Yao et al. 2023) introduces a significant agent paradigm. This paradigm is grounded in the established idea of designing AI agents around thoughts, actions, and observations (Russell and Norvig 2016, p. 1044 – p. 1051; Mitchell 1990; Laird, Newell, and Rosenbloom 1987).

The ReAct paradigm operates on a simple, circular model, that aligns well with the three components described in Section 3.5.1. In this model, a reasoning step leads to an action, which generates observations. These observations then inform subsequent reasoning, creating a continuous loop of interaction and adaptation. This approach enhances the agent's ability to make informed decisions and adapt to dynamic environments.

Thought Component This component constructs and updates plans using verbal reasoning traces. It represents the profile module and maintains short-term memory through generated text and observed data. By leveraging short-term and long-term memory, it formulates plans to achieve goals and executes actions accordingly.

Action Component This component interacts with the environment to generate observations that feed back into the thought process. The continuous feedback loop enables agents to refine plans and actions iteratively.

ReAct-based agents dynamically adapt to complex tasks by integrating reasoning and acting, which maintains robust decision-making capabilities over time. Equipped with short-term memory (planning output, executed actions, and observations) and long-term memory (access to databases and other environmental data sources), along with a well-defined, task-specific profile and clearly defined actions, ReAct agents can effectively solve complex tasks.

4 Related Work

This section reviews existing implementations, technologies, and projects that align with the goals and requirements outlined in Section 2. We analyze commercial products, open source software (OSS), and development frameworks relevant to the thesis, focusing on their features, limitations, and suitability for the intended use case. All reviewed solutions employ technologies discussed in Section 3, providing a representative - though not exhaustive - overview of the current landscape.

We categorize the solutions into commercial products, open-source software, and development frameworks. Each group offers unique advantages and challenges, which we discuss below.

4.1. Commercial Products

Commercial software typically provides contractual commitments from providers, ensuring support, ongoing development, and reliable bug fixes - key factors for organizations considering adoption. Driven by recent market trends, including the launch of Rovo (Reuters 2025), we selected Atlassian Rovo as a representative example.

Rovo[1] integrates tightly with Atlassian's suite, streamlining knowledge discovery and workflow efficiency. Built on Atlassian Intelligence[2], Rovo connects to third-party applications through Connectors[3], letting users access organizational knowledge swiftly and securely.

A key feature of Rovo is its agent-based system. Users can create, share, and modify agents that interact with organizational data to automate tasks, reducing repetitive manual work (Atlassian 2025a). These agents excel at activities like scheduling, milestone tracking, and documentation updates. Rovo appeals especially to organizations within the Atlassian ecosystem or those relying on supported third-party tools.

However, as specified in Section 2.2, critical factors - including seamless infrastructure integration, vendor independence, and strict confidentiality - require careful consideration. Despite its strengths, commercial solutions like Rovo present notable limitations:

[1]https://www.atlassian.com/software/rovo
[2]https://www.atlassian.com/platform/artificial-intelligence
[3]https://www.atlassian.com/software/rovo/connectors

Vendor Lock-In Organizations may become dependent on a single provider, hindering future migration.

Cost Developing on a proprietary platform incurs ongoing expenses and potential hidden costs.

Compliance Stringent regulatory environments or organizational policies may prohibit use; for instance, Rovo does not meet HIPAA requirements (Atlassian 2025b).

Such drawbacks make Rovo less suitable for use cases prioritizing flexibility, vendor independence, or strict compliance. Similar constraints apply to other platforms, such as SAP AI Agents[1] and Salesforce Agentforce[2].

In summary, while commercial tools like Rovo streamline workflows within their target ecosystems, their vendor dependency and opaque data practices may disqualify them for organizations requiring adaptability and full control.

4.2. Open-Source Software

OSS offers transparency, flexibility, and freedom from licensing costs, making it an attractive choice for organizations prioritizing customization and control. We examined PrivateGPT[3] as a prominent example, blending community-driven development with optional paid services for enhanced support and features (PrivateGPT 2023).

PrivateGPT, with over 55,000 stars on GitHub[4], operates under the Apache License 2.0[5] and offers enterprise support via licensing. These attributes make it a strong candidate for enterprise integration.

PrivateGPT delivers an AI assistant powered by RAG, handling private data such as internal documents across various deployment models - local, on-premises, or cloud-based. It supports custom retrieval sources and multiple LLMs, letting users import bespoke models as needed. Key features include:

Privacy-focused architecture Sensitive data remains within the user's infrastructure for maximum security.

Model flexibility Supports various LLMs and user-defined models.

Deployment options Operates locally or in the cloud.

Comprehensive RAG workflows Enables document retrieval and contextual response generation.

[1]https://www.sap.com/products/artificial-intelligence/ai-agents.html
[2]https://www.salesforce.com/agentforce/
[3]https://privategpt.io/
[4]https://github.com/zylon-ai/private-gpt
[5]https://github.com/zylon-ai/private-gpt/blob/main/LICENSE

While PrivateGPT simplifies access to large document collections, it introduces notable operational challenges:

Preprocessing Overhead Users must preprocess, chunk, and embed documents before retrieval.

Data Maintenance Keeping document embeddings current requires ongoing effort.

Access Control Complexity Sophisticated access policies are difficult to enforce, as many vector databases lack advanced authorization mechanisms.

These limitations complicate PrivateGPT's use in environments requiring strict data security or real-time document updates, especially where continuous embedding synchronization is necessary.

Currently, no open-source solutions offer agent-based workflows as robust alternatives to RAG.

4.3. Development Frameworks

Open-source development frameworks contrast sharply with commercial platforms, emphasizing transparency, adaptability, and community-driven improvements at the expense of increased development and maintenance requirements. As neither commercial nor pure open-source solutions met our use-case, we focused on LangGraph - a leading development framework with broad community support and extensive features.

LangGraph[1], extending the LangChain[2] ecosystem, introduces graph-based workflows for modeling agent interactions as nodes and edges. This architecture enables precise control over state transitions and cyclical processes, distinguishing LangGraph from competitors like Microsoft AutoGen's conversation-oriented agents and CrewAI's role-based collaboration. While AutoGen[3] excels in code-generation with containerized execution and CrewAI[4] offers accessible agent role configurations, LangGraph prioritizes explicit workflow modeling.

LangGraph's seamless integration with LangChain brings compatibility with major LLM providers, a wealth of tool integrations, and robust experiment tracking through LangSmith. The framework's MIT license and active community - including over 3,000 contributors - ensure both academic freedom and professional-grade support (LangChain Team 2024a).

[1]https://github.com/langchain-ai/langgraph
[2]https://github.com/langchain-ai/langchain
[3]https://github.com/microsoft/autogen
[4]https://github.com/crewAIInc/crewAI

LangGraph is particularly well suited for implementing ReAct-based architectures, as its persistent state management supports the interleaved reasoning and acting cycles of this paradigm. While frameworks like Agno[1] excel in RAG tasks or Composio[2] enables hierarchical planning, LangGraph strikes a unique balance between low-level workflow control and high-level abstractions. Academic users benefit from thorough documentation spanning concept guides, API references, and deployment strategies - critical when developing novel agentic workflows.

4.4. Text-to-SQL

Recent research empowers LLMs to construct reliable queries in structured query language (SQL), establishing LLMs as robust tools for database interaction. These advances enable users without technical expertise - as well as autonomous systems - to interact seamlessly with SQL databases (Singh et al. 2025). By supporting dynamic data retrieval and manipulation, integration strategies overcome key limitations of LLMs, such as restricted memory and static, outdated knowledge (Hu et al. 2023; Xue et al. 2024).

Researchers initially extended the accessible memory of pre-trained LLMs by incorporating symbolic memory modules and external storage solutions (Hu et al. 2023). Subsequent studies improved SQL query generation by applying fine-tuning and few-shot prompting techniques to LLMs (Sun et al. 2024). Most recently, novel architectures that combine query generation with retrieval-augmented generation (RAG) have further enhanced performance and efficiency in database access tasks (Xue et al. 2024).

[1]https://github.com/agno-agi/agno
[2]https://github.com/ComposioHQ/composio

5 Approach

Building on the literature detailed in Section 3 and related work in Section 4, we systematically evaluated alternative approaches for the proposed PoC. This chapter outlines our two primary strategies, highlights their respective strengths and limitations, and justifies our chosen implementation, which we present in detail in Section 6. We focus our approach on RAG and agent-based solutions.

5.1. RAG

RAG immediately demonstrated a strong fit for our problem, providing a straightforward and robust means to enhance the LLM's generation capabilities. Numerous libraries and platforms support the creation and deployment of powerful RAG workflows. For our initial implementation, we selected LangChain, using QDrant as a vector database and Confluence as our primary data source[1].

5.1.1. Benefits

RAG enabled rapid prototyping and provided critical insights into its practical value. Because setup remained minimal, this approach proved ideal for single-user applications and local deployments favoring simplicity.

LangChain's `ConfluenceLoader` streamlined the downloading, chunking, and embedding of Confluence spaces (LangChain Team 2024b). Additionally, its QDrant integration facilitates efficient on-disk storage, allowing fast iterative development and experimentation (LangChain Team 2025). This seamless pipeline allowed us to prototype and assess RAG quickly, making ease of use and rapid iteration two of its main advantages.

5.1.2. Challenges

Challenges emerged in dynamic environments where data changes frequently, multiple users require distinct access rights, or the embedding model must be updated. Ensuring

[1]SEITENBAU facilitated the integration of various data systems. We prioritized Confluence due to its stable API, rich documentation, and clear structure. Later, we integrated Rocket.Chat, which also offers a robust API but presents a different document structure. The selection of these tools was based on availability, feature set, and research value, with no constraints imposed by SEITENBAU.

data freshness involves constant monitoring, removing outdated records, and re-chunking and embedding new content. While this suffices for sources like Confluence with periodic updates, messaging platforms like chat systems require near-real-time synchronization - a nontrivial hurdle.

Access control introduces further complexity. With modern institutions employing intricate authorization schemes, maintaining consistent and secure data access grows increasingly difficult as sources accumulate.

Finally, updating or replacing the embedding model necessitates reprocessing the entire dataset. Each model upgrade demands full re-embedding of all documents, incurring significant computational costs and discouraging frequent improvements - thereby limiting the adoption of the latest NLP advancements.

5.2. Agents

After examining RAG pipelines, we extended our research to LLM-based autonomous agents. We started with query-based agents, drawing from our observations on related work, discussed in Section 4.4. Following the methodology from Section 5.1, we integrated Confluence as an initial data source and developed a suite of tools for agent interaction with the LLM.

This agent-driven approach streamlines both user authorization and data freshness. By connecting the agent to a Confluence client authenticated through the user's API token, we guaranteed that agent access always matched the user's permissions and reflected the latest available information.

5.2.1. Initial Approach

We prompted the LLM to formulate search queries in Confluence query language (CQL), as shown in Listing 4. This allowed targeted searches for specific content types and locations using the Confluence API (Atlassian 2019).

Crucially, CQL supports ranked fuzzy search. For example, the query in Listing 4, "`foobar~`", gives priority to exact matches of "*foobar*" over partial matches like "*foo*", but still returns both (Atlassian 2017).

```
text ~ "foobar~" AND type = page AND space = SPACE
```

Listing 4: A simple CQL query, fuzzily searching the word "foobar" in all pages within the space "SPACE"

We equipped a pre-built ReAct agent (via LangGraph) with two main tools: the CQL search and a summarizer that generates responses from retrieved content.

This setup highlighted several limitations in contemporary LLMs and generic tool-use agents. The LLM occasionally skipped searches, hallucinated results, or repeated actions until surpassing iteration limits. Because the generic agent generated both the tool choice and its parameters in a single step, it often produced imprecise queries and irrational tool selection.

Additionally, integrating with the Confluence API posed extra hurdles. Query results included only page references and brief excerpts. We then applied LLM-based ranking to these excerpts, selecting only the top n pages for download. However, this process sometimes missed relevant content or delivered imprecise rankings, indicating a clear need for improvement.

5.2.2. Improved Tool

In response, we improved our CQL tool by combining features from query-based agents with the embedding-based ranking used in RAG.

Leveraging Confluence's Ranking We adopted Confluence's built-in ranking to order pages by relevance based on the CQL query. This strategy allows the agent to download the top $n = 100$ pages in full, rather than relying solely on short excerpts.

Chunking and Embedding The tool then chunks and embeds these pages, comparing each embedding with the user's request to further refine ranking. This hybrid approach blends embedding-based ranking-similar to RAG-with dynamic queries to yield richer results.

Selection and Summarization We then narrow the selection to the top $k = 10$ ranked pages using the computed embeddings. An LLM summarizes each shortlisted page in the context of the user's query, filtering out irrelevant information and reducing prompt size.

This multi-step enhancement produced more accurate and robust retrieval, better aligning outputs with user needs. Despite these improvements, the pre-built ReAct agent continued, at times, to invent information, neglect available tools, or redundantly repeat actions.

5.2.3. Graph Architecture

To address these shortcomings, we experimented with constraining agent autonomy. Instead of granting full control, we designed a deterministic workflow[1] that relies minimally on LLM-driven decisions. This agent strictly follows these steps:

1. Extract the user's intent and identify relevant keywords.
2. Formulate and execute a CQL query with the identified keywords.
3. Terminate the workflow if results are sufficient or the iteration limit is reached; otherwise, continue searching.

We describe the implementation, based on this initial approach, in Section 6.

[1]LLMs are inherently non-deterministic, especially at higher temperatures. Determinism here applies solely to the outlined workflow.

6
Implementation

This section details the implementation of the PoC, focusing on system architecture, agent orchestration, and prompt engineering. You can access the complete PoC code in a git repository[1], with an overview of the code structure provided in Appendix 01.

6.1. System Architecture

Figure 2 illustrates the system architecture, which adopts the model-view-controller (MVC) pattern. The frontend, built with `Vue.js`, serves as the view layer and guides users through an interface with three primary states. It communicates with the API, implemented in Python using FastAPI. The API exposes a single endpoint that streams agent events and delivers a final message to the user. The agent, acting as the model, interacts with the infrastructure and emits messages using LangGraph's `adispatch_custom_event`. This design enables the agent to share user-relevant events efficiently.

To fulfill security and integration requirements (Section 2.2), the agent accesses on-premise infrastructure, including LLMs, embedding models, and data sources. Additionally, in its current form, the agent is deployed as a local Docker container, further discussed in Section 6.1.1. This means that every user must set their own credential, in the form of API tokens, locally. This averts the risk associated with confidentiality and access control.

[1]https://github.com/VoigtSebastian/master-thesis/tree/submission

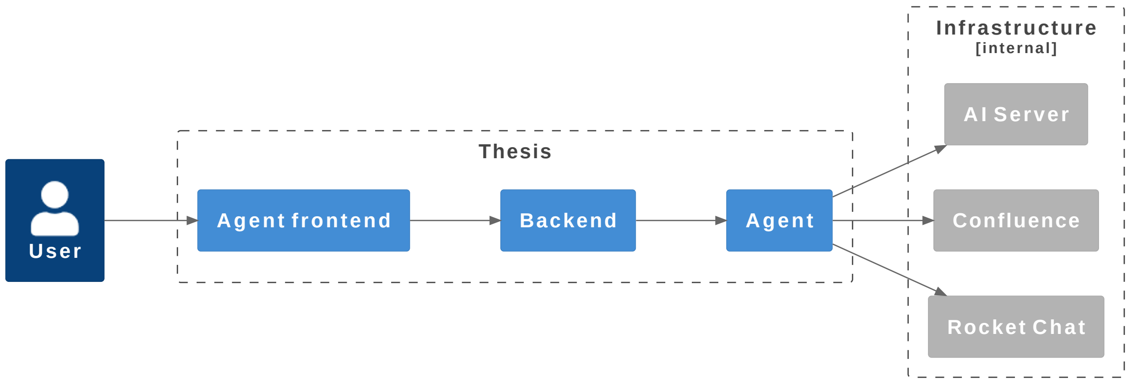


Figure 2: System Architecture based on MVC. Relationship descriptions appear beside or above the arrows.

Users begin by submitting input through a text field (Figure 3). The API receives this input and creates the necessary configuration, state, and agent instances, transitioning the UI to its second state.

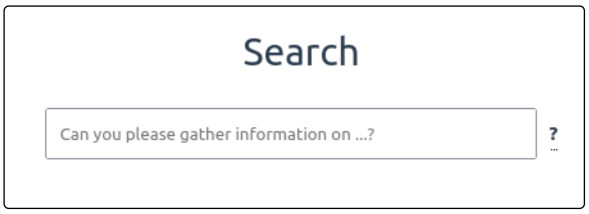
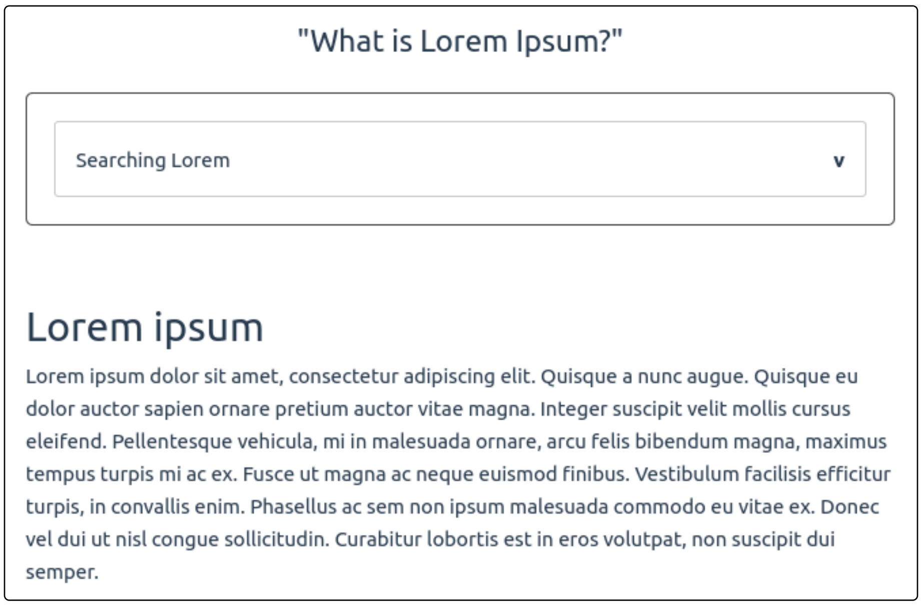


Figure 3: First UI state: the UI prompts the user for an information request.

The UI, entering the processing state, displays a spinner and ongoing agent activity. Figure 4 shows an example, where agent activity appears in the "Searching Lorem" field. Events include currently processed sources and completed steps. They are displayed as collapsible containers that expand to reveal detailed information, including search queries and observations.

After the agent completes its task, the interface presents the retrieved results (Figure 4). The agent generates replies in structured markdown, including numbered citations as needed. At the end of each reply, a clickable list of citations allows users to review primary sources.


"What is Lorem Ipsum?"

Searching Lorem

Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque a nunc augue. Quisque eu dolor auctor sapien ornare pretium auctor vitae magna. Integer suscipit velit mollis cursus eleifend. Pellentesque vehicula, mi in malesuada ornare, arcu felis bibendum magna, maximus tempus turpis mi ac ex. Fusce ut magna ac neque euismod finibus. Vestibulum facilisis efficitur turpis, in convallis enim. Phasellus ac sem non ipsum malesuada commodo eu vitae ex. Donec vel dui ut nisl congue sollicitudin. Curabitur lobortis est in eros volutpat, non suscipit dui semper.

Figure 4: Third UI state: the UI displays the results, with placeholder text for demonstration.

6.1.1. Deployment

The system centers on a development container[1], providing a stable environment for development, testing, and execution.

This thesis does not cover further deployment, as the PoC currently lacks multi-user credential support. To run the agent, users launch the development container, configure the environment (including a `.env` file with required API keys), and execute the necessary commands. The code repository[2, 3] documents this process in more detail.

6.2. Agent Architecture

The agent architecture consists of multiple graphs or agents that collaborate to generate responses. These agents share a common configuration[4], granting access to available LLMs, API clients, and prompt templates. They also use a shared state schema[5] to enable parallel execution and result merging. LangGraph handles state merging after parallel execution, requiring custom reducer logic.

[1]https://containers.dev/
[2]https://github.com/VoigtSebastian/master-thesis/tree/submission?tab=readme-ov-file#tldr
[3]https://github.com/VoigtSebastian/master-thesis/tree/submission/dev#tldr
[4]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/shared/configuration.py
[5]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/shared/state.py

The main graph, depicted in Figure 5, orchestrates the process. It validates user input, coordinates two retrieval sub-graphs, composes replies, and performs final validation, drawing inspiration from the ReAct paradigm.

Retrieval sub-graphs (Figure 6), described in Section 6.2.2 and Section 6.2.3, act as secondary decision makers. Each sub-graph queries a specific data source and updates the agent state. They maintain a streamlined workflow consistent with the approach in Section 5.2.

6.2.1. Main Agent

The main orchestration agent[1] comprises four nodes and several specialized retrieval sub-graphs. Figure 5 visualizes node interactions and data flows.

[1]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/main/graph.py

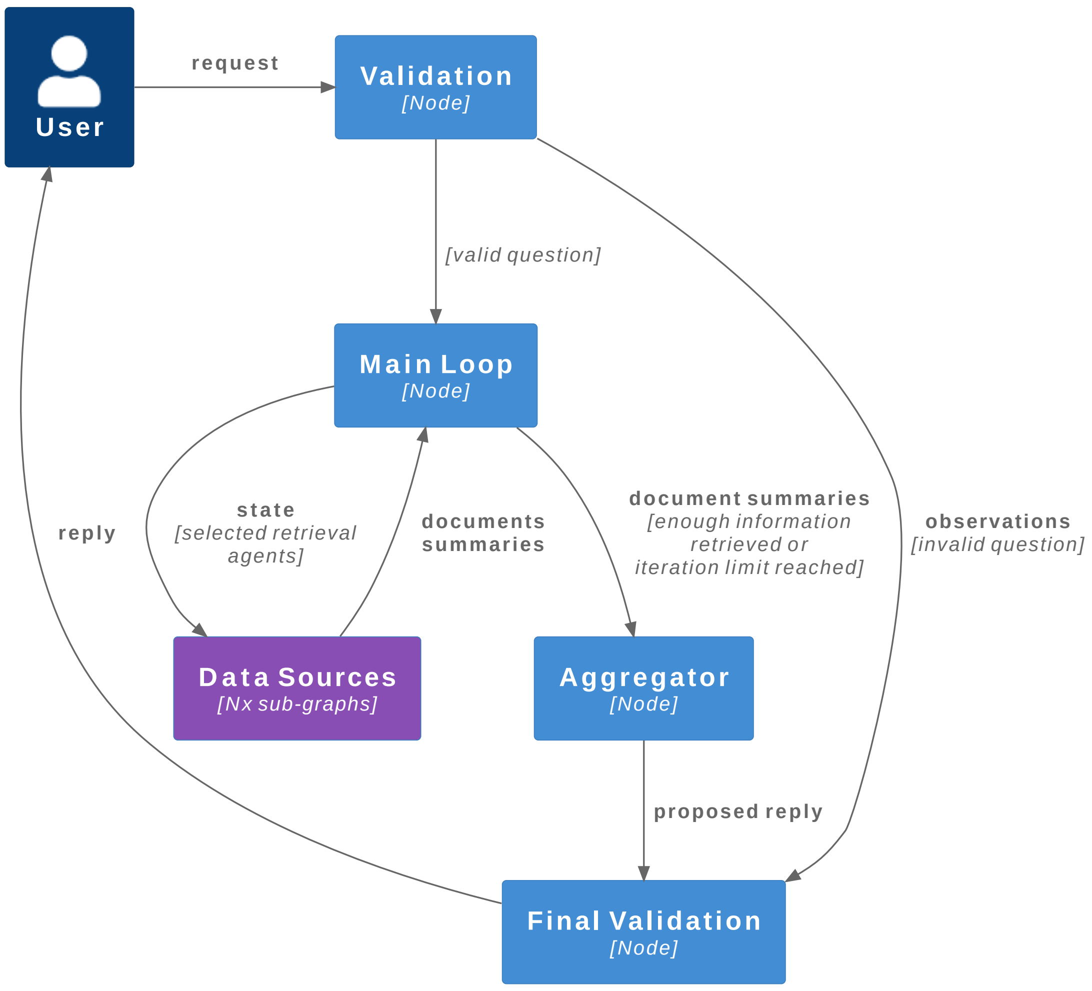


Figure 5: Main agent architecture. Arrows indicate data flow with annotations describing content and conditional execution.

Validation This node checks whether the input requests appropriate information, enhancing trustworthiness (Xi et al. 2025). Initially, the validation node ensured that input was a valid information request. Early testing revealed that users unfamiliar with the technology often entered keywords, which the agent misinterpreted as invalid requests. As a result, we temporarily removed this validation step. If validation passes, the input proceeds to the retrieval agents; otherwise, it moves directly to the final node.

Main Loop Inspired by the ReAct paradigm, this loop selects and executes retrieval agents based on user input and retrieval history. The loop continues until it gathers sufficient information or reaches an iteration limit. Retrieval agents operate in parallel and merge into the main loop upon completion, updating the state with document or chat summaries. This process is managed by LangGraph, utilizing the shared state schema previously discussed.

Aggregator This node assembles a reply from the collected information. It applies a prompt that enforces markdown syntax and source citation. Jinja2 templating defines the input and output format precisely. Each document includes metadata to optimize LLM comprehension.

Final Validation This step evaluates the reply for misinformation and relevance, augmenting the response as needed (see Section 6.3.4 for details on fact-checking).

We implemented custom logic for tool selection and parameter generation instead of using LangGraph's tool node. This approach separates tool choice from parameter generation, improving output quality. A dedicated prompt template with structured output enables precise task definition (see Section 6.3). Each LLM call focuses on a specific task, increasing execution accuracy and efficiency.

6.2.2. Confluence Retrieval Agent

The Confluence retrieval agent[1], shown in Figure 6, uses two nodes to divide responsibilities. It follows the approach in Section 5.2.

[1]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/confluence/graph.py

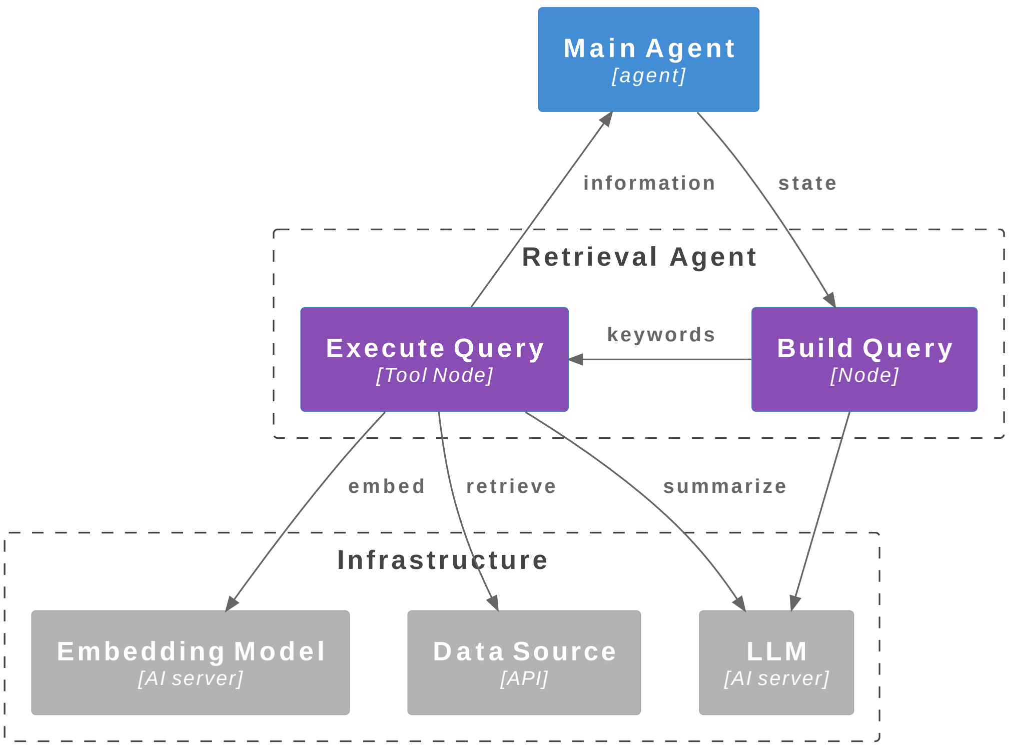


Figure 6: The retrieval agent architecture features two nodes - Build Query and Execute Query. Each node processes and updates the shared state. Retrieval agents start with the main agent's state and merge results after execution. Relationship descriptions accompany arrows.

Build Query This node extracts keywords from the user's request using a prompt based on few-shot and CoT techniques (see Section 6.3). The extracted keywords can later be used to construct a valid CQL query. The structured output includes observations and candidate keywords.

Execute Query Using the extracted keywords, this node constructs a CQL query and queries Confluence with thetool[1] defined in Section 5.2, integrating results into the shared agent state. To reduce the number of pages that are repeatedly retrieved, keywords are stored in the agent's state so that they are only used once. The tool, executed by this node, follows the established workflow of Section 5.2 without changes.

The main agent handles decision-making and tool selection to maintain efficiency. Earlier versions used ReAct agents for independent query refinement, but this increased runtime without improving retrieval quality. Simplifying the workflow improved overall performance.

[1]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/confluence/tool.py

6.2.3. Rocket.Chat Retrieval Agent

The Rocket.Chat retrieval agent[1] uses the same architectural pattern as Confluence but adapts to the unique, short-lived, chat-based data source. It benefits from the experience gathered by the CQL implementation and therefore extends its capabilities. The agent handles complex queries such as "*Did I ever chat with alice or bob about foo,*" filtering chats by participant usernames and searching relevant keywords. This approach reduces runtime and API usage by narrowing the set of chats to search.

Build Query (search pattern) Assuming that modern LLMs are trained on regular expressions, we attempted to generate multiple complete regex patterns in one step. However, this reduced accuracy, so we split tasks and adopted a keyword-based approach similar to the CQL implementation. We instruct the LLM to extract and shorten keywords, which we use to build simpler regex patterns for filtering and search (see Section 6.3.3). This approach closely follows the approach, developed for the Confluence tool. The results are reliable search queries, with mitigated duplication, as we remove already used keywords in subsequent queries.

Build Query (channel filter) To filter chats, we initially provided chat metadata lists to the LLM, but the results were inconsistent. We specified either the channel name or chat participants along with the corresponding chat ID. With this approach, the LLM often identified some, but not all, relevant chat IDs. Switching to a regex-based filter, similar to the search pattern improved the accuracy. Especially when we activated case-insensitivity for the regular expression. The results of which are reliable, simple patterns like `(alice|bob)`.

Execute Query The actual search tool[2], executed by the `Execute Query` node, closely follows the Confluence implementation. The agent executes the generated regex queries on filtered chats, using the Rocket.Chat API. Retrieving up to $n = 100$ messages per chat, and ranking results by embedding similarity to the query. Of those ranked chats, it summarizes the top $k = 3$ chats with respect to the user's request. As with the CQL implementation, the LLM can leave summaries empty for chats that lack relevant information, which reduces the length of successive prompts.

Implementing the Rocket.Chat agent additionally revealed further limitations of RAG. While Confluence pages typically provide long-lived, self-contained information, chat data presents challenges. Messages are typically fragmented and numerous, each with rich metadata and no explicit structure. They are not meant to be used as documentation or as a source of documentation due to a lack of context, unless threaded.

[1]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/rocket/graph.py
[2]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/rocket/tool.py

6.3. Prompt Templates

We developed prompts inspired by (P. Liu et al. 2023), utilizing Jinja2 for data integration and transformation. Most prompts incorporate few-shot demonstrations, CoT reasoning, and structured output separation. We iteratively refined these templates by adjusting wording, emphasizing key steps for the LLM, and providing better examples. Additionally, we instructed the LLMs to enhance the templates by addressing observed shortcomings or learning from examples.

To address LLMs recency bias, we followed recommendations and positioned instructions after large inputs (Zhao et al. 2021; Ma, Yang, and Kästner 2024). For example, summary prompts present the document first, followed by the summarization instructions, improving model focus on large context inputs.

The global Jinja2 environment provides two variables:
- `now`: the current time in human-readable form,
- `agent_personality`: the agent's persona, modifiable via environment variable (default: "`a helpful researcher`").

These variables ensure consistent and context-aware prompt generation.

6.3.1. Prompt Templates for the Main Agent

The main agent uses four key prompt templates, each mapped to a workflow node:

Validation[1] A few-shot prompt with CoT guidance ensures user input is work-appropriate and informational. It returns structured outputs, including a list of `observations` and a binary `passed` flag. We introduced this prompt to improve trustworthiness, as described in (Xi et al. 2025). It previously validated whether the input was an information request, but user feedback led us to relax this constraint, improving satisfaction.

Validation Failed[2] This prompt generates a markdown-formatted error message using the `observations`, the global `agent_personality`, and user input. It explains why the request failed validation and also handles cases where retrieval yields no results, with empty `observation` fields.

[1]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/prompts/main/validation.j2

[2]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/prompts/main/validation_failed.j2

[1]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/prompts/main/tool_choice.j2

Tool Choice[1] After validation, this prompt summarizes the current agent state, including Confluence and Rocket.Chat summaries and executed queries. It incorporates CoT guidelines to support reasoning. The structured output includes `observations` and a set of tools to execute (e.g., “`ROCKET`”, “`CONFLUENCE`”, or empty if sufficient data was retrieved).

Aggregate[2] This zero-shot prompt consolidates all gathered information into a single reply. It provides clear instructions for markdown formatting and source citation, enhancing response clarity and traceability.

6.3.2. Confluence Prompts

Confluence retrieval uses two main prompt templates:

CQL Prompt[3] A few-shot CoT prompt extracts keywords and plans incremental queries. It outputs structured observations and keywords to refine search strategies. This prompt is the earliest and most iteratively refined in our set.

Page Summary[4] To address the limited context window of LLMs and reduce recency bias ((Zhao et al. 2021), (Ma, Yang, and Kästner 2024)), we present Confluence pages first, followed by instructions. This zero-shot prompt instructs summarization tailored to the user query, prioritizing conciseness and relevance.

6.3.3. Rocket.Chat Prompts

The Rocket.Chat agent uses three core prompt templates:

Regex Search[5] A few-shot CoT prompt instructs the LLM to analyze the user’s request and extract keywords accordingly.

Channel Selection[6] Structurally similar to the Regex Search prompt, this template specializes in filtering the chat channel list, reducing resource consumption and speeding up queries.

[2]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/prompts/main/aggregate.j2
[3]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/prompts/confluence/cql_prompt.j2
[4]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/prompts/confluence/page_summary.j2
[5]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/prompts/rocket/regex_search.j2
[6]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/prompts/rocket/channel_selection.j2

Summarize Chat[1] Multiple sub-templates (`dm.j2`, `group.j2`, `thread.j2`)[2] format chat messages into markdown tables. These templates ensure consistent, readable output and exclude URLs to avoid overwhelming the LLM with extraneous data.

6.3.4. General Fact-Check Prompt

All agents use a shared hallucination detection and fact-checking prompt[3], based on three principles:

1. A zero-shot CoT prompt checks whether statements appear in the provided source document. Assuming that claims are shorter than the original documents and considering recency bias, the original is provided first, after which the claim and finally the instruction.
2. It outputs an `observations` string, guiding its final decision with a boolean `hallucination` flag. This CoT structured allows the LLM to inform its own decision.
3. Optional leniency settings allow for nuanced matching, which is useful during aggregation to balance strictness and flexibility. This is currently used to verify the final output of the agent, as it should be checked, but may contain slight inconsistencies from the original documents. This allows the LLM to include real-world information from it's training process.

[1]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/prompts/rocket/summarize_chat.j2

[2]https://github.com/VoigtSebastian/master-thesis/tree/submission/dev/src/prompts/rocket/dto

[3]https://github.com/VoigtSebastian/master-thesis/blob/submission/dev/src/prompts/general/is_hallucination.j2

7 Evaluation

This chapter presents the methods used in evaluating the PoC. We compare the effectiveness and UX of our implementation to traditional search methods.

Because the PoC specializes in retrieving and aggregating data from sensitive sources, existing benchmarks and automated evaluation methods do not suit our context. Instead, we adopted a human-centered approach, using a randomized control trial (RCT) design to collect both quantitative and qualitative user feedback. We also assess the quantitative requirements defined in Section 2.2.

7.1. Automated Evaluation

We determined that evaluating the PoC without human feedback, such as via a continuous integration (CI) pipeline, would not provide relevant insights. Existing agent benchmarks emphasize reasoning skills (Shridhar et al. 2021; Mehta et al. 2024; Liang, Zhu, and Yang 2023), but our architecture centers on secure data access and aggregation, so these metrics do not align with our objectives.

Automated evaluation methods for LLM outputs, such as using additional LLMs for grading (e.g., the Jury method) (Zheng et al. 2023; Verga et al. 2024), remain unreliable (Schaik and Pugh 2024; Shankar et al. 2024). Given these limitations, we chose human evaluation. Participants used the PoC and traditional methods in a RCT setup to solve tasks and provided feedback through questionnaires.

7.2. Trial Design and Methodology

Participants in our study came from diverse professional and academic backgrounds, enhancing the generalizability of our findings.

We based our questionnaire on the UEQ+, an extension of the UEQ that allows adaptation of UX scales for specific research goals (Schrepp and Thomaschewski 2020). The study followed an AB/BA randomized crossover design to compare UX between traditional and AI-assisted search tools (Sibbald and Roberts 1998). We randomly assigned participants into two groups:

Group A Used the traditional tool first, then switched to the AI-assisted tool.
Group B Followed the reverse sequence.

Twelve participants ($N = 12$) completed the study. Each participant worked on two research tasks per tool and then filled out a tailored UEQ+ questionnaire, which measured UX across 13 relevant scales (listed in Table 5; see Appendix 02 for the UI).

slow	/	fast
inefficient	/	efficient
unpredictable	/	predictable
obstructive	/	supportive
not helpful	/	helpful
not rewarding	/	rewarding
difficult	/	easy
illogical	/	logical
not plausible	/	plausible
inappropriate	/	suitable
unintelligent	/	intelligent
poorly prepared	/	well prepared
unambiguous	/	ambiguous

Table 5: The 13 scales evaluated in the tailored UEQ+ questionnaire.

The experimental setup for the trials in this thesis is described in detail in Table 6. It outlines the infrastructural parameters, offering a structured breakdown of experimental conditions. Each row in the table corresponds to a distinct aspect of the setup, thereby facilitating reproducibility and transparent interpretation of results.

Name	Value
Embedding Model	deepset-mxbai-embed-de-large-v1
LLM	Mistral-Small-3.1-24B-Instruct-2503
LLM Quantization	fp8
LLM Temperature	0.1

Table 6: This table outlines the infrastructural parameters, offering a structured breakdown of experimental conditions. Each row in the table corresponds to a distinct aspect of the setup.

All participants answered the same initial research question, designed to reflect a typical workplace problem with a reliable solution. For the second task, users either selected their own topic or worked on an assigned one if they had no current question. This dual-task structure demonstrated how each tool addressed both general and user-specific challenges.

We included an optional open-text field for participants to share strengths, weaknesses, or usability issues they observed.

Each user had up to three minutes per topic to reach an answer. If they ran out of time, we recorded the topic as unsolved, allowing users to factor this into their questionnaire responses.

7.2.1. Rationale for Design Choices

We randomized the tool order to mitigate order effects and bias in UX ratings. This design ensures a fair comparison of the two approaches.

The modular UEQ+ covered key dimensions of UX for knowledge work, such as usability, efficiency, and satisfaction. This allowed us to focus on practical aspects most relevant to our target context.

Offering both fixed and user-generated topics tested the PoC's flexibility and adaptability, demonstrating its strengths across a range of query types.

7.3. Results

Figure 7 and Figure 8 summarize questionnaire responses for each category. Ratings range from 1 (negative) to 5 (positive), with 3 as neutral. A one-point difference equals a $25\% = \left(\frac{100}{5-1}\right)\%$ change on this scale. See Table 11 and Table 12 in Appendix 02 for the raw data.

Each figure contains two boxplots per category - one for the AI trial and one for the traditional trial - depicting overall response distributions. The box spans the interquartile range (IQR) (middle 50%), with the median as a horizontal line. Whiskers extend to the most extreme values within 1.5 times the IQR from each quartile, and outliers appear as circles.

To facilitate direct comparison, we split the results across two diagrams and labeled all categories and trials clearly.

For example, Figure 7 shows that users perceived the AI approach as slower than traditional search, with greater variability in this category. However, the AI assistant scored higher for efficiency, and responses clustered more tightly than in the traditional trial.

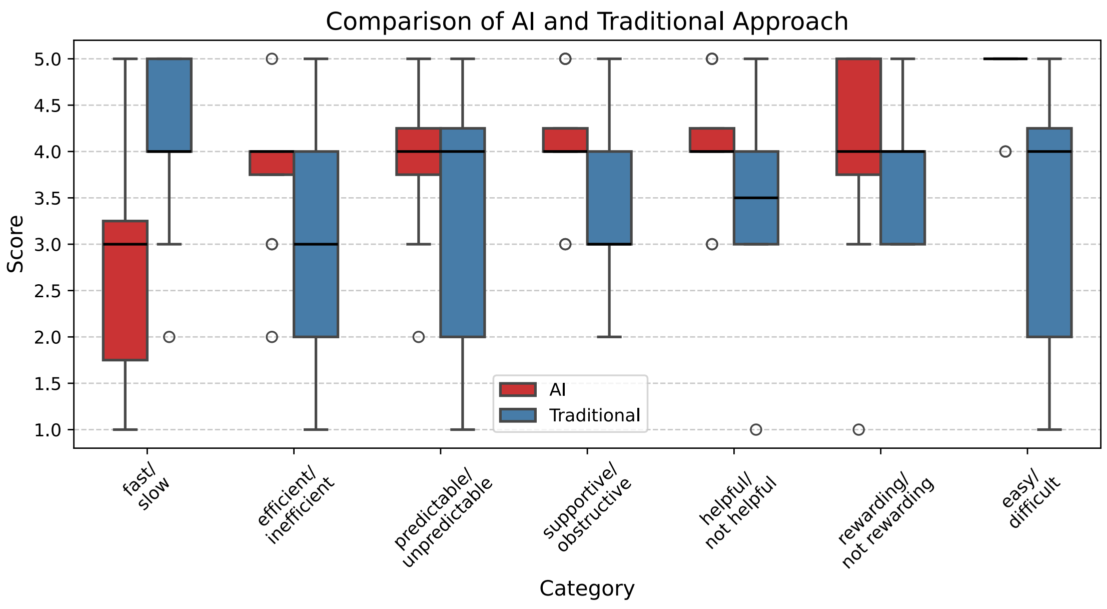


Figure 7: First seven questionnaire categories, visualized as boxplots for the AI and traditional trials. Box shows median and IQR whiskers mark 1.5 IQR from each quartile. Circles denote outliers. Larger values indicate more positive ratings (e.g., fast over slow).

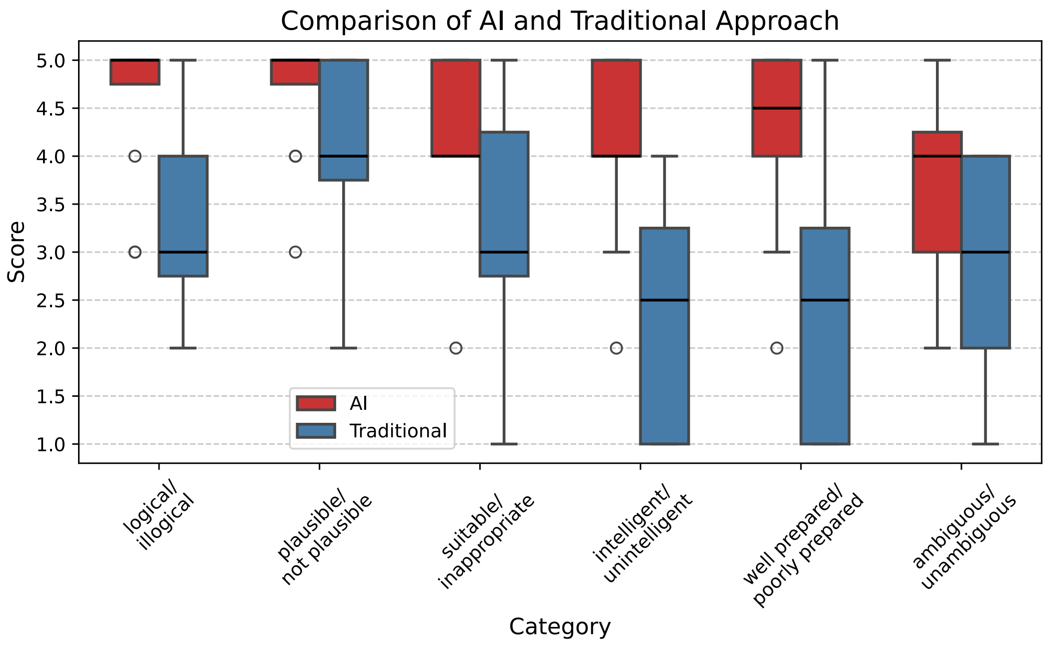


Figure 8: Last six questionnaire categories, visualized as above. Larger values similarly indicate more positive ratings.

To highlight individual-level effects, Figure 9 and Figure 10 show the per-user difference between the AI and traditional tool scores for each category.

Here, each boxplot summarizes the distribution of user-specific differences: positive values favor the AI approach, negatives favor traditional. Values range from -4 to 4, as responses vary from 1 to 5 per questionnaire item. This analysis emphasizes not just group trends but also individual variability - important for nuanced understanding.

To generate these plots, we subtracted each user's traditional trial score from their AI trial score per category, then calculated summary statistics for the differences. We chose this method to visualize both average effects and the spread of user experiences, which is critical in user-centered evaluations.

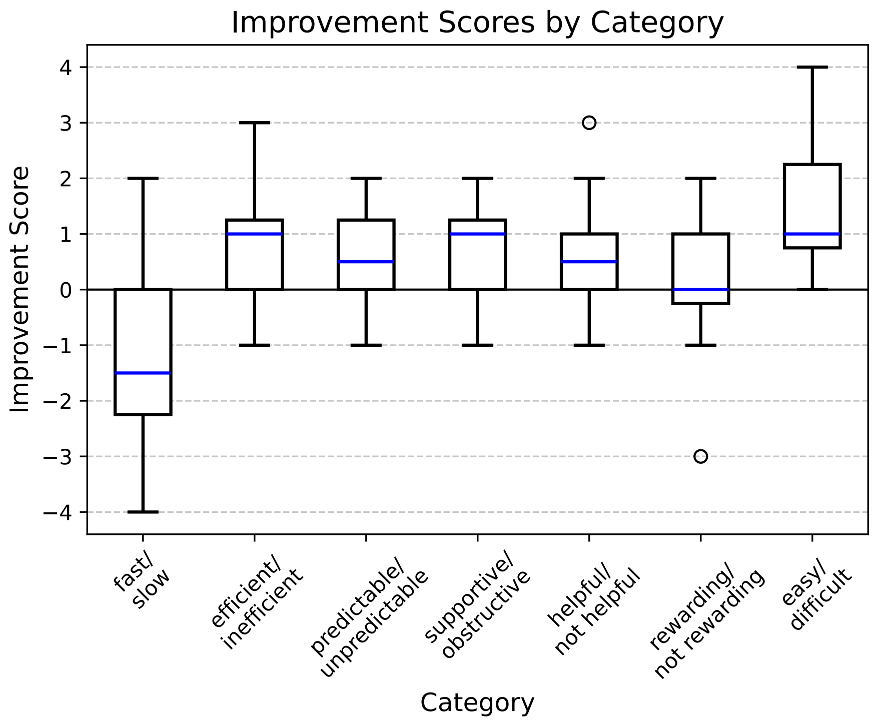


Figure 9: First seven categories: per-user differences in scores (AI – traditional), visualized as boxplots. Box shows median and IQR whiskers mark 1.5 IQR from each quartile. Circles denote outliers. Positive values mean the PoC outperformed the traditional approach.

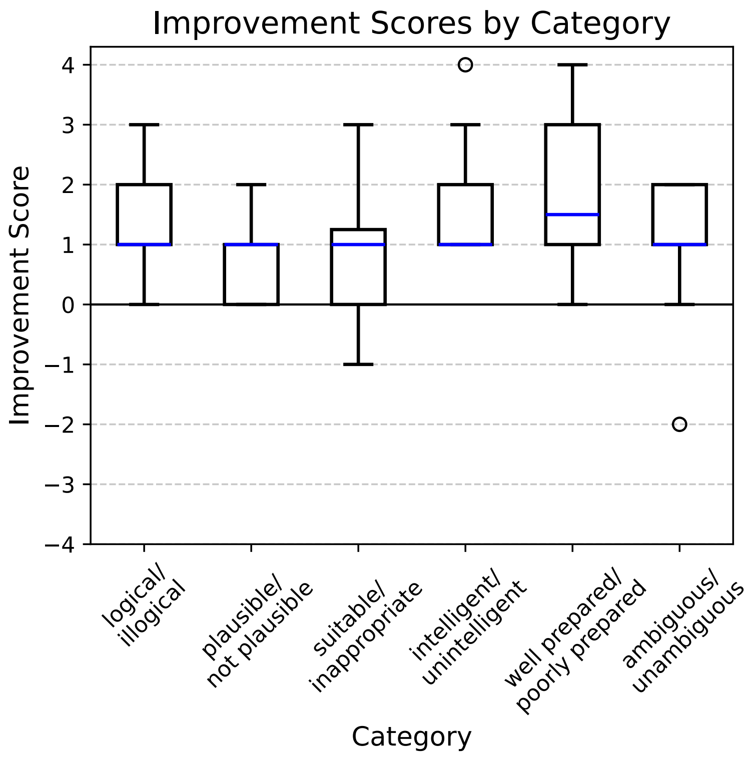


Figure 10: Last six categories: per-user differences, as above. Positive values again favor the AI system.

7.4. Quantitative Evaluation

We use the following tables to evaluate the quantitative requirements defined in Section 2.2. A ✓ indicates a fulfilled requirement, ✗ an unfulfilled one, and ∿ a partially fulfilled one. We present requirements in the same structure as Section 2.2, excluding non-quantitative entries.

7.4.1. General Requirements

Three general requirements guided our agent's development, all binary in character and linked to UX. We fully achieved iterative refinement of queries and German language fluency.

Semantic search, however, remains only partially fulfilled. To prioritize trustworthiness and robustness (see (Xi et al. 2025)), we deliberately limited the LLM's interpretative scope. Some users critiqued this conservative approach, noting that the agent often interpreted keywords literally and occasionally missed their intended meaning.

Requirement	Description	✓/∼/✗
Iterative Refinement	Queries are iteratively refined based on previous results.	✓
Semantic Search	The agent purposefully restricts its own interpretation to maximize trustworthiness, partially limiting semantic search for some tasks.	∼
German	The LLM processes queries fluently in German.	✓

Table 7: Evaluation of General Requirements. ✓ indicates a fulfilled requirement, × an unfulfilled requirement, and ∼ a partially fulfilled requirement.

7.4.2. Data Security

We treated data security as a top priority since the PoC operates on sensitive information. We achieved all goals for confidentiality, access control, and vendor independence.

Requirement	Description	✓/∼/✗
Confidentiality	All data remains on-premise with no external access.	✓
Access Control	Consistent, secure access is maintained via APIs.	✓
Vendor Independence	We built the tools, prompts, and workflows independently, minimizing reliance on external vendors.	✓

Table 8: Evaluation of Data Security Requirements. ✓ indicates a fulfilled requirement, × an unfulfilled requirement, and ∼ a partially fulfilled requirement.

7.4.3. Performance

We assessed all performance requirements through user feedback, except for efficiency. Participants noted that while the agent reduces research workload, it retrieves information more slowly than desired. As a result, we classified the economic impact as only partially fulfilled.

Requirement	Description	✓/∿/✗
Economy	The PoC reduces researchers' workload but is perceived as slow based on user feedback.	∿

Table 9: Evaluation of Performance Requirements. ✓ indicates a fulfilled requirement, ✗ an unfulfilled requirement, and ∿ a partially fulfilled requirement.

7.4.4. Integration

Integration features are critical for system versatility and compliance. All were achieved, as shown in the table below. These include multi-source integration, developer customizability, and exclusive use of on-premise services to ensure data sovereignty.

Requirement	Description	✓/∿/✗
Multi-Source	The system integrates with Confluence and Rocket.Chat.	✓
Customizability	Developers can extend the agent with custom data sources.	✓
LLM Infrastructure	On-premise LLM infrastructure guarantees data sovereignty and organizational compliance.	✓

Table 10: Evaluation of Integration Requirements. ✓ indicates a fulfilled requirement, ✗ an unfulfilled requirement, and ∿ a partially fulfilled requirement.

8
Discussion

This chapter synthesizes, interprets, and contextualizes the findings from Section 7. It addresses the implications for our initial research questions, highlights limitations, and presents recommendations for future research.

8.1. Summary

Our evaluation in Section 7 showed clear improvements across most tested scales. The novelty of the AI tool appears to have elevated overall ratings of the PoC. In contrast, users' familiarity with keyword search enhanced the performance of traditional tools.

The "fast/slow" metric illustrates this distinction. Although the PoC generally outperformed traditional tools, users rated traditional tools higher in perceived speed. Users expect rapid results from keyword-based searches due to pre-indexed documents. While the PoC did not significantly increase total task time, users perceived longer waiting times, likely because they could not interact with the system during response generation.

Users identified both strengths and weaknesses in the new approach. Their familiarity with fast, keyword-based searches shaped their experiences, but they also recognized the PoC's advantages in various scenarios.

8.2. Interpretation

We interpret the quantitative results from Section 7.3 alongside users' written feedback. Three trends emerged:

Deterioration Users rated the PoC as less effective than traditional tools in specific categories.

Stability User satisfaction and performance remained similar for both the PoC and traditional tools.

Improvement The PoC outperformed traditional tools in several categories.

We based these observations on the median, which best represents typical outcomes, and supplemented our analysis with IQR values and qualitative feedback.

8.2.1. Deterioration

Speed was the only category that declined noticeably in perceived quality (37.5% reduction). One user remarked that the PoC felt slower, even though objective measurements did not always confirm this perception.

This result aligns with expectations: users anticipate instant results, which current LLMs do not consistently deliver. Reducing LLM calls, optimizing the tool, or narrowing the document scope would yield only marginal gains. Instead, clarifying delays and highlighting prompt quality in the UI could improve user experience and reduce perceived waiting times.

8.2.2. Stability

The “rewarding/not rewarding” scale remained neutral but trended slightly positive for the PoC, as shown in Figure 7. However, dissatisfaction with waiting times likely limited this effect. When users do not receive precise results quickly, they may prefer manual searches, which they find more rewarding.

8.2.3. Improvement

The following scales reveal the PoC’s strengths and highlight opportunities to improve speed and user communication. Targeted improvements in these areas can further increase user satisfaction.

inefficient / efficient The median improved by 25%, though the IQR suggested some users still perceived inefficiency. Users cited slow speed but appreciated the PoC’s support, intelligence, positivity, and preparation. Although users found the PoC slower, they valued the overall outcome. Long loading times and slow responses contributed to perceptions of inefficiency, but the quality of provided links mitigated the negative impact of speed issues.

unpredictable / predictable The median improved by 12.5%, with a neutral IQR. Keyword-based search feels predictable when successful, but unpredictable when it fails. Users noted the need for exact keywords and the unpredictability of results when search terms did not match documentation structure. The agent also requires precise keywords, which explains the relatively small improvement.

obstructive / supportive The median improved by 25%, though the IQR indicated a more neutral spread. Users found the PoC obstructive due to long waits and the need for precise queries. However, traditional search, especially Confluence, appeared more obstructive because of unpredictable results. This explains why the median improved while the spread remained wide.

not helpful / helpful The median improved by 12.5%, with a neutral IQR. Users noted occasional inconsistencies and the lack of specific chat message citations, which affected helpfulness. Nevertheless, they found the summarized and well-presented results beneficial.

difficult / easy The median improved by 25%, with IQR and maximum values indicating a very positive trend. Users appreciated the well-summarized and orderly presentation, which made the system easier to use despite initial difficulties.

illogical / logical The median improved by 25%, with a positive IQR. Users praised the logical structure of the agent's replies and criticized the confusing ranking in traditional tools. This contributed to the perception that the PoC is logical.

not plausible / plausible The median improved by 25%, though the IQR suggested a more neutral trend. Users found the LLM summary plausible, but considered its validity lower than that of original Confluence pages, which affected perceived plausibility.

inappropriate / suitable The median improved by 25%, but the IQR indicated a more neutral trend. Users sometimes found the system confusing and frustrating, especially when results did not meet expectations. This issue often arose when users searched for specific information without explicitly requesting it from the PoC.

unintelligent / intelligent The median improved by 25%, with a positive IQR. Users valued the PoC's ability to provide well-structured, detailed information, which contributed to its perceived intelligence-especially compared to traditional search options.

poorly prepared / well prepared The median improved by 37.5%, with a positive IQR. Users appreciated the well-organized and summarized results, which indicated good preparation. Compared to the list of links and brief excerpts from traditional search, this feature appears especially important.

ambiguous / unambiguous The median improved by 25%, with a positive IQR. Users highlighted the clear and well-structured presentation, which contributed to the system's unambiguous nature.

8.3. Research Questions

In Section 1.2, we defined two research questions to guide our investigation. We previously discussed some of the concepts introduced by these questions as part of our requirement evaluation in Section 7.4. In this section, we address the results to explore the remaining aspects of each question.

8.3.1. Question 1

The first question asked whether a LLM-based system can provide relevant, accurate information from institutional document storage. It also considered reliable query construction, merging and ranking information from multiple sources, consistent access control, and timeliness.

We answered these sub-questions affirmatively in most cases. Although response speed often fell short of user expectations, the agent reliably retrieves, ranks, and combines information from two knowledge sources. It uses CQL and regular expressions as query languages and enforces access control to maintain data integrity.

8.3.2. Question 2

The second question explored whether a LLM-based agent can iteratively refine and control a complex workflow, improving reply accuracy and relevance through autonomous information retrieval and adaptive querying. It also compared performance to state-of-the-art keyword search methods.

We can answer this question affirmatively. The PoC autonomously retrieves information from sources it selects based on user input. Additionally, it refines its queries based on earlier results. User evaluations showed that it matched or surpassed traditional keyword search in several key areas.

8.4. Limitations

Although our findings highlight the PoC's benefits, several limitations emerged during our trial.

Carryover Effects The randomized crossover design cannot fully eliminate carryover effects between tools. Behavioral states are more difficult to reset than physiological states, making it challenging to ensure that experience with one tool does not influence perceptions of the other (Shi and Ye 2024).

Self-Report Bias UEQ+ ratings are susceptible to social desirability bias, recall errors, and tool limitations ("APA Dictionary of Psychology" 2018; Bound, Brown, and Mathiowetz 2001). Novelty bias may have inflated ratings for the AI tool. Users may also conflate related scales (e.g., Perspicuity versus Efficiency), and open text feedback can be ambiguous or lack engagement.

8.5. Recommendations

Based on these limitations, we recommend the following strategies for future research:

Alternate Study Design Compare a RAG system and an agentic approach using a shared UI to standardize UX and directly compare technological differences.

Limit Sources Use an artificial dataset to control information precision and recall, enabling more precise evaluation in a defined environment.

Familiarization Period Include a familiarization phase to reduce novelty bias and ensure participants provide informed, reliable feedback.

Longitudinal Study Conduct longitudinal research to assess long-term usage effects, user adaptation, and the durability of observed benefits.

These approaches will strengthen future studies and provide deeper insight into agentic LLM-based systems.

9
Conclusion and Future Work

This chapter synthesizes the thesis' key findings, situates their importance within the field, and presents avenues for future research. We revisit the developed tools, summarize the evaluation process, and analyze how our results address the initial research questions. Based on these outcomes, we propose concrete directions for further development and investigation.

9.1. Conclusion

Inspired by the helpful librarian metaphor introduced in Section 1, this thesis set out to design an agentic research assistant capable of navigating, retrieving, and synthesizing institutional knowledge. We outlined critical research questions - focusing on the feasibility of LLM-driven systems for autonomous information retrieval, ranking, and adaptive workflow refinement-which guided our review of existing technologies and identified areas for innovation.

Recognizing the limitations of prevalent approaches, we engineered a novel architecture that blends RAG with query-based autonomous agents. Our PoC demonstrated the ability to autonomously gather information from platforms such as Confluence and Rocket.Chat, enabling users to pose natural language queries and receive detailed, source-cited answers. Throughout development and evaluation, we emphasized both technical rigor and practical usability.

To assess our system, we conducted a RCT, directly comparing our agent to traditional keyword-based retrieval tools. Participants reported notably reduced manual effort and higher satisfaction with the relevance and quality of the generated responses. However, they also encountered slower retrieval speeds, highlighting current challenges in LLM inference and potential opportunities for improvement regarding our proposed architecture.

Our architecture supports robust access control and resilient workflows, as discussed in Section 6. Nevertheless, we observed limitations, including potential carryover effects from the crossover study design and the inherent biases of self-reported user feedback, which may constrain the generalizability of our findings. We provide a detailed analysis of these limitations in Section 8.

This research establishes that LLMs can meaningfully support knowledge workers in sensitive, institution-internal contexts. By combining ReAct and RAG mechanisms, our architecture raises the efficiency and accessibility of research tasks while ensuring traceability and compliance with access requirements. While our solution represents a significant advance, further optimization and integration remain necessary to fully realize the economic and practical potential of AI in organizational knowledge management.

In summary, this work offers a foundation for the development of next-generation research assistants-agents that, much like a skilled librarian, help users discover and contextualize relevant knowledge within complex information landscapes.

9.2. Future Work

Building on our findings, we identify four primary directions for future work: improving retrieval speed, expanding integrations, strengthening security, and conducting further studies.

9.2.1. Retrieval Speed

Participants frequently cited slow retrieval as a major drawback. To address this, future work should focus on optimizing agent performance, reducing LLM inference latency, and applying prompt engineering to minimize superfluous tool calls. Techniques such as more efficient batching, model distillation, or hybrid retrieval pipelines could provide substantial improvements.

9.2.2. Broader Integration

Integrating additional data sources and developing support for diverse formats will extend the system's capabilities. For example, direct downloads of referenced links, integration with vector databases using OpenID authentication, and public internet search with human-in-the-loop moderation (e.g., via searxng) can greatly enhance flexibility.

Further abstraction of retrieval agents will empower users to easily enable or disable data sources as required. Application of this architecture to domains such as healthcare or legal research may also unlock new opportunities and challenges for agent-based information management.

9.2.3. Security

Our work emphasized practical security measures such as access control with API keys, input validation, and strict enforcement of institutional access policies, as detailed in Section 6. We defined the key security requirements for our PoC in Table 2, ensuring the agent reliably protects sensitive information.

Looking ahead, a major improvement will be the addition of multi-user credential management. Currently, the system requires self-hosting by each user. By supporting secure, institution-wide deployments where users authenticate individually, we can improve both accessibility and security.

Finally, as LLM-based systems evolve, investigating potential adversarial risks and testing the resilience of our agent against attacks like prompt injection will be essential future work.

9.2.4. Future Studies

As detailed in Section 8.5, further studies should compare RAG systems with agentic approaches using standardized UI and UX designs to rigorously assess technological benefits and limitations.

Additionally, longitudinal research on user adaptation, workflows, and satisfaction will offer deeper insights into the practical impact and long-term viability of autonomous research assistants.

By pursuing these lines of inquiry, future work can advance the state of the art in autonomous knowledge retrieval, delivering increasingly robust, adaptable, and secure research tools-truly embodying the value of a digital librarian.

References

Alderson-Day, Ben, and Charles Fernyhough. 2015. “Inner Speech: Development, Cognitive Functions, Phenomenology, And Neurobiology”. *Psychological Bulletin* 141 (5): 931–65. https://doi.org/10.1037/bul0000021.

Atlassian. 2017. “Performing Text Searches Using CQL”. December 8, 2017. https://developer.atlassian.com/server/confluence/performing-text-searches-using-cql/.

Atlassian. 2019. “Advanced Searching Using CQL”. August 6, 2019. https://developer.atlassian.com/server/confluence/advanced-searching-using-cql/.

Atlassian. 2025a. “Agents”. January 27, 2025. https://support.atlassian.com/rovo/docs/agents/.

Atlassian. 2025b. “Rovo Data, Privacy and Usage Guidelines | Rovo”. 2025. https://support.atlassian.com/rovo/docs/rovo-data-privacy-and-usage-guidelines/.

Bast, Hannah, Björn Buchhold, and Elmar Haussmann. 2016. “Semantic Search on Text and Knowledge Bases”. *Foundations and Trends® in Information Retrieval* 10 (January):119–271. https://doi.org/10.1561/1500000032.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. “A Neural Probabilistic Language Model”. *J. Mach. Learn. Res.* 3 (null): 1137–55.

Bommasani, Rishi, Drew A. Hudson, E. Adeli, R. Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, et al. 2021. “On the Opportunities and Risks of Foundation Models”. *Arxiv*, August. https://www.semanticscholar.org/paper/On-the-Opportunities-and-Risks-of-Foundation-Models-Bommasani-Hudson/76e9e2ec3de437ffb30d8b7b629f7fe3e61de5c2.

Bound, John, Charles Brown, and Nancy Mathiowetz. 2001. “Chapter 59 - Measurement Error in Survey Data”. Edited by James J. Heckman and Edward Leamer. *Handbook of Econometrics*. Elsevier. https://doi.org/10.1016/S1573-4412(01)05012-7.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, et al. 2020. “Language Models Are Few-Shot Learners”. In *Advances in Neural Information Processing Systems*, 33:1877–1901. Curran Associates, Inc.. https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Cambridge University Press and Assessment. 2025. “Prompting”. In . https://dictionary.cambridge.org/dictionary/english/prompting.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, et al. 2025. “DeepSeek-R1: Incentivizing Reasoning Capability in Llms via Reinforcement Learning”. 2025. https://arxiv.org/abs/2501.12948.

Dhillon, Inderjit S., James Fan, and Yuqiang Guan. 2001. “Efficient Clustering of Very Large Document Collections”. Edited by Robert L. Grossman, Chandrika Kamath, Philip Kegelmeyer, Vipin Kumar, and Raju R. Namburu. *Data Mining for Scientific and Engineering Applications*. Boston, MA: Springer US. https://doi.org/10.1007/978-1-4615-1733-7_20.

Dong, Yixin, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. 2024. “XGrammar: Flexible and Efficient Structured Generation Engine for Large Language Models”. November 27, 2024. https://doi.org/10.48550/arXiv.2411.15100.

Franklin, Stan, and Arthur Graesser. 1996. “Is It an Agent, Or Just a Program?: A Taxonomy for Autonomous Agents.”. In , 21–35.

Gao, Jianfeng, Michel Galley, and Lihong Li. 2019. “Neural Approaches to Conversational AI”. *Found. Trends Inf. Retr.* 13 (2–3): 127–298. https://doi.org/10.1561/1500000074.

Hu, Chenxu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. 2023. “ChatDB: Augmenting LLMs with Databases as Their Symbolic Memory”. June 7, 2023. https://doi.org/10.48550/arXiv.2306.03901.

Huang, Xu, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. “Understanding the Planning of LLM Agents: A Survey”. February 5, 2024. https://doi.org/10.48550/arXiv.2402.02716.

Kojima, Takeshi, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. “Large Language Models Are Zero-Shot Reasoners”. In *Advances in Neural Information Processing Systems*, 35:22199–213.

Koo, Terry, Frederick Liu, and Luheng He. 2024. “Automata-Based Constraints for Language Model Decoding”. August 5, 2024. https://doi.org/10.48550/arXiv.2407.08103.

Laird, John E., Allen Newell, and Paul S. Rosenbloom. 1987. “SOAR: An Architecture for General Intelligence”. *Artificial Intelligence* 33 (1): 1–64. https://doi.org/10.1016/0004-3702(87)90050-6.

LangChain Team. 2024b. “Confluence”. December 17, 2024. https://python.langchain.com/docs/integrations/document_loaders/confluence/.

LangChain Team. 2024a. “Langchain-Ai/langchain: Build Context-Aware Reasoning Applications”. 2024. https://github.com/langchain-ai/langchain.

LangChain Team. 2025. “Qdrant”. March 19, 2025. https://python.langchain.com/docs/integrations/vectorstores/qdrant/.

Lev Semyonovich Vygotsky. 1987. *Thinking and Speech*.

Lewis, Patrick, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, et al. 2020. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. May 22, 2020. https://arxiv.org/abs/2005.11401v4.

Li, Bohan, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. 2020. “On the Sentence Embeddings from Pre-trained Language Models”. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, edited by Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, 9119–30. Online: Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.733.

Li, Jia, Ge Li, Yongmin Li, and Zhi Jin. 2025. “Structured Chain-of-Thought Prompting for Code Generation”. *ACM Trans. Softw. Eng. Methodol.* 34 (2): 1–23. https://doi.org/10.1145/3690635.

Liang, Yuanzhi, Linchao Zhu, and Yi Yang. 2023. “Tachikuma: Understading Complex Interactions with Multi-Character and Novel Objects by Large Language Models”. July 24, 2023. https://doi.org/10.48550/arXiv.2307.12573.

Liu, Bo et al. 2023. “LLM+P: Empowering Large Language Models with Optimal Planning Proficiency”. September 27, 2023. https://doi.org/10.48550/arXiv.2304.11477.

Liu, Michael Xieyang, Frederick Liu, Alexander J. Fiannaca, Terry Koo, Lucas Dixon, Michael Terry, and Carrie J. Cai. 2024. “"We Need Structured Output": Towards User-centered Constraints on Large Language Model Output”. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 1–9. CHI EA '24. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3613905.3650756.

Liu, Pengfei et al. 2023. “Pre-Train, Prompt, And Predict: A Systematic Survey of Prompting Methods in Natural Language Processing”. *ACM Comput. Surv.* 55 (9): 1–35. https://doi.org/10.1145/3560815.

Liu, Vivian, and Lydia B Chilton. 2022. “Design Guidelines for Prompt Engineering Text-to-Image Generative Models”. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–23. CHI '22. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3491102.3501825.

Ma, Wanqin, Chenyang Yang, and Christian Kästner. 2024. “(Why) Is My Prompt Getting Worse? Rethinking Regression Testing for Evolving LLM APIs”. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*, 166–71. CAIN '24. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3644815.3644950.

Mehta, Nikhil, Milagro Teruel, Patricio Figueroa Sanz, Xin Deng, Ahmed Hassan Awadallah, and Julia Kiseleva. 2024. “Improving Grounded Language Understanding in a Collaborative Environment by Interacting with Agents Through Help Feedback”. February 5, 2024. https://doi.org/10.48550/arXiv.2304.10750.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. “Efficient Estimation of Word Representations in Vector Space”. September 7, 2013. https://doi.org/10.48550/arXiv.1301.3781.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. “Distributed Representations of Words and Phrases and Their Compositionality”. October 16, 2013. https://doi.org/10.48550/arXiv.1310.4546.

Mitchell, Tom M. 1990. “Becoming Increasingly Reactive”. In *Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 2*, 1051–58. AAAI'90. Boston, Massachusetts: AAAI Press.

OpenAI. 2025. “Structured Outputs - OpenAI API”. 2025. https://platform.openai.com/.

OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, et al. 2024. “OpenAI O1 System Card”. December 21, 2024. https://doi.org/10.48550/arXiv.2412.16720.

Petroni, Fabio, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. “Language Models as Knowledge Bases?”. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2463–73. Hong Kong, China: Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1250.

PrivateGPT, Zylon by. 2023. “PrivateGPT”. May 2023. https://github.com/zylon-ai/private-gpt.

Reimers, Nils, and Iryna Gurevych. 2019. “Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks”. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3980–90. Hong Kong, China: Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1410.

Reuters. 2025. “Atlassian Issues Upbeat Forecasts on Strong Demand, Shares Soar”. *Reuters*, January. https://www.reuters.com/technology/artificial-intelligence/atlassian-issues-upbeat-forecasts-strong-demand-shares-soar-2025-01-30/.

Reynolds, Laria, and Kyle McDonell. 2021. “Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm”. February 15, 2021. https://doi.org/10.48550/arXiv.2102.07350.

Rozière, Baptiste, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, et al. 2024. “Code Llama: Open Foundation Models for Code”. January 31, 2024. https://doi.org/10.48550/arXiv.2308.12950.

Russell, Stuart J., and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach*. Third edition, Global edition. Prentice Hall Series in Artificial Intelligence. Boston Columbus Indianapolis: Pearson.

Schaik, Tempest A. van, and Brittany Pugh. 2024. “A Field Guide to Automatic Evaluation of LLM-Generated Summaries”. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2832–36. SIGIR '24. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3626772.3661346.

Scholak, Torsten, Nathan Schucher, and Dzmitry Bahdanau. 2021. “PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models”. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, edited by Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, 9895–9901. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.emnlp-main.779.

Schrepp, Martin, and Jörg Thomaschewski. 2020. “Handbook for the Modular Extension of the User Experience Questionnaire”, July.

Schulhoff, Sander, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, et al. 2025. “The Prompt Report: A Systematic Survey of Prompt Engineering Techniques”. February 26, 2025. https://doi.org/10.48550/arXiv.2406.06608.

Shankar, Shreya, J.D. Zamfirescu-Pereira, Bjoern Hartmann, Aditya Parameswaran, and Ian Arawjo. 2024. “Who Validates the Validators? Aligning LLM-Assisted Evaluation of LLM Outputs with Human Preferences”. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, 1–14. UIST '24. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3654777.3676450.

Shen, Yongliang, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. “HuggingGPT: Solving AI Tasks with ChatGPT and Its Friends in Hugging Face”. December 3, 2023. https://doi.org/10.48550/arXiv.2303.17580.

Shi, Danni, and Ting Ye. 2024. “Behavioral Carry-over Effect and Power Consideration in Crossover Trials”. *Biometrics* 80 (2): ujae23. https://doi.org/10.1093/biomtc/ujae023.

Shridhar, Mohit, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. “ALFWorld: Aligning Text and Embodied Environments for Interactive Learning”. March 14, 2021. https://doi.org/10.48550/arXiv.2010.03768.

Sibbald, Bonnie, and Chris Roberts. 1998. “Understanding Controlled Trials Crossover Trials”. *BMJ : British Medical Journal* 316 (7146): 1719–20. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1113275/.

Singh, Aditi, Akash Shetty, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. “A Survey of Large Language Model-Based Generative AI for Text-to-SQL: Benchmarks, Applications, Use Cases, And Challenges”. In *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*, 15–21. https://doi.org/10.1109/CCWC62904.2025.10903689.

Singhal, Amit. 2001. “Modern Information Retrieval: A Brief Overview”. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 24 (4).

Stack Overflow. 2024. “2024 Stack Overflow Developer Survey”. May 2024. https://survey.stackoverflow.co/2024/.

Sun, Ruoxi, Sercan Ö Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, et al. 2024. “SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL (Extended)”. March 30, 2024. https://doi.org/10.48550/arXiv.2306.00739.

Tam, Zhi Rui, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. 2024. “Let Me Speak Freely? A Study On The Impact Of Format Restrictions On Large Language Model Performance.”. In *Proceedings of the 2024*

Conference on Empirical Methods in Natural Language Processing: Industry Track, edited by Franck Dernoncourt, Daniel Preoţiuc-Pietro, and Anastasia Shimorina, 1218–36. Miami, Florida, US: Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.emnlp-industry.91.

Tao, Chaofan et al. 2024. “Scaling Laws with Vocabulary: Larger Models Deserve Larger Vocabularies”. November 1, 2024. https://doi.org/10.48550/arXiv.2407.13623.

Tao, Chongyang et al. 2024. “LLMs Are Also Effective Embedding Models: An In-depth Overview”. December 17, 2024. https://doi.org/10.48550/arXiv.2412.12591.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. “Attention Is All You Need”. August 2, 2023. https://doi.org/10.48550/arXiv.1706.03762.

Verga, Pat, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. 2024. “Replacing Judges with Juries: Evaluating LLM Generations with a Panel of Diverse Models”. May 1, 2024. https://doi.org/10.48550/arXiv.2404.18796.

vLLM Team. 2025. “Structured Outputs”. 2025. https://docs.vllm.ai/en/latest/features/structured_outputs.html.

Wang, Bailin, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. 2023. “Grammar Prompting for Domain-Specific Language Generation with Large Language Models”. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 65030–55. NIPS '23. Red Hook, NY, USA: Curran Associates Inc.

Wang, Lei et al. 2024. “A Survey on Large Language Model Based Autonomous Agents”. *Frontiers of Computer Science* 18 (6): 186345. https://doi.org/10.1007/s11704-024-40231-1.

Wang, Zichong et al. 2024. “History, Development, And Principles of Large Language Models: An Introductory Survey”. *AI and Ethics*, October. https://doi.org/10.1007/s43681-024-00583-7.

Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. January 10, 2023. https://doi.org/10.48550/arXiv.2201.11903.

White, Jules, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2025. “A Prompt

Pattern Catalog to Enhance Prompt Engineering with ChatGPT”. In *Proceedings of the 30th Conference on Pattern Languages of Programs*, 1–31. PLoP '23. USA: The Hillside Group.

Xi, Zhiheng, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, et al. 2025. “The Rise and Potential of Large Language Model Based Agents: A Survey”. *Science China Information Sciences* 68 (2): 121101. https://doi.org/10.1007/s11432-024-4222-0.

Xue, Siqiao, Caigao Jiang, Wenhui Shi, Fangyin Cheng, Keting Chen, Hongjun Yang, Zhiping Zhang, et al. 2024. “DB-GPT: Empowering Database Interactions with Private Large Language Models”. January 3, 2024. https://doi.org/10.48550/arXiv.2312.17449.

Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. “ReAct: Synergizing Reasoning and Acting in Language Models”. March 10, 2023. https://doi.org/10.48550/arXiv.2210.03629.

Zhao, Wayne Xin, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, et al. 2024. “A Survey of Large Language Models”. October 13, 2024. https://doi.org/10.48550/arXiv.2303.18223.

Zhao, Zihao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. “Calibrate Before Use: Improving Few-shot Performance of Language Models”. In *Proceedings of the 38th International Conference on Machine Learning*, 12697–706. PMLR. https://proceedings.mlr.press/v139/zhao21c.html.

Zheng, Lianmin, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, et al. 2023. “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena”. December 24, 2023. https://doi.org/10.48550/arXiv.2306.05685.

“APA Dictionary of Psychology.” 2018. April 19, 2018. https://dictionary.apa.org/.

Appendix

Appendix 01: Code

This section delves into the details of the code structure and the rationale behind it. The complete code can be found in the GitHub repository[1]. An overview of the directory structure is provided in Listing 5.

The `dev` directory is organized into four main sub-directories:
`agent-frontend/` Contains the `Vue.js` code responsible for interacting with the agent.
`questionnaire/` Contains the `Vue.js` code used to evaluate the master's thesis.
`evaluation/` Contains the Jupyter notebook utilized to evaluate the questionnaire.
`src/` Contains the agent's core Python code.

Upon running the agent, a `log/` directory will be generated, storing logs for each execution.

```
.
├── agent-frontend
├── evaluation
├── questionnaire
└── src
    ├── confluence
    ├── main
    ├── prompts
    │   ├── confluence
    │   ├── general
    │   ├── main
    │   └── rocket
    ├── rocket
    ├── shared
    └── util
```

Listing 5: Structure of the `dev/`subdirectory, containing the code in the GitHub repository.

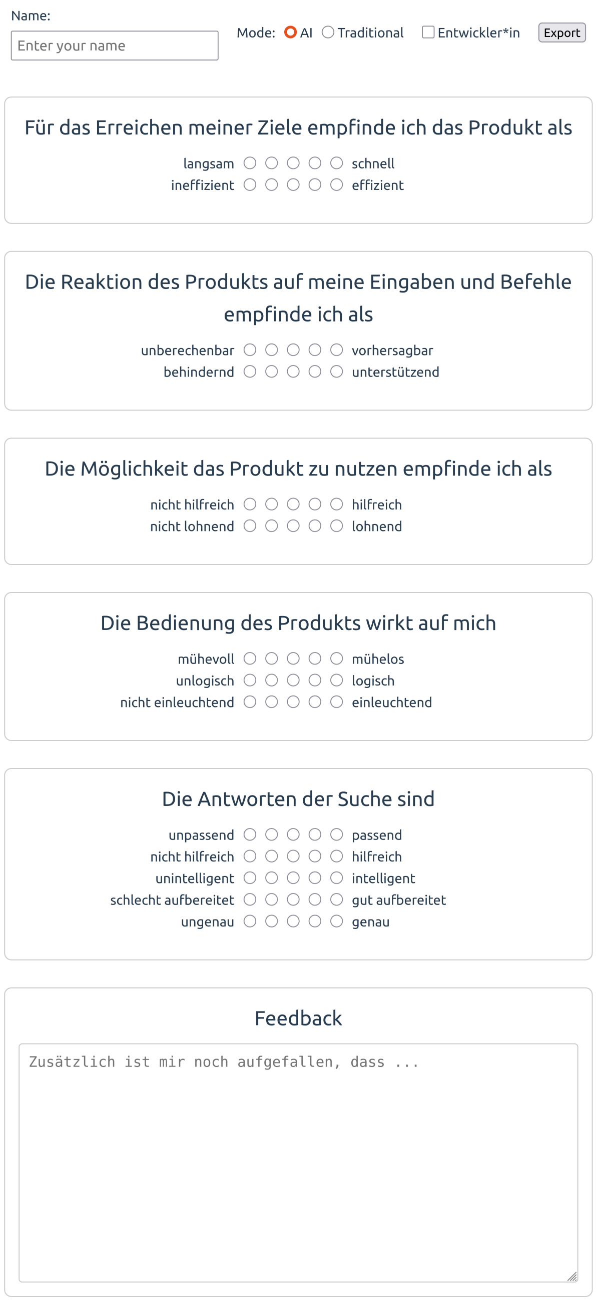
The `src` directory contains all relevant Python code, subdivided into multiple directories:
`confluence/`, `rocket/`, and `main/` These directories contain `LangGraph` and agent-specific code.
`prompts/` This directory contains relevant prompts in the `jinja` templating language, following the structure of the `src/` directory.

[1]https://github.com/VoigtSebastian/master-thesis/tree/submission

`shared/` This directory contains information shared by the agents, such as configuration, state, and the final output.

`util/` This directory contains utility functions, including API clients, logging, and general operations.

Appendix 02: Questionnaire


Name: Enter your name

Mode: [✓ AI] Traditional ☐ Entwickler*in Export

Für das Erreichen meiner Ziele empfinde ich das Produkt als

langsam	○	○	○	○	○	schnell
ineffizient	○	○	○	○	○	effizient

Die Reaktion des Produkts auf meine Eingaben und Befehle empfinde ich als

unberechenbar	○	○	○	○	○	vorhersagbar
behindernd	○	○	○	○	○	unterstützend

Die Möglichkeit das Produkt zu nutzen empfinde ich als

nicht hilfreich	○	○	○	○	○	hilfreich
nicht lohnend	○	○	○	○	○	lohnend

Die Bedienung des Produkts wirkt auf mich

mühevoll	○	○	○	○	○	mühelos
unlogisch	○	○	○	○	○	logisch
nicht einleuchtend	○	○	○	○	○	einleuchtend

Die Antworten der Suche sind

unpassend	○	○	○	○	○	passend
nicht hilfreich	○	○	○	○	○	hilfreich
unintelligent	○	○	○	○	○	intelligent
schlecht aufbereitet	○	○	○	○	○	gut aufbereitet
ungenau	○	○	○	○	○	genau

Feedback

Zusätzlich ist mir noch aufgefallen, dass ...

Figure 11: The questionnaire interface, based on (Schrepp and Thomaschewski 2020)

fast/ slow	efficient/ inefficient	predictable/ unpredictable	supportive/ obstructive	helpful/ not helpful	rewarding/ not rewarding	easy/ difficult
2	2	2	3	3	3	1
4	1	1	2	1	3	4
5	2	4	3	3	3	3
3	3	4	3	3	3	2
4	4	4	4	4	4	4
5	5	5	5	5	5	5
4	3	4	4	4	4	2
5	2	2	3	3	4	2
4	3	2	3	3	4	4
4	3	2	2	4	4	4
5	4	5	3	5	5	5
4	4	5	4	4	4	5

logical/ illogical	plausible/ not plausible	suitable/ inappropriate	intelligent/ unintelligent	well prepared/ poorly prepared	ambiguous/ unambiguous
3	2	3	1	1	2
2	3	1	3	1	1
2	4	2	1	5	2
3	5	5	4	3	3
4	4	4	3	3	3
5	5	5	4	4	4
3	3	3	1	1	4
2	5	3	1	1	2
4	4	3	2	2	3
3	4	2	2	2	2
4	4	5	4	4	4
4	5	4	3	3	4

Table 11: Raw data from the questionnaire, collected from $N = 12$ participants after using the traditional search tools. Each scale ranges from 1 to 5, where 1 indicates a negative value and 5 a positive value. Each row corresponds to a specific user's responses to the evaluated scales.

fast/ slow	efficient/ inefficient	predictable/ unpredictable	supportive/ obstructive	helpful/ not helpful	rewarding/ not rewarding	easy/ difficult
4	4	3	4	4	5	5
4	4	2	4	4	5	5
1	4	4	5	5	4	4
3	4	5	4	4	4	5
3	4	4	5	4	5	5
3	4	5	5	5	5	5
3	4	3	4	3	4	4
1	3	4	4	4	4	5
2	3	4	4	4	3	5
2	2	4	4	3	1	5
5	5	5	3	5	5	5
1	4	4	3	4	3	5

logical/ illogical	plausible/ not plausible	suitable/ inappropriate	intelligent/ unintelligent	well prepared/ poorly prepared	ambiguous/ unambiguous
3	3	4	2	3	3
5	5	4	4	4	3
3	4	5	5	5	4
5	5	4	5	5	5
5	5	4	4	4	4
5	5	5	5	5	4
4	4	2	3	4	2
5	5	4	4	5	3
5	5	4	4	5	4
5	5	4	4	2	4
5	5	5	5	5	5
5	5	5	4	4	5

Table 12: Raw data from the questionnaire, collected from $N = 12$ participants after using the PoC. Each scale ranges from 1 to 5, where 1 indicates a negative value and 5 a positive value. Each row corresponds to a specific user's responses to the evaluated scales.