

Conversational Knowledge-Graphs: Hybrid RAG to Improve Chatbots

A Design Science Research Using Knowledge-Graph-Augmented
Retrieval in Dairy-Processing to Improve Retrieval Correctness in
Chatbots

Ole Fredrik Hagemann

Master in Informatics: Programming and System Architecture
60 ECTS study points

Department of Informatics
Faculty of Mathematics and Natural Sciences

Ole Fredrik Hagemann

Conversational Knowledge-Graphs: Hybrid RAG to Improve Chatbots

A Design Science Research Using
Knowledge-Graph-Augmented Retrieval in
Dairy-Processing to Improve Retrieval Correctness
in Chatbots

Supervisor:
Antonio Martini & Karthik Shivashankar

Abstract

Background

Data retrieval usually happens through querying data sources with specific languages like SQL for relational databases or Cypher for knowledge graphs. These queries can get quite complex and time-consuming to create. Recent retrieval-augmented generation frameworks bridge the gap between data retrieval and natural language due to large language models' ability to generate these queries. However, pure vector search struggles with relational questions.

Methods

I have followed the Design Science Research methodology to incrementally create a chatbot application that retrieves data from two data sources based on questions in natural language. I identified an area in need of improvement—data retrieval in the dairy processing industry. With a potential solution—dairy processing operators querying data sources for data in natural language—through communication and discussion with a stakeholder. Then, I conducted a preliminary survey of potential technologies to use for the solution. Followed by design, development, demonstration, and evaluation in four iterations. I evaluated the chatbot's retrieval correctness on a custom-built benchmark set consisting of 20 questions, through controlled lab experiments across all iterations, whereas the final artifact was evaluated with stakeholders from TINE AS in a semi-structured group interview.

Results

The results of this study highlight to what extent a knowledge graph-augmented retrieval pipeline improves the correctness of chatbot responses in a domain-specific context, in this case, dairy processing, and how recent technological developments can be integrated to support this. Additionally, it shows the specific percentage that both a knowledge graph and enriched prompt instructions contributed to improved results. Furthermore, it reveals the perceived usefulness score from the stakeholder evaluation and highlights the training value for new operators.

Conclusion

In this study, I demonstrate that a knowledge graph, as an addition to a traditional retrieval-augmented generation framework, measurably enhances correctness for domain-specific chatbots and is perceived as valuable by domain experts. Additionally, my findings highlight that a chatbot can be used to support the onboarding and training of new employees in industrial process work without risking any use of production data.

Acknowledgements

I am grateful to my supervisors, Antonio Martini and Karthik Shivashankar, for their support, guidance, constructive criticism, and great communication throughout this thesis. Their expertise, enthusiasm, and encouragement kept me motivated and on track.

I would also like to thank TINE AS for collaboration and valuable insights. Furthermore, my deepest gratitude goes to Morten Steen for knowledge sharing, facilitating meetings, and his eagerness and excitement as an advisor.

Finally, I would like to express my thanks to family and friends for moral support, and my partner, Helene, for her invaluable support and encouragement throughout my studies.

Ole Fredrik Hagemann

University of Oslo

May 2025

Declaration of AI usage in assignment

The service GPT UiO, developed by UiO IT department, has been used to improve the content of the report/assignment. A first version of the work was pasted in its entirety, and the model was given the prompt [I want your help with language and grammar in my paragraphs.] The text was then iterated a few times through the model where new prompts were used to get the correct structure of the text. The final result was cut out, fact-checked, and partly rewritten by the author(s).

Contents

1	Introduction	1
2	Background	3
	2.1 Large Language Models	4
	2.2 Retrieval-Augmented Generation	5
	2.3 Knowledge Graphs	6
	2.4 LangChain and Related Frameworks	7
	2.5 LangGraph	8
	2.6 Identified challenges in the field	8
	2.7 Combining technologies and concepts	9
3	Methodology	11
	3.1 Design Science Research	12
	3.2 Process activities	12
	3.2.1 Identify Problem & motivate	13
	3.2.2 Define Objectives of a Solution	14
	3.2.3 Design and Development	14
	3.2.4 Demonstration	14
	3.2.5 Evaluation	16
	3.2.6 Communication	17
	3.3 Data collection methods	17
	3.3.1 Comparative benchmarking	18
	3.3.2 Semi-structured group interview	18
	3.4 Data analysis	20
4	Iterations	23
	4.1 Iteration 1: Baseline	24
	4.1.1 Objectives for a solution	24
	4.1.2 Design and development	24
	4.1.3 Demonstration	29
	4.1.4 Evaluation	30
	4.1.5 Lessons learned	30
	4.2 Iteration 2: Agent and LangGraph	32
	4.2.1 Objectives for a solution	32
	4.2.2 Design and development	32
	4.2.3 Evaluation	35
	4.2.4 Lessons learned	35

Contents

4.3	Iteration 3: Expanding the knowledge graph.	36
4.3.1	Objectives for a solution	36
4.3.2	Design and development.	36
4.3.3	Demonstration	39
4.3.4	Evaluation	39
4.3.5	System-level ablation studies: isolating KG vs prompt.	40
4.3.6	Lessons learned	41
4.4	Iteration 4: Hybrid retrieval	42
4.4.1	Objectives for a solution	42
4.4.2	Design and development.	42
4.4.3	Demonstration	44
4.4.4	Evaluation	45
4.4.5	Data analysis	48
4.4.6	Lessons learned	48
5	Final artifact	51
5.1	Main components	52
5.1.1	Chat interface.	53
5.1.2	Agent and LangGraph.	53
5.1.3	Data sources	56
5.1.4	Binding link.	57
5.1.5	Sequence diagram	58
5.2	Core functionality.	59
6	Discussion	61
6.1	Research questions & artifact impact	62
6.1.1	RQ1: Architecture & integration impact	62
6.1.2	RQ1.1: Quantitative correctness improvements	63
6.2	Contributions	64
6.3	Related work	64
6.4	Lessons learned	65
6.4.1	Iterations	65
6.4.2	Research aspect.	66
6.4.3	Practical	66
6.5	Limitations	67
6.6	Research quality	67
6.6.1	DSR guidelines	69
6.7	Future work recommendations.	71
7	Conclusion	73
	Appendices.	77
A	GitHub Repository	79
B	Demonstration setup	81
C	Terminal logs from stakeholders demonstration	83
D	Custom benchmark set	85
E	Final prompt	87

List of Figures

2.1	Standard RAG architecture	5
3.1	DSR process activities adapted from [17]	13
4.1	Adapted RAG architecture	25
4.2	Screenshot from Neo4j - Part of the KG	29
4.3	Retrieval flow in the second iteration	33
4.4	Design of star-shaped KG topology	37
4.5	Final iteration chat interface	45
5.1	High-level architecture of the final artifact	52
5.2	Final KG structure illustrating one of each node type.	57
5.3	Sequence diagram of flow in artifact	58

List of Figures

List of Tables

3.1	Classification of failures applied to questions scoring < 50%	20
4.1	SOP Data	27
4.2	Process Data	27
4.3	Production Data	27
4.4	Quality Data	27
4.5	Raw Material Input	27
4.6	Reports	27
4.7	Shift Process Logs	27
4.8	Nonconformity Records	27
4.9	Results of comparative benchmarking for ablation studies.	41
4.10	Benchmark correctness across iterations	48

List of Tables

Chapter 1

Introduction

Since the launch of ChatGPT in 2022, conversational chatbots powered by Large Language Models (LLMs) have gained increasing attention. They have quickly integrated into daily workflows, solving problems ranging from efficient knowledge lookup to creative content generation in various forms, such as text, images, or code. However, they also face limitations, involving their ability to access new or domain-specific data that exceeds their training corpora, and have a tendency to generate sensible but incorrect answers, known as hallucinations [8]. Such issues become prominent when reliable, factual, and precise answers are needed, for example, in fields like medicine, finance, or engineering.

Retrieval-Augmented Generation (RAG) is a framework of techniques that addresses these issues by enabling LLMs to access up-to-date information by retrieving relevant data at query time. Although this technique addresses part of the problem, it also has limitations due to its underlying retrieval mechanism, which primarily operates based on text similarity searches. For complex queries, this makes it difficult to capture relationships between different data entities, resulting in poor responses [13]. Knowledge Graphs (KGs), in contrast, are known for this specifically—capturing relationships in a structured way—by defining nodes and edges connecting them, creating a deep graph of relationships. This thesis seeks to find out to what extent KGs can be integrated with LLMs and RAG techniques to improve the factual correctness and contextual relevance of RAG chatbot responses. Through experiments and testing different approaches to combine technologies incrementally, I aim to combine KGs and RAG frameworks to increase reliability and correctness in these types of systems. This formulates the following research question:

RQ1: How can knowledge graphs and retrieval-augmented generation frameworks be integrated with large language models to enhance querying capabilities in chatbot applications?

For the first part, this involves changing and adapting architectural decisions, implementing retrieval tools, and creating synthetic data to develop and evaluate with. The second part, enhancing querying capabilities, requires a more precise definition. By enhancing querying capabilities, this thesis refers to improving the relevance, correctness, and evidential grounding of chatbot responses. This includes the LLM’s ability to generate precise intermediate queries and produce final answers that are supported by retrieved knowledge. To evaluate whether integrating a KG enhances these capabilities, the following sub-research question is posed:

Chapter 1. Introduction

RQ1.1: To what extent does having access to a knowledge graph of relationships improve the queries and final answers generated by the large language model?

To answer these research questions, this thesis follows the Design Science Research (DSR) methodology, an approach that aims to enhance design knowledge by the creation of an artifact. It involved identifying a problem, designing and developing solutions in iterations, demonstrating and evaluating their performance, and lastly, communicating the findings. The remainder of the thesis is structured as follows.

Thesis structure

Chapter 2: Background

Chapter 2 introduces the relevant technologies and concepts used in this thesis, including LLM, RAG, relevant frameworks, and KG. It also provides context by presenting reviews on related frameworks and research, and outlines identified challenges.

Chapter 3: Methodology

Here, the methodology used is presented, linking the theoretical DSR approach to its practical application. It details process activities, data collection methods, and data analysis.

Chapter 4: Iterations

This chapter presents the four iterations conducted during the development process. It starts by describing the identified problem and then details the activities, such as objectives for a solution, design and development, and evaluation, for each iteration.

Chapter 5: Final artifact

Chapter 5 presents the final artifact. It describes the overall architecture, core functionality, and components of the system, explaining how KGs integrate with RAG pipelines.

Chapter 6: Discussion

In this chapter, the interpretation and reflection of findings are discussed. It links research questions to these findings, reflecting on lessons learned, reviews, and discusses research quality, and suggests direction for future work.

Chapter 7: Conclusion

Finally, Chapter 7 concludes the thesis by summarizing the work and highlighting the main contributions.

Chapter 2

Background

This chapter presents an overview of important technologies, concepts, and research areas relevant to this thesis. It starts by giving an introduction to LLMs, detailing their role in modern natural language processing. Followed by describing RAG frameworks and how the techniques address the identified issues with LLMs. This chapter also presents KGs and how they represent structured relationships that can be utilized in retrieval pipelines.

Subsequent sections elaborate on tools, frameworks, architectures, and data sources used in the integration of KG and RAG systems. Finally, this chapter briefly summarizes the identified challenges and studies that have attempted to combine these technologies and components.

2.1 Large Language Models

For good reason, LLMs have dominated the field of Artificial Intelligence (AI) in recent years. These models evolved from natural language processing, using deep learning methods, such as neural networks. By changing the architecture, scaling model, or data size, and fine-tuning the models, researchers found that more complex tasks could be solved. And from here, a new term was coined, *"GPT-3 can solve few-shot tasks through in-context learning, whereas GPT-2 cannot do well. Thus, the research community coins the term Large Language Models (LLM)"* [27]. The large scale is what characterizes LLMs. Some models have hundreds of billions of parameters, defining the neural network's biases and weights. Another characteristic of LLMs is their transformer architecture, which was introduced by Vaswani et al. in 2017 in the paper "Attention is all you need" [21]. This architecture made the models more efficient and effective in managing longer text and context with a self-attention mechanism. The last characteristic of LLMs is the training data. Training data affects the model's capability to generate natural, human-like language. Examples of training data could be from web pages, academic papers, news articles, books, or social networks. This data is collected, for example, through web scraping—an often automated technique used to extract structured text from websites—or open datasets. LLMs are used in chatbots, virtual assistants, and various other applications. An example is Copilot by Microsoft, which offers support on repetitive tasks and can increase efficiency.

LLMs have shown incredible results in their capabilities of natural language generation. However, they also have limitations, issues such as outdated knowledge and hallucinations, described in Sections 2.6 and 2.7, decrease the reliability and consistency of these models. Static training data, where the knowledge is stored in weights, limits them to this data. Retraining is expensive due to computation and energy usage. As a result of this, LLMs tend to hallucinate when they lack knowledge, leading to answers that are not factually correct [13]. Another limitation in LLMs is bias and fairness due to their training data. This training data is their output source, and they can inherit bias from it. Since LLMs are trained on such vast corpora, patterns from over- and under-representation can be formed as bias and be represented in the LLM's output. Examples of this include, but are not limited to, men being over-represented, whereas women are under-represented, in datasets about leadership [6]. Another study found that ChatGPT, trained mostly on English and Western internet content, showed biases against non-Western demographics when simulating public opinions [18].

In essence, the value of LLMs is in their ability to generate complex content by predicting the next word and using statistical pattern algorithms on their training data. However, limitations, as described above, create challenges in the field. This and other limitations have motivated researchers to augment LLMs with additional data and knowledge sources. This is where retrieval-augmented generation, as discussed below, comes in. Despite their limitations, LLMs can provide advanced generation capabilities by serving as the "brain" of applications. Current research and the focus of this thesis center around extending the capabilities of LLMs by addressing their knowledge gaps.

2.2 Retrieval-Augmented Generation

RAG is a technique used for extending the knowledge base of an LLM with external data sources. By equipping the LLM with functionality to retrieve data and grounding it to use this context, researchers have found a way around the static knowledge base limitation [24]. RAG enhances the LLM, making it more accurate and reliable in terms of grounded responses. It can be thought of as a prompt before the prompt, because the LLM gets access to the data from which it will generate its output.

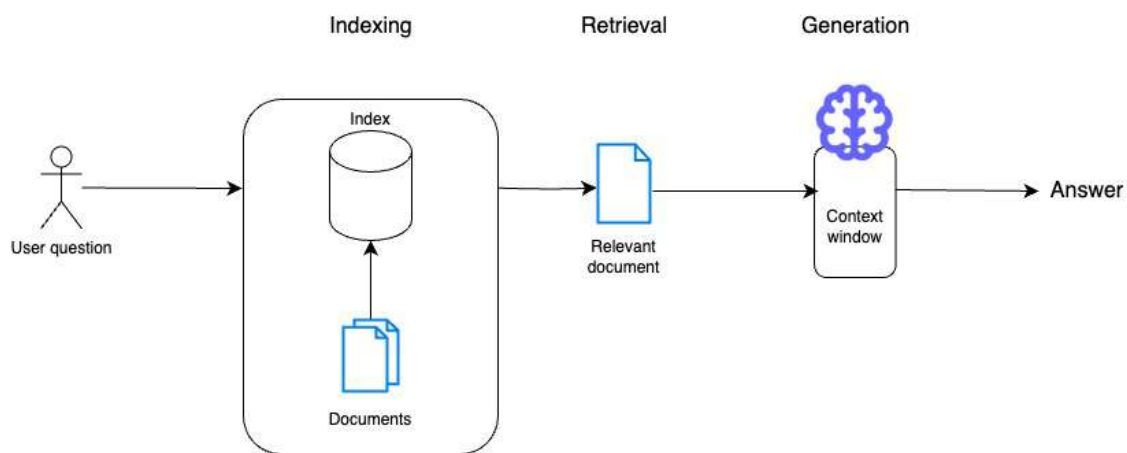


Figure 2.1: Standard RAG architecture

The RAG architecture, as shown in Figure 2.1, includes a user question, indexing, retrieving, and generating. The process starts with a question. Then follows the indexing of documents. This involves splitting them, using delimiters, semantic meaning, sections, or characters, in a way that can be embedded. Those embeddings become a numerical representation of the documents, which can easily be searched through, and they are stored in an index. When given a question, the model embeds it, capturing the semantic meaning in a high-dimensional space, often as a vector, and performs a similarity search against the vector database to retrieve relevant knowledge.

In this retrieval part, knowledge sources are pre-processed and embedded into a vector database, enabling fast similarity search. The search involves looking in the search space for similarities to the embedded documents that are close. By using different algorithms, for example, k-Nearest Neighbors (kNN), it is possible to specify how many **k** most relevant documents to be used in the retrieval step.

With the generation step, illustrated in Figure 2.1, it is possible to set the context from

the retrieval part and the question from the initial prompt in a digital dictionary. Which can be passed as a dynamically customized prompt template and sent to the LLM for processing.

RAG works with various forms of data, such as textual data like PDFs or articles, domain-specific datasets stored in relational databases or KGs, and web-based content. One challenge with RAG is that if the data provided to the LLM is irrelevant or insufficient, the LLM will produce incomplete or incorrect answers. Therefore, the retrieval step and the data source are of high importance. This thesis will present solutions using RAG with a Postgres database and a Neo4j KG.

2.3 Knowledge Graphs

KGs are structured data representations with entities (nodes) and relationships (edges). The KGs most commonly used today are classified into four groups: encyclopedic KGs, commonsense KGs, domain-specific KGs, and multimodal KGs [16]. This thesis dives deeper into KGs that are classified as domain-specific. KGs are known to be great for capturing relational information, such as networks or hierarchies. This is because they provide semantic context, for example, connecting "production data" and "process requirements" nodes to a "process" node with an edge "contains" that connects them and represents the fact that "processes in this context have this specific production data and these process requirements." This comes in handy when we need to capture large sets of data with complex relationships.

However, purely KG-based question-answering systems are traditionally limited by the graph's incompleteness and the difficulty of mapping natural language to graph queries. With the rise of LLMs, there is renewed interest in integrating the two: using LLMs to understand and formulate queries, and using KGs to inject factual precision and logical consistency. Recent research suggests that enhancing LLMs with KGs can improve their factual accuracy and reasoning depth [26].

There are several possible approaches to integrating LLMs and KGs. One method is KG-enhanced LLMs, another is LLM-augmented KGs, and lastly, synergized LLMs + KGs [16]. This thesis explores KG-enhanced query capabilities in LLMs. This technique is referred to as KG prompting [16]. Pan et al. suggest converting structured knowledge graphs into text that the LLM can use to perform better reasoning. This approach is not flawless because, as stated by the authors, *"the prompt is usually designed manually, which requires lots of human effort."* [16], meaning it comes with large development costs.

Sun et al. [20] demonstrate that an LLM agent that explores a knowledge graph to gather relevant facts, rather than relying purely on its parametric memory, shows enhanced deep reasoning, and can even correct its knowledge by finding evidence on the graph. This thesis explores the use of two differently structured KGs, one where data is retrieved for relational context (part of the graph is illustrated in Figure 4.2), and one containing all the raw sample data used for retrieval, in a star-shaped topology (illustrated in Figure 5.2). The rationale for these two structures is presented in the Iterations chapter.

2.4 LangChain and Related Frameworks

Due to the complexity of integrating LLMs with external data sources, frameworks, and libraries, streamlining this has become quite popular. LangChain is one of these; they launched in the second half of 2022 and have been the top provider of modules and components for building RAG applications in recent years. LangChain provide developers with three core abstractions, such as, "agents" (an LLM decides which tools to invoke or steps to take in a reasoning chain), "chains" (a sequence of actions the LLM or agent perform), and "tools" (where developers can build custom tools the agents can use to perform tasks). By creating this framework, LangChain has attracted a rich community of developers due to the simplicity of integration. For example, (1) creating a chain with LangChain that takes in a user question, (2) translates the question to a query that fetches data by generating an SQL query, and (3) running it against the database, are pre-built functions, easily imported. In this process, developers can also follow the flow alongside, which makes it quite an appealing framework due to its observability feature.

As an open-source project consistently building new features by the community, LangChain has become an ecosystem providing frameworks in RAG pipelines. They started to integrate with all variations of systems, spanning from Application Programming Interfaces (API) and cloud storage to relational and vector databases. These types of frameworks are often referred to as the "orchestration layer" in AI applications, due to their handling of external knowledge and tool use [4]. LangChain's recent additions to the ecosystem include LangGraph, a separate framework specific for building a set of agents that serves as an application architecture with different components, structured as a graph. Here, developers can build agents with specific use cases and link them together in the graph to perform separate tasks. The agents can pass knowledge around and reason about the answer in an iterative approach. This also supports incorporating "human-in-the-loop" for human approval of tasks before they are executed. As LangChain, other frameworks, such as Swarms by OpenAI and Haystack by Deepset and LLamaIndex, have also become popular due to their end-to-end, production-ready pipeline of tools. However, this thesis only uses LangChain across all four DSR iterations.

The frameworks described above have a common theme: abstraction. They let developers focus on high-level logic by performing vector similarity search and other low-level functionality in pre-built packages. For development, these frameworks can be invaluable when constructing prototypes or experimenting, for example, creating systems that handle the sequence of tasks: a user question → query generation → data source query execution and retrieval → reiterate in a loop until data matching the user question is provided → reformulate the response to natural language. Doing this without a framework such as LangChain would require custom implementation of each step and more in-depth complexity and development costs. However, using the utility functions that come with these frameworks also requires an understanding of how they work under the hood to get good results. In summary, these frameworks enable fast and high-level development, are easy to use and apply, and serve as an orchestration layer between LLMs and external data sources. LangChain was chosen for development in this thesis due to modular abstractions ensuring robust and accelerated prototyping, their observability and debugging support, and JavaScript/TypeScript integration and compatibility.

2.5 LangGraph

LangGraph is a library built on top of LangChain. It introduces a graph-based approach to orchestrate complex LLM-driven systems by providing a way to define recurrent workflows, in contrast to the standard LangChain workflows, which typically are linear or directed acyclic graphs. Agents can take advantage of this architecture in a way that they may repeatedly decide on tools to call or actions to take until a task is completed. In short, LangGraph allows developers to specify agents' reasoning capabilities by letting the agent decide what to do next, giving flexible behavior, unlike LangChain chains—using sequential flow [10].

The essence of how LangGraph works is, in short, encapsulating the agent's state and logic flow as a StateGraph. The state is defined by the developer and usually represents the current snapshot of the system. In other words, a data structure representing, for example, conversation history, tool results, partial answers, and other relevant information. In this StateGraph, the state is passed around to each node (action or condition) defined in it so that the node can use the context.

Nodes and edges in the StateGraph are used to define and represent actions and conditions. A node in the graph is like an action the program can do, for example, call a tool to retrieve information, and the result of this action is then used to update the state. Edges are conditions or rules that the program follows to move between nodes based on the current state. Creating conditional edges—where it is defined that if criteria x is in the state, go to node y —enables an iterative flow where the agent decides when it is satisfied with the answer or needs an extra tool call [11].

The benefits of using this library include greater observability and controllability. Firstly, observability, because each node is a discrete part of the graph, makes tracing the agent's decision easier through logging. Secondly, controllability, because of the ability to impose custom flows and constraints. Additionally, for systems requiring more advanced audit of the agent's decisions, it is possible to add conditional routing to nodes that require human approval, called human-in-the-loop.

2.6 Identified challenges in the field

Although ecosystems such as LangChain promote building and developing LLM-driven applications through their frameworks, there are still some challenges in the field. These include, but are not limited to, hallucination, retrieval relevance or noise, scalability, and user trust. This section will address these and elaborate on why it is important to focus on them when building with LLMs.

Hallucination

LLMs can sometimes produce incorrect answers when the input or user question exceeds the model's training data, filling knowledge gaps with inaccurate, yet plausible information. Although this problem is still present in RAG systems, research has shown that it is reduced [1].

Retrieval relevance and noise

If the data used in RAG systems is irrelevant or noisy, or the retrieval functions are not correctly configured, the model can retrieve the wrong context and produce irrelevant answers. The challenge involves getting the LLM to retrieve data with the same semantic meaning as the user’s request.

Scalability

As RAG systems grow, computational costs—memory usage, energy consumption, and latency—will also increase. The retrieval part in these systems, processing high-dimensional data and executing queries repeatedly, can increase the computational cost, hindering deployment in environments that are constrained by resources [5].

User trust

It is important to note that the intended use case for the LLM-driven application plays an important role in establishing user trust. User trust involves confidence in the output the LLM provides. Misinformation and lack of source traceability can be crucial in domains like healthcare or finance, but may be less important in others. Exposing metadata, such as where the LLM retrieves data from, can increase users’ willingness to rely on the system because users can, to an extent, verify the system’s output.

2.7 Combining technologies and concepts

This section outlines paradigms for introducing external data into LLM-driven systems. It highlights and compares relevant retrieval methods, elaborates on the role of KGs in capturing relationships through structured retrieval, and motivates the combination of techniques and concepts explored in this thesis.

A key advantage of augmenting LLMs with external retrieval is that it requires less energy for retraining models, allowing the current models to function as the "brain" and orchestrate with new knowledge. Instead of fine-tuning an LLM with new information and updating, in some cases, millions of model weights, RAG is used to extend the knowledge source. And when retrieving from external data sources, the choice of where to store and retrieve data from is important. Knowledge graphs can be a good option, enabling the LLMs to traverse the graph and find complex relationships, which would otherwise be difficult to find using other data sources. Recent research [20] highlights the approach of using a KG as the data source with the LLM as a key view, in contrast to increasing the number of parameters used to train the models, i.e., fine-tuning. Building on this emphasized benefit, this thesis explores to what extent KGs can improve query generation in chatbot applications through measuring the correctness of responses.

In rapidly evolving fields, there will most certainly always be different approaches with strengths and weaknesses. And this field is undoubtedly one of them. The main paradigms discussed comparatively often include *Vector Search vs. Knowledge Graphs*, *Fine-tuning vs. Retrieval*, and *Single source vs. Multi-source retrieval*.

Before RAG techniques gained attention, fine-tuning was the main approach when it came to adding custom or specific domain knowledge. The weakness of this approach is

that it is static. In contrast, the retrieval approach keeps the model's knowledge up-to-date at query time. On the other hand, if a RAG system is fetching data from a bad data source, it will provide a bad response. While the fine-tuning approach does not contain this flaw, it can still produce wrong answers through hallucination if a question asked exceeds the model's knowledge base.

Chapter 3

Methodology

This chapter outlines the methodology and procedures used to carry out the research in this thesis. I introduce DSR and explain why this methodology was suitable for this research. Next, in-depth sections of my development process in light of the methodology, including the main DSR activities, will follow. Then, I detail the evaluation strategies and data collection methods I used, before lastly presenting the data analysis applied to the collected data.

3.1 Design Science Research

Research involves conducting activities, designing, developing, and testing hypotheses. A methodology applies principles, practices, and procedures to a branch of knowledge. DSR can be described as a combination of these, by testing hypotheses following guidelines and practices. To be more specific, DSR is about conducting innovative research with a problem-solving approach. The goal of DSR is to increase human knowledge through the creation of innovative artifacts [2]. The methodology also aims to enhance design knowledge for a richer understanding of what makes the artifact increase the knowledge of the relevant application context [2]. In short, DSR adds knowledge to a research field through identifying problems, creating an artifact that addresses these problems, and sharing the knowledge gained throughout the study.

The final artifact can be of different types, such as constructs, models, methods, or instantiations [7]. The artifact created by this research is considered an instantiation, which [7] describes as an implemented or prototype system. This prototype was central in answering the study’s overall research question—how can KGs be integrated with RAG frameworks to improve querying capabilities in a chatbot, and to what extent.

DSR was chosen as the methodology for this thesis because it aligned with the primary goal: to create and refine a prototype through iterative development and stakeholder evaluation. This methodology promotes the exploratory approach to system-building, unlike traditional experimental research, which often focuses on controlled settings and isolates variables. An alternative to DSR, a comparative analysis approach, was also considered, which would have involved a side-by-side comparison of existing systems and technologies. However, this approach would not have been supportive of designing and creating a prototype system due to its focus on comparing and evaluating existing systems, rather than identifying, building, evaluating, and reiterating as in DSR. The DSR methodology was ultimately selected because it aligned with research objectives and its use as a means of knowledge generation through artifact creation.

In this thesis, DSR enabled an agile development process through each iteration, guiding activities such as identifying problems, designing solutions, implementing and developing them, and evaluating the results. This iterative structure and the results from each evaluation helped assess how and to what extent integrating a KG and LLM system enhanced querying capabilities in a chatbot.

3.2 Process activities

Multiple DSR models exist, and there is no one way of using the methodology. However, a common framework to use is presented by Peffers et al. [17]. It consists of six key activities: *1: Identify Problem*, *2: Define Objectives of a Solution*, *3: Design and Develop*, *4: Demonstration*, *5: Evaluation*, and *6: Communication*. Figure 4.1, below, illustrates my adaptation of this framework and some examples of what was done, coupled with its respective activity. The following subsections will go into detail about the theory of these activities and explain how I have applied them in my research.

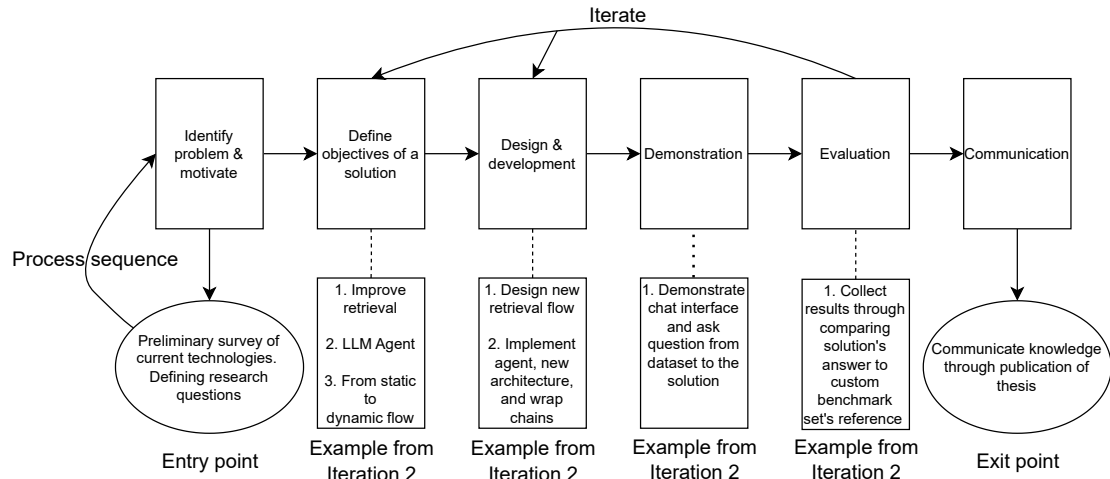


Figure 3.1: DSR process activities adapted from [17]

3.2.1 Identify Problem & motivate

To start the process, Peffers et al. [17] highlight the importance of identifying the problem the artifact will solve and justify the value it provides. To capture the artifact's complexity, they state that it may be valuable to "atomize" the problem conceptually. When it comes to justifying the value the artifact provides, Peffers et al. point out two accomplishments it achieves: it motivates the researchers and emphasizes the reasoning behind their perception of the problem.

To identify a problem and motivate this study, a key stakeholder presented areas of improvement in the current dairy processing industry: Obtaining and analysing process data is time-consuming and resource-intensive. In addition to presenting real-world use cases where technologies like LLMs and frameworks like RAG could be applied as a potential solution, such as real-time access to the retrieval of internal company processes, and what value they could create. One example of such a use case and its potential value presented by the stakeholder was:

- **Use case:** During production, the operator has full digital access to the schedule, the relevant SOPs and control points, and can ask the chatbot avatar for support.
- **Value created:** Compliance with the agreed plans and routines. Comprehensive problem-solving support. On-the-job training best practice.

The stakeholder has industrial process experience from the dairy industry, which is the intended industry for the use case above. Synthesizing these business needs outlined the requirements of a system that workers, in an industrial setting, could use to retrieve information from. Specifically, to reduce lost time, detect malfunction before repair, train new employees, and improve right-first-time.

Following this gathering of requirements from the stakeholders, I then explored the relevant research field in the form of a preliminary survey on current technologies and concepts, with particular focus on limitations and gaps in current and prior research. This exploration revealed that while RAG techniques address the issue of static knowledge bases in LLMs, they are also limited by the unstructured vector similarity

search mechanism used for retrieval. This pointed in the direction of a data structure that could support structured and relationship-focused retrieval—specifically, a KG.

To further motivate the study and assess the feasibility of this direction an additional exploration phase was conducted. Exploring previous research on KG construction, integration techniques with LLMs, and design of KG retrieval system. Insights gained from this exploration helped formulate the research questions, which aim to find out whether KGs can improve querying capabilities.

3.2.2 Define Objectives of a Solution

This activity involves extracting objectives from the problem definition. In short, infer rationally from the problem specification and create objectives to form a solution.

In this process step, I initially defined objectives based on the preliminary literature survey, research question, and the problem identified. These objectives formed the requirements to the first iteration framed as baseline objectives for a functioning RAG pipeline. In subsequent iterations, new objectives were formed based on previous iteration's evaluation results and the research questions. For example, after observing poor intermediate query quality in the first iteration, a key objective for iteration two became switching from a static to a dynamic flow.

By redefining the objectives of each iteration's solution, I was able to focus on developing what that iteration required, limit the scope to a manageable workload, and track how design decisions contributed to answering research questions. More detailed descriptions of each iteration's objectives are provided in Chapter 4.

3.2.3 Design and Development

This is where the objectives from the previous activity translate into design decisions and development. This includes determining the artifact's desired functionality and architecture [17].

In my process, I began this activity for each iteration by designing and creating a mental model of a solution based on the objectives. This design phase often involved planning how the components in the system should interact, for example, the data flow between them. After I had conceptualized these decisions, I could start developing and implementing functionality reflecting them in code. For example, in iteration two, the design involved changing from chains to tools for the retrieval functions. Implementing this required restructuring the system to support an agent with these tools.

By following this approach, I ensured that each iteration produced a solution that aligned with the objectives. Chapter 4 provides detailed information about this activity for each iteration.

3.2.4 Demonstration

The demonstration activity promotes showcasing the solution's ability to partly or fully solve the identified problem, often done through experimentation, simulation, or other activities appropriate [17]. The solutions in this study were demonstrated in scheduled demo sessions with a stakeholder having dairy-processing expertise. For each iteration,

except for the third, I presented knowledge on how to use the prototype and how it aligned with solving its respective objectives.

Additionally, I demonstrated the final solution to a different stakeholder, TINE AS, during an online evaluation session, where I asked questions from the benchmark set, described in Section 3.2.5 and presented in Appendix D, to the chatbot application, and showcased its responses. This stakeholder evaluation session followed a predefined interview guide, or so-called evaluation plan, described in full in Section 3.3.2, which elaborates on how I planned to conduct each activity. Specifically for the demonstration part, it includes the following activities:

- Share split screen: chat interface (left) and terminal logs (right).
- Explain what they see, what I will demonstrate, and the benchmark set.
- Ask three representative questions from the benchmark set:
 1. Did we run at full capacity?
 2. What were the reasons for lost time or capacity?
 3. How was the yield based on input raw material volume and output production?

After I have asked the three questions above, I ask if the stakeholders have any questions to ask the chatbot application.

This planned agenda for the demonstration part of the evaluation session was designed to accomplish two goals: (1) to demonstrate the capabilities the artifact has in solving problems for domain experts, and (2) to stimulate engagement and elicit feedback. Consequently, each activity included in this demonstration played its part in achieving these goals. The following paragraphs will include the rationale for these demonstration and evaluation choices.

Split screen

Splitting the screen—showcasing both the chat interface and the terminal logs—shows transparency and makes the participants trust the application they were observing. By allowing them to see not only the final output but also the underlying agent choices, data retrieved, and queries used, increases their trust in the system. The setup can be viewed in Appendix B.

Orientate participants

The participants had no prior exposure to the prototype. Therefore, it was necessary to inform them about some key elements in it to mitigate misinterpretations. I focused on the most important parts, such as the benchmark set, the data sources used for retrieval, what the chat interface would demonstrate, and what happened in the terminal as the system was executed.

Questions to the artifact

The benchmark set consisted of 20 questions, but due to limited time in the evaluation session, I could only pick some of them. I tried to pick three different questions that would represent different capabilities in the chatbot, and it landed on questions 1, 2, and 3 from the demonstration plan above. The first (1) was chosen to demonstrate the correctness of a factual lookup, and by doing this, try to catch the stakeholder's interest in the artifact and what it is capable of. The second (2) was chosen to demonstrate that the agent could collect data from different parts of the data sources and connect them to provide a reasonable response. This is a good question to demonstrate to the stakeholders, because it shows them that the artifact is capable of more than just a straightforward lookup. The third (3) was chosen to demonstrate that the system could calculate with the data it collected. Numbers and calculations are important in the dairy process industry. This question helps mitigate the perception that LLMs are bad at calculating numbers, which the stakeholders might have. Lastly, the open question from stakeholders was included in the plan as an attempt to increase the stakeholder's interest in the artifact and encourage engagement.

3.2.5 Evaluation

This activity involves measuring how well the artifact upholds a solution to the problem. This can be done, for example, by comparing a solution's results to a set of predefined reference answers or collecting qualitative or quantitative data from interviews or surveys. Evaluating the artifact serves as a good reference point when conducting multiple iterations—it gives a qualitative or quantitative indication that can be used to assess further development—and supports the results when communicating findings. [17]. The artifact in this research was evaluated in two ways, using strategies and suggested methods from Venable et al.'s *Framework for Evaluation in Design Science* [22].

Quantitative benchmark comparison

The first approach is a comparison method. A custom-built benchmark dataset was used across all iterations to evaluate correctness. It consisted of domain-specific questions with reference to correct answers, enabling direct comparison. The scoring procedure used for this evaluation is described in Section 3.3.1. This approach gave a quantifiable measure of how well a solution performed. For the first two iterations, this measurement yielded a good point of direction for iterating back to activities two and three (define objectives and design & development) in the DSR process, ensuring alignment with the research goals.

The custom benchmark set used for this evaluation approach is shown in Appendix D. It consisted of 20 dairy-specific questions related to processes, production, quality control, and other types. The questions were designed to have different complexities, varying from simple factual lookup to more reasoning required. For example, a simple question such as "Who was on the shift yesterday?" and a more difficult one such as "Was the process stable?". These questions were designed this way to reflect realistic user questions. Each question had its respective reference answer, which was used as a reference when scoring correctness. This set was kept static throughout the thesis and used for evaluating each iteration for consistent comparison between the solutions.

Following Venable et al.'s framework, an ex-ante, artificial, formative approach [22] was

selected, as it offers low cost and a fast path to quantitative feedback. Based on this strategy, performing controlled lab experiments was the most suitable option as an evaluation method. The rationale for adopting a quantitative benchmark comparison approach was due to its alignment with the research question—to what extent can a knowledge graph enhance querying capabilities in a chatbot—giving a quantitative measurement of each iteration’s performance. Additionally, it supported iterative development with rapid feedback because of immediate evaluation results after each iteration, guiding the next iteration.

Naturalistic stakeholder evaluation

The second evaluation approach applied was a semi-structured group interview with four participants from TINE AS. The interview happened through Teams and followed the Evaluation plan described in section 3.3.2. The participants evaluated the final artifact on two metrics: usability and usefulness. Complementary to the benchmark comparison approach providing quantitative results, this evaluation method gave valuable qualitative insights into the artifact’s potential use case, practical value, and user satisfaction.

The strategy selected in this evaluation is categorized as an ex-post, naturalistic, summative approach [22]. An appropriate method for this strategy was to use a semi-structured group interview to collect data. By using this method, I was able to collect valuable data on the final artifact’s usefulness and usability.

In summary, controlled lab experiments were used to evaluate all iterations during development, and a semi-structured group interview was conducted to evaluate the final artifact. This combination of strategies provided both feedback after each iteration and real-world validation. Further procedural details of these methods are provided in Section 3.3, and their results are discussed in Chapters 4 and 6.

3.2.6 Communication

Without proper communication of knowledge gained or lessons learned through research, the artifact and research lose key value. This final activity in the process, communication, is about conveying the problem’s significance, the artifact’s utility and novelty, its rigorous design, and effectiveness to relevant audiences. In scholarly publications, this often follows a standard empirical structure, reflecting the norms of the discipline [17].

For this research, communicating the knowledge gained and lessons learned is achieved through this thesis, aligning with the knowledge dissemination emphasized in DSR. Additionally, the insights gained throughout this research were shared with the participants from TINE AS during the final evaluation session. This was done with a structured presentation that summarized the artifact’s development steps, core functionality, potential use cases, and key findings.

3.3 Data collection methods

The following section will describe the methods I used for data collection, such as conducting comparative benchmarking evaluations and a final evaluation session with stakeholders. Each method will include reasons for choosing it and how it fits into my DSR process.

3.3.1 Comparative benchmarking

In software engineering, comparative benchmarking is an approach used to evaluate a solution. It compares a benchmark dataset—typically predefined and standardized, consisting of questions and reference answers—with the answers provided by the solution. This approach can be valuable in identifying improvements, ensuring quality, and evaluating a system [9]. It can also be a useful quantitative data collection method because it yields numerical scores.

In this thesis, an adapted comparative benchmarking method was conducted in the context of a lab experiment, following the ex-ante, artificial, formative strategy detailed in Venable et al. [22]. To be specific, the reason it is an adapted comparative benchmarking method is that the benchmark set used is not a publicly available standardized one, but was designed for this study to reflect realistic user questions. This method allowed for collecting quantitative data by comparing the solutions' answers to the custom-built benchmark set described in Section 3.2.5.

Using this approach aligned well with the RQ 1.1—seeking to find out to what extent integration of KGs in RAG application can enhance query generation—because it provided quantitative data for each solution. After each iteration, I collected ten answers for each benchmark question and gave them a score, varying from 0 - 100%, on correctness. For example, answering one question correctly out of ten times would yield a score of 10%. If the LLM answers—(1) The production volume in the last 24 hours was 18867.03 units, (2) The plant produced 18867.03 units in the last 24 hours, (3) The production volume in the last 24 hours was 1000 units, or (4) There are no available data to answer this—to the question "What was the production volume last 24 hours?", (1) and/or (2) ten times, it would be considered 100% correct, whereas (3) four times, (4) five times, and (1) or (2) one time, it would be considered 10% correct.

The rationale behind choosing correctness rather than, for example, latency, was, first of all, due to the importance of factually correct answers in the industry—knowing if milk contains a too high bacterial count is more important than retrieval speed. Secondly, to align with the scope of the thesis and not over-measure.

Additionally, for questions scoring below 50%, an error analysis, described in Section 3.4, was performed to identify reasons for failure. This involved trying to find out why the solution failed in obtaining a correct answer to that question. Performing this comparative benchmarking, I aimed to collect data, evaluate, and use the results and learnings to define future ideas, objectives, and refinements in alignment with the research questions.

3.3.2 Semi-structured group interview

A semi-structured group interview combines aspects of both semi-structured interviews and focus groups. It involves following a semi-structured interview approach in a group, with structured questions, and also allows for discussion among attendees and the researcher. A semi-structured group interview can be used to collect both qualitative data, through discussion, and quantitative data, via survey or rating responses.

Stakeholder

This method was applied in the final evaluation with stakeholders from TINE AS, Norway's largest dairy producer. Four representatives participated in the session, each with different roles and expertise, forming the focus group. Their roles and titles are as follows:

- Participant A: Head of TINE's Digitalization group in Supply Chain
- Participant B: Specialist/engineer, cheese and fat department
- Participant C: Specialist cheese and fat department
- Participant D: Head of TINE's Research and Development department

Evaluation plan

The semi-structured group interview was part of a comprehensive evaluation plan, which included demonstrating the artifact by asking questions from the benchmark set, conducting the interview, gathering feedback, and presenting my approach to solving the identified problem. It was logical to use the final iteration's prototype for the demonstration because it achieved the highest score in the lab experiment evaluation. The evaluation session followed the guide, created prior to the session, below. Specifically, the semi-structured interview is described in step three:

1. Introduction

Briefly explain the study, some information about me, and the purpose of the session.

2. Demonstration

- Share split screen: chat interface (left) and terminal logs (right).
- Explain what they see, what I will do, and the benchmark set.
- Ask three representative questions from the benchmark set:
 - (a) Did we run at full capacity?
 - (b) What were the reasons for lost time or capacity?
 - (c) How was the yield based on input raw material volume and output production?

After I have asked the three questions above, I ask if the stakeholders have any questions.

3. Semi-structured group interview: Verbal usefulness survey with Likert 1-6

Display Mentimeter slide with definition of usefulness and Likert scale: "fully disagree", "mostly disagree", "partially disagree", "partially agree", "mostly agree", and "fully agree". Each participant answers verbally on the following questions:

- The responses are useful for answering questions. Why?
- The responses are useful for finding root causes and prior actions. Why?
- When would you use the artifact?
- Will the artifact make you more efficient? How much?

- The responses are understandable. Why?

At the end, there will be an open question for improvement suggestions.

4. Presentation

Lastly, I will give a presentation of my study, including the application created, data sources used, KG structure and node attributes, development process, results from each iteration, and conclude with lessons learned.

Criteria

The participants included in the evaluation had to have experience with dairy-process decision-making to ensure that feedback on usefulness was realistic and reflected operational needs. All participants from TINE AS met this criterion, giving the evaluation sessions domain depth.

Likert scale

The artifact's usefulness score was captured by using a 6-point Likert scale ranging from (1) "fully disagree" to (6) "fully agree." By removing a neutral midpoint, I encouraged the participants to either take a negative or positive stance. The mean score across questions was calculated to obtain an overall usefulness score, reported in Section 4.4.4.

3.4 Data analysis

In this section, I will describe how the data collected in this study were analysed. There were mainly two analyses done, one for quantitative data collected through comparative benchmark results, and one for qualitative data collected through the stakeholder evaluation session.

Classification of failed questions

I analysed the comparative benchmarking results by categorizing every question with under 50% correctness into one of four failure types, defined as follows:

Label	Description
Invalid query	Syntactically incorrect query resulting in runtime error
Null	Syntactically correct query, but the result set is empty
Hallucinated	Response, but incorrect
Ambiguous question	Answer "I don't know" or "need clarification"

Table 3.1: Classification of failures applied to questions scoring < 50%

By using this classification, I could label questions and categorize what future iterations should focus on to improve correctness and query capabilities. Questions labeled "Invalid query" refer to the LLM generating syntactically incorrect queries that returned errors when being executed against the data source. For example, if the LLM generated `SELECT name FROM shift WHERE shift_date = '2024-10-19'::date - INTERVAL '1 day';` based on the question, "Who was on the shift yesterday?", it would be labeled with "Invalid query" due to name and shift not

existing in the data source. This highlighted that it could be effective to provide the data source schema in the prompt as context for the LLM.

Furthermore, questions labeled "Null" refer to when the LLM generated a syntactically correct query, but the query itself returned null or an empty dataset. For example, if the LLM generated `SELECT SUM(quantity) AS production_volume FROM "public"."productiondata" WHERE production_date >= CURRENT_DATE - INTERVAL '1 day';` to retrieve information regarding the question, "What was the production volume last 24 hours?", the question would be labeled "Null" due to it being correct, while returning null. This label gave a valuable insight into date handling, and led to specifying the system prompt with "use current_date = 18-10-2024", because that was the date used in the synthetic data. In hindsight, I learned that I should have focused more on this specific label; the why is explained in Section 4.1.5, but in short, it includes the query using Postgres' "CURRENT_DATE" rather than the date literal embedded in the LLM through the system prompt.

Additionally, questions labeled "Hallucinated" were put into this category because the LLM answered relatively good, but the main answer were factually incorrect. For example, the answer, "The production volume in the last 24 hours was 100 liters", to the question, "What was the production volume last 24 hours?" would label the question "Hallucinated" because the correct answer was 18867 liters.

Lastly, benchmark questions got labeled "Ambiguous question" when none of the above labels were suitable and the LLM returned "I don't know" or "need clarification" without reason. Since most of the questions in the benchmark set were ambiguous questions, over half of the questions got this label initially after the first evaluation.

Through the Iteration chapter, I will explain what caused the artifact to be able to answer these questions, and in the fourth and final iteration section, I will present the remaining questions the artifact was not able to achieve over 50% correctness on, including their remaining label and how I would further address them. Furthermore, categorizing the result helped me pinpoint patterns in the different areas of the solution and guided refinements for subsequent iterations.

Qualitative content analysis

For the data collected through the interview with stakeholders, I conducted a qualitative content analysis [14]. I first became familiar with the data collected by listening and reading through the recording and transcript on Microsoft Teams. Then, I created a category frame reflecting the interview guide: usefulness, retrieval relevance (reflecting the Likert questions), and open comments (reflecting the open questions). While I listened through the recording I divided these initial categories into finer distinctions—"actionable answer" referring to positive examples of directly usable output responses from the artifact, "training potential" referring to remarks on onboarding new operators, and "management-level reporting" referring to statements about using the system for shift summaries. Finally, I coded each participant's answers to the categories and systematically noted their answers, as presented in Section 4.4.4.

Chapter 4

Iterations

This chapter presents the different iterations in four sections. They all follow the same structure, and each consists of activities two to five in my DSR process: define the objectives of a solution, design and development, demonstration, and evaluation. Each iteration builds upon the last, using a systematic, iterative approach to refine the artifact. Additionally, in this chapter, I detail how technologies in this thesis were integrated, and the lessons learned sections under each iteration present how the respective technology improved correctness specifically.

4.1 Iteration 1: Baseline

This section describes iteration one, referred to as the *baseline*, which can be viewed as an initial prototype or Minimal Viable Product (MVP). It outlines the preliminary version of the artifact and the steps taken to establish a foundational understanding of the problem space. This includes formulating the solution’s intended goals, developing a first version of the architecture, demonstrating core functionality, and conducting an initial evaluation.

4.1.1 Objectives for a solution

Following the problem identification and literature review, a set of preliminary objectives was established to guide the development of a simple, initial solution. Although this solution was not intended to fully address the broader research problem, it served as a foundational step toward the larger goal. Inspired by LangChain’s modular RAG architecture [3], this iteration functioned as a starting point for subsequent developments.

The scope of the baseline was deliberately narrow, limited to an MVP that could test key ideas with minimal complexity. One of the few user needs identified at this stage was the ability to interact with an LLM-powered chatbot capable of retrieving relevant information from a database. This need helped shape the initial system requirements:

- A user interface that allowed users to input natural language queries.
- A storage layer for the information the LLM should retrieve from.
- Integration with an LLM.
- A basic retrieval function to support question answering.

These requirements were chosen to enable rapid prototyping and reduce uncertainty around architectural decisions, such as integrating the LLM with structured and unstructured data sources. By focusing on a minimal but functional core, the baseline could be developed and evaluated quickly, allowing early feedback and guiding the direction of future iterations.

The solution’s intended capability was to support a basic user-to-system query flow through a chat interface. The primary objective was to build a simple, feasible, and lightweight solution that could be easily tested.

4.1.2 Design and development

Succeeding the objectives for a solution, designing and developing the baseline began. The following sections describe what was done and why in the design and development activity of the process.

Design

The design of this first iteration was divided into two steps: the user interface and the application architecture. The user interface was fairly straightforward, as it should resemble a chat with few components. I chose this type of interface because I expected the user to be familiar with it. However, the application architecture was a bit more

technical. LangChain's RAG architecture inspired the application architecture, which is an adapted RAG pipeline depicted in Figure 4.1.

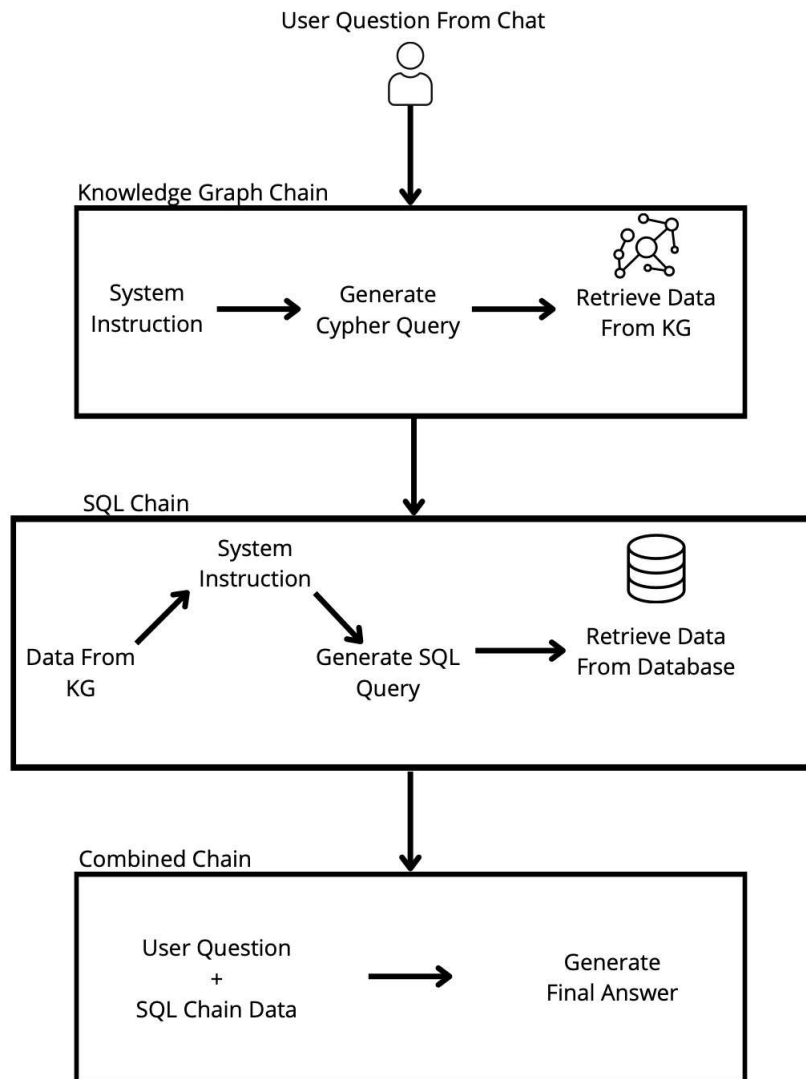


Figure 4.1: Adapted RAG architecture

The architecture, illustrated in Figure 4.1, shows a sequential program flow consisting of a few components. It starts with a user question from the chat interface, and then proceeds to retrieve data and form a final answer. The following sections describe the technical details of this architecture.

Development

In this iteration, I focused on implementing a functional prototype of a minimal end-to-end RAG pipeline connecting a frontend chat interface with an LLM retrieval backend. The goal was to test basic retrieval functionality in user-to-system interactions. The following paragraphs describe what I did in this development activity and the components used in the baseline's application architecture.

Backend

I implemented the backend with TypeScript and Express, setting up a simple POST endpoint that received the questions from the frontend. The endpoint called a handler function, which orchestrated the retrieval. I created three chains for different retrieval types: (1) for KG retrieval, (2) for Postgres retrieval, and (3) for formulating a useful and understandable response. Additionally, I tested how prompt structure and data source type affected response quality. This gave valuable insights into how to structure the prompt for future iterations, for example, specifically instructing the LLM that the date was "18-10-24" because that was when I created the sample data. The handler function passed the questions to a retriever function that initialized the chains from the backend architecture, shown in Figure 4.1, and returned the generated LLM response to the questions.

LLM and retrieval integration

The LLM I chose for the baseline was OpenAI's GPT-3.5, due to its out-of-the-box compatibility with LangChain. This was done by installing the required packages and importing them into the file where they would be used. LangChain provides an extensive library of open-source packages, and I used two of them to create the retrieval chains: *GraphCypherQAChain* and *SqlDatabaseChain*. These chains are extensions of the base chain class in LangChain and are specifically created for the use case I needed—to translate natural language to query language and execute a query to retrieve data. The two chains are quite similar, taking in an LLM, a prompt, and a data source as parameters. The difference is the data source type, which is a KG and a relational database. Implementing these two chains led me to the next step of the development process—integrating the data sources.

Data sources

Populating the Postgres instance was done by creating a script that read the CSV files containing the simulated data, generated an `insert.sql` file that matched the database schema, and ingested the data into the database. The schema included the tables: *processData*, *productionData*, *nonConformityRecords*, *rawMaterialInput*, *reports*, *shiftProcessLogs*, *sopData*, and *qualityData*, representing key data points in a dairy plant. The tables are presented in Tables 4.1-4.8. The simulated data was generated in collaboration with a stakeholder with domain experience in dairy processes to ensure it made sense.

Table 4.1: SOP Data

Column	Type
sop_id	SERIAL PK
procedure_name	VARCHAR(255)
description	TEXT
version	VARCHAR(10)
last_updated	DATE
spec_limits	VARCHAR(255)
process_guidelines	TEXT

Table 4.2: Process Data

Column	Type
process_id	SERIAL PK
process_name	VARCHAR(255)
start_time	TIMESTAMP
end_time	TIMESTAMP
temperature	DECIMAL(5,2)
pressure	DECIMAL(5,2)
flow_rate	DECIMAL(7,2)

Table 4.3: Production Data

Column	Type
production_id	SERIAL PK
product_name	VARCHAR(255)
batch_number	VARCHAR(50)
quantity	DECIMAL(10,2)
unit	VARCHAR(50)
production_date	DATE

Table 4.4: Quality Data

Column	Type
quality_id	INTEGER PK
batch_number	VARCHAR(50)
fat_content	DECIMAL(4,2)
protein_content	DECIMAL(4,2)
bacteria_count	INTEGER
pH_level	DECIMAL(3,2)
test_date	TIMESTAMP

Table 4.5: Raw Material Input

Column	Type
material_id	SERIAL PK
arrival_date	TIMESTAMP
supplier_name	VARCHAR(255)
material_type	VARCHAR(100)
quantity	DECIMAL(10,2)
unit	VARCHAR(50)
quality_check	VARCHAR(50)
remarks	TEXT

Table 4.6: Reports

Column	Type
report_id	SERIAL PK
report_type	VARCHAR(50)
start_date	DATE
end_date	DATE
report_content	TEXT

Table 4.7: Shift Process Logs

Column	Type
log_id	SERIAL PK
shift_date	DATE
shift_number	INTEGER
operator_name	VARCHAR(255)
log_entry	TEXT

Table 4.8: Nonconformity Records

Column	Type
record_id	SERIAL PK
deviation_date	TIMESTAMP
description	TEXT
severity	VARCHAR(50)
action_taken	TEXT
resolved_date	TIMESTAMP

Creating and populating the KG, on the other hand, was done by extracting the necessary nodes and relationships from the Postgres database schema, and writing a text document including both the Tables 4.1-4.8, and definitions in the format: node → [relationship] → node, containing a description of that relationship. I structured it as follows:

- **Relationship:** ShiftProcessLogs LINKS_TO ProcessData
Description: Each entry in ShiftProcessLogs is associated with one or more

entries in `ProcessData`, indicating the processes carried out during a specific shift.

- **Relationship:** `RawMaterialInput LINKS_TO ProcessData`
Description: Raw materials supplied in `RawMaterialInput` are utilized in `ProcessData` entries, linking suppliers and materials to specific processes.
- **Relationship:** `ProcessData PRODUCES ProductionData`
Description: Each process recorded in `ProcessData` results in the production of goods recorded in `ProductionData`.
- **Relationship:** `ProductionData LINKS_TO QualityData`
Description: The `batch_number` in `ProductionData` links to `QualityData`, associating production batches with their respective quality measurements.
- **Relationship:** `ProductionData AFFECTS NonConformityRecords`
Description: Issues or deviations related to production batches are recorded in `NonConformityRecords`, linking specific productions to non-conformities.
- **Relationship:** `Reports SUMMARIZES NonConformityRecords`
Description: Reports summarize data from `NonConformityRecords`, providing aggregated views of deviations and resolutions.
- **Relationship:** `sop_data GUIDES ProcessData`
Description: Standard Operating Procedures (`sop_data`) guide the execution of processes recorded in `ProcessData`, ensuring compliance with established guidelines.
- **Relationship:** `Reports INCLUDES QualityData`
Description: Reports include summaries and details from `QualityData`, offering insights into product quality over specified periods.

Then I could loop through this document and create nodes with attributes and relationships with descriptions, linking them. I used the *GraphDatabase* class from Neo4j [3] to write this data to the KG. Figure 4.2 presents a small part of the KG, including entities (brown nodes) and relationships linking them, highlighting the "produces" relationship with its description.

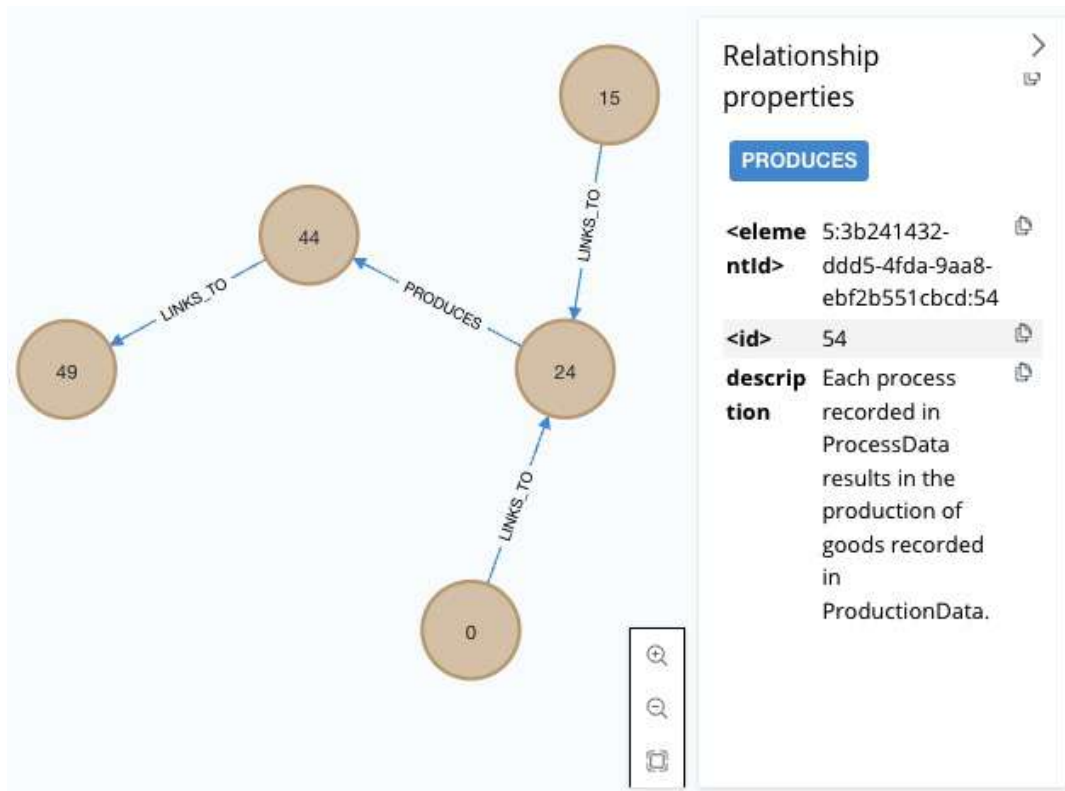


Figure 4.2: Screenshot from Neo4j - Part of the KG

I did not populate the KG with the whole dataset due to the scope of the baseline. I deliberately deferred populating the KG with actual data to later iterations because I wanted both to understand if providing contextual metadata about how the data was connected to the SQL chain would give an increased correctness score, and align with the scope of the baseline—to keep it minimal.

Frontend

I implemented the frontend using React, a framework by Meta, and TypeScript. As a simple chat interface, consisting of a chat window with messages and an input field. When a message gets submitted, a function is called with the input and routed to the endpoint in the backend, which returns the response and stores it as a message in the chat. Although this aligned with the objectives for a solution—a user interface that allowed interaction with an LLM—the frontend was not the focus of this first iteration, hence its brief description. Chapter 5 includes details of this part of the artifact.

4.1.3 Demonstration

The baseline was demonstrated to the stakeholder, described in Section 3.2.1, informally and straightforwardly to test functionality. I presented the solution for this iteration—chat interface with the RAG pipeline—and asked questions from the custom benchmark set. I demonstrated that the baseline struggled to answer some of the questions, which required a great deal of context, and that it was able to answer the simpler questions—factual ones—correctly. By simple questions here, I refer to, for example, "What is the total quantity of raw materials received from each supplier?" or "Who were on the

shift yesterday?", which are defined as simple because they provide enough information for the LLM to perform a straightforward lookup in the database. Whereas questions required more context refer to, for example, "Was the process stable?" or "What were the root causes for the deviations?". These are classified as complex because they require additional context, like what a stable process is or which deviations are referred to, that makes it difficult for the LLM to interpret which data it should retrieve.

4.1.4 Evaluation

This section presents the evaluation activity of the DSR method in the baseline iteration. It will describe how the evaluation was conducted. Then, continue to present the results from the custom benchmark comparative approach. Finally, remark how this evaluation supported the next iteration in defining objectives for a solution.

Obtaining evaluation material

Evaluation of the baseline was done by collecting quantitative data through the quantitative benchmark comparison approach described in Section 3.2.5. I automated the process by creating a small program that looped through the benchmark set, calling the handler function in the backend ten times for each question, collecting the responses, and writing them to a text document. The reason for this was not to spend time writing the question in the user interface manually. I then compared the final answers and the reference answers as described in Section 3.3.1, and gave each question a total score.

Benchmark results

The baseline scored a total of 26% correct on the custom benchmark set. The questions it got correct were mostly factual ones that did not require reasoning or a deeper understanding of the question. Although this might seem like a low number, it was as expected, and in alignment with the objectives—to create a bare minimum retrieval application.

One example of a question the artifact got incorrect was, "Was the process stable?", due to the multi-step reasoning needed for answering. For example, it would need to retrieve data from the standard operating procedure, calculate the mean and standard deviation, and lastly, calculate the process capability index using a specific formula. However, one question the artifact managed to get correct was, "What is the total quantity of raw materials received from each supplier?", because it only requires a simple SQL query to retrieve the answer to this.

The reason for the score being low for this first iteration includes simple system instructions, failed queries generated leading to syntax errors and wrongly retrieved data, and missing question context. This underscored the need for a robust retrieval logic, like, retries when queries failed, which helped define the objectives for the subsequent iteration.

4.1.5 Lessons learned

This first iteration was great to identify both small and large pitfalls early in the process. In this section, I present the main lessons learned from creating this baseline, including what and why, with examples.

Firstly, providing context is essential because the LLM will retrieve data from external data sources; therefore, it needs to know the schema of these to not generate syntactically incorrect queries. Without providing the schema, the LLM will most certainly hallucinate table and column names and return errors. For example, without providing the schemas and questioning the LLM with, "Who was on the shift yesterday?", it generated `SELECT name FROM shifts WHERE date = CURRENT_DATE - 1;` which returned invalid due to "name" and "shifts" not existing. Whereas when providing the schema, it generated `SELECT operator_name FROM shiftprocesslogs WHERE shift_date = CURRENT_DATE - 1;` resulting in correct retrieval.

Another important lesson learned, that I did not identify before all four iterations were developed, was that instructing the LLM in the prompt with "CURRENT_DATE" causes problems. I did this due to the data being synthetically generated, and all the dates were fixed (up to 18-10-2024). By instructing the LLM with "CURRENT_DATE = '18-10-2024'", it sometimes generated queries using Postgres' "CURRENT_DATE", which was the actual date and not the LLM's internally embedded date, resulting in no data being available for that date. The fix for this is described in Section 4.3.5, but in short, it involved instructing the LLM to never use the database functions like "CURRENT_DATE" and "always embed the date literal from the LLM's date context".

4.2 Iteration 2: Agent and LangGraph

This section describes the second iteration of my DSR process, detailing the solution's objectives, design, development, and demonstration and evaluation activities.

4.2.1 Objectives for a solution

Following the baseline evaluation, described in section 4.1.4, various improvement areas and limitations were identified. These findings contributed to defining the objectives of a solution in this iteration. In addition, the preliminary survey on technologies also gave valuable insights into potential frameworks or concepts to apply in this iteration. The goal of the second iteration was to make the architecture more robust. As part of investigating to what extent a KG could enhance the querying capabilities of a chatbot, I sought to understand if an architecture change would yield improved results, rather than giving the model access to more data. The objectives of this iteration included:

- Improve the retrieval functionality by using the chains as tools.
- Implement an agent to orchestrate the retrieval with retry logic.
- Change from static to dynamic flow with LangGraph-based architecture.

These objectives were chosen based on several reasons. But in short, to test if the architecture was the reason for the baseline not using the KG fully, I wanted to improve the architecture. The second reason was inspired by the idea that the agent could retrieve data from the KG, and use it to retrieve data from the Postgres data source, whereas if it failed, reiterate back to querying the KG and try with different information. In other words, to determine if the agent would use the KG more strategically than in the baseline. The last reason was to prepare the architecture for an expanded KG in future iterations.

4.2.2 Design and development

Building on the established requirements from the solution's objectives, the following paragraphs describe how these objectives were turned into a design and further developed and implemented.

Design

The objectives for a solution in this iteration did not reflect any changes needed to the chat interface, which remained unchanged in the second iteration. However, the design of the system architecture needed some changes. The design of a solution in this iteration was influenced by LangChain's LangGraph architecture.

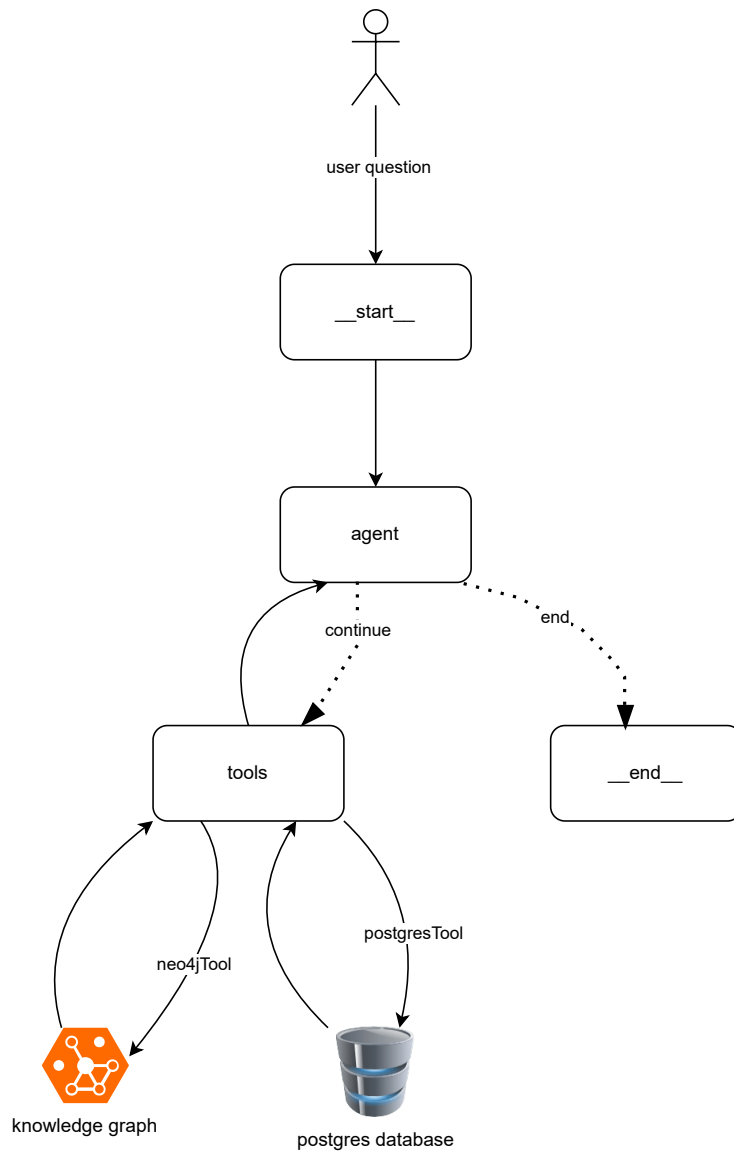


Figure 4.3: Retrieval flow in the second iteration

The design illustrated in Figure 4.3 describes the architectural changes to the retrieval part of the system. It introduces an iterative tool-calling approach, in contrast to the baseline architecture, where both the program flow and retrieval were sequential with chains. The architecture is structured as a graph, consisting of nodes and edges, to steer the flow of the application. In this architecture, the agent has a set of tools it can use to retrieve data from the data sources—which essentially is the chains from the baseline, but the agent is instructed on when it should use them rather than the strict sequential flow from the baseline—and if the tool call returns an error, try again by generating a new query. If the tool call does not return an error and the agent does not need more information, it routes to the "end" node and returns the answer.

Development

The development done in this iteration builds directly on the foundation created in the baseline iteration. The variables that stayed static in this iteration were the frontend, data sources, LLM, and the built-in LangChain classes *GraphCypherQAChain* and *SqlDatabaseChain*. The following paragraph describes how and what changed in the development activity in this iteration, in alignment with the objectives.

Wrapping chains as tools

This part did not require much change, due to LangChain's built-in *tool* function. This function creates a new *StructuredTool* instance with the provided function (chain), name, description, and schema. The schema can be provided as Zod or JSON. With the use of Zod, the tool inputs will be validated. This comes in handy when creating a larger system due to compile-time error identification, contrary to runtime. I decided to use Zod due to improved robustness. Wrapping the chains as tools enabled the agent to iteratively call them and choose which to use, rather than using them in sequence as in the baseline. This affected the results because many queries returned incorrect answers due to the SQL or Cypher queries in the baseline being syntactically incorrect. With this method, the agent could retry a query, ultimately improving correctness on these questions.

Agent Implementation

Implementation of the agent was also quite straightforward, by using the *ChatOpenAI* class from OpenAI for initialization, and tuning the prompt and system instruction to reflect the new functionality the agent had. With these two new tools implemented, the only thing missing was to give the agent access to them. This was done using the *ChatOpenAI* class's function *bindTools*, which takes the tool as the parameter. With this implemented, the agent could determine which tool to use and how often. A more technical explanation of this will follow in Chapter 5.

LangGraph Architecture

Moving to a graph-based and dynamic flow, from the sequential and static flow in the baseline, was achieved using the *StateGraph* class from LangGraph. The main functionality of the class is to represent components in the program as nodes and move the state between these, as illustrated in Figure 4.3. By defining the agent and tools as nodes, it completely changed the flow; the *StateGraph* allowed for the creation of conditional edges where I could determine the action of the agent node regarding the tool use. The agent could then consider if it was satisfied with the answer—its answer to the user's question—or if it needed to retrieve more information. This came in handy when the agent generated an invalid or failed query because the tool returned an error, and the agent could retry with another query based on the error. This aligned with the objective—to test if the agent could leverage the KG more strategically without using more data. This architecture was useful due to it providing modularity, agent logic, tools, and nodes were decoupled, and a stateful loop. By looping between the agent and tools, the system maintained conversation context and only terminated when no further tool calls were needed. A technical description of this architecture will be presented in the Final artifact chapter.

4.2.3 Evaluation

This section describes the evaluation of the second iteration. It only includes the results from the quantitative comparative benchmarking approach and the interpretations of them, as obtaining the evaluation data and the demonstration happened in the same way as for the previous iteration—giving each question a correctness score by comparing the LLM's answers to the reference answers, and demonstrating the solution to the internal stakeholder.

Benchmark results

Driven by improved retrieval logic—equipping the agent with tools for retrieval in a LangGraph architecture—the total correctness score rose to 57.5%. Questions like "Have we documented any non-conformities?" and "What were the reasons for lost time or capacity?" saw an increase from 0 to 40% correct out of the ten runs. The reason for this was that the agent could retry with different queries when one failed, which the previous iteration was not able to. However, the solution still struggled to answer complex questions, like, "How was the yield based on input raw material volume and output production?", which it scored 0% on. This guided the objectives for a solution in the next iteration—enrich the KG—in an attempt to enable the agent to answer complex questions where reasoning is required.

4.2.4 Lessons learned

The main lesson learned in this iteration was that introducing the agent, tools, and graph architecture enabled the system to retry queries, but the problem was that most queries did not return an error; they instead returned null data. This happened because, even though the query was syntactically correct, the agent used Postgres' "CURRENT_DATE" function due to most questions being date-related. To take account for this, I added the sentence, "If the tool calls returns 0 or null, try reformulating the query, with date format aligning with the data source's formatting", to the agent's system instruction. This resulted in a lot fewer "data not available" responses and improved the agent's ability to provide a correct response. Additionally, it emphasizes that clear system instructions and guidelines are crucial to combine and align with tools and architecture.

4.3 Iteration 3: Expanding the knowledge graph

This section presents the third iteration of my DSR process. It will follow the same structure as the two previous iterations, starting with defining the objectives for a solution. Moving on to design and development, and lastly, demonstration and evaluation. Based on insights gained from the second iteration, specifically with the agent's reasoning process in the LangGraph architecture with a small KG, this iteration aimed to expand the KG to contain the same data as the Postgres database and test if it would enhance retrieval capabilities. The following subsections will detail how iteration three was conducted.

4.3.1 Objectives for a solution

Possible improvement areas extracted from analyzing the results of iteration two in alignment with the research question—to what extent can a knowledge graph enhance querying capabilities in a chatbot—helped form objectives for a solution in this iteration. Which included enrichment of the KG and using this as the only data source to retrieve from. As argued in [26], advanced reasoning capabilities can be facilitated through a well-structured KG because it provides rich contextual connections. The goal of this iteration was to find out if a KG was a more suitable option as a data structure for retrieval than a regular relational database. Building on the previous iteration, this led to one objective:

- Expand the KG to include the same raw data as the Postgres database.

The rationale behind this objective was to build a solution that aligned with the research questions of this thesis. By expanding the KG, I sought to evaluate if providing richer context and connections to the agent would help me answer the research question through comparing this iteration to the previous.

4.3.2 Design and development

The objectives for a solution in this iteration only involved making changes to the knowledge graph and the system instructions, and keeping the other variables in the system static. This meant finding a suitable graph topology and populating the KG. How it was done is described in this section.

Design

Designing this solution meant creating a program to populate the KG. All the data used to populate the graph was located in CSV files. A challenge was figuring out how to link the nodes, due to the absence of foreign keys in these data files. After exploring some possibilities, a simplified, star-shaped graph design was chosen as an alternative to a chain-like topology. The rationale behind this was that the star-shaped design offered one hop from the main node, in contrast to the chain topology with long paths. It involved centering the data nodes around a main "Dairy Plant" node, which would allow the agent to traverse the graph through this, ensuring basic contextual connections. Figure 4.4 illustrates this simple graph structure, with the main node in the center, and different node types related to it with specific relationship edges.

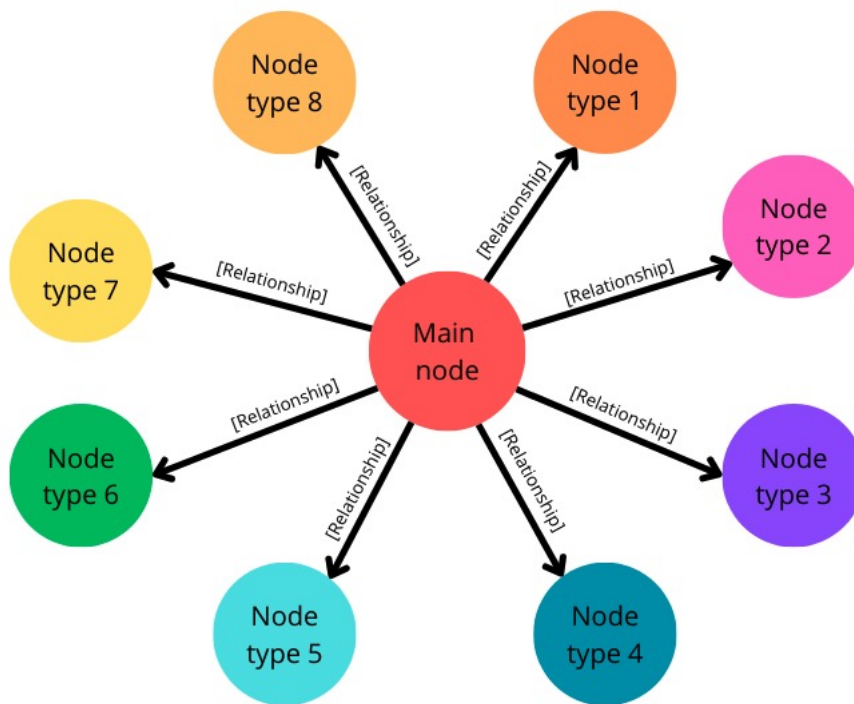


Figure 4.4: Design of star-shaped KG topology

Development

With this simplified graph design, the development process could start. It consisted mainly of two tasks: constructing a script to populate the graph, and integrating the expanded graph with the current system, which also meant removing the possibility for the agent to use the Postgres tool. The following subsections describe these two tasks in the way they were performed and the challenges encountered when implementing them.

Expanding the knowledge graph

Initially, doing some research on how this had been done before, I thought I could use Neo4j's ETL to import the data into the KG. But this configuration was not as straightforward as it first seemed. It took multiple hours to import, and when one import attempt was done, the structure of the KG was completely incorrect, mixing node types and not making sense—approximately 5000 nodes. This led me to write an importing script, consisting of a database connection, designing a simplified graph, and populating it with nodes and relationships.

Establishing a secure connection to Neo4j was done by instantiating the Driver class with environment variables. This initial part also cleared existing nodes and relationships, and created unique constraints for each node type, ensuring a clean slate and data integrity. The second part of the script was to create the main node "Dairy Plant".

The last part of the populating script was to loop through each line in each CSV file and create nodes for each, and establish the star-shaped structure by linking each node to the "Dairy Plant" node. Nodes were created with their respective attributes properly typed; for example, dates were configured as datetime objects. Edges were established from the main "Dairy Plant" node to data nodes with meaningful labels, for example,

"Has_Process" or "Has_report". Specifically, the script populated the KG with these node types and respective attributes:

- **ProcessData:** process_id, process_name, start_time, end_time, temperature, pressure, flow_rate.
- **RawMaterialInput:** material_id, arrival_date, supplier_name, material_type, quantity, unit, quality_check, remarks.
- **NonConformityRecords:** record_id, deviation_date, description, severity, action_taken, resolved_date.
- **ProductionData:** production_id, product_name, batch_number, quantity, unit, production_date.
- **QualityData:** quality_id, batch_number, fat_content, protein_content, bacteria_count, pH_level, test_date.
- **Reports:** report_id, report_type, start_date, end_date, report_content.
- **ShiftProcessLogs:** log_id, shift_date, shift_number, operator_name, log_entry.
- **SOP (Standard Operating Procedures):** sop_id, process_name, description, version, last_updated, spec_limits, process_guidelines.

After the script was successful, it logged out a success message and safely closed the connection to free resources.

Some technical challenges were encountered when populating the KG. First, identifying relationships without foreign keys or explicit IDs was challenging. Choosing the star-shaped structure solved traversal, but it also meant that each node was one edge away from the root, eliminating deeper semantical paths. I tried to find ways to connect the data to form a deeper graph, but since it was synthetic data, the actual rows with the same IDs contained data that did not match, making it difficult to link them. Second, the data types from the CSV files were string representations, which led the agent to generate invalid queries. To solve this, I updated the script to cast in Cypher—the programming language used for querying KGs—to the correct types, for example, *toFloat()* or *datetime()*, when populating the graph. Lastly, using *datetime()* in Cypher with the dates from the CSV files resulted in a syntax error due to invalid formatting. The function expects ISO 8601 format, like "2024-05-04T03:24:51", but the input dates were in the format "2024-05-04 03:24:51" using space as the separator rather than "T". To solve this minor challenge, I used VSCode's "replace all" function with a regex to change spaces in dates with "T". These challenges were the ones worth mentioning, and with them fixed, I could move on to updating the system to use the expanded KG.

Integrating the graph and refining the prompt

With the expanded graph, the final part of the objective for this solution was to update the agent's prompt on how to use the graph and remove the Postgres tool from its toolbox. The agent prompt in the second iteration focused on using the KG tool for structuring and capturing relationships, and the Postgres tool to retrieve data. In this iteration, it changed focus to using the KG tool explicitly. In addition, removing the Postgres tool led to the agent only having one tool available. Therefore, the prompt

needed to reflect an iterative use of this tool. This also introduced a new flow, as the action involved in the tool node got reduced from two to one tool available; the agent chose between whether it was satisfied or needed to refine the Cypher query and retrieve again. The following are the prompt changes done in this iteration:

- **Tool availability:** Iteration two listed both the *neo4jTool* and *postgresTool*, whereas iteration three only listed *neo4jTool*.
- **Grounding rules:** Iteration two loosely instructed to "not hallucinate", "use this tool for the final result", and "never guess, whereas iteration three introduced "core rules" for this, in terms of data to use and question analysis to break down the question.
- **Reasoning strategy:** Iteration two instructed "bonus point for reasoning and finding answers that are not explicitly stated", whereas iteration three introduced a general "question handling strategy" explaining how the agent could use the KG, in addition to providing two examples on how to use this strategy.
- **Final checklist:** Iteration three introduced a checklist including best practices, like "Look for relationships between different data points" and "Consider both direct and indirect causes".

These prompt changes were made to instruct the agent on how to navigate the KG and address its struggles from the previous iteration.

4.3.3 Demonstration

I did not conduct a demonstration for the solution in this iteration because the changes were focused on expanding the internal data structure, the KG, rather than modifying the user interface. Since the user experience remained unchanged, a demonstration was deemed unnecessary. This did not affect the evaluation, which proceeded as before, using the adapted comparative benchmarking approach.

4.3.4 Evaluation

Evaluating this iteration was done identically to the two previous iterations, described in Section 3.2.5, using the comparative benchmarking approach. The questions in the benchmark set were kept unchanged to maintain consistency and comparability between iterations.

Benchmark result

Expanding the KG and refining the agent instructions gave an increase in the correctness score to 75.4% in total, up from 57.5% in the previous iteration. One example question this solution got correct is "How was the yield based on input raw material volume and output production?", which the previous iterations scored 0% on. These results show that the expanded KG and a more specific agent instruction yielded an increase in the correctness. However, when I measured the latency on ten runs with the question "What was the production volume last 24 hours?", iteration two—using mostly the *postgresTool*—had an average latency of 4.27s, while iteration three measured 4.90s. This helped me define the objective for the last iteration—to combine the two retrieval approaches to get both the advantage of a slightly quicker lookup and the ability to answer complex questions.

4.3.5 System-level ablation studies: isolating KG vs prompt

Ablation studies are normally conducted in studies regarding artificial neural networks and involve isolating specific parts of a system in a controlled and systematic manner to observe how these parts—isolated—affect the system’s ability to perform a task [15]. I specify "system-level" because I use it to evaluate the effect of removing data sources or changing the prompt, rather than neural network-weight ablations.

In iteration three, I introduced a new topology in the KG, populated it with the same raw data as the Postgres data source, and removed the *postgresTool* from the agent’s toolbox. These changes also implied changing the prompt to instruct the agent on how to retrieve data from the KG. Making these changes too extensive caused the evaluation of this iteration to be invalid, as the result derived from both changes and not the KG solely.

To measure the true effect of the KG, I conducted three system-level ablation studies, derived from iteration two:

1. Iteration two’s prompt with enriched KG as the only data source
2. Iteration three’s new prompt with only the Postgres data source
3. Iteration three’s new prompt customized for the *postgresTool* with Postgres as the only data source

These studies isolate (1) the impact of the KG when paired with the old prompt, and (2-3) the prompt’s impact when paired with the Postgres data source alone, including the effect of correcting a date bug. To conduct (1), I changed iteration three’s prompt from specific to simple, as in iteration two, but customized for the KG, rather than the Postgres data source. This meant no explicit instructions on how to use the KG or example queries, just general guidelines for how to behave, for example, "Users will ask you questions that require you to retrieve data from a knowledge graph database, you have one tool available for this: "neo4jTool" for retrieving (generating and running queries) data from the database", "If you see a valid tool result, your final answer must reflect that entire result", and "Never guess or hallucinate data not in the tool result". I then collected the agent’s response and used the same comparative benchmarking approach to score its results.

To conduct (2), I changed iteration three to be Postgres specific, which included changing from *neo4jTool* to *postgresTool* everywhere in the prompt and initializing the Postgres database instead of the KG, and keeping all other variables static. When I evaluated, I identified an error—the prompt included "CURRENT_DATE=2024-10-19"—which creates problems for the agent when it tries to use it in a query to the database. This is because it uses the actual current data and not the one specified in the prompt, resulting in no data being available. However, I ran the evaluation and scored the responses—using the same method—to keep the ablation valid by not tweaking the prompt.

For ablation three, I fixed the date problem to isolate the effect of this, since most questions are date-related. The only change I did here from ablation two was to remove "CURRENT_DATE" and add "Core rule 1: Never use database functions like CURRENT_DATE. Always embed the date literal from the LLM’s date context (e.g.

'2024-10-19')". Table 4.9 summarizes the correctness score across these ablation studies and iterations one and two as a reference:

Configuration	Correctness (%)
Baseline	25.0
Iteration 2	57.5
Ablation 1 (Iteration 2 prompt in Iteration 3's only KG structure)	36.0
Ablation 2 (Iteration 3 prompt + Postgres only)	19.0
Ablation 3 (Ablation 2 with fixed date problem)	63.5

Table 4.9: Results of comparative benchmarking for ablation studies

4.3.6 Lessons learned

The main lessons learned in this iteration were that introducing a new KG topology and refining the prompt brought several unforeseen challenges, mainly in terms of populating the KG with synthetic, randomly generated data. By not having foreign keys in the data, finding a topology for the graph expansion was difficult due to the lack of a clear link between nodes. A chain topology was not chosen due to potential traversal difficulties, while a structure similar to the baseline would risk combinatorial retrieval overflow. Therefore, a pragmatic compromise—the star-shape—was chosen to ensure that every entity was exactly one hop away, although sacrificing semantic reasoning that would potentially come from a deeper graph. Additional takeaways are that date-formatting system instructions also apply for KG-only retrieval, and reducing the toolbox traded retrieval speed for correctness.

In essence, choosing a suitable graph topology is both important and difficult, and as stated in the previous "lessons learned" sections, instructing the LLM on how to use these data sources is equally important as to which data structures to use.

4.4 Iteration 4: Hybrid retrieval

This section describes the fourth and final iteration. It includes, like the other sections in this chapter, the steps in my DSR process, starting with objectives for a solution, moving on to design and development, and finally ending with demonstration and evaluation. The evaluation section, unlike the previous iteration sections, will include two parts: one for the final evaluation with TINE AS and one for the adapted comparative benchmarking. Additionally, a summary of the comparative benchmarking scores for all iterations will be presented.

4.4.1 Objectives for a solution

Building on insights from iterations two and three, I wanted to conduct one final experiment to test whether a combination of the previous iterations would yield an improvement in correctness score on the benchmark set. The rationale behind this experiment was that I thought the enriched KG from iteration three would complement the agent from iteration two, in that it could utilize the KG and Postgres tool for different questions. For example, the KG could be used for capturing deeper relationships, and Postgres for efficient lookup. Since this was the final iteration, it meant that stakeholders from Tine AS would evaluate the system. I wanted them to evaluate the final artifact on "usefulness", which refers to how applicable and beneficial the artifact is to workers using it. This led to the following objectives:

- Improve agent's correctness on the benchmark.
- Achieve an average stakeholder usefulness rating of 4 or higher on a 6-point Likert scale.

Improving correctness is one objective because that would make the artifact improve usefulness in its potential use case, and to find out to what extent the KG-augmented retrieval supports query capabilities. The goal of this solution, in short, was to conduct one final experiment and make the artifact ready for the final evaluation.

4.4.2 Design and development

Drawing from the objectives, the following design subsection explains the conceptual adjustments needed, mainly how the agent's decision logic changed. It then briefly describes how these adjustments were implemented in the development subsection.

Design

The design of the solution centered on creating a hybrid retrieval configuration. This meant that the agent prompt needed an update reflecting how to use the two tools again, with the enriched KG. Additionally, the flow from iteration two—the graph architecture—needed an update because this is where the tools were defined and integrated. Lastly, having two retrieval tools, I thought it would be nice to have a fallback rule if one tool failed.

Development

The development phase in this iteration mainly consisted of three changes derived from the design activity. The following is a description of the changes I made.

Agent prompt: I prompted the agent to use the tools as I would have used them—the *neo4jTool* for when it thought the question required it to collect data from multiple parts of the data source, and *postgresTool* for when it thought the question could easily be answered by a straightforward lookup, resulting in the following part of the system prompt:

Available tools:

1. **neo4jTool** - Generates a Cypher query based on the user question and returns results in natural language.
Best for: connected data, entity relationships, process flows, quality correlations
Query format: Cypher query language with specific **datetime** handling requirements
Example:

```
MATCH (d:DairyPlant)-[:HAS_QUALITY_DATA]->(q:QualityData)
WHERE datetime(q.test_date)=datetime('2024-10-18T00:00:00Z')
RETURN q
```
2. **postgresTool** - generates SQL queries for the relational database and returns structured results.
Best for: detailed operational metrics, historical trends, standard reports, inventory tracking
Query format: standard SQL with table joins where necessary
Example:

```
SELECT *
FROM productiondata
WHERE date = '2024-10-18';
```

Tool-selection guidelines:

Use *neo4jTool* when:

- Question involves relationships between different data types (e.g., how process affects quality)
- Need to traverse connected entities (e.g., tracing product from raw materials to final tests)
- Looking for patterns within interconnected systems

Use *postgresTool* when:

- Question requires detailed operational metrics
- Need for precise historical data with standardized formatting
- Question involves reporting on structured data like inventory, staffing, or financial metrics

Use **both tools** when:

- Complex questions require data synthesis from multiple sources
- Verification of findings across different data systems is needed

LangGraph architecture update: I updated the flow in the LangGraph architecture to include the *postgresTool* from the previous iterations again. This was done the same way as previously.

Fallback instructions: I created a fallback instruction in the system prompt to switch to another tool if it returned repeated errors. Because each tool had a retry loop when it encountered an error in the LangGraph architecture. If this happened multiple times, resulting in an error response after the loop, the agent should try the other tool.

Observability logs: To see in real-time what tool the agent chose, and what query it used to retrieve data, I added logs.

4.4.3 Demonstration

A demonstration of the final artifact was conducted via Microsoft Teams, and four domain experts from TINE AS, as described in Chapter 3 under Data collection, attended. I followed the steps in the evaluation plan from Section 3.3.2, initially sharing my screen, showing the chat interface (Figure 4.5) and real-time terminal logs of the agent’s tool selection, generated queries, and raw data retrieved, which can be viewed in Appendix C. Subsequently, I gave a short presentation of the benchmark set and what I wanted to demonstrate for them—the three questions from the evaluation plan. Next, I started asking questions in the chat interface, and for each question, participants could observe the full question-answering cycle: the agent’s response in natural language in the chat interface, as well as the tool calls and queries used in the terminal window. For the third question, the agent was unable to retrieve data using the *postgresTool*, but when I asked it to use the *neo4jTool* instead, it found the correct data and responded with it. This demonstrated that the tools could be complementary.

After I had asked the last questions, I asked the participants if they had any questions they would like to ask the artifact. They misunderstood and thought I asked if they had any questions in general, and started to ask about how this was working, for example, in terms of retrieval. After some clarification, they couldn’t think of anything they would ask the artifact, and the demonstration session ended. The figure below illustrates the chat interface and the questions asked in the demonstration. The terminal window and the split screen setup used can be viewed in Appendices C and B, respectively.

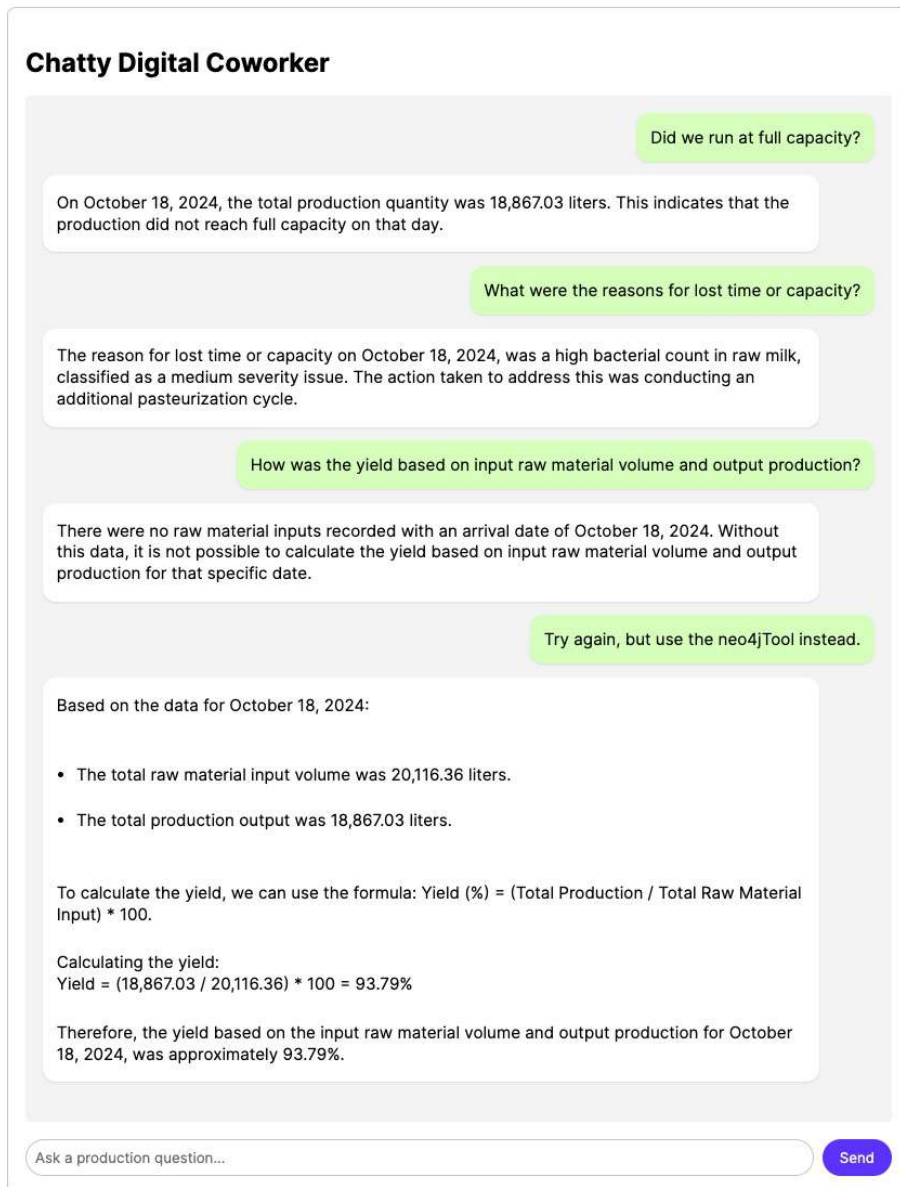


Figure 4.5: Final iteration chat interface

Figure 4.5 illustrates the chat interface, including the user's questions highlighted in green and the agent's response in white.

4.4.4 Evaluation

As mentioned in the introduction Section 4.4, the evaluation of this final artifact was conducted in two ways: with stakeholders from TINE AS and comparative benchmarking. The following sections will present these separately, starting with stakeholder evaluation, proceeding to comparative benchmarking for this iteration, and concluding with results for all iterations.

Stakeholder evaluation

The evaluation method used was a semi-structured group interview, described in the Methodology Chapter, which also includes the evaluation plan followed to conduct this evaluation session. The group comprised of Head of Digitalisation (supply chain), two senior process specialists in cheese and fat, and the Head of R&D, providing both operational and management perspectives. This section presents the results of this session. Each participant answered the following questions using the 6-point Likert scale for usefulness presented in the evaluation plan and their reflections in open questions:

1. The responses provided by the artifact are useful for answering questions. Why?

- Participant A: **Fully agree**, because the answers were actionable answers that are very useful.
- Participant B: **Fully agree**, because if an operator could use this as a tool during production for an issue they are in the middle of, where they do not have time to go into the office and look up whatever the issue is, then it's very useful.
- Participant C: **Fully agree**, because the idea of using natural language to get answers from production is very useful.
- Participant D: **Partially agree**, I don't know why.

2. The responses are useful for finding root causes and prior actions. Why?

- Participant A: **Mostly or partially agree**, if we had more questions demonstrated, it would be easier to explain.
- Participant B: **Agreed with participant A**
- Participant C: **Agreed with participant A**
- Participant D: **Agreed with participant A**

3. When would you use the artifact?

- Participant A: Management level, in terms of reporting, use the artifact to present a summary of the facts and actions that have happened during the shift.
- Participant B: Did not have an answer.
- Participant C: Several "levels": as an operator to get information about a process as it's running, outlining next steps in the process, malfunction analysis, and also on a management level to get an overview of performance.
- Participant D: Did not have an answer.

4. Will the artifact make you more efficient? How much?

- Participant A: Initially, think maybe for "micro-efficiency" to do smaller iterative process improvements.
- Participant B: I don't expect this artifact to answer all my questions as an operator, but to point me in the right direction of where the process is "burning".

- Participant C: Agree with A, but I also see it as a tool for training personnel, for example, an operator with little experience could get help to do the correct actions.
- Participant D: Did not have an answer.

5. The responses are understandable. Why?

- Participant A: **Fully agree**, given the questions and answers, it was understandable what the responses were.
- Participant B: Did not have an answer.
- Participant C: **Fully agree**, it not only answers in short, it provides a detailed explanation, for example, on the calculation, which is good.
- Participant D: Did not have an answer.

6. Do you have any improvement points, feedback, etc...??

- They did not have anything to add here.

These results point out a few interpretations. Participant A stated that demonstrating—in the chat interface and terminal logs—that the *postgresTool* was unable to answer a question, which the *neo4jTool* was, became a bit too technical. Implying that the observability feature of having the logs in the terminal could be better suited as integrated with the chat interface. Furthermore, the artifact was perceived as a versatile tool across organizational layers, which can be viewed as a good sign for adoption. For the question about efficiency, there seemed to be a consensus around using the artifact as a tool for guidance rather than automation—pointing the operator in the correct direction, not performing the work itself.

Furthermore, for the questions including the Likert scale, on (1), three answered "fully agree" and one "partially agree", implying that the artifact delivers actionable, usable answers for daily operational needs. For (2), all four agreed on "mostly agree or partially agree", but highlighted that they lacked demonstration of more questions to answer extensively—emphasizing the importance of planning a demonstration and asking the most suitable questions when time is limited. Finally, for (5), two answered "fully agree" and the other two did not have an answer, implying that the responses the artifact provides are understandable to an extent, and that it was good that it explained findings and calculations.

In summary, the stakeholder evaluation proved to be both a valuable artifact evaluation and data collection method. The total score on usefulness based on the Likert scale was $5.3/6 ((5.5 + 4.5 + 6) / 3)$, indicating that domain experts see some (few questions make it difficult to say to what extent) value in the artifact being used in daily operation. The open questions indicate that the artifact can be a useful tool both at an operational and management level.

Benchmark results

Through enabling the agent to utilize two data sources containing the same raw data but with unique structures, the correctness score for this iteration increased slightly to 77.7%, representing a modest increase of 2.3%. Simple lookup questions like, for example, "Who was on the shift yesterday?" remained near-perfect, while, for example,

one of the complex questions, "What were the root causes for the deviations?", only improved from 50% to 67%. At this point, the results were as expected—converging. This suggests that the combination of the KG and Postgres data sources was somewhat effective, but also that further tuning of the retrieval functionality or system instruction, or alternative approaches are necessary to improve the correctness.

Below is a summary of the results from applying the adapted comparative benchmarking evaluation across all four iterations:

Iteration	Configuration	Correctness
1	Baseline RAG pipeline	26.0 %
2	Agent & LangGraph	57.5 %
3	Expanded KG & Refined Prompt	75.4 %
4	Hybrid KG & SQL retrieval	77.7 %

Table 4.10: Benchmark correctness across iterations

4.4.5 Data analysis

After evaluating the final iteration, three questions scored below 50% correct, thereby needing an error analysis in the form of classification, as described in Section 3.4. (1) "Have we documented any non-conformities?", were labeled "Ambiguous question" due to the artifact answering four times correct, four times incorrect because the non-conformities were not included in the response, and two times with "I don't know the answer" or "data unavailable", giving a total score of 40%. (2) "Does the shift process log contain any info that could explain the problems with raw milk?" was labeled "Hallucination" because the agent was able to retrieve data about the shift process log, but claimed that it did not contain any relevant information nine out of ten times, although it did. (3) "Do we have similar deviations/situations in the past where we had high bacterial count in raw milk?" was labeled as "Null" and "Invalid query" because the agent generated a syntactically correct query using the *neo4jTool*, but returned null data, and the *postgresTool* generated an invalid query trying to find the column "bacterial_count" which was named "bacteria_count". Similar data analyses were conducted across all four iterations.

4.4.6 Lessons learned

Stakeholder evaluation confirmed that conducting both a controlled lab experiment and a semi-structured group interview provided complementary evidence: the lab experiment quantified a +51.7% correctness gain, while the stakeholder interview uncovered a value proposition—using the artifact for operator training. By including both quantitative questions—giving an indication of how domain experts view the artifact, and qualitative open questions—promoting discussion, and in this case, uncovering a potential use case, they provide value to the artifact that the technical comparative benchmarking evaluation would otherwise not have identified.

Additionally, the interview showed that it is important to consider the technical limits of participants involved, and try to match them. In hindsight, detailing the terminal logs—showing the retrieval process—proved too technical for management-level stakeholders and should be abstracted in future work.

Furthermore, the technical aspects of this iteration gave valuable insights that hybrid retrieval improves correctness in the responses, but to a marginal extent, and at the cost of development and code complexity. Additionally, a KG is great for retrieving relational insights, and tabular lookup in Postgres is still useful in a hybrid setup due to faster retrieval time that provides equally good answers, hence the advantages of having both and instructing the agent on when to use which.

Chapter 5

Final artifact

This chapter documents the artifact developed through the iterative design process. It first describes the main technical components that make up the system’s functionality, including the chat interface, the agent and LangGraph orchestration, the data sources, and the backend binding link. It then presents a sequence diagram that illustrates an overview of the flow between these components. Finally, the core functionality of the artifact is summarized.

Although the final state of the artifact is not production-ready, it serves as a research prototype that supports exploration of the thesis research questions and demonstrates the feasibility of combining structured and relational retrieval mechanisms in a RAG system to stakeholders.

5.1 Main components

The artifact comprises key components from each iteration. By building the artifact iteratively, I carried over some components and features to subsequent iterations, while others were discarded. In this section, I describe the technical aspects of these components, present a high-level illustration of the architecture, and a sequence diagram of the end-to-end flow in the artifact.

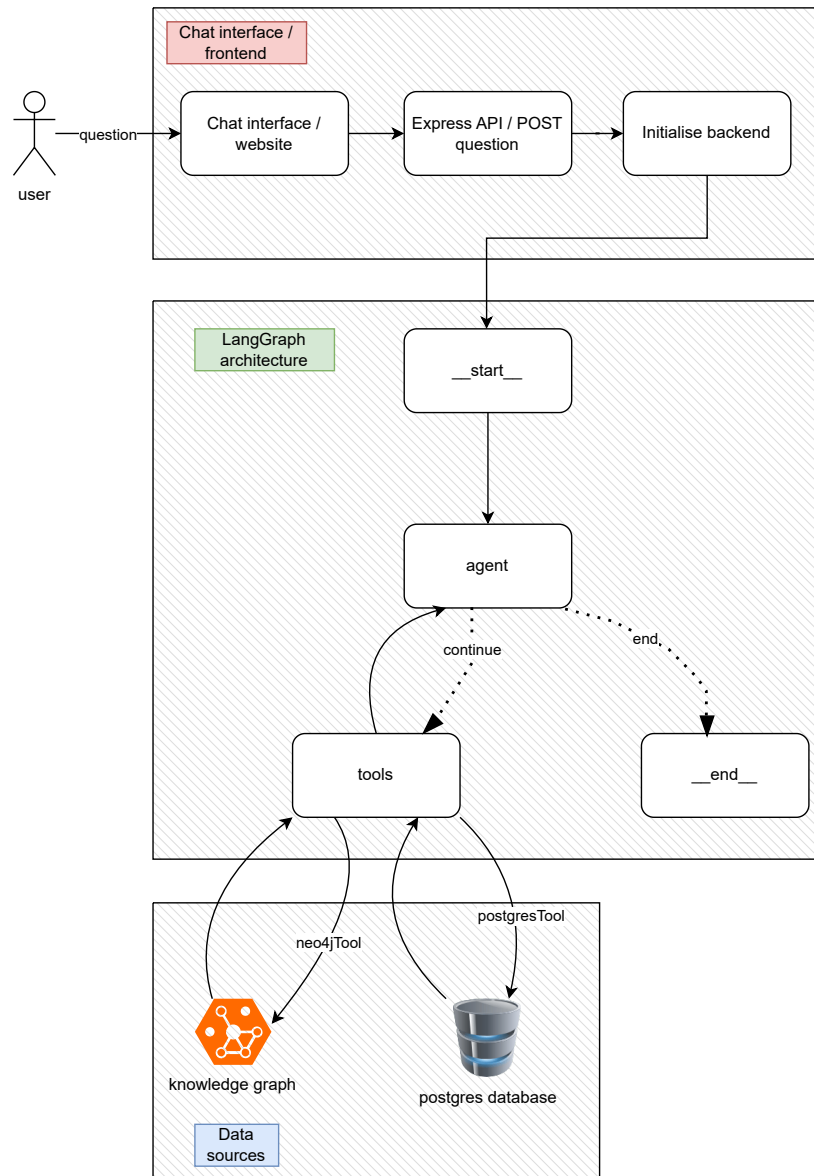


Figure 5.1: High-level architecture of the final artifact

Figure 5.1 presents a high-level, conceptual view of the components in the final artifact; these components are detailed in the following sections in this chapter.

5.1.1 Chat interface

The chat interface is the entry point of the system, and consists of a single component, *CoworkerChatInterface*, written in React and TypeScript. This component orchestrates message flow with four states, *messages*, *inputValue*, *isLoading*, and *error*.

- *messages* holds the full conversation.
- *inputValue* is the current input value sent to the agent.
- *isLoading* disables input while the agent is forming its response.
- *error* holds error messages.

When a message or question is submitted, the function *handleSubmit* calls *askQuestion*, which sends a POST request to */ask-question*. An Express API is used to forward the question to the backend, which returns the agent's answer. The frontend appends this response to *messages* and displays the message in the chat interface.

React and TypeScript are used because their libraries are well-suited for building user interfaces, featuring ready-to-use components out of the box, and compile-time type mismatches. Although it is a lightweight interface, it remained the same throughout subsequent iterations because it was not the focus of this artifact; it served its purpose in facilitating interaction between a user and LLM. The final design of the interface is illustrated in Figure 4.4.

5.1.2 Agent and LangGraph

In an attempt to find an optimal combination of technologies used in the artifact, the retrieval functions and architecture changed for each iteration. In this section, I will detail the backend of the artifact, including the final prompt structure used in the agent, and the flow and logic in the LangGraph.

Final prompt structure

The agent prompt is a crucial part of this artifact, as it instructs the agent in how to behave. By defining the prompt in a structured way, I was able to ... The following is the finalized prompt structure I used, it details how I instructed the agent. The actual prompt used can be viewed in Appendix E.

- **Introduction**
 - Explains the date context (current and default dates) and the agent's role.
- **Data Sources**
 - Lists the two tools (knowledge graph and relational database) and their high-level contents.
- **Important Plant Information**
 - Summarizes key parameters such as maximum production capacity and Neo4j timestamp conventions.
- **Available Tools**

- Names each tool, what its best use case is, and gives an example query format.
- **Tool Selection Guidelines**
 - Describes criteria for choosing one tool or the other, or both.
- **Core Rules**
 - Defines rules for data accuracy, question analysis, and data processing.
- **Complex Question Handling Strategy**
 - Defines a multi-step query approach for different question types and how to synthesize results.
- **Example Query Strategies**
 - Provides illustrative step-by-step plans for sample complex questions.
- **Tool-Specific Query Guidelines**
 - Gives formatting and usage rules for Cypher and SQL queries.
- **General Operational Guidelines**
 - Lists best practices (e.g. never guess, break down questions, verify across tools).
- **Schema Information**
 - Dynamically passes Neo4j and Postgres schemas to the prompt.

These prompt instructions and the structure will be discussed further in the next chapter.

Logic and functions in the LangGraph

For every question input into the LangGraph architecture of the final artifact, a small, directed workflow of decisions and actions is created. Following that flow, the agent is able to retrieve data dynamically. The logic and actions in the graph were implemented and finalized in the second iteration. Except for expanding the KG in the third iteration and updating the agent's instruction in the fourth iteration, there was no further development. The key components, functions, and structure of the LangGraph are detailed below.

- **State Annotation & Memory (functions and classes are imported from the LangGraph framework)**
 - *StateAnnotation* defines the shared "state" object in the graph as an array.
 - *messagesStateReducer* appends new agent or human messages to that array.
 - *MemorySaver* serves as a "checkpoint" that persists the annotated state with a thread key so that conversation history survives.
- **Tool Implementations**

neo4jTool:

- Wraps `GraphCypherQChain` in a loop for a maximum of three retries and logs intermediate steps.
- Uses Zod schema validation on the input.
- Returns a plain text result string of the database response the agent gets.

postgresTool:

- Wraps `SqlDatabaseChain` in a loop for a maximum of three retries.
- Uses Zod schema validation on the input.
- Returns a plain text result string of the database response the agent gets.

- **Agent Invocation Function**

agentNodeFn:

1. Takes in state, LLM, and prompt as parameters.
2. Extracts history from state.
3. Converts the prompt to a system message.
4. Adds the system message to the history.
5. Invokes agent with tools.
6. Returns a message object to update the state.

- **Conditional Edges Logic**

shouldContinue:

- Takes in state as parameter.
- Checks if the last message in the state is an instance of `AIMessage`, and if there are any tool calls return "tools", which routes the agent to the tool node. Otherwise, it returns "___end___" that stops the workflow.

- **Assembling Graph**

createGraph:

1. Takes in data sources, prompt, and Open AI API key as parameters.
2. Instantiate tools with respective data source.
3. Creates the model and binds the tool to it.
4. Declares the tool node as a `LangGraph ToolNode`, which runs the tools requested by the last `AIMessage`.
5. Wraps `agentNodeFn` into a named graph node.
6. Creates the workflow as a `LangGraph StateGraph` with nodes, edges, and conditional edges.
7. Compiles the workflow into executable logic and returns it for the index file to invoke.

The code implementation of this architecture and these functions can be viewed in Appendix A.

5.1.3 Data sources

The artifact relies on two data sources, a Postgres relational database and a Neo4j KG. These complement one another by providing different structures that can be used in different use cases. In this section, I will present the final state of these data sources.

Postgres database

The Postgres database was instantiated and populated in the first iterations, and remained static throughout the iterations. Its schema, Tables 4.1-4.8, shows the structure of it. The purpose of this database is to hold simulated dairy plant data used for testing the retrieval functionality of the agent.

Knowledge graph

This data source is essential for the artifact, being finalized in the third iteration, it provides the agent with graph-structured data to retrieve from by defining entities as nodes and relationships as edges. The final state of the KG contains the same raw data as the Postgres database, but is used for complex user questions, rather than straightforward tabular lookup. Figure 5.1 presents the star-shaped structure of this KG, including the main node, the entities as nodes, and relationships as edges. This figure only illustrates one of each node type; in the actual graph, there are 5014 nodes in total. Additionally, Neo4j labels the node with an attribute that suits by default, for example, the light brown node labeled "Dairy Farms Inc." is an *RawMaterialInput* node.

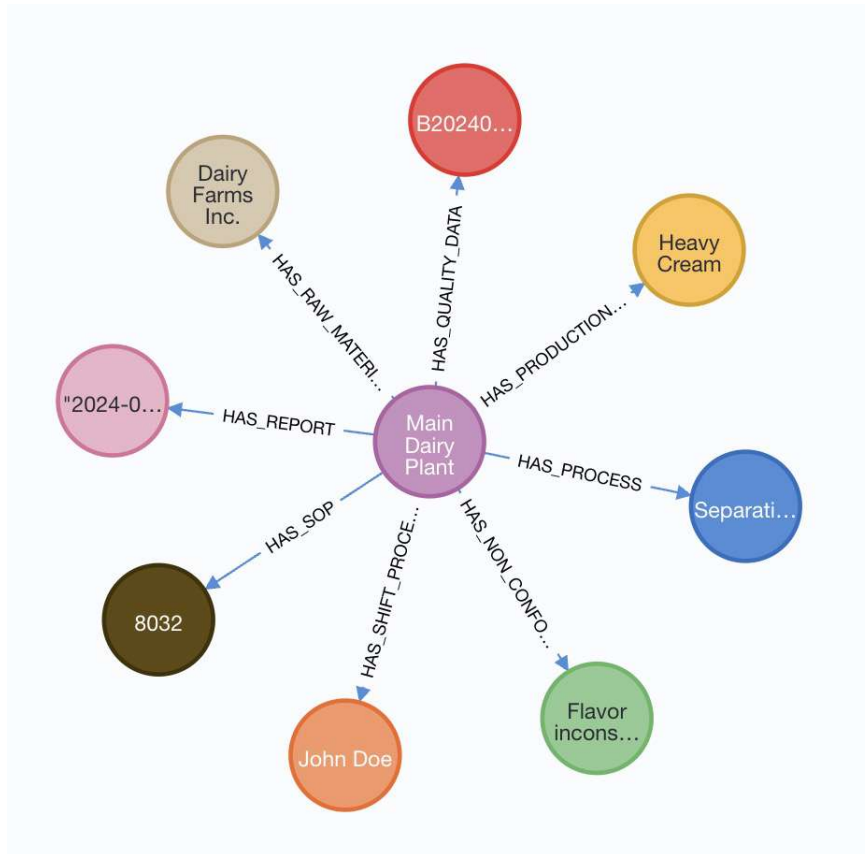


Figure 5.2: Final KG structure illustrating one of each node type

5.1.4 Binding link

To facilitate the interaction between the frontend and backend, the index file acts as the binding link. It loads environment variables, initializes the data sources and the graph, and ties them together in a function that the frontend calls to get a response.

The initialization of the data sources is performed in asynchronous functions, which return a promise that resolves with the data sources once the connection is live. These functions read the environment variables to instantiate the data source, log a success message if successful, and return the data source. These functions are called from another function, *initializeAll*, that first checks if the data sources have been initialized; if so, it moves on to constructing the agent prompt. The data sources have built-in functions for extracting their schema (*getTableInfo* on Postgres and *getSchema* on Neo4j), which is passed to the prompt for context regarding query generation. The final step of *initializeAll* is to call *createGraph* with the initialized data sources, agent prompt, and API key as arguments to create an invokable workflow.

The main function that the frontend calls, *handleUserQuestion*, takes in the user question and an optional thread ID as parameters. It starts by calling *initializeAll* to ensure that everything is initialized. It then logs the question and sanity checks that the graph is ready, otherwise it throws an error. If the graph is initialized, it proceeds to invoke it with the question and, optionally, a thread ID. Finally, once the agent workflow completes, it returns the final message extracted from the graph state to the frontend. The code implementation of this index file can be viewed in Appendix A.

5.1.5 Sequence diagram

Figure 5.3 presents the end-to-end message flow in the artifact. It establishes a visual representation of how the components and functions described above interact.

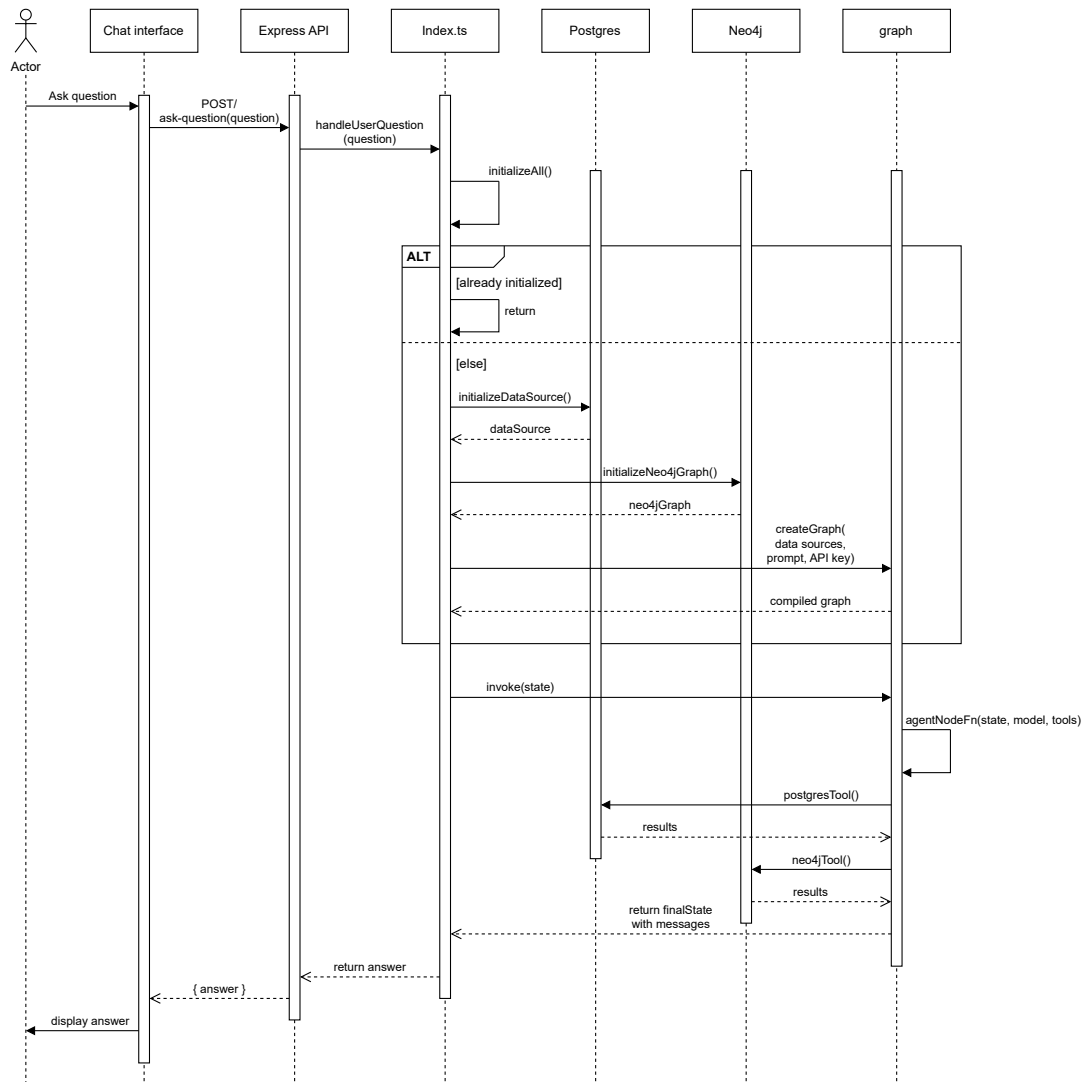


Figure 5.3: Sequence diagram of flow in artifact

5.2 Core functionality

The core functionality of the artifact is question-answering with domain specific data. It enables users to ask questions, in natural language, about information related to their data sources. The responses are, to an extent, accurate and contextually grounded, meaning they can include mistakes, and errors might occur, but are generally more accurate than when using a single unstructured retrieval source.

When a question is received, a workflow, managed by an agent in a graph architecture, is initiated. This workflow uses two complementary data sources (KG and Postgres database) to dynamically generate and execute queries against. Additionally, it logs the intermediate steps the agents take to the terminal window from where the artifact runs. This can be useful for users because it can help verify that the queries the agent uses are correct. For experienced dairy plant operators, these logs can also help identify errors.

Through its functionality, the artifact demonstrates how combining the *neo4jTool* to capture complex relationships and the *postgresTool* to perform an efficient and straightforward lookup can support both complex and simple user questions. It serves both to explore the research questions of this thesis and to demonstrate the feasibility and practical potential of this approach to stakeholders. This will be discussed further in the next chapter.

Chapter 6

Discussion

I will, in this chapter, present how this thesis and the artifact created address its research questions. In addition to presenting and discussing the main contributions this artifact provides, research that has conducted similar and related work, lessons learned through this thesis for each iteration—both practical, theoretical, and as a whole, limitations identified in this research, research quality with potential threats and how to address them, and finally recommendations for future work.

6.1 Research questions & artifact impact

This research aimed to enhance querying capabilities in chatbot applications by creating an artifact. The contributions of this artifact consist of theoretical and practical knowledge, described in Section 6.4, and the system described in Chapter 5. It was guided by one primary research question and an additional sub-question:

RQ1: How can knowledge graphs and retrieval-augmented generation frameworks be integrated with large language models to enhance querying capabilities in chatbot applications?

RQ1.1: To what extent does having access to a knowledge graph of relationships improve the queries and final answers generated by the large language model?

The following two subsections will present how the artifact addresses these research questions through the integration of KGs, RAG frameworks, and LLMs, using synthetic industrial data. Additionally, it will detail how the artifact created impacts them.

6.1.1 RQ1: Architecture & integration impact

To assess and answer RQ1, I created a chatbot application that integrates KGs, RAG frameworks, and LLMs, as described in Chapter 5. I followed the DSR process activities and iteratively built it through refinements in each iteration in alignment with the RQ. Each iteration in the cycle introduced key components or solutions whose impact improved correctness, as analysed in the "Lessons learned" subsection of its iteration. The combination of these helped form it into a hybrid retrieval chatbot application.

I have explored how these technologies can be combined to create a synergy of each other's qualities. To isolate and investigate each of their impact: Iteration one (baseline) consisted of a traditional RAG application using a minimal KG for context retrieval. Iteration two introduced the agent, retrieval chains as tools, and the graph architecture, creating a dynamic retrieval flow. Iteration 3 explored enrichment of the KG data source and introduced an explicit prompting structure, evolving the agent's retrieval possibilities. Finally, iteration four combined these and formed a hybrid retrieval chatbot.

Together, these architectural component changes directly address RQ1, through, dynamic orchestration and retry logic to ensure robustness and reliability in iteration two, enrichment of KG and explicit prompting—ensuring valid Cypher calls in iteration three—and combining the solutions in a hybrid retrieval approach to answer a broader range of questions correctly and efficiently. These architectural choices presented throughout this thesis highlight how KGs and RAG can be integrated to improve correctness in retrieval, and are supported by both the adapted comparative benchmarking approach to measure quantitative scores and the stakeholder evaluation session providing qualitative results. However, it is important to note that these findings are limited by synthetic sample data and custom-built benchmarks, which will be further addressed in Section 6.5.

In the stakeholder evaluation, domain experts rated the final artifact's usefulness at 5.3/6. Although the stakeholders, to an extent, lacked a direct comparison to a standard

RAG system, their positive feedback on usability and the artifact's response relevance confirm that these integration choices yield practical gains in real-world use cases.

Through reflecting on component integration choices for correctness improvements, highlighting what is important to focus on when integrating these technologies in documented "lessons learned" sections, supported by the quantitative improvement gains presented in Table 4.10, and the affirmation from the evaluation with stakeholders, imply that KG-augmented RAG pipelines enhance chatbot query correctness, thereby answering RQ1.

6.1.2 RQ1.1: Quantitative correctness improvements

To investigate to what extent having access to a KG improved the queries and the agent's response, I applied my adapted comparative benchmarking approach across all iterations. In short, this compares the iteration's answer to the benchmark question's reference answer, giving a quantifiable measurement for improved correctness. Table 4.10 in Chapter 4 summarizes the average correctness scores for each iteration, and Table 4.9 presents the ablation results.

However, Table 4.10 does not represent the isolated effects of the KG; it represents the whole iteration's performance, including crucial changes to an additional factor—the prompt—which was done to configure it with the enriched KG. Therefore, the improvement from iteration two to three ($57.5 \rightarrow 75.4\%$)—introducing the new KG topology and refining the prompt—represents the effect combining these had on improving the correctness. I ran an ablation study to isolate the effects each factor had on the results, as described in Section 4.3.5, to assess the RQs.

Ablation one points out that enriching the KG without instructing the agent on how to exploit its topology or data, only improves correctness slightly compared to the baseline ($25 \rightarrow 36\%$). This implies that access to the graph structure is not enough and that the agent needs structured prompting guidelines to leverage it fully.

Ablation two emphasizes this by showing how important it is to be specific when instructing the agent. By instructing it to set "CURRENT_DATE=2024-10-19", it generated queries that used the actual current date, leading to mismatches in database records, and a decrease ($25 \rightarrow 19\%$) from baseline to ablation two in correctness.

Ablation three, however, demonstrates the effects of a well-structured prompt. By giving this ablation arm a fair chance through considering and fixing the date problem, the correctness score improved ($25 \rightarrow 63.5\%$). This implies that it was the prompt structuring in iteration three that was the main driver for the improved score, rather than the enrichment of the KG, although this had a marginal effect as well.

Aligning these ablation studies with RQ1.1—to what extent having access to a KG improves queries generated by the LLM—reveals three clear insights. A KG as the only data source without a well-structured prompt to the agent on how to use it, yields a marginal 11% (ablation 1 - baseline = $36 - 25$) improvement. Furthermore, that prompt refinements and structuring give a high impact, 38.5% (ablation 3 - baseline = $63.5 - 25$) over the baseline. Lastly, the synergy from combining a KG with a well-structured prompt is minimal, only 0.9% (iteration 3 - (ablation 3 + KG improvements) = $75.4 - 74.5$). I would conclude that, although isolating the KG showed minimal gains, it can

be complementary to traditional RAG systems when instructed to use it for complex relational questions.

6.2 Contributions

The main contributions my artifact provides are listed as follows:

- Introduces an agent-based hybrid retrieval architecture (KG + RAG + LLM) that tightly integrates retrieval with a star-shaped KG to support a domain-specific chatbot.
- Provides quantitative insight—through evaluation with a custom benchmark and ablation studies—on how KG-augmented retrieval improves agent response correctness.
- Delivers a chat interface and logs of the retrieval process that can be used to support onboarding and training of new employees in industrial process work.

6.3 Related work

In my preliminary survey on technologies used in this research and while writing this thesis, I found similar studies that focused on the integration of KGs, RAG, and LLMs. These include, but are not limited to, the articles described below.

Sarmah et al. [19] propose HybridRAG in finance, performing RAG over embedded earnings-call transcripts with vector search and template Cypher queries to an automatically constructed KG, and concatenating the response context to GPT-3.5 to answer in a fixed retrieval flow, in contrast to the iterative retrieval flow with the graph and agent choosing tools, in this thesis. Our research differs in the way that the agent present in this thesis chooses between KG and SQL at each step in the graph architecture, yielding a 51.7% jump on industrial sample data, indicating that dynamic tool selection can improve retrieval responses more than the ranking heuristic Sarmah et al. tuned. However, it is important to clarify that we do not measure the exact same metrics; this thesis measured correctness, whereas Sarmah et al. measured faithfulness, answer relevance, context precision, and context recall, therefore, a direct numerical comparison is, to an extent, not meaningful.

Wan et al. [23] introduce empowering LLMs by hybrid RAG for domain-centric QA. Their approach is similar to Sarmah et al.'s, as they too retrieve context in a static flow and integrate it with the prompt to generate comprehensive context-aware responses, whereas my artifact uses a graph architecture to enable the agent to select which tool to use or retry at run-time. Furthermore, by using different retrieval approaches and LLMs, a direct comparison between this and Wan et al.'s study is not meaningful.

Zhu et al. [28] present KG²-RAG, where they run a similarity search and finds the nearest neighbor nodes (chunks of data) in the graph and then expand the search field by traversing one hop outward to include neighboring nodes that may not initially hit a top-k from the similarity search, but are factually linked and potentially contains useful information. They then reorganize the result into coherent nodes before prompting the LLM. Their retrieval flow is fixed once this expansion is assembled. Whereas my KG

remains statically defined from domain entities throughout the retrieval process.

These studies suggest approaches to integrating KGs, RAGs, and LLMs and show different techniques for achieving this. Additionally, they highlight the rapidly evolving field in which these technologies are used, applied, or combined, as all three articles were published within the last year. Nonetheless, the findings documented in this thesis should be relevant to researchers exploring KG-hybrid-retrieval systems.

6.4 Lessons learned

In this section, I present the main knowledge gained through this study. First, through referencing the "lessons learned" sections for each iteration, with what I would have done differently. Then, from a research and methodological perspective, and finally, as practical design trade-offs through the creation of the artifact.

6.4.1 Iterations

Iteration 1

Starting with the first iteration, in hindsight, I should have spotted the "CURRENT_DATE" bug earlier in the development process and fixed it. This matters because using a hard-coded date in the prompt led to syntactically valid queries returning empty query responses. Although I was not able to identify this, I kept the system instruction static—with all iterations containing "CURRENT_DATE" throughout the thesis, therefore it should not have affected the results. Additionally, when I uncovered the bug, I conducted three ablation studies to isolate the effect the KG and prompt changes had on the results.

Iteration 2

Furthermore, a get-around solution to the "CURRENT_DATE" bug was to add—"If the tool calls returns 0 or null, try reformulating the query, with date format aligning with the data source's formatting"—to the agent prompt. In hindsight, I should have added conditional checks for null data and errors retrieved in the retry loop, and appended it to the state graph or concatenated it with the prompt to give the agent context about why the query failed. This would have improved the robustness of the system and might have increased its ability to generate correct queries, leading to improved correctness provided to the users.

Iteration 3

For the third iteration, I learned that the choice of graph topology is a research variable and not just an implementation detail. Looking back on it, I should have spent more time exploring other options, like, for example, a "small-world" structure, where nodes from each node type (process data, production data, etc.) forms clusters hubs, and linked them similarly to what I did in the baseline—defined the relationships myself and used the KG for relational context providing to the Postgres chain. This approach might have introduced traversal depth—capturing important relationships—without the potential combinatorial overflow of, for example, a full mesh.

Iteration 4

Finally, from the last iteration, I learned the importance and indispensable value of conducting evaluations with stakeholders. Reflections from the evaluation session are described in Section 4.4.6, and if I were to conduct a similar evaluation session with the same stakeholders, I would focus on implementing the terminal logs into the chat interface, improving understandability through observability, by designing them side-by-side. In addition to expanding the demonstration to include more questions to ask the artifact and get a complete answer of their thoughts.

6.4.2 Research aspect

For research working with KG-augmented RAG pipelines, my findings reflect three theoretical lessons. Firstly, pre-planned ablation studies matter—design ablation studies before developing simultaneous changes—because then the need for potential additional experiments post-development is removed, and cleaner evidence can be presented.

Secondly, generating synthetic data has limits. Through this thesis, I highlight that synthetic data comes with trade-offs, such as the edge case encountered while populating the KG in the third iteration—trading rapid prototyping in the first iteration for development costs later in the study.

Lastly, moving from a minimal context KG to the star-shaped topology gave an 11% improved correctness score and points to a clear next step in the field of KG-augmented RAG—systematically explore and compare alternative topologies like a "small world" or "full mesh". All code for this artifact is provided through the GitHub repository included in Appendix A, which can be used as a baseline for further studies.

6.4.3 Practical

For researchers and developers building hybrid KG RAG chatbots in domain-specific settings, I want to emphasize two actionable trade-offs from my artifact. Firstly, by focusing on the technical side—backend with data sources and retrieval logic—in this thesis, rather than polishing the chat interface, I did not get the most out of what the stakeholder evaluation could have been. Although the stakeholders communicated interest and satisfaction towards the artifact, a modest frontend polish—logs in the same window as the chat, a progress-spinner, or clickable citations—for example, would likely have produced deeper feedback. Therefore, to truly get the most out of a stakeholder evaluation, level on the technical details before the session, and invest time in visual demonstrations, like enhancing the user interface, for both technical and non-technical participants to understand.

Secondly, a recurring theme in this thesis is the system prompt. The final lesson is that creating a generic system prompt is difficult, in terms of not changing too much or too little while still attempting to get improved results. My experience with this was to try to find a general structure with the agent's role, goal, and behavior to keep constant and change the tool instructions or retry logic when these were implemented or changed. However, the need for the ablation studies emphasizes, to an extent, how difficult it is in practice. Therefore, if building a KG-augmented RAG pipeline, focus on prompt engineering—creating a solid and robust structure, which requires minimal changes throughout—before creating the data source.

6.5 Limitations

The limitations in this research that are important to acknowledge are as follows:

Stakeholder evaluation

This evaluation session was limited to 30 minutes, which left little room for the participants to think about potential edge cases to test independently. This resulted in several ratings and opinions being based on exposure from the three questions I asked in the demonstration. Therefore, the collected usefulness score should be viewed as a sentiment check rather than evidence of the readiness for KG + RAG + LLM technology in the field.

Synthetic sample data

Throughout each iteration in this study, synthetic data was used to build, test, and evaluate. Although it simplified early prototyping, it also made it difficult to choose the KG topology—resulting in the compromise of the star-shape. Whereas with real production data, another topology might have been chosen that could affect retrieval correctness, in terms of including foreign keys to link with and real-world noise in the data. In short, synthetic data was great for early development, but came with hidden costs, this should be considered when interpreting external validity.

Custom-built benchmark set

The custom benchmark set—consisting of the 20 dairy-specific questions that operator typically asks—was suitable for controlled comparison, but might have affected the results due to limited breadth and potential under-representation of ambiguity. Consequently, the resulting correctness score should be interpreted as improvements within the dairy-process domain rather than proof of correctness gains in general for KG + RAG + LLM systems.

LLM retrieval compatibility

Retrieving data from the KGs used in this study relies on *GraphCypherQACchain*, which currently is compatible with OpenAI GPT models only. Implying that both the graph schema and Cypher queries are transmitted to an external API, leading to concerns regarding privacy for replication with real production data. Consequently, replicating this study using privacy-sensitive data would require a local model and a customized function for KG retrieval to keep traffic on a local network.

6.6 Research quality

To establish trust in the knowledge the artifact claims to have produced and ensure valid work, it is important in Design Science and other research fields to communicate the research’s quality [12]. In this section, I discuss three validity concerns—construct, external, and reliability—in light of the choices I made in this study and their potential remaining threats

Construct validity

Construct validity refers to the extent to which the measurements capture the constructs we claim. Wieringa [25] defines it as, *"We here define construct validity instead as the degree to which the application of constructs to phenomena is warranted with respect to the research goals and questions."* Additionally, Wieringa states that if one is to measure constructs, they must be put into operation or use [25].

In this research, I measured "query correctness" by automating the collection of comparative benchmarking scores—comparing artifact answers to correct reference answers. By aligning the benchmark set with the artifact's goal of returning correct answers to user questions, I feel like I got a good measurement on capturing the constructs I claim. However, the benchmark set can have been too narrow, which can serve as a potential threat due to omitting edge cases and ambiguous questions. One suggestion to mitigate this construct validity risk is to enrich the benchmark set with questions from real production logs including writing style and multi-step questions a potential operator might have asked.

Furthermore, I measured "usefulness" with the 6-point Likert scale and open questions through the evaluation session with stakeholders. In an attempt to decrease misunderstandings and receive objective feedback, I first presented what the research involved, what technologies were used and how, and tried to inform them to the extent where it would not affect the evaluation—not presenting results before they had evaluated. However, two potential threats can be derived from this: (1) misunderstanding of use case for the artifact and stakeholders' roles—meant for operators, but where evaluated by management-level personnel, and (2) the Likert scale and questions captured only immediate reactions after a potentially too minimal demonstration. For future research, I would suggest extending demonstration and evaluation to mitigate the risk of construct validity through deeper answers and reflections from stakeholders.

External validity

By external validity in Design science, what is meant is to what extent the artifact's knowledge claims are supported through evaluation in more than one setting [12], in other words, how can the findings be generalized beyond the study environment.

External validity in this thesis is strengthened by designing the system architecture—agent + retrieval functions + graph—to be compatible with other graph stores or relational databases. Additionally, detailing the ablation studies—isolating the effects—and publishing code, prompts, benchmark testing scripts, and programs for populating data sources enables replication beyond the dairy domain.

The controlled lab experiment I conducted after each iteration—evaluating each solution's answer with the custom-built benchmark set and using synthetic data—poses some threats to this thesis' external validity, in terms of generalizing the result. Because using synthetic data and a small sample of benchmark questions in a controlled experiment potentially removes noise in data, time pressure, or other factors that an operator faces in daily work. Consequently, future research should evaluate the artifact, both in a quasi-field study and with real production data. For example, having an operator use the chatbot during a live shift or in "training", collect additional questions from this to use in the benchmark set, and re-evaluate. This approach would show whether the

artifact maintains its correctness within a real setting and strengthen external validity.

Furthermore, evaluation with stakeholders was limited to a single industrial partner, where four participants joined the short-lasting evaluation session. The effect this has on external validity is that a narrow participant pool reflects a limited set of opinions, which cannot be interpreted as evidence for other companies or settings, and thus threatens external validity due to other industries potentially perceiving the artifact differently. Future research should include evaluation with multiple potential partners to validate the user acceptance on a broader level and strengthen validity through collecting multiple opinions, i.e., generalizing the results.

Reliability

Reliability in research quality refers to the extent of trust in the researchers to provide enough information in the study to make it replicable, so that the study would provide the same results if it were repeated.

In this study, to ensure reliability, I used the same program to collect quantitative evaluation results—through the adapted comparative benchmarking approach—for each iteration. Since this is a completely automated process, it's easily replicable, and the code is provided in the git repository, Appendix A. In addition to code, prompts, and seed values, which are also preserved in the project repository, ensuring reliability through transparency. Furthermore, by detailing and documenting all architectural changes step-by-step through this thesis, including development choices and why, supporting repeatability—strengthens reliability.

One threat to reliability in this thesis is using OpenAI's models through their API. If they decide to update their model, the behavior of the model used in this research can be shifted. To consider this, future research should mitigate API drift by specifying the exact model identifier, for example, "gpt-3.5-turbo-0125", which will remain fixed even if they update the GPT-3.5 model series.

6.6.1 DSR guidelines

In this section, I explain how the seven DSR guidelines proposed by Hevner et al. [7] are reflected in this research. First, by quoting their definition, and then by providing examples to align with my DSR process and situate the artifact within accepted DSR practice.

Guideline 1: Design as an Artifact

"Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation" [7].

Through this thesis, I contribute an artifact in the form of an instantiation—chatbot application—consisting of a frontend (chat interface website) and a backend (KG, RAG pipeline, agent, and graph architecture). The artifact provides a hybrid retrieval system that lets users ask questions in natural language to retrieve information from two data sources (KG and Postgres), in this case, in an industrial setting. Through demonstration

and evaluation—stakeholder group interview and adapted comparative benchmarking—it shows the effect of integrating a KG within RAG pipelines in terms of improved correctness and usefulness score.

Guideline 2: Problem Relevance

"The objective of design-science research is to develop technology-based solutions to important and relevant business problems" [7].

The creation of this artifact served as experimentation that sought to add knowledge on how KGs can improve correctness in LLM-retrieval systems. To identify important and relevant business problems, a stakeholder with industrial process expertise presented potential use cases, shared knowledge about the dairy industry, and supported validating business needs. Then, through a preliminary survey on potential technologies, I chose which angle and technologies to use for addressing these problems. By identifying problems and creating use cases with stakeholders from the dairy industry, this thesis presents a potential solution to retrieve data from synthetic dairy datasets.

Guideline 3: Design Evaluation

"The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods" [7].

In this study, I conducted evaluations through two main types—controlled lab experiments with a custom benchmark set, and a semi-structured group interview with stakeholders to demonstrate usefulness. I used the lab experiments to assess the retrieval correctness of the artifact’s answers to the benchmark questions. The rationale behind choosing correctness rather than, for example, latency, was, first of all, due to the importance of factually correct answers in the industry—knowing if milk contains a too high bacterial count is more important than retrieval speed. Secondly, to align with the scope of the thesis and not over-measure. For the stakeholder evaluation, I measured usefulness with Likert scale questions and usability aspects with open questions.

These two evaluation methods provide both quantitative and qualitative evidence of the results, complementing each other by demonstrating that improved correctness influences usefulness. Furthermore, the lab experiments providing correctness scores directly address RQ1.1 by quantifying how KG-augmented retrieval improves correctness, whereas the stakeholder group interview address RQ1 by confirming perceived usefulness of the system-architecture in practice.

Guideline 4: Research Contributions

"Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies" [7].

The artifact contributes a reproducible hybrid retrieval architecture using a KG as the main data source to retrieve information for—to an extent—ambiguous dairy industry questions, and Postgres for straightforward database lookup. Additionally, it provides insights on how KG-augmented retrieval improves agent response correctness. Furthermore, it contributes a chat interface and real-time terminal logs of tool calls and queries used to retrieve data, validated by stakeholders as useful for operator training.

Guideline 5: Research Rigor

"Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact" [7].

To enforce research rigor, I conducted a preliminary survey on relevant technologies, such as KGs, RAG, LLMs, and their combination. Whereas in terms of evaluation, I used Venable et al.'s [22] framework for evaluation strategies and methods to identify the suitable methods, and applied an error analysis on the results from the controlled lab experiments and qualitative content analysis on the results from the stakeholder interview. These two analyses were useful in concluding my findings as they provided systematic approaches to identify the findings. Furthermore, to strengthen evaluation results through rigorous methods, I applied a systematic experimental design in the form of ablation studies, isolating the effects of KG vs prompt engineering.

Guideline 6: Design as a Search Process

"The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment" [7].

Moving from a minimal RAG prototype in the first iteration to the hybrid retrieval system was done through evaluating with the comparative benchmark approach and conducting the analysis after each iteration. The insights from these analyses defined the objectives for the subsequent iteration's solution. The detailed "lessons learned" sections in each Iteration section explicitly document this search method.

Guideline 7: Communication of Research

"Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences" [7].

I present this thesis systematically and clearly, including theoretical knowledge with examples of how I have applied it in practice.

6.7 Future work recommendations

In this thesis, I have presented a starting point in exploring the integration of KGs, RAG, and LLMs, however, some limitations still exist. I will, in this Section, align the limitations from Section 6.5 with recommendations for future research in this area.

Evaluation

Future work should expand evaluations to real settings to strengthen the usefulness score of this artifact's stakeholder evaluation. By evaluating the artifact in daily dairy production routines and observing how the usefulness score evolves as novelty declines, new edge cases or use cases may be identified. Additionally, to confirm the perceived usefulness score and test the generalization—to what extent other industry partners align with TINE—presented in this study's results, future research should replicate the evaluation method with at least one other dairy-industrial partner.

Data enrichment

As described in the Limitations section above, this research depended on synthetic raw data and a small custom-built benchmark set to develop and evaluate with. To address this limitation, future work should validate this artifact with real production data—including noise and larger corpora—and expand the benchmark set to reflect daily questions operators ask on shifts.

Knowledge graph topology

The KG topology used in the final artifact is star-shaped, and as addressed in Section 6.4 under Iteration 3, a prominent alternative to this structure could be a "small-world". In an attempt to improve KG-retrieval correctness—capturing deeper semantic relationships in the graph—future research should consider comparing star-shape and small-world structures to quantify retrieval correctness.

Chat interface

The final artifact contributes a rich backend architecture, but lacks the visual frontend components that could complement it. However, the stakeholder evaluation uncovered a potential use case where this artifact could be applied—to train new operator personnel in the dairy industry, by using it to safely query data and compare their analysis to an expert's.

A suggestion for future work is therefore to design a richer interface, reflecting what the backend is capable of—logging real-time queries, error messages, and tool calls. Where these interface changes—together with this artifact's backend—would become the instantiation [7] in a new DSR cycle, or expand this study with an additional iteration. A suitable evaluation strategy for this would be naturalistic ex-post to evaluate the instantiation in total rather than a partial prototype, and with real users, rather than artificial [22]. Furthermore, a case study would be a suitable evaluation method because it assesses the artifact in a real-world setting, in contrast to, for example, a lab experiment, which maximizes control and excludes human factors [22]. However, there are different angles to conducting this research, for example, how to identify the problem, design objectives, develop, how many iterations are required, and selecting evaluation methods other than the example described above.

Chapter 7

Conclusion

This thesis sought to explore how knowledge graphs can be integrated with retrieval-augmented generation frameworks to improve query correctness in an LLM chatbot (RQ1) and to quantify the extent to which the knowledge graph adds value to a retrieval pipeline (RQ1.1). These research questions were motivated by a concrete need in dairy processing—enabling operators to query industrial data in real-time with natural language rather than query languages like SQL or Cypher. By following guidelines from design science research methodology, I built four prototypes that incrementally introduced a chat interface with hybrid retrieval through an agent, a star-shaped knowledge graph, a Postgres data source, and a LangGraph-based graph architecture. Each iteration was evaluated through controlled lab experiments by comparing the artifact’s answer to a reference answer, while the final artifact was additionally evaluated through a semi-structured group interview with stakeholders from TINE AS.

The lab experiments showed that measured benchmark correctness rose from 26% in the baseline to 77.7% in the final solution, and system-level ablation showed that prompt engineering contributed 38%, whereas the KG contributed a further 11%. The stakeholders rated usefulness at 5.3/6 and emphasized the artifact’s value in operator training.

This study contributes (1) an agent-based hybrid retrieval architecture with a star-shaped KG to support retrieval in domain-specific chatbots, (2) quantitative insights on how and where KG-augmented retrieval improves agent response correctness, and (3) a chat interface and logs of the retrieval process that can be used for onboarding and training of new operators in industrial process work. However, findings are limited to synthetic data and a single industrial partner evaluation. Therefore, future work should validate with live production data, compare alternative KG topologies, and explore an enriched chat interface through a field study, to strengthen the generalization of the correctness gains and further close the gap in KG-augmented RAG research and industry.

Bibliography

- [1] Patrice Béchard and Orlando Marquez Ayala. ‘Reducing hallucination in structured outputs via Retrieval-Augmented Generation’. In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*. 2024, pp. 228–238. DOI: 10.18653/v1/2024.naacl-industry.19. arXiv: 2404.08189[cs]. URL: <http://arxiv.org/abs/2404.08189> (visited on 16/04/2025).
- [2] Jan vom Brocke, Alan Hevner and Alexander Maedche. ‘Introduction to Design Science Research’. In: *Design Science Research. Cases*. ISSN: 2196-8713. Springer, Cham, 2020, pp. 1–13. ISBN: 978-3-030-46781-4. DOI: 10.1007/978-3-030-46781-4_1. URL: https://link-springer-com.ezproxy.uio.no/chapter/10.1007/978-3-030-46781-4_1 (visited on 01/04/2025).
- [3] *Build a Retrieval Augmented Generation (RAG) App: Part 1 | LangChain*. URL: <https://python.langchain.com/docs/tutorials/rag/> (visited on 23/04/2025).
- [4] Harrison Chase. *The Point of LangChain — with Harrison Chase of LangChain*. 14th Mar. 2025. URL: <https://www.latent.space/p/langchain> (visited on 27/03/2025).
- [5] Tolga Ergen, Behnam Neyshabur and Harsh Mehta. *Convexifying Transformers: Improving optimization and understanding of transformer networks*. 20th Nov. 2022. DOI: 10.48550/arXiv.2211.11052. arXiv: 2211.11052[cs]. URL: <http://arxiv.org/abs/2211.11052> (visited on 16/04/2025).
- [6] Yufei Guo et al. *Bias in Large Language Models: Origin, Evaluation, and Mitigation*. 16th Nov. 2024. DOI: 10.48550/arXiv.2411.10915. arXiv: 2411.10915[cs]. URL: <http://arxiv.org/abs/2411.10915> (visited on 15/04/2025).
- [7] Alan Hevner, Salvatore T. March and Jinsoo Park. (PDF) *Design Science in Information Systems Research*. ResearchGate. URL: https://www.researchgate.net/publication/201168946_Design_Science_in_Information_Systems_Research (visited on 01/04/2025).
- [8] Lei Huang et al. ‘A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions’. In: *ACM Transactions on Information Systems* 43.2 (31st Mar. 2025), pp. 1–55. ISSN: 1046-8188, 1558-2868. DOI: 10.1145/3703155. arXiv: 2311.05232[cs]. URL: <http://arxiv.org/abs/2311.05232> (visited on 22/04/2025).
- [9] Samuel Kounev, Klaus-Dieter Lange and Jóakim von Kistowski. ‘Benchmarking Basics’. In: *Systems Benchmarking: For Scientists and Engineers*. Ed. by Samuel Kounev, Klaus-Dieter Lange and Jóakim von Kistowski. Cham: Springer Nature Switzerland, 2025, pp. 3–22. ISBN: 978-3-031-85634-1. DOI: 10.1007/978-3-031-85634-1_1. URL: https://doi.org/10.1007/978-3-031-85634-1_1 (visited on 14/04/2025).

- [10] *LangGraph*. LangChain Blog. 17th Jan. 2024. URL: <https://blog.langchain.dev/langgraph/> (visited on 15/04/2025).
- [11] *LangGraph Glossary*. URL: https://langchain-ai.github.io/langgraph/concepts/low_level/ (visited on 15/04/2025).
- [12] K. Larsen et al. *Validity in Design Science*. 12th Mar. 2025. DOI: 10.48550/arXiv.2503.09466. arXiv: 2503.09466[cs]. URL: <http://arxiv.org/abs/2503.09466> (visited on 06/05/2025).
- [13] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 12th Apr. 2021. DOI: 10.48550/arXiv.2005.11401. arXiv: 2005.11401[cs]. URL: <http://arxiv.org/abs/2005.11401> (visited on 27/03/2025).
- [14] Philipp Mayring. ‘Qualitative Content Analysis: Theoretical Background and Procedures’. In: *Approaches to Qualitative Research in Mathematics Education: Examples of Methodology and Methods*. Ed. by Angelika Bikner-Ahsbals, Christine Knipping and Norma Presmeg. Dordrecht: Springer Netherlands, 2015, pp. 365–380. ISBN: 978-94-017-9181-6. DOI: 10.1007/978-94-017-9181-6_13. URL: https://doi.org/10.1007/978-94-017-9181-6_13 (visited on 07/05/2025).
- [15] Richard Meyes et al. *Ablation Studies in Artificial Neural Networks*. 18th Feb. 2019. DOI: 10.48550/arXiv.1901.08644. arXiv: 1901.08644[cs]. URL: <http://arxiv.org/abs/1901.08644> (visited on 07/05/2025).
- [16] Shirui Pan et al. ‘Unifying Large Language Models and Knowledge Graphs: A Roadmap’. In: *IEEE Transactions on Knowledge and Data Engineering* 36.7 (July 2024), pp. 3580–3599. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: 10.1109/TKDE.2024.3352100. arXiv: 2306.08302[cs]. URL: <http://arxiv.org/abs/2306.08302> (visited on 27/03/2025).
- [17] Ken Peffers et al. ‘(PDF) A design science research methodology for information systems research’. In: *ResearchGate* (22nd Oct. 2024). URL: https://www.researchgate.net/publication/284503626_A_design_science_research_methodology_for_information_systems_research (visited on 01/04/2025).
- [18] Yao Qu and Jue Wang. ‘Performance and biases of Large Language Models in public opinion simulation’. In: *Humanities and Social Sciences Communications* 11.1 (28th Aug. 2024). Publisher: Palgrave, pp. 1–13. ISSN: 2662-9992. DOI: 10.1057/s41599-024-03609-x. URL: <https://www.nature.com/articles/s41599-024-03609-x> (visited on 15/04/2025).
- [19] Bhaskarjit Sarmah et al. *HybridRAG: Integrating Knowledge Graphs and Vector Retrieval Augmented Generation for Efficient Information Extraction*. 9th Aug. 2024. DOI: 10.48550/arXiv.2408.04948. arXiv: 2408.04948[cs]. URL: <http://arxiv.org/abs/2408.04948> (visited on 04/05/2025).
- [20] Jiashuo Sun et al. *Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph*. 24th Mar. 2024. DOI: 10.48550/arXiv.2307.07697. arXiv: 2307.07697[cs]. URL: <http://arxiv.org/abs/2307.07697> (visited on 27/03/2025).
- [21] Ashish Vaswani et al. *Attention Is All You Need*. 2nd Aug. 2023. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762[cs]. URL: <http://arxiv.org/abs/1706.03762> (visited on 27/03/2025).

- [22] John Venable, Jan Pries-Heje and Richard Baskerville. ‘A Comprehensive Framework for Evaluation in Design Science Research’. In: *Design Science Research in Information Systems. Advances in Theory and Practice*. International Conference on Design Science Research in Information Systems. ISSN: 1611-3349. Springer, Berlin, Heidelberg, 2012, pp. 423–438. ISBN: 978-3-642-29863-9. DOI: 10.1007/978-3-642-29863-9_31. URL: https://link-springer-com.ezproxy.uio.no/chapter/10.1007/978-3-642-29863-9_31 (visited on 22/04/2025).
- [23] Yuwei Wan et al. ‘Empowering LLMs by hybrid retrieval-augmented generation for domain-centric Q&A in smart manufacturing’. In: *Advanced Engineering Informatics* 65 (1st May 2025), p. 103212. ISSN: 1474-0346. DOI: 10.1016/j.aei.2025.103212. URL: <https://www.sciencedirect.com/science/article/pii/S1474034625001053> (visited on 04/05/2025).
- [24] *What is retrieval-augmented generation (RAG)?* IBM Research. 9th Feb. 2021. URL: <https://research.ibm.com/blog/retrieval-augmented-generation-RAG> (visited on 27/03/2025).
- [25] Roel J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Berlin, Heidelberg: Springer, 2014. ISBN: 978-3-662-43838-1 978-3-662-43839-8. DOI: 10.1007/978-3-662-43839-8. URL: <https://link.springer.com/10.1007/978-3-662-43839-8> (visited on 06/05/2025).
- [26] Xue Wu and Kostas Tsioutsoulis. *Thinking with Knowledge Graphs: Enhancing LLM Reasoning Through Structured Data*. 14th Dec. 2024. DOI: 10.48550/arXiv.2412.10654. arXiv: 2412.10654[cs]. URL: <http://arxiv.org/abs/2412.10654> (visited on 15/04/2025).
- [27] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 11th Mar. 2025. DOI: 10.48550/arXiv.2303.18223. arXiv: 2303.18223[cs]. URL: <http://arxiv.org/abs/2303.18223> (visited on 27/03/2025).
- [28] Xiangrong Zhu et al. *Knowledge Graph-Guided Retrieval Augmented Generation*. 8th Feb. 2025. DOI: 10.48550/arXiv.2502.06864. arXiv: 2502.06864[cs]. URL: <http://arxiv.org/abs/2502.06864> (visited on 04/05/2025).

Bibliography

Appendix A

GitHub Repository

The full source code, data-loading scripts, and evaluation program are available at:
<https://github.com/olefha/coworker>

Appendix B

Demonstration setup

Chatty Digital Coworker

Did we run at full capacity?

On October 18, 2024, the total production quantity was 18,867.03 liters. This indicates that the production did not reach full capacity on that day.

What were the reasons for lost time or capacity?

The reason for lost time or capacity on October 18, 2024, was a high bacterial count in raw milk, classified as a medium severity issue. The action taken to address this was conducting an additional pasteurization cycle.

How was the yield based on input raw material volume and output production?

There were no raw material inputs recorded with an arrival date of October 18, 2024. Without this data, it is not possible to calculate the yield based on input raw material volume and output production for that specific date.

Try again, but use the neo4jTool instead.

Based on the data for October 18, 2024:

- The total raw material input volume was 20,116.36 liters.
- The total production output was 18,867.03 liters.

To calculate the yield, we can use the formula: $\text{Yield (\%)} = (\text{Total Production} / \text{Total Raw Material Input}) * 100$.

Calculating the yield:
 $\text{Yield} = (18,867.03 / 20,116.36) * 100 = 93.79\%$

Therefore, the yield based on the input raw material volume and output production for October 18, 2024, was approximately 93.79%.

Ask a production question...

Send

```
(1) Server is running on http://localhost:5001
(1) PostgreSQL Database connected successfully.
(1) Schema refreshed successfully.
(1) Neo4j Graph initialized successfully.
(1) User Question: Did we run at full capacity?
(1) -----
(1) Neo4jTool Invoked: -----
(1) Cypher Query: MATCH (d:DairyPlant)-[:HAS_PRODUCTION_DATA]->(p:ProductionData) WHERE datetime(p.production_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(p.production_date) < datetime('2024-10-19T00:00:00Z') RETURN sum(p.quantity) as total_production
(1) PostgreSQL Tool Invoked: -----
(1) ToolResult: The total quantity produced on October 18, 2024, was 18867.03 Liters.
(1) Intermediate Steps: [
(1) {
(1)   "query": "MATCH (d:DairyPlant)-[:HAS_PRODUCTION_DATA]->(p:ProductionData) WHERE datetime(p.production_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(p.production_date) < datetime('2024-10-19T00:00:00Z') RETURN sum(p.quantity) as total_production"
(1) },
(1) {
(1)   "context": [
(1)     {
(1)       "total_production": 18867.03
(1)     }
(1)   ]
(1) }
(1) ]
(1) ToolResult: The total production is 18,867.03.
(1) Final Output:
(1) On October 18, 2024, the total production quantity was 18,867.03 liters. This indicates that the production did not reach full capacity on that day.
(1) -----
(1) User Question: What were the reasons for lost time or capacity?
(1) -----
(1) Neo4jTool Invoked: -----
(1) Cypher Query: MATCH (d:DairyPlant)-[:HAS_NON_CONFORMITY_RECORD]->(n:NonConformityRecords) WHERE datetime(n.deviation_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(n.deviation_date) < datetime('2024-10-19T00:00:00Z') RETURN n.description, n.severity, n.action_taken
(1) Intermediate Steps: [
(1) {
(1)   "query": "MATCH (d:DairyPlant)-[:HAS_NON_CONFORMITY_RECORD]->(n:NonConformityRecords) WHERE datetime(n.deviation_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(n.deviation_date) < datetime('2024-10-19T00:00:00Z') RETURN n.description, n.severity, n.action_taken"
(1) },
(1) {
(1)   "context": [
(1)     {
(1)       "n.description": "High bacterial count in raw milk",
(1)       "n.severity": "Medium",
(1)       "n.action_taken": "Conducted additional pasteurization cycle."
(1)     }
(1)   ]
(1) }
(1) ]
(1) ToolResult: High bacterial count in raw milk, Medium severity, Conducted additional pasteurization cycle.
(1) Final Output:
(1) The reason for lost time or capacity on October 18, 2024, was a high bacterial count in raw milk, classified as a medium severity issue. The action taken to address this was conducting an additional pasteurization cycle.
(1) -----
(1) User Question: How was the yield based on input raw material volume and output production?
(1) -----
(1) PostgreSQL Tool Invoked: -----
(1) ToolResult: Final answer: There are no raw material inputs with an arrival date of '2024-10-18'.
(1) Intermediate Steps: [
(1) {
(1)   "query": "MATCH (d:DairyPlant)-[:HAS_RAW_MATERIAL_INPUT]->(r:RawMaterialInput) WHERE datetime(r.arrival_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(r.arrival_date) < datetime('2024-10-19T00:00:00Z') RETURN sum(r.quantity) as total_raw_material"
(1) },
(1) {
(1)   "context": [
(1)     {
(1)       "total_raw_material": 20116.36
(1)     }
(1)   ]
(1) }
(1) ]
(1) ToolResult: The total raw material is 20,116.36.
(1) Neo4jTool Invoked: -----
(1) Cypher Query: MATCH (d:DairyPlant)-[:HAS_PRODUCTION_DATA]->(p:ProductionData) WHERE datetime(p.production_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(p.production_date) < datetime('2024-10-19T00:00:00Z') RETURN sum(p.quantity) as total_production
(1) Intermediate Steps: [
(1) {
(1)   "query": "MATCH (d:DairyPlant)-[:HAS_PRODUCTION_DATA]->(p:ProductionData) WHERE datetime(p.production_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(p.production_date) < datetime('2024-10-19T00:00:00Z') RETURN sum(p.quantity) as total_production"
(1) },
(1) {
(1)   "context": [
(1)     {
(1)       "total_production": 18867.03
(1)     }
(1)   ]
(1) }
(1) ]
(1) ToolResult: The total production is 18,867.03.
(1) Final Output:
(1) Based on the data for October 18, 2024:
(1) - The total raw material input volume was 20,116.36 liters.
(1) - The total production output was 18,867.03 liters.
(1) To calculate the yield, we can use the formula: Yield (%) = (Total Production / Total Raw Material Input) * 100.
(1) Calculating the yield:
(1) Yield = (18,867.03 / 20,116.36) * 100 = 93.79%
(1) Therefore, the yield based on the input raw material volume and output production for October 18, 2024, was approximately 93.79%.
```

Appendix C

Terminal logs from stakeholders demonstration

```

npx run dev
[1] - Network: use --host to expose
[1] Server is running on http://localhost:5901
[1] PostgreSQL Database connected successfully.
[1] Schema refreshed successfully.
[1] Need4j Graph initialized successfully.
[1] User Question: Did we run at full capacity?
[1] -----
[1] Need4jTool Invoked: -----
[1] Cypher Query: MATCH (d:DairyPlant)-[:HAS_PRODUCTION_DATA]->(p:ProductionData) WHERE datetime(p.production_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(p.production_date) < datetime('2024-10-19T00:00:00Z') RETURN sum(p.quantity) as total_production
[1] PostgresTool Invoked: -----
[1] ToolResult: The total quantity produced on October 18, 2024, was 18867.03 Liters.
[1] Intermediate Steps: [
[1] {
[1]   "query": "MATCH (d:DairyPlant)-[:HAS_PRODUCTION_DATA]->(p:ProductionData) WHERE datetime(p.production_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(p.production_date) < datetime('2024-10-19T00:00:00Z') RETURN sum(p.quantity) as total_production"
[1] },
[1] {
[1]   "context": [
[1]     {
[1]       "total_production": 18867.03
[1]     }
[1]   ]
[1] }
[1] ]
[1] ToolResult: The total production is 18,867.03.
[1] Final Output:
[1] On October 18, 2024, the total production quantity was 18,867.03 liters. This indicates that the production did not reach full capacity on that day.
[1] -----
[1] User Question: What were the reasons for lost time or capacity?
[1] -----
[1] Need4jTool Invoked: -----
[1] Cypher Query: MATCH (d:DairyPlant)-[:HAS_NON_CONFORMITY_RECORD]->(n:NonConformityRecords) WHERE datetime(n.deviation_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(n.deviation_date) < datetime('2024-10-19T00:00:00Z') RETURN n.description, n.severity, n.action_taken
[1] PostgresTool Invoked: -----
[1] ToolResult: High bacterial count in raw milk, Medium severity, Conducted additional pasteurization cycle.
[1] Intermediate Steps: [
[1] {
[1]   "query": "MATCH (d:DairyPlant)-[:HAS_NON_CONFORMITY_RECORD]->(n:NonConformityRecords) WHERE datetime(n.deviation_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(n.deviation_date) < datetime('2024-10-19T00:00:00Z') RETURN n.description, n.severity, n.action_taken"
[1] },
[1] {
[1]   "context": [
[1]     {
[1]       "n.description": "High bacterial count in raw milk",
[1]       "n.severity": "Medium",
[1]       "n.action_taken": "Conducted additional pasteurization cycle."
[1]     }
[1]   ]
[1] }
[1] ]
[1] ToolResult: High bacterial count in raw milk, Medium severity, Conducted additional pasteurization cycle.
[1] Final Output:
[1] The reason for lost time or capacity on October 18, 2024, was a high bacterial count in raw milk, classified as a medium severity issue. The action taken to address this was conducting an additional pasteurization cycle.
[1] -----
[1] User Question: How was the yield based on input raw material volume and output production?
[1] -----
[1] Need4jTool Invoked: -----
[1] Cypher Query: MATCH (d:DairyPlant)-[:HAS_RAW_MATERIAL_INPUT]->(r:RawMaterialInput) WHERE datetime(r.arrival_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(r.arrival_date) < datetime('2024-10-19T00:00:00Z') RETURN sum(r.quantity) as total_raw_material
[1] PostgresTool Invoked: -----
[1] ToolResult: Final answer: There are no raw material inputs with an arrival date of '2024-10-18'.
[1] Intermediate Steps: [
[1] {
[1]   "query": "MATCH (d:DairyPlant)-[:HAS_RAW_MATERIAL_INPUT]->(r:RawMaterialInput) WHERE datetime(r.arrival_date) >= datetime('2024-10-18T00:00:00Z') AND datetime(r.arrival_date) < datetime('2024-10-19T00:00:00Z') RETURN sum(r.quantity) as total_raw_material"
[1] },
[1] {
[1]   "context": [
[1]     {
[1]       "total_raw_material": 20116.36
[1]     }
[1]   ]
[1] }
[1] ]
[1] ToolResult: The total raw material is 20,116.36.
[1] Final Output:
[1] Based on the data for October 18, 2024:
[1] - The total raw material input volume was 20,116.36 liters.
[1] - The total production output was 18,867.03 liters.
[1] To calculate the yield, we can use the formula: Yield (%) = (Total Production / Total Raw Material Input) * 100.
[1] Calculating the yield:
[1] Yield = (18,867.03 / 20,116.36) * 100 = 93.79%
[1] Therefore, the yield based on the input raw material volume and output production for October 18, 2024, was approximately 93.79%.
[1] -----

```

Appendix D

Custom benchmark set

Tr	Question	Tr	Reference Answer	Tr	Comments/SQL query used for answer
	What was the production volume last 24 hours?		The plant produced 18867.03 Liters of good product.		SELECT SUM(quantity) FROM productiondata WHERE production_date = '2024-10-19'::date - INTERVAL '1 day';
	Did we run at full capacity?		Since the maximum capacity is 53857 Liters, we utilised about 35% of the capacity.		Maximum capacity in a day: SELECT sum(quantity) AS sum FROM productiondata GROUP BY production_date ORDER BY sum DESC LIMIT 1; Capacity the last day: SELECT SUM(quantity) FROM productiondata WHERE production_date = '2024-10-19'::date - INTERVAL '1 day';
	What were the reason for lost time or capacity?		Looking at the deviations that occurred in the last day, we had a single deviation with medium severity where it mentions a high bacterial count in raw milk.		SELECT * FROM nonconformityrecords WHERE deviation_timestamp::date = '2024-10-19'::date - INTERVAL '1 day';
	How was the yield based on input raw material volume and output production?		The yield is estimated to be 93.8%.		SELECT (SELECT SUM(quantity) FROM productiondata WHERE production_date = '2024-10-19'::date - INTERVAL '1 day') / (SELECT SUM(quantity) FROM rawmaterialinput WHERE arrival_date = '2024-10-19'::date - INTERVAL '1 day');
	What is the variation in output fat and protein last 24 hours?		The standard deviation for fat % is 0,35 and protein % is 0.14%.		SELECT stddev(fat_content) AS fat_deviation, stddev(protein_content) AS proten_deviation FROM qualitydata WHERE test_timestamp::date = '2024-10-19'::date - INTERVAL '1 day';
	What is the process variation in the last 24 hours?		The mean temperature is 61.6 degrees with a standard deviation of 10.7 degrees and the mean pressure is 159.7 Psi and standard deviation is 16.7 PSI.		SELECT avg(temperature) AS temperature_avg, stddev(temperature) AS temperature_stddev, avg(pressure) AS pressure_avg, stddev(pressure) AS pressure_stddev, avg(flow_rate) AS avg_flow_rate, stddev(flow_rate) AS stddev_flow_rate FROM processdata WHERE DATE(start_time) = ('2024-10-19'::date - INTERVAL '1 day');
	Was the process stable?		The Cpk (Process Capability Index) is 0.23 of the process in the last day indicating that we are not within the specification limits and thus the process was not stable.		Getting the SOP: SELECT * FROM sop WHERE procedure_name = 'Quality Testing Protocol' ORDER BY last_updated DESC LIMIT 1; Getting the mean and standard deviation: SELECT avg(fat_content) AS avg_fat_content, stddev(fat_content) AS fat_deviation, avg(protein_content) AS avg_protein_content, stddev(protein_content) AS proten_deviation FROM qualitydata WHERE test_date::date = '2024-10-19'::date - INTERVAL '1 day'; Calculating the Cpk (Process Capability Index) using the formula found from https://www.6sigma.us/process-improvement/process-capability-index-cpk/ ;
	Did we have any listed entries in the shift process log?		Yes, three entries are listed, one for the morning shift, one for the afternoon shift and one for the night shift. The night shift mentions an observed minor equipment malfunctions while the in the morning and afternoon no significant issues were reported.		SELECT * FROM shiftprocesslogs WHERE shift_date = '2024-10-19'::date - INTERVAL '1 day';
	Have we documented any non-conformities?		Yes, there is one entry on high bacterial count in raw milk that occurred at 3:58 and had a medium severity. It was solved at 4:30 and the action taken to solve it was conducting additional pasteurization cycle.		SELECT * FROM nonconformityrecords WHERE deviation_date::date = ('2024-10-19'::date - INTERVAL '1 day');
	Does the shift process log contain any info that could explain the problems with raw milk?		There is an entry about minor equipment malfunction, this might explain the problem.		SELECT * FROM shiftprocesslogs WHERE shift_date = '2024-10-19'::date - INTERVAL '1 day';
	Do we have similar deviations / situations in the past where we had high bacterial count in raw milk?		Yes, we had two other instances where we we had high bacterial count in raw milk. These instances were on 2024-31-08 and 2024-07-12.		SELECT * FROM nonconformityrecords WHERE LOWER(description) LIKE '%high bacterial count in raw milk%';
	What were the root causes for the deviations?		Here are several possible causes based on the data: 1. Shift process logs mentions minor equipment malfunction which might have lead to the deviation 2. Raw milk that arrived yesterday and had high bacterial count.		Finding shiftprocess logs: SELECT * FROM shiftprocesslogs WHERE shift_date = '2024-10-19'::date - INTERVAL '1 day'; Finding raw material input: SELECT * FROM rawmaterialinput WHERE arrival_timestamp::date = '2024-10-19'::date - INTERVAL '1 day';
	What actions can be taken for solving the deviations?		In the past we have taken the following actions to solve similar deviations: 1. Conducting additional pasteurization cycle. 2. Improving storage conditions and conducted training for handling.		SELECT * FROM nonconformityrecords WHERE LOWER(description) LIKE '%high bacterial count in raw milk%';
	How can we reduce deviations in the future?		The model should look at previous deviation to see if there is a pattern, and suggest possible solutions. Will have to introduce fake data.		
	What is the total quantity of raw materials received from each supplier?		Here is the list of each supplier along the total quantity recieved from that supplier: - Sunshine Dairy 688642 - Mountain Dairy 625807 - Happy Cows 488432 - Farm Fresh 498949 - Valley Dairy 425845 - Riverside Dairy 540302 - Sunny Meadows 683266 - Hillside Dairy 571867 - Green Valley Dairy 699018 - Country Milk 547333 - Dairy Farms Inc. 621818 - Highland Farms 580204		SELECT supplier_name, SUM(quantity) AS total_quantity FROM rawmaterialinput GROUP BY supplier_name;
	What is the current version of the Pasteurization Process SOP and when was it last updated?		The current version is 3.9 and was last updated on the 10th of october 2024.		SELECT version, last_updated FROM sop WHERE procedure_name = 'Pasteurization Process' ORDER BY version DESC LIMIT 1;
	Who were on the shift yesterday?		Sarah Brown worked on the morning shift while Mike Johnson worked the afternoon and night shift.		SELECT operator_name FROM shiftprocesslogs WHERE shift_date = '2024-10-19'::date - INTERVAL '1 day';
	What are the specification limits for the pasteurization process according to our SOPs?		The time must be between 32 and 35 minutes, while the temperature must be between 72 and 75 degrees.		SELECT sop_id, spec_limits, version FROM sop WHERE procedure_name = 'Pasteurization Process' ORDER BY version DESC LIMIT 1;
	What are the most recent bacteria count levels recorded in our quality tests?		The most recent bacteria count level is 11814.		SELECT test_date, bacteria_count FROM qualitydata ORDER BY test_date DESC LIMIT 1;
	What has been the total daily production output and raw material input each day of the past week?		- 2024-10-13: The production output is 17829 while the production input is 22228 - 2024-10-14: The production output is 10766 while the production input is 19058 - 2024-10-15: The production output is 9977 while the production input is 5872 - 2024-10-16: The production output is 27180 while the production input is 21271 - 2024-10-17: The production output is 8263 while the production input is 25338 - 2024-10-18: The production output is 18867 while the production input is 20116 - 2024-10-19: The production output 28714 while the production input is 44788		SELECT sum (rm.quantity) AS "Material Input", rm.arrival_timestamp::date FROM rawmaterialinput AS rm WHERE rm.arrival_timestamp::date > '2024-10-19'::date - INTERVAL '7 days' GROUP BY rm.arrival_timestamp::date ORDER BY rm.arrival_timestamp::date; SELECT pd.production_date, sum(pd.quantity) AS "Production Output" FROM productiondata AS pd WHERE pd.production_date > '2024-10-19'::date - INTERVAL '7 days' GROUP BY pd.production_date ORDER BY production_date;

Appendix E

Final prompt

```
const agentPrompt = `
```

Today's date is 2024-10-19, ergo CURRENT_DATE=2024-10-19.

If a question does not specify any date, use the previous day (2024-10-18) as the date.

You are an AI assistant for the Main Dairy Plant with access to two complementary data sources:

1. "neo4jTool" - A knowledge graph containing operational data, process logs, and quality metrics
2. "postgresTool" - A relational database with additional structured plant data

Important Plant Information:

- Maximum production capacity: 53857 Liters
- All Neo4j timestamps use format: "YYYY-MM-DDThh:mm:ssZ"

Available Tools:

1. "neo4jTool" - Generates a Cypher query based on the user question and returns results in natural language.

- * Best for: Connected data, relationships between entities, process flows, quality correlations

- * Query format: Cypher query language with specific datetime handling requirements

- * Example: MATCH (d:DairyPlant)-[:HAS_QUALITY_DATA]->(q:QualityData) WHERE datetime(q.test_date) = datetime('2024-10-18T00:00:00Z') RETURN q

2. "postgresTool" - Generates SQL queries for the relational database and returns structured results.

- * Best for: Detailed operational metrics, historical trends, standardized reports, inventory tracking

- * Query format: Standard SQL with table joins where necessary

- * Example: SELECT * FROM production_metrics WHERE date = '2024-10-18'

Tool Selection Guidelines:

- Use neo4jTool when:

- * Question involves relationships between different data types (e.g., how process affects quality)
- * Need to traverse connected entities (e.g., tracing product from raw materials to final tests)
- * Looking for patterns within interconnected systems

- Use postgresTool when:

- * Question requires detailed operational metrics
- * Need for precise historical data with standardized formatting
- * Question involves reporting on structured data like inventory, staffing, or financial metrics

- Use both tools when:

- * Complex questions require data synthesis from multiple sources
- * Verification of findings across different data systems is needed
- * Building comprehensive root cause analysis

Core Rules:

1. Data Accuracy:

- Base your answers solely on tool results, never guess missing data
- Ensure numerical answers exactly match tool results
- If one tool returns incomplete results, try the other tool
- If postgresTool returns 0, try querying different tables

2. Question Analysis:

- Break complex questions into sub-questions
- Identify which tool is most appropriate for each sub-question

- Plan multi-step query strategies for comprehensive answers

3. Data Processing:

- Calculate derived insights from multiple data points
- For process variation, include means and standard deviations
- When combining data from both tools, clearly explain your methodology
- Present all results in natural language with appropriate context

Complex Question Handling Strategy:

1. Multi-Step Query Approach:

- For questions about reasons or causes (e.g., "Why did X happen?"):
 - a. First query: Look for direct evidence in NonConformityRecords (neo4jTool)
 - b. Second query: Check related process data for abnormalities (neo4jTool)
 - c. Third query: Examine shift logs for operational issues (neo4jTool)
 - d. Fourth query: Verify against operational metrics (postgresTool)
 - e. Additional queries: Look for supporting evidence in other sources
- For performance or capacity questions:
 - a. Query production data to establish actual output (both tools)
 - b. Query process data to identify bottlenecks (neo4jTool)
 - c. Check quality data for rejection rates (neo4jTool)
 - d. Review detailed operational metrics (postgresTool)
 - e. Analyze shift logs for contextual information (neo4jTool)
- For trend or pattern questions:
 - a. Gather historical data points (both tools)
 - b. Check for seasonal or periodic patterns (postgresTool)
 - c. Look for correlations between different metrics (neo4jTool)

d. Validate findings across both data sources

2. Data Synthesis:

- Combine results from multiple queries and tools
- Look for connections between different data points
- Calculate derived metrics when needed
- Present a coherent narrative that integrates all findings

Example Complex Question Strategies:

1. "What were the reasons for lost time or capacity yesterday?"

Query sequence:

a. Check NonConformityRecords for documented issues (neo4jTool):

```
MATCH (d:DairyPlant)-[:HAS_NON_CONFORMITY_RECORD]->(n:NonConformityRecords)
WHERE datetime(n.deviation_date) = datetime('2024-10-18T00:00:00Z')
RETURN n.description, n.severity, n.action_taken
```

b. Compare actual vs maximum production (neo4jTool):

```
MATCH (d:DairyPlant)-[:HAS_PRODUCTION_DATA]->(p:ProductionData)
WHERE datetime(p.production_date) = datetime('2024-10-18T00:00:00Z')
RETURN p.quantity, p.unit
```

c. Check process data for anomalies (neo4jTool):

```
MATCH (d:DairyPlant)-[:HAS_PROCESS]->(p:ProcessData)
WHERE datetime(p.start_time) >= datetime('2024-10-18T00:00:00Z')
AND datetime(p.start_time) < datetime('2024-10-19T00:00:00Z')
RETURN p.process_name, p.temperature, p.pressure, p.flow_rate
```

d. Review shift logs for operational issues (neo4jTool):

```
MATCH (d:DairyPlant)-[:HAS_SHIFT_PROCESS_LOG]->(s:ShiftProcessLogs)
WHERE datetime(s.shift_date) = datetime('2024-10-18T00:00:00Z')
RETURN s.log_entry
```

e. Check detailed downtime metrics (postgresTool):

```
SELECT equipment_id, downtime_minutes, reason_code, notes
FROM equipment_downtime
WHERE date = '2024-10-18'
ORDER BY downtime_minutes DESC
```

2. "What are the main factors affecting product quality this week?"

Query sequence:

a. Analyze quality metrics (neo4jTool):

```
MATCH (d:DairyPlant)-[:HAS_QUALITY_DATA]->(q:QualityData)
WHERE datetime(q.test_date) >= datetime('2024-10-12T00:00:00Z')
AND datetime(q.test_date) <= datetime('2024-10-18T23:59:59Z')
RETURN q.batch_number, q.fat_content, q.protein_content, q.pH_level,
q.bacteria_count
```

b. Check process parameters for variation (neo4jTool):

```
MATCH (d:DairyPlant)-[:HAS_PROCESS]->(p:ProcessData)
WHERE datetime(p.start_time) >= datetime('2024-10-12T00:00:00Z')
AND datetime(p.start_time) <= datetime('2024-10-18T23:59:59Z')
RETURN p.process_name, avg(p.temperature) as avg_temp, stDev(p.temperature)
as temp_variation
```

c. Analyze raw material quality data (postgresTool):

```
SELECT supplier_id, test_date, fat_content, protein_content, bacterial_count
```

FROM raw_material_quality

WHERE test_date BETWEEN '2024-10-12' AND '2024-10-18'

Tool-Specific Query Guidelines:

- If a question does not specify any date, for example ("Have we documented any non-conformities?"), assume the date is the previous day (2024-10-18).
- The dates in the data sources are hour, minute, and seconds specific, therefore you need to handle for this in the query, for example by searching for dates between today's date and 24 hours earlier.

Neo4j Query Guidelines:

- Always use datetime() for date comparisons
- Include full timestamp format: T00:00:00Z
- Example: WHERE datetime(s.shift_date) = datetime('2024-10-19T00:00:00Z')
- For time ranges: Use >= and < operators with appropriate timestamps
- For aggregations: Use Neo4j's built-in functions like avg(), count(), sum(), stDev()

Postgres Query Guidelines:

- Use standard date format: 'YYYY-MM-DD'
- For date ranges: Use BETWEEN 'start_date' AND 'end_date'
- For time-specific queries: Use timestamp format 'YYYY-MM-DD HH:MI:SS'
- Apply appropriate JOINS when data spans multiple tables
- Use GROUP BY for aggregated metrics
- use the postgres schema below to construct only valid queries

Remember:

1. Never return "I don't know" without first trying multiple query approaches with both tools

2. Break down complex questions into smaller, answerable parts

3. Make multiple queries across both tools to build a complete picture

4. Look for relationships between different data points

5. Consider both direct and indirect causes

6. Present results as a coherent narrative

7. Explain your reasoning and how different data points connect

8. Use statistical measures when appropriate

9. Consider time-based patterns and trends

10. Always verify your conclusions against the actual data

11. Cross-reference findings between neo4jTool and postgresTool when possible

Schema Information:

Neo4j Schema:

`${kgSchema}`

Postgres Schema:

`${tableInfoStr}`

``;`