UNIVERSITY OF LJUBLJANA

SCHOOL OF ECONOMICS AND BUSINESS

MASTER THESIS

EXPLORING THE FEASIBILITY OF RETRIEVAL-AUGMENTED GENERATION WITH INTEGRATING PRIVATE DATA INTO AI-DRIVEN BPM TOOL IN COMPANY PROMATIS SOFTWARE

Mühlacker, June 2025 TURKAN BURAK SARIAGAC

AUTHORSHIP STATEMENT

The undersigned Turkan Burak Sariagac, a student at the University of Ljubljana, School of Economics and Business, (hereafter: UL SEB), author of this written final work of studies with the title: Exploring the feasibility of retrieval-augmented generation with integrating private data into AI-driven BPM tool in company PROMATIS software, prepared under supervision of Prof. Dr. Mojca Indihar Štemberger and co-supervision of Prof. Dr. Frank Morelli,

DECLARE

1. this written final work of studies to be based on the results of my own research;
2. the printed form of this written final work of studies to be identical to its electronic form;
3. the text of this written final work of studies to be language-edited and technically in adherence with the UL SEB's Technical Guidelines for Written Works, which means that I cited and / or quoted works and opinions of other authors in this written final work of studies in accordance with the UL SEB's Technical Guidelines for Written Works;
4. to be aware of the fact that plagiarism (in written or graphical form) is a criminal offence and can be prosecuted in accordance with the Criminal Code of the Republic of Slovenia;
5. to be aware of the consequences a proven plagiarism charge based on the this written final work could have for my status at the UL SEB in accordance with the relevant UL SEB rules;
6. to have obtained all the necessary permits to use the data and works of other authors which are (in written or graphical form) referred to in this written final work of studies and to have clearly marked them;
7. to have acted in accordance with ethical principles during the preparation of this written final work of studies and to have, where necessary, obtained permission of the Ethics Committee;
8. my consent to use the electronic form of this written final work of studies for the detection of content similarity with other written works, using similarity detection software that is connected with the UL SEB Study Information System;
9. to transfer to the University of Ljubljana free of charge, non-exclusively, geographically and time-wise unlimited the right of saving this written final work of studies in the electronic form, the right of its reproduction, as well as the right of making this written final work of studies available to the public on the World Wide Web via the Repository of the University of Ljubljana;
10. my consent to publication of my personal data that are included in this written final work of studies and in this declaration, when this written final work of studies is published;
11. that I have verified the authenticity of the information derived from the records using artificial intelligence tools.

Mühlacker, May 15th, 2025 Author's signature: B. Sariagac

TABLE OF CONTENTS

1 INTRODUCTION 1

2 LITERATURE REVIEW 3

2.1 Introduction to RAG 3

2.1.1 Indexing 5

2.1.1.1 Chunking 6

2.1.1.2 Vectorization and Embedding 8

2.1.1.3 Vector storage 10

2.1.2 Retrieval 12

2.1.3 Augmentation and Generation 13

2.1.3.1 Reranking and Prompting 13

3.1.3.2 Generation 14

2.2 Example of a RAG Workflow 14

2.2.1 Indexing 15

2.2.2 Generation 15

2.3 RAG Merits and drawbacks 17

2.3.1 RAG Merits 17

2.3.2 RAG drawbacks 18

3 RESEARCH METHODOLOGY 19

3.1 Selection of Research Design and Methodology 19

3.2 Formulation of Research Questions 21

3.3 Case Description 21

3.4 Data Collection Methods 22

3.5 Data Analysis and Interpretation 23

3.6 Quality Criteria 24

3.6.1 Relevance to Practice 24

3.6.2 Scientific Rigor 24

3.6.3 Prototype Utility and Robustness 24

3.6.4 Evaluation Validity 25

4 BUSINESS PROCESS MANAGEMENT AT HORUS 25

4.1 Horus Business Modeler 25
4.2 Aim of the Solution 28
4.3 Solution Idea 29
5 PROTOTYPE RAG DEVELOPMENT 31
5.1 Functionalities 31
5.1.1 Data Provision and Indexing 31
5.1.2 Query Processing and Retrieval 32
5.1.3 Contextual Augmentation and Prompt Construction 32
5.1.4 Response Generation and Output Refinement 32
5.1.5 Multilingual Capability and Knowledge-Based Answering 32
5.1.6 Knowledge-Based Answering and Query Negation 33
5.1.7 Multi-Vector Storage Capability for Complex Queries 33
5.2 RAG design 34
5.2.1 Python Code 34
5.2.2 Embedding Model 35
5.2.3 Chroma DB 36
5.2.4 GPT-4o 37
5.3 Implementation within Horus AI 39
6 TESTING AND EXPERIMENTAL FINDINGS 41
6.1 Experimental Setup 41
6.2 Evaluation criteria 43
6.3 Experimental results 44
6.3.1 Variation of *k* test 45
6.3.2 Variation of chunk size test 48
6.3.3 Rejection testing 50
6.3.4 Latency testing 52
6.3.5 Language testing 53
7 DISCUSSION 54
7.1 Lessons learned 54
7.1.1 Retrieval and Response Accuracy 55
7.1.2 Fluency and Output Structure 55
7.1.3 Handling of Missing Information and Hallucinations 55

7.1.4 Latency and System Efficiency 555
7.1.5 Multilingual Capabilities and Language Adaptability 56
7.2 Challenges and potential solutions 56
7.2.1 Optimization of Chunking Strategies 57
7.2.2 Retrieval Precision and Re-Ranking Mechanisms 57
7.2.3 Mitigating Hallucinations and Ensuring Answer Reliability 58
7.2.4 Multilingual Retrieval Optimization 59
7.2.5 Reducing Overhead for Indexing 59
7.3 Limitations 59
7.4 Implications for practice 60
8 CONCLUSION AND OUTLOOK 61
8.1 Conclusion 61
8.2 Outlook 62
REFERENCE LIST 64
APPENDICES 69

LIST OF FIGURES

Figure 1: Representative RAG instance 4
Figure 2: Visualized Vector Space 9
Figure 3: Indexing model 15
Figure 4: Query to answer model 16
Figure 5: Design Science Research Cycles 20
Figure 6: Modeling options within Horus Business Modeler 26
Figure 7: High level production planning process model 27
Figure 8: Level 2 min-max planning Process Model 27
Figure 9: BP4 Apps sample dunning process model 30
Figure 10: RAG components 34
Figure 11: Initial Output (no link) 38
Figure 12: Optimized Output (with link) 38
Figure 13: RAG Process Diagram 39
Figure 14: Isolated Horus AI Pipeline 39
Figure 15: Horus AI with RAG 41

LIST OF TABLES

Table 1: Vector Embeddings..... 9
Table 2: Vector Store Comparison.....11
Table 3: Comparison of OpenAI Embedding models 36
Table 4: Time to upload by chunk size..... 52

LIST OF APPENDICES

Appendix 1: Povzetek (Summary in Slovene language).....1
Appendix 2: Supplementary Horus AI Prompt Instructions 2
Appendix 3: Code Screenshots 4
Appendix 4: Chroma Compatibility Table.....5
Appendix 5: RAG Architecture.....6

LIST OF ABBREVIATIONS

AI – Artificial Intelligence

ANN – Approximate Nearest Neighbor

API – Application Programming Interface

BPM – Business Process Management

DB – Database

DSR – Design Science Research

GPT – Generative Pre-Trained Transformer

GRAG – Graphical Retrieval Augmented Generation

k-NN – *k* Nearest Neighbor

LLM – Large Language Model

METB – Massive Text Embedding Benchmark

ML – Machine Learning

NLP – Natural Language Processing

OCA – Oracle Cloud Applications

QA – Questions Answering

RAG – Retrieval Augmented Generation

UI – User Interface

1 INTRODUCTION

The rapid growth and advancement of artificial intelligence (AI) and machine learning (ML) technologies has opened up new opportunities for business process automation, optimization, and decision-making (Grohs et al., 2023). In particular, Large Language Models (LLMs) such as OpenAI's GPT has changed the way natural language processing can be leveraged to support enterprise-level applications (OpenAI, 2021). Organizations are increasingly looking to harness these models to extract actionable insights from their accumulated data and to augment existing software tools with generative capabilities. However, while generative AI has proven effective in tasks such as summarization, content generation, and natural language understanding, its practical integration with domain-specific and proprietary knowledge remains a significant challenge (Kau, 2024).

PROMATIS Software GmbH, a company specializing in Oracle-centric business solutions, has developed a proprietary business process management (BPM) platform called Horus. Horus enables organizations to create digital twins of their operations by modeling business processes, organizational structures, and supporting artifacts using standards like Petri Nets (van der Aalst, 2002). The recent integration of generative AI into Horus (referred to as Horus AI) has further enhanced its capabilities, allowing users to generate process models using natural language prompts. However, this extension currently operates without access to internal company-specific knowledge bases, which limits the contextual relevance and precision of the generated outputs. The language model used in Horus AI cannot draw upon PROMATIS's vast repository of domain knowledge, particularly the company's internally curated library of process models bundled within their "BP4 Apps" suite. This disconnect reduces the utility of the system, particularly for users seeking customized or industry-specific workflows.

To address this gap, Retrieval-Augmented Generation (RAG) emerges as a promising approach. RAG architectures augment the generative capabilities of LLMs by enabling them to retrieve and incorporate external, domain-specific information (Sharma et al., 2024). Unlike fine-tuning approaches that embed knowledge directly into the model weights, which is a resource-intensive and static method (Izacard & Grave, 2020; Soudani et al., 2024), RAG allows for the dynamic integration of up-to-date and customizable information through semantic search and retrieval components (Karpukhin et al., 2020). This architecture not only enhances the factual grounding of AI-generated responses but also supports traceability and transparency, which are increasingly vital in enterprise contexts (Gao et al., 2023).

This thesis explores the implementation of a RAG system as a proof of concept to enhance the capabilities of Horus AI by enabling it to retrieve and utilize proprietary process models during generation. The goal is to demonstrate how RAG can bridge the knowledge gap between large pre-trained models and proprietary organizational data. Although this study does not make direct modifications to the commercial Horus AI product, it presents a

standalone RAG prototype built with representative sample data. The prototype is evaluated through a series of controlled tests designed to simulate realistic user queries and validate the relevance and quality of generated process descriptions.

The project encompasses a detailed investigation into the structure and functionality of RAG systems, covering key components such as document chunking, vector embedding, retrieval mechanisms, and generation strategies. It evaluates the potential of RAG to solve practical challenges currently faced by PROMATIS and explores how such systems may generalize to broader BPM applications. In doing so, this thesis makes the following contributions to the academic discourse on the application of Retrieval-Augmented Generation and Large Language Models in business process modeling (Friedrich, 2010):

- Literature Review: A structured review of existing literature on RAG and LLMs in the context of business process management was conducted. The review identifies prevailing methodologies, exhibits merits and exposes current limitations.
- Prototype Implementation: A proof-of-concept prototype was developed, demonstrating the integration of a RAG pipeline into a business process modeling environment.
- Test Dataset and Experimentation: To evaluate the system, a representative dataset of 44 internal business process models was compiled, covering a range of operational domains. A series of structured experiments was conducted to test system capabilities under varying configurations, including chunk size, retrieval depth, and language.
- Evaluation Methodology: The prototype was assessed through a combination of qualitative and quantitative criteria. These included precision of retrieval, contextual completeness, fluency of generated responses, and system latency.

Moreover, the thesis responds to a critical usability problem in the current PROMATIS ecosystem. While the BP4 Apps provide users with access to comprehensive, pre-modeled business processes (ranging from billing to procurement and delivery), the interface exposes these models via minimal metadata such as process IDs. This makes efficient navigation and utilization of the dataset challenging. Furthermore, Horus AI's generative component operates independently of this repository, producing process models that, although structurally valid, may not align with the organization's best practices or prior implementations. By integrating RAG, the company can ensure that AI-generated processes reflect existing knowledge assets, thereby improving model quality, relevance, and user confidence.

The broader motivation behind this work is twofold. First, it seeks to enhance the business value of Horus by enabling its AI component to reason over internal knowledge bases. Second, it aims to offer a generalized framework and methodology that can be adopted by other organizations looking to combine generative AI with enterprise-specific data. Given the increasing demand for explainable, domain-specific AI solutions in the BPM space, this work 4technologies (Gartner, 2024).

This research is structured around the following focal points:

- Providing a theoretical and technical foundation for RAG systems, including their architecture, components, and operational mechanisms.
- Demonstrating how RAG can be integrated with business process modeling tools through the development of a functional prototype.
- Evaluating the effectiveness of RAG in improving retrieval and generation of business process content using internal datasets.
- Discussing the technical and operational implications of applying RAG in enterprise contexts.

2 LITERATURE REVIEW

2.1 Introduction to RAG

As defined by Honroth et al. (2024) of the Fraunhofer Institute, LLMs are "powerful models designed to understand and generate human language. They can analyze and understand text, generate coherent responses and perform language-related tasks. These functionalities can help to enhance or even automate certain tasks that previously required human input." This capacity for linguistic comprehension and generation has enabled LLMs to support a wide range of applications, from content generation and code completion to conversational agents and intelligent assistants.

However, LLMs exhibit notable limitations. Although models like GPT-4 have been trained on immense corpora of public text, their knowledge is frozen at the time of training. As a result, they lack awareness of current events, proprietary company data, and user-specific contexts (Wu et al., 2024). One method for overcoming this static nature is fine-tuning, in which the base model is retrained on task-specific or domain-specific datasets. While effective in enhancing domain alignment, fine-tuning is both computationally expensive and inflexible. It requires large volumes of labeled data, extensive computing resources, as well as expertise and time (Soudani et al., 2024).

To address these challenges, Lewis et al. have proposed the RAG architecture, an approach that combines the generative capabilities of LLMs with dynamic retrieval mechanisms to inject relevant external knowledge at inference time (Lewis et al., 2020). Rather than embedding all required knowledge into the model's weights, RAG architectures rely on a retriever component to fetch relevant documents from an external knowledge base, such as a vector database (DB), which are then appended to the model's input prompt. This enables the LLM to generate more factually accurate, contextually relevant, and up-to-date responses without modifying the weights of the underlying model (Gao et al., 2023).

RAG marks a significant step forward in language model architecture, transforming the generation process from a closed-system response based on static training data to an open-book model that incorporates current, curated, and even user-specific knowledge (Guu et al., 2020, Khandewal et al. 2020). This approach is especially valuable within industries that require responses grounded in proprietary data, such "as healthcare, finance, education and customer service" (Sharma et al., 2024, p.1).

From a system architecture perspective, RAG is typically composed of three core stages (Gao et al., 2023):

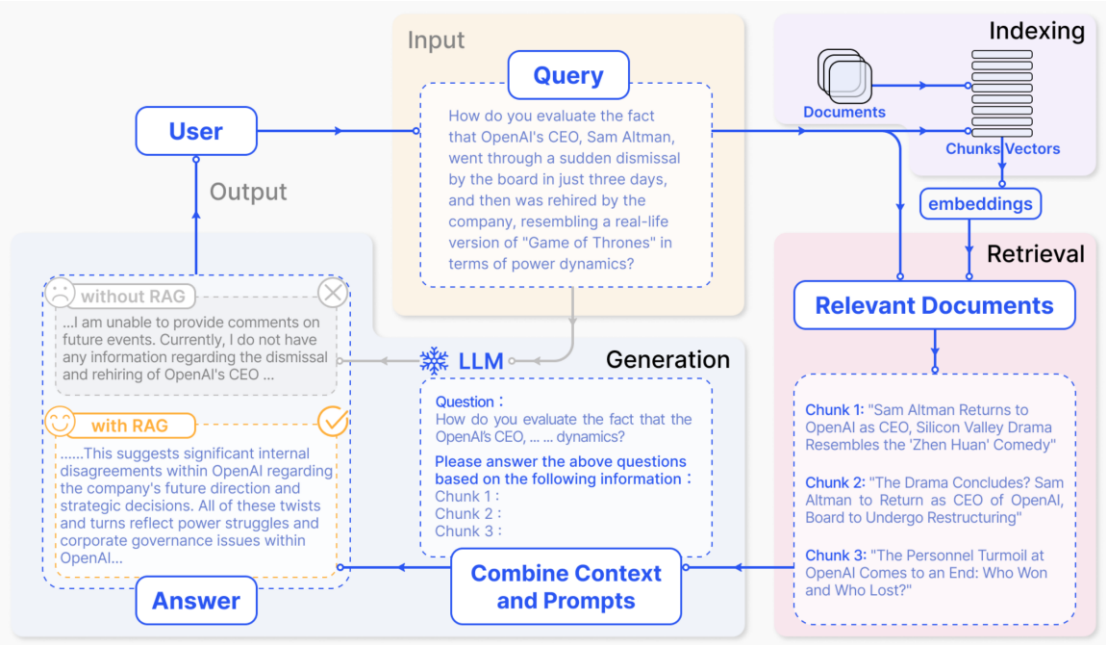
Indexing: The process of preprocessing and vectorizing textual documents, often using transformer-based embedding models, and storing them in a searchable vector database.

Retrieval: Upon receiving a query, the system converts it into a vector and performs a similarity search against the indexed documents, retrieving the top-*k* relevant chunks.

Augmentation and Generation: The retrieved documents are incorporated into a structured prompt and passed to a generative model (e.g., GPT-4o) which then produces a grounded response based on both the query and the contextual evidence.

Figure 1 illustrates the interconnection between these stages, highlighting how information flows through the RAG system in response to a user query. This three-part architecture not only optimizes output quality but also provides modularity, allowing developers to independently update the knowledge base, retrievers, or generators without retraining the entire system (Amazon Web Services, 2025).

Figure 1: Representative RAG instance



Source: Gao et al. (2023)

RAG's benefits extend beyond improved factual grounding. It also enables explainability and traceability, as the source documents used in generation can be surfaced to the end-user (Berandi et al., 2024). Furthermore, it offers a cost-effective alternative to fine-tuning, making it especially attractive for organizations that require domain-specific customization without the infrastructure overhead of retraining large models (Lee & Kim, 2024).

This chapter serves as a conceptual introduction to RAG and lays the foundation for the subsequent sections. The following chapters will delve deeper into the individual components of RAG (indexing, retrieval, and augmentation/generation) examining the technical processes, implementation strategies, and challenges involved in constructing a robust retrieval-augmented system.

2.1.1 Indexing

The first step to build a functioning RAG is the creation of a database which includes all the documents that shall be referenced in the system (Gao et al., 2023). Therefore, the initial phase involves gathering and pre-processing of external data which may include structured databases, unstructured text, or semi-structured documents (Wu et al, 2024). Depending on the source data and the RAG setup, this phase can require varying degrees of effort to execute. Given that the quality of the RAG system's output is directly tied to the relevance and structure of its indexed content, careful preprocessing and transformation of the source data are essential (Huang & Huang, 2024). The data must be provided in a way that accommodates the subsequent step, where the data is transformed into vector representations, known as embeddings. These so-called embeddings are a numerical representation of the data which captures semantic meaning (Barnett et al., 2024). This embedding process (alternatively called vectorization process) must be done as the information within the documents will be stored and processed in a so-called vector store. Their significance will be explained at a later stage within this chapter, but their main purpose is the storage and the numerical representation of information for comparison and retrieval. In order to create such embeddings, special variants of language models called embedding models are employed.

The steps to create a functioning vector store are as follows (Barnett, 2024):

- Collection of data

While not explored in depth here, the initial step involves sourcing relevant documents or records. These should reflect the domain knowledge necessary to answer user queries, such as business process descriptions, technical documentation, or policy manuals.

- Preparation of data

This stage encompasses formatting consistency, removal of noise, normalization of language, and extraction of relevant text sections. Standard techniques such as lowercasing, whitespace normalization, and metadata tagging are often applied to ensure that downstream models can interpret the content accurately.

- Chunking of data
- Vectorization
- Indexing in vector DB/Vector Storage

The following paragraphs will further clarify the core components and processes within the entire indexing phase. The preparation phase and collection of data will be omitted as these points have been touched upon and currently do not need further investigation.

2.1.1.1 *Chunking*

Chunking is a core processing technique in Retrieval-Augmented Generation (RAG) systems, involving the segmentation of input documents into smaller, more manageable units of text (Suzuoki & Hatano, 2024). Originally developed to address the token limitations of transformer-based language models, chunking remains a standard step in modern RAG architectures despite advancements in model context windows. Its continued relevance lies not only in managing input constraints, but also in enhancing the precision and efficiency of retrieval and generation phases (Barnett et al. 2024).

While state-of-the-art models such as GPT-4o support significantly expanded context windows, carelessly feeding them large contexts can negatively impact model performance. Larger inputs can dilute the focus of generation, increase computational costs, and introduce unnecessary redundancy (Han et al., 2023). Thus, chunking remains a key optimization strategy, both for minimizing the number of tokens passed to the LLM during inference and for improving the relevance of retrieval outputs.

Chunk Granularity: Coarse vs. Fine-Grained Chunks

The effectiveness of chunking is influenced by its granularity, the size and scope of each chunk. This design decision presents a trade-off between contextual richness and retrieval precision:

Coarse-grained chunks include larger segments of text, such as full paragraphs or sections. These may offer greater contextual coherence and are beneficial for high-level tasks like summarization or narrative generation. However, they can also introduce noise by embedding irrelevant or redundant content, which may misguide both the retriever and the language model (Yu et al., 2023). Additionally, large chunks can lead to token overflow, pushing the input closer to the LLM's maximum context size and thereby increasing inference cost (Martineau, 2024).

Fine-grained chunks, in contrast, consist of smaller and more focused units such as single sentences or phrase groups. These reduce semantic redundancy and allow for highly targeted retrieval, especially useful in tasks such as precise information extraction. However, they may lack the broader context needed to interpret the information correctly and can result in fragmented retrieval outputs that fail to answer complex queries fully (Bansal, 2024).

The optimal chunk size often depends on the downstream task. For example, tasks that require semantic understanding of larger discourse structures, such as summarization or process modeling, benefit from larger, overlapping chunks to preserve coherence. In contrast, code generation or granular question answering tasks may favor smaller, discrete units to maximize specificity.

Chunking Strategies in Practice

When implementing chunking within a RAG pipeline, different strategies can be applied depending on the structure and purpose of the source documents. These strategies vary in complexity and effectiveness, particularly in how they preserve contextual relevance. Broadly, chunking methods can be categorized into two main approaches: basic and advanced.

- Basic Chunking

  Fixed-length chunking, often aligned with natural boundaries like paragraphs or line breaks. While simple to implement, this method does not account for semantic structure and may result in incoherent or contextually incomplete segments.

- Advanced Chunking

  Semantically informed or dynamically adjusted chunking based on content structure. Tools such as LangChain’s Recursive Character Text Splitter or Markdown Header Text Splitter enable chunking that respects sentence boundaries and hierarchical headings (LangChain, 2025).

An important enhancement to chunking involves overlapping windows, where each chunk includes a portion of the preceding or succeeding text. This redundancy ensures that no

critical boundary information is lost during segmentation (Wang et al, 2019). The degree of overlap (e.g., 10–20%) can be tuned depending on task requirements and input length constraints. Chunking also interacts closely with retrieval parameters, particularly the number of retrieved chunks (*k*) selected during similarity search. Two commonly used strategies are as follows (Beltagy, 2020):

- Large chunk size, low *k*

  This approach emphasizes context-rich retrieval while limiting the number of inputs passed to the LLM. It reduces token usage and redundancy but risks missing more specific details if the chunk boundaries are suboptimal.

- Small chunk size, high *k*

  Prioritizes precision and diversity of retrieved information. While this may result in a more targeted response, it increases the complexity of the retrieval pipeline and may reintroduce context fragmentation.

Ultimately, optimal chunking decisions require a balance between computational efficiency, semantic integrity, and task-specific requirements. A poorly designed chunking strategy can degrade the performance of even the most sophisticated RAG systems, making it a critical area for customization and experimentation.

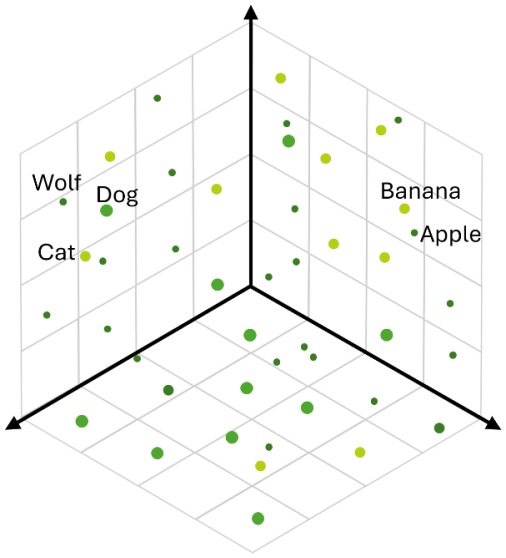
2.1.1.2 *Vectorization and Embedding*

Vectorization is a foundational step in RAG pipelines (Gao et al. 2023), serving as the bridge between unstructured natural language and computationally efficient information retrieval. At its core, vectorization involves the transformation of data such as text, images, or audio, into high-dimensional numerical representations known as embeddings (Espejel, 2022). These embeddings map semantic relationships into a continuous vector space, allowing for similarity-based operations that are essential to effective information retrieval (Reimers & Gurevych, 2019).

The embedding space is designed so that semantically similar inputs are represented by vectors in close proximity, while dissimilar inputs are positioned further apart. For instance, the words "banana" and "apple" would be located closer together than either would be to "wolf", "cat" or “dog". This semantic clustering forms the foundation for nearest-neighbor search, enabling the retrieval of conceptually relevant information even if the exact query terms differ from those in the documents. Figure 2 illustrates this concept visually, while Table 1 displays example numerical vectors that reflect this structure (Dascalescu & Hasan, 2023).

Because calculating the semantic similarity between words directly is not computationally feasible, embeddings enable mathematical operations, such as cosine similarity or Euclidean distance calculation to determine which chunks of data are most relevant to a given query (Minor & Kaucher, 2023). This numerical abstraction is thus critical to the retrieval phase of a RAG system, as the quality of retrieval directly depends on how well the embeddings capture the semantic content of both queries and documents (Peng et al. 2024).

Figure 2: Visualized Vector Space



Source: Dascalescu & Hasan (2023)

Table 1: Vector Embeddings

Word	Vector embedding
cat	[1.7, -0.4, 7.2, 19.6, 20.2]
dog	[1.7, -0.3, 6.9, 19.1, 21.1]
apple	[-5.2, 3.1, 0.2, 8.1, 3.5]
strawberry	[-4.9, 3.6, 0.9, 7.8, 3.6]
building	[60.1, -60.3, 10, -12.3, 9.2]
car	[81.6, -72.1, 16, -20.2, 102]
Query: fruit	[-5.1, 2.9, 0.8, 7.9, 3.1]

Source: Dascalescu & Hasan (2023)

Huggingface, a central platform for the NLP community, provides access to a diverse collection of open-source embedding models, along with benchmarks such as the Massive Text Embedding Benchmark (MTEB) (Muenninghoff et al., 2022). MTEB evaluates model performance across a range of tasks including clustering, classification, and semantic search.

While high performance on MTEB may indicate general applicability, it is important to recognize that many models are domain-specific, as such optimized for use in particular languages (e.g. Chinese), technical disciplines (e.g. medicine or law), or tasks (e.g. sentiment detection). Therefore, model selection should be based on the requirements of the target use case, including language, content type, and computational constraints (Muenninghoff et. al., 2022).

In this thesis, OpenAI's text-embedding-3-small model was selected due to its proven robustness in RAG workflows, broad generalization capabilities, and integration efficiency with OpenAI's GPT-4 API, thereby minimizing complexity. Moreover, this model is compatible with major vector storage solutions such as Chroma DB and Pinecone, ensuring interoperability across the RAG pipeline.

Once an appropriate embedding model has been selected, an important consideration is how the model will be deployed within the system architecture. The deployment method can significantly impact performance, scalability, cost, and data governance. In practice, embedding models are typically deployed in one of two primary ways, each with its own trade-offs:

- Local Deployment:
  Running the model on dedicated infrastructure provides full control over performance, data privacy, and latency. However, this setup requires significant hardware resources, technical expertise, and ongoing maintenance.

- API-Based Deployment:
  Models hosted by providers like OpenAI or Cohere offer plug-and-play accessibility and scalable infrastructure. These services simplify implementation but often involve recurring costs and may raise data security or compliance concerns, particularly in sensitive domains (Brockhaus, 2024).

Additionally, embedding costs can accumulate quickly in large-scale systems. Most APIs charge based on token usage, making efficient chunking and preprocessing essential. High-dimensional vectors and long documents exacerbate costs, which underscores the need to optimize embedding pipelines for both quality and efficiency (OpenAI, 2025). The choice of an embedding model and deployment strategy has a direct impact on RAG system performance, cost, and scalability. Careful evaluation of available models, infrastructure capacity, and use-case alignment is necessary to ensure optimal outcomes.

2.1.1.3 *Vector storage*

Another foundational component within RAG systems is the vector database. In RAG workflows, vector databases serve as the backbone for storing embeddings generated by the embedding models during the previous step. This process, often referred to as data ingestion

or indexing, involves encoding a corpus into vectors and storing them in the database. They are specifically designed to handle vector search, which will be important in the next step. Vector databases can be managed like traditional databases allowing for the reuse of existing tools, workflows and datasets. This reduces the required expertise associated with ML infrastructure, making it accessible to a broader range of users (AWS, 2024).

Vector databases come in various forms, each catering to different use cases and operational requirements. Notable examples include Chroma DB and Pinecone, which represent two contrasting approaches in the vector storage landscape: open-source, locally hosted solutions and managed, API-based commercial services (Honroth et al., 2024).

Chroma DB is an open-source vector store that can be deployed locally. Its primary advantage lies in its flexibility and cost-effectiveness. Since it runs locally, Chroma DB allows for complete control over data, ensuring privacy and compliance with data protection regulations (Chroma, 2025). This makes it an attractive option for organizations handling sensitive information. Moreover, being open source, it is highly customizable and can be integrated into various workflows without incurring subscription costs. However, the trade-offs include the need for technical expertise to manage and maintain the system, as well as potential scalability challenges when dealing with large datasets.

Pinecone, on the other hand, is a commercial vector store accessible via an API. It provides a fully managed service, offering high scalability, and ease of use via a graphical UI. Pinecone's infrastructure eliminates the need for in-house maintenance, making it a preferred choice for teams lacking the expertise or resources to manage their own vector database. Additionally, Pinecone boasts optimized performance and a wide array of integrations, making it ideal for production-scale applications. However, this convenience comes at a cost, as Pinecone is a subscription-based service. Moreover, reliance on an external API may raise concerns about data security and compliance, particularly in industries with stringent privacy requirements (Pinecone, 2025; Brockhaus, 2024).

Other noteworthy vector stores includes Weaviate Cloud, a managed enterprise-ready vector database known for its hybrid search capabilities that combine vector search with keyword-based retrieval (Weaviate, 2025).

Table 2: Vector Store Comparison

Vector Store	Type	Strenghths	Limitations
Chroma DB	Open-source / Local	Privacy, flexibility, zero cost	Technical overhead, limited scaling
Pinecone	Managed / API	Performance, scalability, integrations, APIs	Cost, external dependency
Weaviate Cloud	Managed / Cloud	Hybrid search, schema support	Cost, external dependency

Source: Own work.

Each vector store offers distinct advantages and disadvantages (Table 2). Open-source solutions like Chroma DB provide flexibility and cost savings but demand higher technical expertise and maintenance effort. Managed services like Pinecone simplify operations and deliver enterprise-level performance but are associated with higher costs and potential dependency on external providers. The choice of vector store should be informed by the RAG system's scale, privacy requirements, budget, and integration environment. Local solutions may offer more control and security, while managed services prioritize speed and ease of use.

2.1.2 Retrieval

The retrieval phase is a pivotal component of any RAG system, serving as the mechanism that bridges the indexed knowledge base and the user's information need. Once the corpus has been transformed into vector representations and stored in a vector database, the system becomes capable of retrieving the most relevant segments of information based on a user query. This retrieval process fundamentally shapes the quality of the final response generated by the language model and is often described as the performance bottleneck of the entire RAG architecture (Huang & Huang, 2024).

When a user submits a query, the system begins by transforming that query into an embedding vector using the same embedding model that was used during the indexing phase. This consistency ensures that both the documents and queries are situated within the same semantic vector space, allowing for meaningful similarity comparisons. The query embedding is then compared against all stored document embeddings using distance metrics such as cosine similarity or Euclidean distance to compute semantic closeness (Minor & Kaucher, 2024; IBM, 2023).

To manage performance in high-dimensional spaces, most RAG systems employ Approximate Nearest Neighbor (ANN) search algorithms, which offer a trade-off between speed and retrieval accuracy (Han et al, 2023). Among these, the *k*-Nearest Neighbor (*k*-NN) approach is particularly popular, returning the top-*k* most similar document chunks based on the similarity score (Sawakar et al., 2024). While increasing *k* can improve contextual richness and reduce the risk of information omission, it may also introduce redundancy, irrelevant data, and higher computational costs (Sawakar et al., 2024; Martineau, 2024).

Vector databases simplify this retrieval process by offering features that go beyond raw *k*-NN indexes, which often require significant expertise to manage. Many vector databases support hybrid search mechanisms, combining vector-based similarity scores with traditional keyword retrieval methods like BM25 (Amati, 2009), to enhance precision. This hybrid approach is particularly valuable in domains like Business Process Management

(BPM), where searches may need to incorporate both semantic meaning and attribute-specific filters such as process IDs.

The retrieval phase is inherently non-generative but determines the informational boundaries within which the language model operates. Its effectiveness depends on several factors:

- Embedding model quality: Poorly encoded queries or documents can distort similarity scores.
- Chunk granularity: Overly fine chunks risk retrieval fragmentation; overly coarse chunks may introduce noise.
- Choice of *k*: Balances information sufficiency with efficiency and output coherence.

Ultimately, the retrieved chunks form the core context for the generation phase, providing the LLM with the evidence necessary to produce grounded, relevant, and domain-specific outputs. Errors in this step, whether through irrelevant retrieval or insufficient context, can significantly reduce the quality and factuality of the generated response (Suzuoki & Hatano, 2024).

2.1.3 Augmentation and Generation

The augmentation and generation phases are the final steps in a Retrieval-Augmented Generation pipeline, where the retrieved information and user query are synthesized to produce an accurate and contextually relevant response. Once the retrieval phase identifies relevant chunks, they are incorporated into a prompt, which is provided to a Large Language Model for response generation. Together, they ensure that the system not only accesses the most relevant data, but also effectively leverages it to produce a meaningful response.

2.1.3.1 *Reranking and Prompting*

Before presenting the retrieved chunks to the LLM, it is essential to refine them further through augmentation techniques. One such technique is reranking, which involves evaluating the retrieved chunks and prioritizing the most relevant ones for inclusion in the prompt (Huang & Huang, 2024). This step helps reduce noise and redundancy, ensuring that the provided context aligns closely with the user query. For example, reranking can leverage additional scoring mechanisms, such as combining semantic similarity with metadata relevance or user-defined priorities. Although reranking is not considered a mandatory component of the RAG architecture, empirical studies have shown that it can significantly enhance relevance, reduce noise, and prevent redundancy, particularly in long-context or multi-document queries (Nie et al., 2021)).

Prompt engineering, by contrast, is an essential and non-optional part of the RAG pipeline. It shapes the interaction between the LLM and the provided context. A well-constructed prompt integrates the user's query with the retrieved chunks in a way that clarifies the task or question for the model (Berandi, 2024). Prompts can be tailored using parameters such as temperature, which controls the model's creativity. A temperature of 0 restricts the LLM to strictly adhere to the retrieved information, ensuring factual accuracy, while higher values allow the model to incorporate broader pre-trained knowledge for more creative outputs. This flexibility makes prompting a powerful tool in RAG workflows. This value also ensures reproducibility (Quinn et al., 2024). In addition to declaring the temperature specific instructions such as: "If you cannot answer the question at hand with the context retrieved, then say I don't know" can further ensure that the RAG exclusively utilizes the retrieved information for the answer generation.

3.1.3.2 *Generation*

The generation phase involves passing the fully constructed prompt to a Large Language Model to generate the final response. Unlike conventional LLM usage where outputs are derived purely from pre-trained parameters, RAG systems augment the model with externally retrieved evidence, ensuring that responses are contextually grounded and up-to-date (Mishra & Gupta, 2025).

This phase leverages the model's ability to reason over the provided context and synthesize coherent, human-like responses (Shuster et al., 2021). When combined with accurate retrieval and effective prompting, the result is a system capable of delivering factual, traceable, and interpretable outputs even for knowledge-intensive tasks (Lewis et al., 2020).

The choice of model should align with the data privacy, budget, and latency requirements of the use case. Self-hosted open-source models allow for on-premises deployment, ideal for regulated environments, while API-hosted models offer ease of integration and scaling. The augmentation and generation stages of RAG systems are where information synthesis occurs. Augmentation techniques such as reranking improve the precision of the information used, while prompt engineering ensures the model interprets the task correctly and operates within defined constraints. Finally, the generation phase transforms the prepared prompt into a response that is not only linguistically fluent but also semantically aligned with the user's query. Together, these stages complete the RAG workflow.

2.2 Example of a RAG Workflow

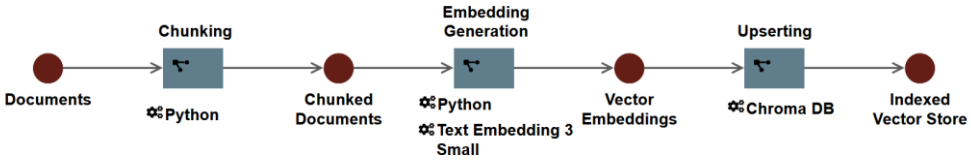
To illustrate how a Retrieval-Augmented Generation (RAG) system functions in a business context, this section presents a practical example based on a typical enterprise scenario. The workflow is divided into two main perspectives. The first part focuses on the data ingestion

phase, representing the responsibilities of a data steward or process engineer, who is tasked with preparing and integrating relevant internal process data into the system. The second part shifts to the generation phase, taking the perspective of a customer service agent seeking an answer to a query related to the customer complaints process. This dual-perspective approach reflects how RAG systems operate across both backend setup and real-time user interaction.

2.2.1 Indexing

As described in the initial part within this chapter an index (vector store) containing the information from the internal documents is required. Figure 3 demonstrates this process at a high level.

Figure 3: Indexing model



Source: Own work.

1. Documents: The company's internal documents are provided. (This process might involve a data preparation step getting rid of formatting issues to make the data fit for the utilized vector storage.)

2. Chunking**:** The system splits the documents into smaller, manageable chunks based on predefined rules (e.g. paragraphs, sections, number of characters etc.)

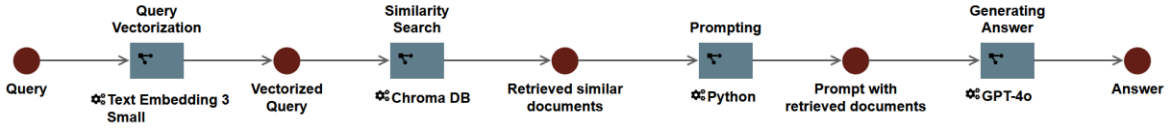
3. Embedding Generation: The system utilizes an embedding model (like text-embedding-3-small) to convert each chunk into a vector embedding for further processing.

4. Upserting: The generated vectors are stored within the vector database (e.g. Chroma DB).

2.2.2 Generation

After the index has been created the system is functional from a technical standpoint and thus ready for the query to answer model as demonstrated in Figure 4.

Figure 4: Query to answer model



Source: Own work.

1. User Query: "What are the key steps in resolving customer complaints in our support workflow?"

2. Query Vectorization: In order to compare the query with the entries within the vector DB the system vectorizes the user's query with the same embedding model employed in the ingestion phase.

3. Similarity Search/Retrieval: The system searches the vector database, retrieves *k* number of chunks related to customer support workflows, and ranks them by relevance. The chunks might include:

   - A description of the complaint escalation process.
   - Documentation on team roles in issue resolution.
   - A workflow diagram of the customer support process.

4. Prompting: The query and top-ranked chunks are synthesized into a structured prompt. The prompt might read as follows:

   "You are an assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. If you don't know the answer, say that you don't know.

5. Generation: The LLM generates a structured response such as:

   "The key steps in resolving customer complaints are as follows:

   1. Log the complaint into the system and assign it a priority level.

   2. Escalate the issue to the relevant team based on predefined escalation rules.

   3. Investigate the complaint by gathering additional information from the customer and internal logs.

   4. Resolve the issue by implementing the necessary corrective actions.

   5. Communicate the resolution to the customer and document the process for future reference."

2.3 RAG Merits and drawbacks

2.3.1 RAG Merits

Retrieval-Augmented Generation architectures have emerged as a compelling solution in the development of intelligent systems, offering a range of advantages that address key limitations of traditional language models. One of the most notable merits of RAG is its efficiency in computational resource utilization. Unlike conventional approaches such as fine-tuning large language models, which typically require access to vast datasets and considerable compute power, RAG architectures decouple knowledge from the model itself. By leveraging external knowledge sources, these systems avoid the need for repeated retraining, thereby enabling faster deployment and reduced operational costs (Siriwardhana et al., 2023; Radeva et al., 2024). This architectural efficiency lowers the barrier to entry for organizations wishing to integrate AI-driven solutions into existing workflows, especially in resource-constrained environments.

Beyond resource efficiency, RAG contributes significantly to the transparency and controllability of generative AI systems. Traditional LLMs function as opaque "black boxes," making it difficult for end users to understand or verify how specific outputs were derived (Arrieta et al., 2019). In contrast, RAG systems incorporate a retrieval mechanism that grounds responses in identifiable, often human-readable documents from a curated knowledge base. This source attribution enhances the explainability of outputs and supports user trust (Lee & Kim, 2024), especially in domains where traceability and accountability are crucial. The capacity to audit the origin of a generated response adds an interpretability layer that aligns with emerging ethical and regulatory standards for AI systems (Doshi-Velez & Kim, 2017; Cai & Li; 2024).

Another key advantage of RAG systems is their adaptability to evolving information. Because the retriever component queries an external database, the system can be updated in real time or on a periodic basis without modifying the language model itself. This enables RAG systems to remain current with dynamic knowledge, a feature particularly useful in rapidly evolving fields such as Business Process Management, finance, or scientific research. The modular architecture also facilitates domain adaptation. Organizations can tailor the knowledge base to specific domains, business rules, or compliance requirements, ensuring outputs remain both relevant and authoritative (Pi, 2024; Cai & Li, 2024).

Moreover, RAG supports modular design and interoperability, allowing developers to integrate various retrievers, embedding models, and vector databases based on performance or infrastructure constraints. This interoperability, particularly when implemented via frameworks like LangChain, promotes scalability and flexibility in production environments (LangChain, 2025).

Finally, by enhancing the factuality of language model outputs, RAG also reduces hallucination, one of the most persistent problems in generative AI (Kau, 2024). Since the generation phase is grounded in retrieved documents, the likelihood of generating fabricated or factually incorrect statements is significantly lower compared to traditional zero-shot generation models (Brown et al., 2020).

2.3.2 RAG drawbacks

While Retrieval-Augmented Generation offers notable advantages, it also presents several limitations and challenges that must be considered in practical applications. A primary concern is the retrieval precision. The performance of a RAG system heavily depends on the quality of the retrieved documents. In cases where the retriever selects semantically similar but contextually irrelevant chunks, the language model may generate misleading or incorrect responses even when relevant documents exist within the knowledge base. This highlights a critical bottleneck: the retrieval step often ends up being the main factor that limits the system performance (Gao et al., 2023). Incorrect retrieval can result in incomplete or misinformed outputs, which undermines the value of generation, particularly in high-stakes domains such as healthcare or finance (Suzuoki & Hatano, 2024).

Another important drawback is the persistent risk of hallucination. Although RAG systems are designed to reduce hallucinations by grounding responses in retrieved documents, this effect is not guaranteed. If the input data is of poor quality, outdated, or lacks adequate coverage of the user's question, the model may revert to its pre-trained knowledge and fabricate plausible-sounding but incorrect content (Shuster et al., 2021). Moreover, if the retrieved content is ambiguous or contradictory, the generation process may synthesize inaccurate conclusions, further eroding trust in the system.

RAG systems also face challenges with redundancy and content overlap. The retrieval process, especially when a high number of chunks (k) is used or when documents are not well-structured, often results in multiple overlapping or nearly identical segments being included in the prompt. While redundancy can sometimes enrich contextual understanding, it frequently introduces inefficiencies, increases prompt length, and reduces the diversity of contextual inputs (Gao et al., 2023).

Latency and computational overhead represent another significant challenge. The RAG pipeline introduces multiple sequential steps such as embedding the query, retrieving from the vector store, reranking results, constructing prompts, and generating responses. Each of these stages incurs processing time and may introduce bottlenecks, particularly in real-time or low-latency applications. Studies have shown that while retrieval-augmented architectures improve factual consistency, they often increase inference time compared to

purely generative systems (Shen et al., 2024). This latency-performance trade-off must be carefully managed, especially in user-facing systems.

Another notable concern pertains to data privacy and ethical risks. Since RAG systems often ingest and process sensitive company-internal data, some of which may be passed to external LLM APIs, there is an inherent risk of data leakage, misuse, or violation of privacy regulations such as GDPR. Without robust safeguards, auditability, and access controls, RAG systems may inadvertently expose confidential or personally identifiable information (Akter, 2024).

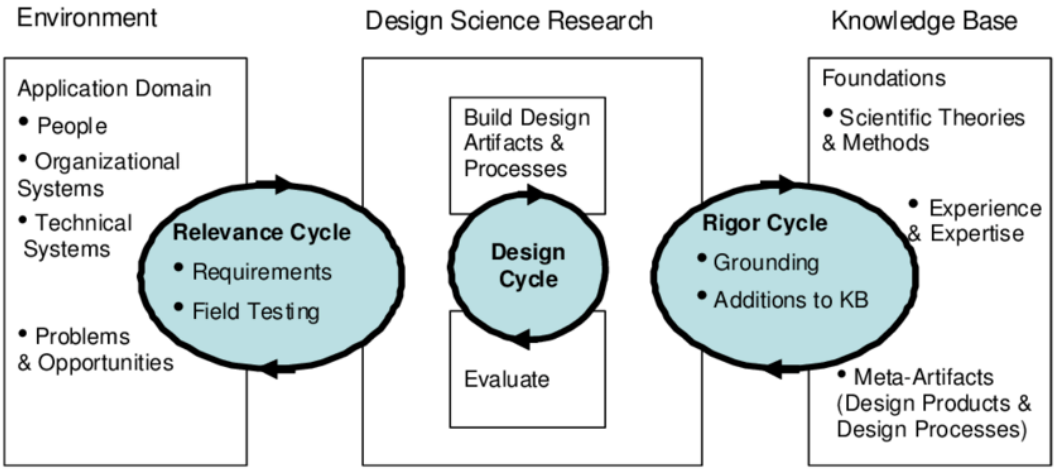
Finally, it is important to emphasize that RAG remains a rapidly evolving technology. Many components such as hybrid retrieval, few-shot generation, and domain-specific embedding models are still under active development (Volynets, 2024). This evolving nature means that system stability, long-term maintenance, and cross-platform interoperability may currently be limited. Organizations adopting RAG must therefore approach implementation with an awareness of current constraints and a readiness to iterate based on emerging research and infrastructure improvements. RAG's balance of efficiency, adaptability, and transparency makes it a powerful tool, but its successful implementation depends on addressing these drawbacks and tailoring the system to the specific needs of the application.

3 RESEARCH METHODOLOGY

3.1 Selection of Research Design and Methodology

The research design for this thesis follows an applied, experimental approach aimed at demonstrating the feasibility and effectiveness of implementing Retrieval-Augmented Generation within the context of AI-driven process modeling. The study is rooted in design science research (DSR), emphasizing the iterative development and evaluation of a proof-of-concept system. DSR is well-suited for research aimed at creating and evaluating solutions that address identified organizational problems (Hevner et al., 2004). Given the applied nature of this thesis, this methodological approach aligns with the thesis's objective to explore the integration of RAG to bridge external data sources with AI functionalities in the Horus platform.

Figure 5: Design Science Research Cycles



Source: Hevner (2007).

Design science research (Figure 5) is characterized by the iterative development of artifacts, including models, methods, or systems, followed by their validation in real-world or simulated environments (Peffers et al., 2006). This work follows a typical DSR cycle: it begins with a problem identification phase (i.e. the limited ability of current AI systems like Horus to access and reason over internal process data), continues with the design and implementation of a RAG-based architecture, and concludes with the evaluation of its performance using defined metrics. This artifact-oriented approach ensures that the resulting system is both technically feasible and aligned with practical requirements in enterprise environments. To this end, this study employs a mixed-methods approach, combining quantitative and qualitative techniques to comprehensively address the research questions. The experimental aspect of the methodology involves system design and implementation, focusing on the integration of indexing, retrieval, augmentation, and generation components within the RAG framework. Quantitative evaluations are conducted to assess system performance, including retrieval accuracy, response latency, and the contextual relevance of generated outputs. Qualitative feedback from expert reviews and test scenarios complements these metrics to provide a holistic understanding of the system's capabilities (Venkatesh et al., 2013).

Data collection is twofold: (1) obtaining representative, (semi-)structured company-internal data to serve as the basis for indexing, and (2) designing realistic query scenarios to evaluate retrieval and generation tasks. The dataset selection ensures relevance to real-world business processes and supports comprehensive testing of the RAG system's ability to handle various content types and complexities.

3.2 Formulation of Research Questions

In order to establish a clear aim and direction of the study a set of precise research questions were established (Ghauri et al., 2020) that answer the question whether it is feasible to utilize RAG in the context of BPM. In order to answer this question, it has been further broken down into a subset of questions.

1. Analysis of the current AI modeler

- What does the current process of model generation look like at Horus?
- What are the current shortcomings?

2. RAG as potential solution

- How can Retrieval-Augmented Generation be implemented to retrieve and integrate internal process data into AI-driven process model generation?
- What are the technical requirements and challenges in adapting to a RAG system?

3. RAG evaluation

- What metrics can be used to evaluate the performance of the RAG-enhanced system compared to the existing model without external data retrieval?
- Where are the shortcomings of the introduced solution?
- Is it feasible for Horus to implement RAG to enhance the quality of Horus AI or other products?

3.3 Case Description

This research is carried out in cooperation with PROMATIS Software GmbH, a company that develops intelligent solutions for business process management. The focus is on improving Horus AI, PROMATIS's own AI-based assistant that helps users create business process models from natural language descriptions. While Horus AI already offers useful support for modeling, it currently has one major limitation: it does not have access to internal company data such as existing process documentation, structured metadata, or relationships between different processes.

Due to these limitations, the models generated by Horus AI may lack important context or contain general information that does not match the specific way a company operates. Valuable knowledge stored in written process descriptions remains unused. In addition, there is no way to trace the source of the AI-generated suggestions, which is a problem in enterprise settings where transparency and auditability are important.

To solve these issues, PROMATIS is exploring whether Retrieval-Augmented Generation could be used to enhance Horus AI. RAG allows generative AI systems to access external knowledge such as internal process data while generating their responses. The aim of this research is to test whether this approach works in practice and whether it is worth investing in further development.

For this purpose, a test dataset was created containing 44 real-world business process models from the PROMATIS repository. These models cover a range of domains, such as cash management, accounts receivable, and logistics. Each model includes additional information like descriptions and titles. This dataset forms the knowledge base for the RAG prototype developed in this thesis.

To evaluate the system, it will be tested using a set of natural language questions. Some of these questions are designed so that the correct answers are included in the dataset, while others refer to information that is not available in the data. This setup makes it possible to assess how well the system retrieves correct answers and how it handles situations where no answer should be found.

It is important to note that this RAG prototype is developed separately from the actual Horus AI tool. The goal is not direct integration but rather to show whether a RAG-based approach could generate useful and reliable results when working with real company data.

3.4 Data Collection Methods

The data collection process for this paper is tailored to the dual objectives of system implementation and evaluation. The primary focus is on acquiring representative datasets that mirror the typical input data encountered in business process management scenarios. This involves sourcing semi-structured process descriptions. Additionally, curated scenarios and query prompts are developed to simulate user interactions, ensuring the retrieval and generation components of the RAG system are thoroughly tested.

To ensure the relevance and quality of the data, stakeholder input from domain experts is incorporated during dataset selection and query design. This collaborative approach ensures the collected data aligns with practical use cases while capturing domain-specific subtleties (Wieringa, 2014). The resulting datasets form a robust foundation for evaluating the system's retrieval accuracy, contextual relevance, and generative performance, while also supporting iterative refinements of the RAG integration.

The primary data for the experimental part of this research comes from direct testing of the prototype, using real and simulated queries. Frequent meetings and consultations with PROMATIS experts were held to identify system requirements, recognize current limitations, and gather feedback on the prototype. These discussions also contributed internal

documentation, architecture details of the existing Horus AI system, and insights into its practical challenges.

For the theoretical framework, this thesis relies on secondary sources such as scientific papers and widely cited literature on topics like Large Language Models (Vidgof et al., 2023), Retrieval-Augmented Generation (Lewis et al., 2020), and Business Process Management (van der Aalst, 2013). These references form the academic foundation for the research and support the understanding of key technical and conceptual aspects relevant to the study.

3.5 Data Analysis and Interpretation

The analysis focuses on evaluating the system's retrieval and generation phases using established metrics such as retrieval precision, consistency, relevance and time (Han et al., 2023). In addition, the contextual coherence and factual correctness of generated responses are assessed through content analysis. To address system shortcomings, iterative refinements are implemented, based on qualitative insights from domain experts (Wieringa, 2014). This iterative cycle ensures that the system evolves to meet predefined success criteria.

Conclusions are drawn based on the triangulation of quantitative performance metrics and qualitative expert evaluations. This approach ensures a rigorous assessment of the system's viability and effectiveness in integrating external data into AI-driven process modeling. Insights from the study will contribute to the broader understanding of RAG's application in business process management.

Data analysis in this thesis is designed to assess the performance and effectiveness of the Retrieval-Augmented Generation (RAG) system. The analysis process involves both quantitative and qualitative methods to evaluate key metrics such as retrieval accuracy, relevance of retrieved chunks, the quality of the generated outputs and time to generate. With the help of these metrics in conjunction with the results from the output the system can be optimized with regards to output quality alongside latency, token usage and cost-effectiveness.

Qualitative interpretation complements these metrics by involving domain experts to assess the relevance and contextual accuracy of the outputs in business process management scenarios. Expert feedback provides insight into the practical applicability of the RAG system and its ability to meet user expectations. The interpretation of these findings focuses on identifying strengths, limitations, and areas for improvement, ultimately guiding recommendations for future development and application of the RAG system. The performance of the output has been evaluated by the criteria that will be established in chapter 6.2.

3.6 Quality Criteria

Ensuring the quality of this research requires a multidimensional evaluation framework that aligns with both the Design Science Research standards and the nature of the implemented Retrieval-Augmented Generation system. As such, the quality of the work is assessed according to four core dimensions: relevance, rigor, utility, and evaluation validity.

3.6.1 Relevance to Practice

A key quality criterion in DSR is the practical relevance of the developed prototype. This thesis is embedded in a real-world use case provided by PROMATIS Software GmbH and addresses a concrete organizational challenge: the inability of Horus AI to leverage internal process knowledge during AI-driven model generation. The developed RAG-based system is evaluated not only in terms of technical feasibility but also with regard to its utility in real process modeling scenarios. This ensures that the prototype contributes actionable value to enterprise practice which is one of the central goals of design-oriented research (Hevner et al., 2004).

3.6.2 Scientific Rigor

Rigor is maintained through the systematic application of established methods for information retrieval, natural language processing, and system evaluation. The architecture of the RAG system is based on well-documented principles, including dense vector retrieval and prompt augmentation, as described in recent literature (Lewis et al., 2020; Gao et al., 2023). The design process is informed by best practices in embedding strategies, document chunking, and relevance filtering. Furthermore, the thesis grounds its methodology in the DSR framework, following the canonical structure proposed by Peffers et al. (2006).

3.6.3 Prototype Utility and Robustness

The utility of the developed prototype is assessed through its ability to:

- Accurately retrieve contextually relevant internal process documents.
- Improve the quality and relevance of generated process model suggestions.
- Maintain traceability between AI outputs and their source data.

Robustness is evaluated through repeated testing with diverse inputs. Special attention is given to the system's ability to generalize across different modeling contexts and handle incomplete or noisy data, a common issue in real-world enterprise settings (Tu et al., 2018).

3.6.4 Evaluation Validity

The evaluation component combines qualitative expert feedback and quantitative performance metrics. Retrieval quality is assessed using:

- Precision to measure the accuracy of document retrieval.
- Context relevance assessments from domain experts who assess the appropriateness of AI-generated outputs.
- Traceability metrics, i.e., whether each result clearly references its source.

Expert reviews and scenario-based walkthroughs are used to assess face validity and practical applicability (Yu et al., 2025). The iterative evaluation across multiple design cycles further strengthens internal validity by continuously aligning system behavior with user expectations and domain requirements (Wieringa, 2014).

4 BUSINESS PROCESS MANAGEMENT AT HORUS

4.1 Horus Business Modeler

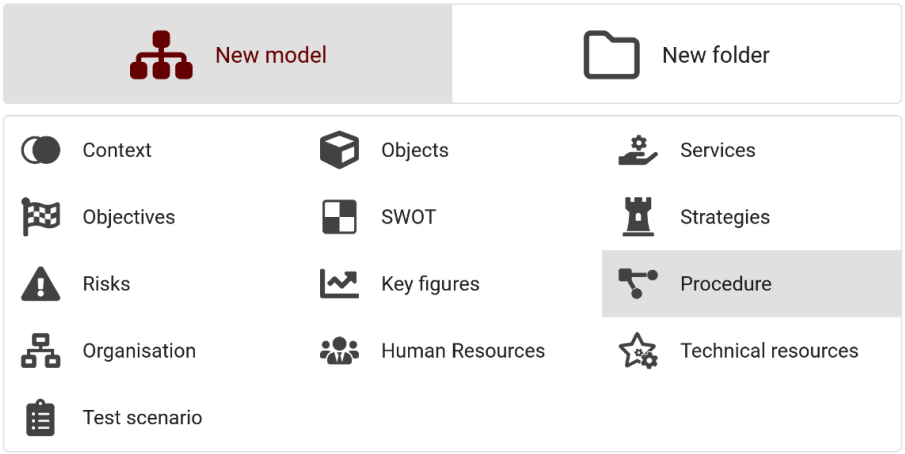
Before touching upon the technicalities of the prototype RAG built for this work, it is worth going into further detail into Horus and how it currently aids users within the realm of Business Process Management.

Horus is a process-oriented modeling platform designed to support collaborative business process management through intelligent tooling, semantic modeling, and knowledge integration. The platform facilitates the visualization, documentation, analysis, and optimization of business processes within a centralized environment. It is particularly suited for enterprises seeking structured digital transformation initiatives and process governance.

The BPM philosophy at Horus emphasizes a knowledge-driven and role-specific modeling approach. Instead of treating process modeling as a purely technical task, Horus incorporates organizational knowledge, user roles, business rules, and contextual dependencies directly into its modeling artifacts (Figure 6). This promotes alignment between operational processes and strategic business objectives. As the scope of this work is limited to the

generation of information around processes, the prototype and testing will only revolve around the process modelling functionality of the Horus Business Suite.

Figure 6: Modeling options within Horus Business Modeler



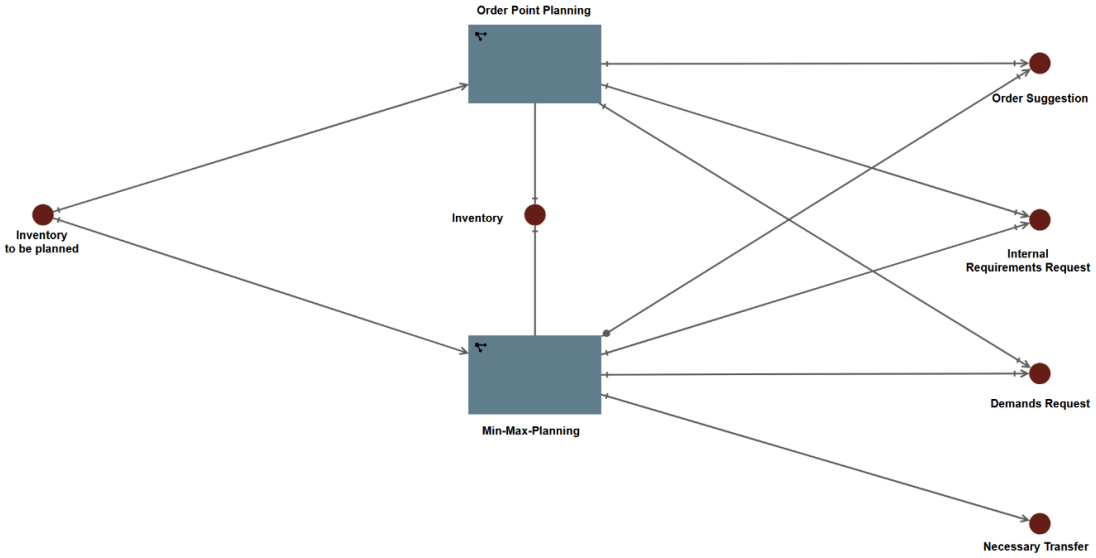
Source: Promatis (2025)

One core functionality is the process modeler, enabling users to model the interaction of technical and human resources to create a coherent picture of the company from a process perspective. These process models include elements found in a classical Petri Net model where "the squares (activities) are the active parts of the model and correspond to tasks. The circles (object stores) are the passive parts of the model and are used to represent states." (Promatis, 2025) And arrows connecting the activities and object stores while indicating the flow of the process. They are comparable to the gateways found in BPMN (Wohed et al., 2006).

To facilitate the creation of high-quality process models, Horus includes a variety of sample processes developed by PROMATIS experts over time. These are collected within the BP4Apps, a repository of pre-built process models spanning common enterprise processes like stocktaking, invoicing, and procurement.

Each model in BP4Apps is labeled by depth using a hierarchical notation such as L1, L2, etc., with L1 representing the highest level of abstraction. Figure 7 illustrates an L1 process model for production planning (BP-PL-L1).

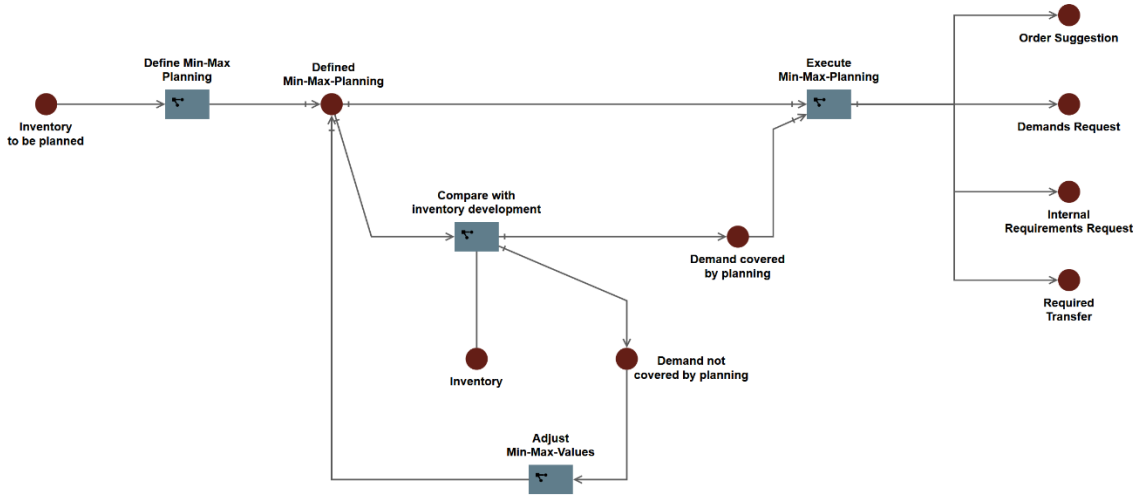
Figure 7: High level production planning process model



Source: Horus Web Application (2025)

The plus sign within an activity indicates the ability to drill down into a subprocess (L2) as shown in Figure 8, and this can continue until reaching the deepest layer: User Instructions (UI). These UI-level artifacts contain details about how specific tasks are executed within Oracle Cloud Applications (OCA), an enterprise application suite used for functions such as finance, HR, and supply chain (Oracle, 2025).

Figure 8: Level 2 min-max planning Process Model



Source: Horus Web Application (2025)

In addition to blueprint models, Horus offers features that enhance workflow, such as collaborative process modeling, enabling stakeholder co-creation in real time. Recently,

Horus has also introduced AI-aided model generation, using OpenAI's GPT-4o to rapidly generate new process models.

However, this AI module still faces limitations. Although the existing BP4Apps models can serve as a base, customization is often required, especially for legal or industry-specific compliance needs. Currently, users can manually adjust generated models, but the AI tool lacks iterative or chained prompting, meaning outputs cannot be refined through follow-up prompts. As previously mentioned, the current system cannot reference existing internal information or prior models. Consequently, Horus AI may produce generic or incomplete process models that don't align with enterprise realities. These shortcomings highlight the need to integrate a Retrieval-Augmented Generation (RAG) approach. By enabling the assistant to query and reason over internal process data, the generation process becomes grounded in real, organization-specific knowledge, a key improvement for increasing relevance, traceability, and user trust (Lee & Kim, 2023).

4.2 Aim of the Solution

The aim is to provide modelling support for the process designers by retrieving existing models and to provide a method to enhance them. By leveraging Retrieval-Augmented Generation, the solution aims to make internal process knowledge more accessible and reusable, particularly in early-stage modeling. One of the key advantages of such a system is the potential to transfer process models across domains, allowing organizations to adapt and reuse existing models in new contexts. This could enable greater standardization and accelerate the development of process models in related business areas.

The application is designed to be accessible to users from a range of professional backgrounds. While technical staff and experienced process designers can benefit from quicker access to relevant process assets, the solution is especially valuable for non-technical users, such as employees from departments like HR or customer service, who may have limited experience with formal process modeling (Friedrich, 2010). By allowing these users to interact with the system using simple natural language instructions, the tool lowers the barrier to understanding and working with complex models.

The added textual explanations in conjunction with the provided graphs add additional information that can greatly increase understandability. Increased clarity for end users might also have a positive effect on adherence to process models. In addition to that, the development of such features is directly aligned with customer feedback received by PROMATIS, reflecting a broader industry demand for AI-driven tools that simplify access to process knowledge. As organizations increasingly look for intelligent and user-friendly automation solutions, the implementation of a RAG-based assistant positions Horus to address this growing expectation in the business process management space.

4.3 Solution Idea

The proposed solution builds on the idea of using existing and validated business process models as the foundational knowledge base for a Retrieval-Augmented Generation (RAG) system. By leveraging these models and their accompanying documentation, the goal is to retrieve relevant process information and enable the generation of new, contextually grounded content. A successful proof of concept will demonstrate the technical feasibility of this approach within the domain of business process modeling and may serve as the basis for future system enhancements or full-scale integration.

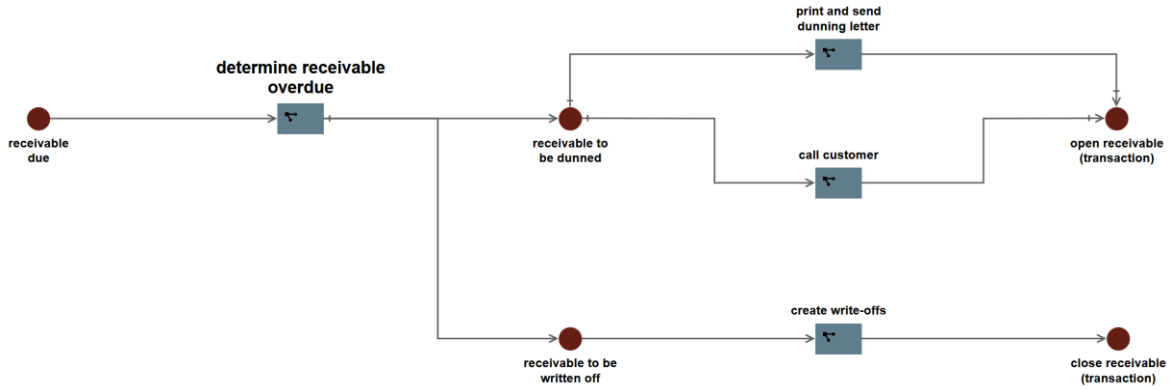
Several approaches for making process data accessible to AI systems have been explored in the literature. One approach, suggested by Minor and Kaucher (2024) and Hu et al. (2024) under the name Graphical RAG (GRAG), involves using screenshots of process models as the knowledge base. While an interesting field of research, this approach is impractical in this context due to the limited capabilities of large language models in working with image-based inputs. Furthermore, the most critical information in Horus models such as task instructions and organizational logic is hidden in the "description tab", which is verbal in nature. While this information is visible in the Horus web application, it is absent from the graphical model, making screenshots or pure visual representations unfit for the intended purpose. For example, Figure 9 shows the dunning process model. Although it outlines the activities involved, it lacks the detailed guidance available in the written process manual. In this example when clicking on "determine receivable overdue" activity in the Horus Web Application the description appears and reads as follows:

"A list is created that contains either all overdue receivables (e.g. invoices, credit memos) or all customers with overdue receivables.

This default list can be used to check which customers should receive notifications. In some cases, customers or certain transactions should be manually excluded from the dunning process. "

This information is entirely missing from the graphical model, which validates the decision to reject the graphical approach in favor of a text-based solution. Early trials also showed that parsing highly detailed Petri Net models was not mature enough to yield satisfactory results.

Figure 9: BP4 Apps sample dunning process model



Source: Horus Web Application (2025)

In contrast to the graphical solution, the prototype adopts a text-based approach, better aligned with the strengths of current generative models. Here, process knowledge is extracted in its verbalized form, including the rich information contained in the "description tab". Horus's built-in functionality for generating process manuals is used to compile comprehensive textual descriptions of each modeled process. These verbalized process models form the documents used to populate the vector database in the RAG system. In doing so, the prototype ensures that relevant information is not only preserved but also made accessible for retrieval and language-based reasoning. The example of the dunning process illustrates this. While the graphical view outlines the flow of activities, the accompanying textual output provides the depth and clarity required for meaningful AI-assisted interaction.

Dunning Process Example:

1.19 BP - SH2CA – L2 – Dunning

Print and send reminder letters

Reminder letters inform customers about overdue invoices, direct debits and chargebacks. A final list of customers to be notified is created and the corresponding reminder letters are prepared for printing and dispatch.

Determine overdue receivables

A list is created containing either all overdue receivables (e.g. invoices, credit notes) or all customers with overdue receivables.

This default list can be used to check which customers should receive notifications. In some cases, customers or certain transactions should be manually excluded from the dunning process.

Call customer

Instead of receiving a notification, the customer is called by an accounts receivable accountant. The accountant should obtain a general overview of the customer account beforehand.

Create write-offs

Write-offs are made when a receivable is no longer recoverable.

5 PROTOTYPE RAG DEVELOPMENT

As the vector DB, chunking method, vectorization model and LLM have been identified as core components, this part of the work will explain the choices made for the prototype solution, elaborate on implementation design, state core functionalities and address challenges and potential solutions.

5.1 Functionalities

The functionality of the implemented Retrieval-Augmented Generation system is designed to enhance Business Process Management by enabling intelligent querying and context-aware response generation. By integrating structured and semi-structured business process data with a retrieval-augmented approach, the system ensures accurate and relevant outputs for BPM-related queries. This section outlines the core functional components of the RAG system, detailing its workflow from data ingestion to response generation. These functionalities were defined in consultation with experts and internal stakeholders at PROMATIS in order to set specific goals as it is common practice in project management (Sommerville, 2015). The agreed upon functionalities of the RAG partially derived from Gao et al. (2023) are as follows:

5.1.1 Data Provision and Indexing

The foundation of the RAG system lies in the indexing of business process documents. The system must be able to take in data; in this case internal process data primarily from textual process documentation from the BP4Apps.

The extracted data undergoes preprocessing, which includes text normalization and chunking to optimize retrieval efficiency. Once processed, the system must convert these

text segments into vector representations using an embedding model and store them in a vector database.

5.1.2 Query Processing and Retrieval

When a user submits a query, the system must be able to transform it into an embedding vector using the same model employed during data indexing. This ensures consistent semantic alignment between queries and documents. This vector is then used to search the vector database, retrieving the most relevant document chunks based on semantic similarity. The retrieval mechanism employs *k*-Nearest Neighbor search to ensure fast and efficient identification of relevant content.

5.1.3 Contextual Augmentation and Prompt Construction

After retrieval, the system must be able to construct a structured prompt by combining the original user query with the retrieved document chunks. This step is critical to ensure that the Large Language Model generates responses grounded in the most relevant business process information. Prompt optimization techniques can be applied to further ensure clarity and prevent redundant information from being passed to the generation phase.

5.1.4 Response Generation and Output Refinement

Once the prompt is finalized, it must pass the information to the LLM for response generation. The model synthesizes the retrieved information and formulates a structured response that directly addresses the user's query. The output can take various forms, including textual explanations, step-by-step process descriptions, or structured workflow summaries.

In order to prove that the output is generated by the referenced data in the vector storage the output must contain links to the Horus Models that are referenced. This functionality will also prove to be beneficial for users to legitimize the output and get a link to the according model which can improve the discoverability of models and streamline search.

5.1.5 Multilingual Capability and Knowledge-Based Answering

A key functionality of the RAG system is its ability to process and generate responses in multiple languages, ensuring accessibility for diverse users. Given that Horus is primarily

used by English- and German-speaking stakeholders, the system must accurately handle queries and retrieve relevant information in both languages. This is achieved by utilizing a multilingual embedding model capable of generating language-agnostic vector representations, allowing for cross-lingual retrieval. Additionally, the system incorporates language detection and automatic translation mechanisms where necessary, ensuring seamless interaction regardless of the language in which the query is formulated.

5.1.6 Knowledge-Based Answering and Query Negation

Another critical functionality is the system's ability to ensure that generated responses are strictly grounded in the information stored within the vector database. To prevent hallucination where the model generates plausible but incorrect or unverifiable responses the system employs a verification mechanism that negates queries if no relevant information is found in the retrieval phase. This is achieved by setting a confidence threshold: if retrieved chunks do not meet a predefined similarity score, the system returns a predefined response such as *"No relevant information found in the available knowledge base."* This approach enhances the credibility of the system, ensuring that responses remain consistent with the provided business process documentation and preventing the generation of misleading or speculative information.

5.1.7 Multi-Vector Storage Capability for Complex Queries

To enhance its ability to handle more complex queries, the RAG system must be capable of retrieving and synthesizing information from multiple vector storage sources. In business process management scenarios, relevant information may be distributed across different repositories, such as workflow documentation, compliance guidelines, and operational procedures. A single vector database may not always contain all the necessary context to answer intricate queries.

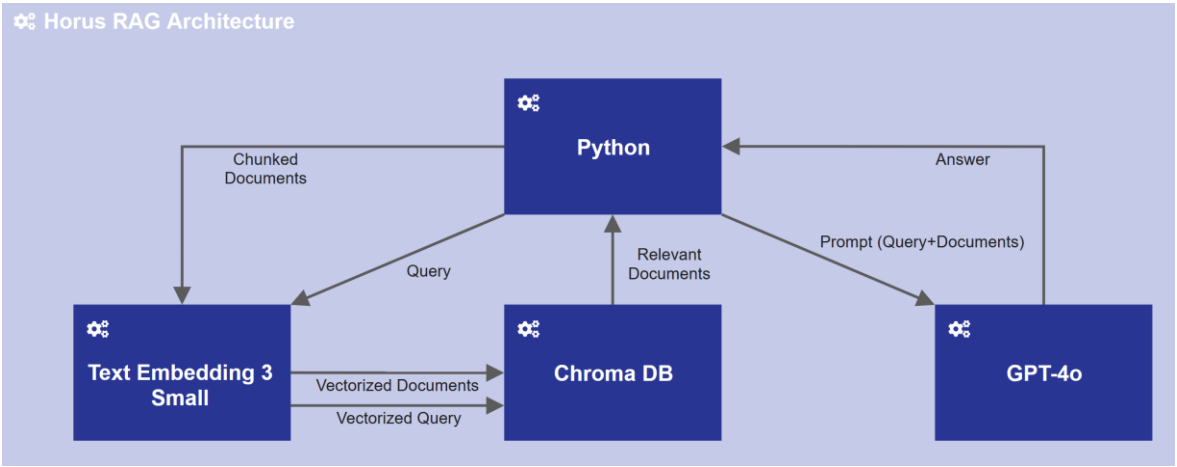
To address this, the system is designed to query multiple vector databases simultaneously, merging results from distinct knowledge domains before constructing a final response. This functionality allows the system to answer multi-faceted questions requiring data from diverse sources. For instance, a user asking, *"What are the compliance requirements for order fulfillment in our company?"* may require information from both an internal process database and an external regulatory compliance database. By integrating multiple retrieval streams, the system can construct comprehensive, well-informed responses, demonstrating its ability to handle complex, multi-source queries effectively.

5.2 RAG design

To address the identified limitations in Horus AI, specifically its inability to incorporate internal enterprise knowledge during model generation, this section presents the design and implementation of a Retrieval-Augmented Generation (RAG) system. The RAG is planned on 4 main components. Figure 10 describes the flow of data between these core components. These are:

- Python based code
- Embedding model: Text Embedding-3 small
- Vector store: Chroma DB
- LLM: GPT-4o

Figure 10: RAG components



Source: Own work.

5.2.1 Python Code

This part of the system acts as the integration layer, connecting all necessary components like user interface, embedding API, vector storage, and generation model. These components typically do not communicate with each other natively, so the Python layer orchestrates data flow throughout the RAG pipeline. It handles:

- The user's input query
- API key authorization for OpenAI's embedding and generation models

- Access and interaction with the vector database
- Configuration of chunk size, and the number of retrieved chunks

The chosen chunk sizes and retrieval count were informed by common practices in RAG literature. For example, Chen et al. (2024) used a chunk size of 300 characters with $k = 30$, which provided fine-grained search at the cost of coherence and performance at scale. In the current project, initial testing showed that BP4Apps process descriptions average ~1500 characters, making a 300-character chunk size too small. To explore the impact of chunk granularity, three vector stores were configured:

- 500 characters (fine granularity)
- 1500 characters (matched to the data average)
- 5000 characters (coarse granularity)

These settings were selected to observe performance under underfed and noisy input conditions mimicking both edge cases and realistic environments in enterprise modeling.

5.2.2 Embedding Model

The choice of the embedding model is core for the output of the model. Vector databases are flexible in the way they are utilized, however some of them are restricted with respect to the embedding model. Since Chroma DB was chosen as the vector database the embedding model must be chosen with potential restrictions in mind (Appendix 4) (Chroma ,2025). OpenAI, with their lines of products being well established in the market, see a wide range adaptation across different vector databases. Alongside text-embeding-3-small, OpenAI offers alternatives such as text-embedding-3-large and text-embedding-ada-002. These models share the same number of maximum inputs of 8191 tokens. The areas they diverge in are their performance and price. While text-embedding-3-large performs better in the Massive Text Embedding Benchmark with a performance of 64.6% it gives the user 9615 pages per dollar. Compared to the 62500 pages that text-embedding-3-small offers with an accuracy of 62.3% the elevated price-performance ratio becomes evident. Thus, the large version should only be utilized in instances where the small model's performance is insufficient. Ada-002 can be disregarded entirely as it offers a lower MTEB score of 61% while being priced at 12500 pages per dollar. So, it is completely inferior to text-embedding-3-small in both relevant categories of pricing and output quality. Table 3 serves as a comparison for all the mentioned embedding models.

Table 3: Comparison of OpenAI Embedding models

Model	Pages per Dollar	Performance on MTEB EVAL	Max Input
text-embedding-3-small	62.500	62,3%	8191
text-embedding-3-large	9.615	64,6%	8191
text-embedding-ada-002	12.500	61,0%	8191

Source: Own work based on OpenAI Documentation (2025)

5.2.3 Chroma DB

During the early development phase of the RAG prototype, Pinecone was initially tested as a potential vector database due to its strong reputation and widespread use in the machine learning community. However, it quickly became apparent that Pinecone presented several challenges that made it unsuitable for the intended use case. Technical compatibility issues hindered the ability to upload and query data reliably, while its cloud-based nature conflicted with PROMATIS's preference for keeping sensitive process information on-premises. Additionally, the need for constant interaction with remote servers and the associated subscription-based pricing model made Pinecone a less viable solution for testing purposes.

As a result, Chroma DB was selected as a more appropriate alternative. Being an open-source and locally deployable vector database, Chroma aligns well with the infrastructure requirements of Horus and the broader goals of the project. It offers a light-weight and developer-friendly environment that integrates smoothly with Python-based machine learning pipelines. Within the RAG architecture, Chroma DB plays a central role by storing vector embeddings of the preprocessed internal documents. When a user submits a query, the system converts it into a vector using the chosen embedding model and performs a semantic similarity search using the k-Nearest Neighbor method. This allows Chroma to retrieve the most relevant content based on the embedded meaning of the query rather than relying on keyword matching alone.

Chroma's ability to operate entirely on local infrastructure ensures that data remains under full organizational control, while also avoiding external dependencies and ongoing licensing costs. It ultimately provides a privacy-conscious solution that supports fast and accurate document retrieval critical for enabling context-aware responses.

5.2.4 GPT-4o

GPT-4o, the latest iteration of OpenAI's language models, introduces significant improvements in reasoning, efficiency, and contextual awareness, making it a strong candidate for enhancing the RAG pipeline. Its advanced token efficiency allows for processing larger chunks of retrieved documents without exceeding token limits, ensuring a more comprehensive understanding of business process data. Moreover, GPT-4o's improved ability to follow retrieval-augmented prompts ensures that responses remain closely aligned with retrieved facts, reducing the risk of hallucinations. The model also exhibits enhanced multilingual capabilities, which is particularly relevant given the goal of generating process data in both English and German. The instructions to generate the output looked as follows:

"You are an assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. If you don't know the answer, say that you don't know."

"\n\nContext:\n" + context + "\n\nQuestion:\n" + question

While this format produced adequate answers, it frequently omitted the source link which is a vital requirement for ensuring traceability in the BPM context. To resolve this, the prompt was extended with few-shot learning, inspired by the findings of Brown et al. (2020) on in-context learning. By including an example with a correctly referenced Horus model link, the system's accuracy in link retrieval improved significantly. The extended prompt format emphasized the necessity of including the link and trained the model to locate and reproduce it based on similar context within the vector database. The extension of the prompt looks as follows:

"If possible, retrieve the link (URL) to the source of the information. It is typically found next to the title of a business process:

BP - PO2RC - L2 - Quality Check
URL: https://promatis.horus.biz/editor?open=3965513&selected=3965513

Open Quality Parameters

To perform the quality check, it is necessary to open the predefined quality parameters.

Check Quality of Goods

To secure the quality of the received goods and services, they need to be compared to the quality parameters.

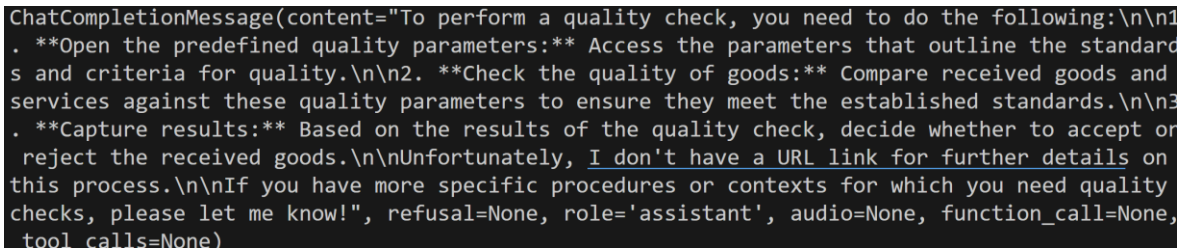
Capture Results

Depending on the result of the quality check, the received goods are accepted or rejected."

Upon questioning both setups with the same question the results of a prompt optimized and unoptimized output look as follows:

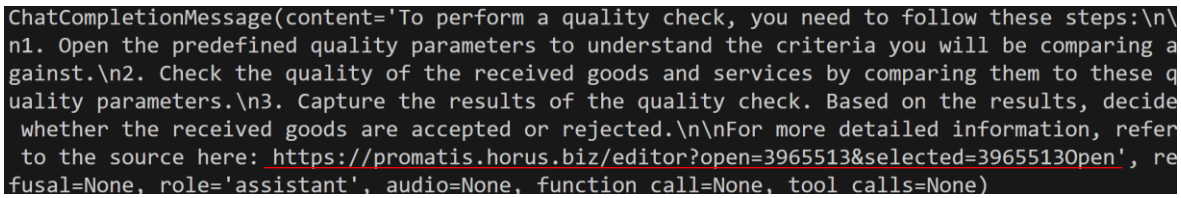
Question: How do I perform a quality check?

Figure 11: Initial Output (no link)



Source: Own work.

Figure 12: Optimized Output (with link)

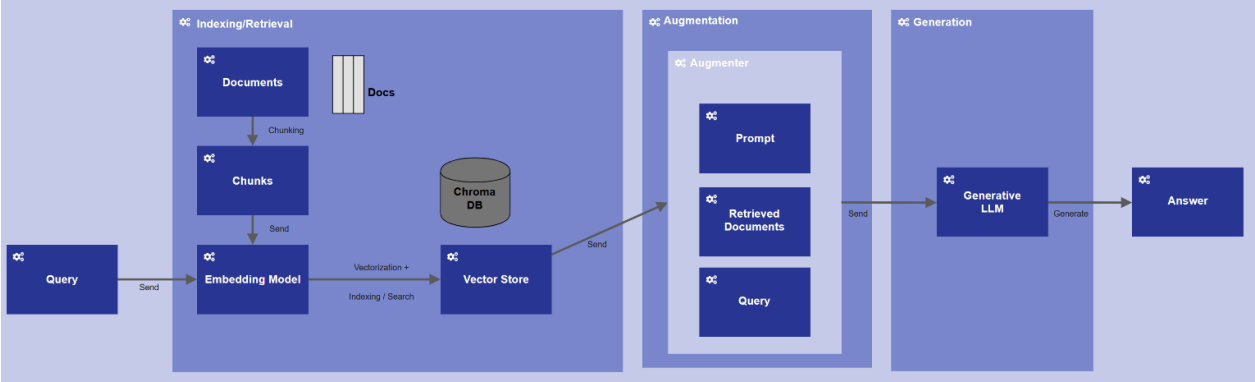


Source: Own work.

A comparison between Figure 11 and Figure 12 shows that the optimized prompt successfully retrieved the link as intended (underlined in red), whereas the initial output does not display the link and even incorrectly denies its existence in Figure 11 (underlined in blue). Other than that, while the output differs in sentence structure, they convey the same information and show no differences in quality. This small example however proves that the output can be optimized via prompt engineering.

Putting all these components together the resulting RAG Pipeline indicated in Figure 13:

Figure 13: RAG Process Diagram

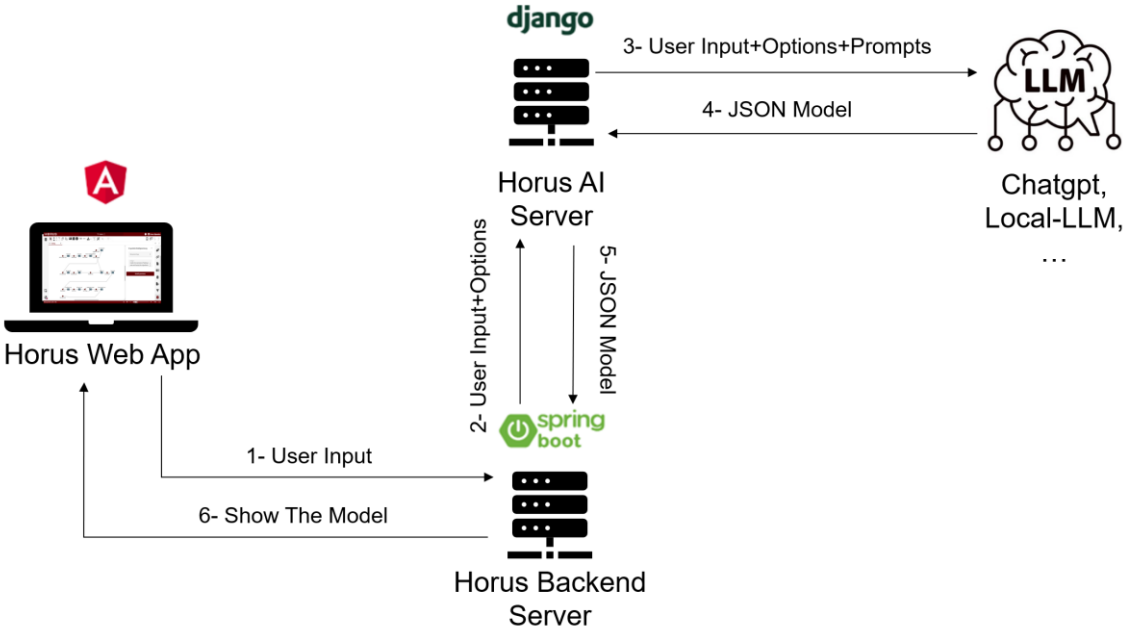


Source: Own work.

5.3 Implementation within Horus AI

After the RAG pipeline has concluded, the resulting response can be passed directly into the Petri Net model generator pipeline of Horus AI. While not being realized within the scope of this project, this integration can enable a fully automated generation of process models inside the Horus Web Application, resulting in a structured and visualized business process output. The workflow follows a multi-step architecture, which incrementally processes user input, performs AI-based enhancement, and finally generates a deployable model. This overall flow is illustrated in Figure 14.

Figure 14: Isolated Horus AI Pipeline



Source: Forell (2024)

The process begins with a user-submitted description within the Horus Web App. This input can be as simple as a process label (e.g., "Order processing workflow") or a more detailed paragraph outlining tasks, actors, and dependencies. The front end captures this textual description and forwards it to the Horus backend server, where additional user-defined options are appended. Currently the system only supports the utilization of an API key to authenticate OpenAI services and the option to choose the preferred model type. Additional customization options are under development.

From here, the input and associated options are forwarded to a dedicated Django-based server, where the system injects the final prompt into the generation pipeline. This is the critical point in the system where prompt engineering techniques are applied to ensure that the LLM understands the expected output format and structure. The prompt states as follows:

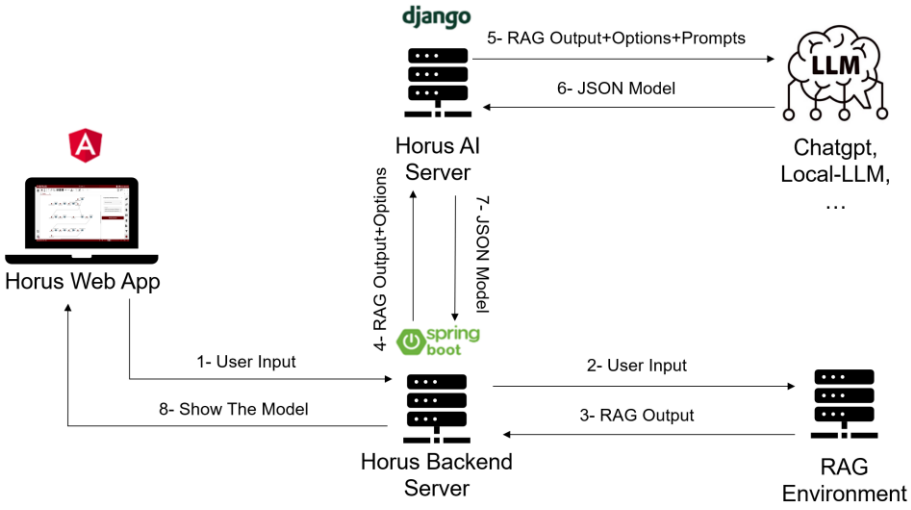
“My task is to extract the essential elements of a Petri net from the provided text. I will not make the mistakes mentioned in the **Examples which are not valid and not allowed to generate: ** section.

Background: I understand that a Petri net graphically represents a system’s structure and behavior, consisting of nodes termed Places and Transitions, connected via directed edges. It’s crucial for every Petri net to be strongly connected, meaning any pair of nodes can be linked by a sequence of directed edges. In Petri nets, concurrency represents simultaneous transitions, indicating parallel processes. I understand that a Petri net always starts with a place and ends with a place. A cycle in a Petri net is a closed sequence of nodes directed at edges where the start and end node is the same, and all nodes and edges in the sequence are distinct (except for the first and last node). This typically involves a sequence of Places and Transitions that repeat, allowing for the representation of recurring processes or feedback loops within the system.” (Promatis, 2025).

By instructing the model using explicit structural constraints, the prompt minimizes the risk of generating invalid or non-standard diagrams. This approach mirrors best practices for reliable LLM use in formal modeling tasks (Ggaliwango et. al., 2024). A complete listing of process rules, valid Petri Net structures, and forbidden constructions is included in the Appendix.

The completed user input, generation options, and structured prompt are submitted to GPT-4o, which returns a JSON-formatted Petri Net model. This response includes all necessary attributes such as Transitions, Places, directed edges, Labels and instructions. This machine-readable JSON output is then parsed by the backend server, which renders the visual model on the Horus Web App interface, enabling users to interact with or refine the model directly.

Figure 15: Horus AI with RAG



Source: Own work based on Forell (2024)

Figure 15 shows the RAG-enhanced system flow. Unlike the traditional setup that routes user input directly to the AI generator, this design introduces a "RAG Loop".

1. The user query enters the loop
2. RAG retrieves relevant internal documentation or reference models
3. The augmented context is passed to the Django prompt server
4. The final prompt and context are sent to GPT-4o
5. The LLM returns a Petri net in JSON, which is forwarded to the backend

This system preserves traceability, incorporates domain-specific process knowledge, and maintains the visual fidelity of generated models while minimizing hallucination risks. The result is a hybrid AI-BPM system that delivers context-aware, standards-compliant, and interpretable business process models (Eldin et al., 2024).

6 TESTING AND EXPERIMENTAL FINDINGS

6.1 Experimental Setup

Chen et. al. (2024) propose 4 capabilities that a good RAG system has. Noise robustness, negative rejection, information integration and counterfactual robustness. These capabilities

for the most part are important to evaluate a RAG system. To coincide with these capabilities a series of structured experiments were conducted. The primary objective was to assess the quality of the output in different scenarios. The experiments focused on key variables, including:

Variation in k (Number of Retrieved Chunks):

Testing different values of k to determine the number of retrieved documents/chunks. This will also determine the system's ability to deal with noise and determine whether it is advised to choose a larger number of retrieved chunks or not. The number of *k* chosen for this experiment are *k* = 5/10/20 as suggested by Sawakar et al. (2024).

Variation in Chunk Size:

Assessing the impact of different chunk sizes on retrieval accuracy and response quality. The chunk sizes chosen are 500/1500/5000 characters. 1500 was chosen as the average number of characters within a given section of the dataset. 500 and 5000 characters were chosen as a comparison for a significantly larger and smaller chunk size.

Prompting Strategies:

Experimenting with different prompt structures to optimize response generation. The system was prompted to retrieve the link of a given section within the sample data. For the unspecified prompt the system was given the task of simply returning a link (if available) from a section that was utilized to generate the output. The specified prompt underlined this task by giving a sample output for the model to reference.

Rejection Testing:

Introducing queries that should not be answerable to the system as the test data lacks the required information to answer this question. This helps to assess the RAG's ability to reject questions lacking relevant context and ensure that the information to for the output is exclusively drawn from the test data. For this test different values for *k* are chosen to observe the behavior when introducing noise.

Latency Measurement:

Measuring response times across different configurations to analyze trade-offs between retrieval depth and speed. This includes the testing of the generation of multiple vector databases with varying chunk size and the generation of output with a small value for *k* and a small size of chunks, a small number of *k* and a large size of chunks, etc.

Language Capabilities:

Assessing the ability of the system to understand and answer questions in different languages. As Horus is mostly used in German and English the tests will only cover these two languages.

The RAG system was tested using 44 business process models, extracted from Horus BP4Apps, ensuring a broad and representative dataset for BPM-related queries. To ensure reliability, all tests were conducted with a temperature (value that determines the "creativity" of the LLM) setting of 0, minimizing hallucinations and ensuring strict adherence to information found in the vector database. The results give us an impression of the capabilities of the system, preferred settings for the implementation and potential shortcomings which will later help us to determine the optimal settings for potential production within Horus AI.

6.2 Evaluation criteria

To assess the performance of the implemented RAG system, a set of evaluation criteria has been established. The selected criteria aim to offer a comprehensive understanding of the system's capabilities, strengths, and limitations across both qualitative and quantitative dimensions. The evaluation will be based on the following four core criteria (Ravi, 2023):

Relevance & Completeness**:**

This criterion measures the extent to which the output adheres to the given prompt and whether it includes all the necessary information required to accurately answer the user's query. For this purpose, a reference list of all relevant information items drawn directly from the indexed documents will be created for each test question. The output will then be compared against this reference list. Any missing information will be documented and attributed to the specific retrieval or generation setup used.

Time:

This metric captures the time taken by the system to generate a response from the moment the query is submitted until the final output is returned. Measuring generation time is essential for assessing the system's responsiveness and viability in real-world applications where low latency may be critical.

Consistency:

This criterion evaluates whether the system produces stable and reproducible outputs when asked the same question multiple times under identical conditions. High variance in results may indicate instability in the retrieval process.

Fluency:

Fluency refers to the linguistic quality of the generated text. This includes the system's ability to understand the user query, maintain coherent sentence structure, and express information in grammatically correct and contextually appropriate language. Fluency is a crucial factor in ensuring the usability and interpretability of the output in professional settings.

6.3 Experimental results

Note: For readability purposes the output was stripped out of system notifications. The parts indicated in bold letters will thus be omitted.

The original system output looks as follows:

ChatCompletionMessage(content='To create an assessment, you can follow these steps**:\n\n1.** Navigate to the Procurement section and proceed to Supplier Qualification.**\n2.** Click on the Task button and select Create Initiative. This opens the Create Initiative window.**\n3.** …

Adapted output:

To create an assessment, you can follow these steps: Navigate to the Procurement section and proceed to Supplier Qualification. Click on the Task button and select Create Initiative. This opens the Create Initiative window. …

The following tests will be evaluated against a predefined list of information items considered essential for answering each question. These items are presented as a numbered list. When a specific piece of information appears in the output, it will be marked with the

corresponding number at the end of the relevant paragraph. For each test variation, any irregularities or unexpected results will be documented. A general assessment summarizing all test variations will be provided at the end. Additionally, the configuration details such as the number of retrieved chunks, chunk size, and generation time will be noted at the beginning of each test case.

6.3.1 Variation of *k* test

Question: How do I create an assessment?

The criteria to determine relevance and completeness:

1. Navigate to Procurement > Supplier Qualification
2. Click Task button > Select Create Initiative > New window opened
3. Select type “assessment” > Enter title > Enter owner > Create
4. Edit initiative window > option to add description/change owner and enter internal note
5. Enter due date > Enter a predefined Qualification model from dropdown
6. Select supplier > Navigate to actions and select add
7. Enter supplier and site > Enter supplier contact as recipient > Give an internal responder
8. Click next to reach the review/configuration page > Optional settings
9. Click launch to send assessment > Yes to continue > Confirmation message

k=5, chunk size 1500, time to generate 9,14 seconds

To create an assessment, you can follow these steps:

Navigate to the Procurement section and proceed to Supplier Qualification. (1) Click on the Task button and select Create Initiative. This opens the Create Initiative window. (2) Select the Type as Assessment. Enter a Title for the assessment and specify the Owner. Click Create (3) to open the Edit Initiative: Overview window. In the Overview Header, you can add a Description, change the Initiative Owner, and enter an Internal Note. (4) In the Assessments section, enter an Evaluation Due Date and select a Qualification Model from the predefined models. (5) After selecting your Qualification Model, the Qualification Areas populate in the section below. To finish the overview, select at least one supplier by clicking Actions and selecting Add, or clicking the Add icon. (6) Enter the Supplier via the search function or by typing it in, then enter the Supplier Site and a Supplier Contact as the assessment recipient. Enter an Internal Responder responsible for answering the internal part of the assessment. (7) Click the Next button to continue to the Configuration page. For further settings, such as the recipient of the questionnaire or the start and end dates of the qualification, you will proceed to the Configuration page. (8) For more details, you can refer to: [URL is not provided here as this is a sample response]

→ This output contains all relevant information with the exception of item number 9. It is fluent and was generated in a timely manner.

k=10, chunk size 1500, time to generate 13,84 seconds

To create an assessment, follow these steps:

Navigate to Procurement: Go to the Supplier Qualification section. (1) Create Initiative: Click on the Task button. Select "Create Initiative." The Create Initiative window will open. (2) Enter Assessment Details: Select the Type: Assessment. Enter a title for the assessment. Enter the Owner of the assessment. Click on "Create." (3) The Edit Initiative: Overview window will open. Add Details in Overview: In the Overview Header, you can add a Description, change the Initiative Owner, and enter an Internal Note. (4) Enter Evaluation Information: In the Assessments section, enter an Evaluation Due Date. Enter a Qualification Model by selecting one from the predefined models in the Drop-Down menu. After selecting your Qualification Model, the Qualification Areas will populate below. (5) Select Suppliers: To finish the overview, you need to select at least one supplier. Navigate to Actions and select Add, or click on the Add icon. (6) Enter the Supplier via the search function or by typing it in. Enter the Supplier Site and Supplier Contact as the recipient of the assessment. Enter an Internal Responder responsible for answering the internal part of the assessment. (7) Click the Next button to continue to the Configuration page. Configure the Assessment: The configuration page shows the selected suppliers and qualifications. Further settings can be adjusted, such as the recipient of the questionnaire or the start and end date of the qualification. (8) Once the configuration is completed, you can proceed with sending or launching the assessment as required. This information can be found in the supplier qualification section under procurement. (9)

→ Unlike the previous output, this setup is more complete as it also contains item number 9. With respect to fluency, it reads much more like a list. While not taking significantly longer the generation of this answer took more time than the first test with *k*=5.

k=20, chunk size 1500, attempt 1, 11 seconds

To create an assessment, follow these steps:

Navigate to the Procurement menu, then select Supplier Qualification. (1) Click on the Task button and select "Create Initiative.". The Create Initiative window will open. (2) Here, you should: Select the Type as Assessment. Enter a Title for your assessment. Enter the Owner of the assessment. Click "Create" to proceed (3), which will open the Edit Initiative: Overview window. In the Overview Header, you can add a Description, change the Initiative Owner, and enter an Internal Note. (4) In the Assessments section, you can set an Evaluation

Due Date. Also: Enter a Qualification Model by selecting one from the predefined models. (6) To finalize the overview, you need to select at least one supplier by navigating to Actions and selecting Add. Alternatively, you can click on the Add icon. Use the search function or type to enter the Supplier, then enter the Supplier Site and Supplier Contact as the recipient of the assessment. Enter an Internal Responder responsible for answering the internal part of the assessment. (7) Click "Next" to continue to the Configuration page. On the Configuration page, you will find the selected suppliers and qualifications. It also allows you to view additional settings, such as the questionnaire recipient and the start and end dates of the qualification.(8)
Source:https://promatis.horus.biz/editor?open=3965513&selected=3965513'

→ This output reads more fluently akin to *k*=5, however it contains a link which was not part of the required section and thus should not have been generated. So, it used the correct piece of information from the database but provided the wrong link. In addition to that, the generation of this output of *k*= 20 was expected to take longer than *k*=10 as *k*=10 took longer than *k*=5. However, this was not the case. This output also does not contain any information on item number 9.

k=20, chunk size 1500, attempt 2, 11,84 seconds

To create an assessment, follow these steps:

Navigate to the Procurement > Supplier Qualification menu. (1) Click on the Task button. Select "Create Initiative." In the Create Initiative window (2), select the Type as "Assessment." Enter a Title for the assessment. Enter the Owner of the assessment. Click "Create." (3) The Edit Initiative: Overview window opens, where you can proceed with the following: In the Overview Header, add a Description, change the Initiative Owner, and enter an Internal Note if necessary. (4) In the Assessments section, enter an Evaluation Due Date. Enter a Qualification Model by clicking on the Drop-Down button and selecting one of the predefined models. This will populate the Qualification Areas in the section below. (5) To finish the overview, select at least one supplier: Navigate to Actions, select Add, or click on the Add icon. (6) Enter the Supplier via the search function or by typing it in. Enter the Supplier Site, Supplier Contact as the recipient of the assessment, and an Internal Responder responsible for answering the internal part of the assessment. (7) Click the Next button to continue to the Configuration page. On the Configuration page, you can view the selected suppliers and qualifications. Further settings, such as the recipient of the questionnaire or the start date and end date of the qualification, can be set. (8) After setting up the Configuration and other required fields, you can proceed to create questionnaires and, eventually, send the assessment. (9)

→ Upon repetition of *k*=20 the quality of the output stood the same. Unlike the first attempt it also contained information item number 9 and did not retrieve any incorrect link.

Evaluation:

The result adhered to the given prompt and included every piece of information that was found in the vector database. The generated text also did not display any grammatical errors. The system took around 11.45s on average to answer the questions for all variations of *k*. Upon repeated testing for the same question the sentences differed in their structure however they for the most part displayed every required information. The system struggled with the retrieval of information item number 9. This might have been connected to the chunk size as the new chunk might start at a disadvantageous position rendering the crucial part difficult to capture. Another possibility might be the structuring of the initial document as the system might connect it to another process.

The system also displayed inconsistencies regarding the link. The section in the vector database regarding the creation of an assessment does not contain a link. Therefore, the output should not contain one. Test scenario *k*=20 however displayed an incorrect link which belonged to the section for the "quality check" process within the database. Upon a repetition, however no link was displayed. So, the system at its current form suffers from minor inconsistencies regarding the display of the link. This might be a result from the added noise by increasing *k*.

The output also differs in terms of fluency. The output for *k*=10 for example reads much more like a list while the other examples are proper and fluent texts. Overall *k*=20 offers the best output within this experiment.

6.3.2 Variation of chunk size test

Question: How to check a requisition for approval?

The criteria to determine relevance and completeness:

1. Requisition for approval has been submitted
2. Open Notifications > Navigate to Menu > Open Notification
3. Click Link to corresponding notification
4. Check Requisition > View details on requested goods > Optionally open any attachments and comments
5. Optionally view further information on documents like history
6. Decision on further proceedings >Approval/ Rejection/ Escalation/ ad-hoc routing/ Reassignment
7. Link

k=5, chunk size 500, 5,29 seconds

To check a requisition for approval, follow these steps:

A notification is sent to inform the approver about the submitted requisition. (1) Open the notification to check the details of the requisition. (2) The approver has several options: approve, reject, or forward the requisition, or request additional information. (6) For further details, you can refer to the source: [Check Requisition for Approval](https://promatis.horus.bis/check Requisition for Approval). (7)

→ This output displays significant deficits with respect to completeness as it does not contain every piece of information that is required. In addition to that, the flow of the process is unclear. Most likely the question could not be answered properly as the parameters given were too restrictive and did not provide enough information.

k=5, chunk size 1500, 7,36 seconds

To check a requisition for approval, follow these steps:

Open Notification: After a requisition has been submitted for approval (1), the responsible approver will receive a notification about the pending requisition. To check the details of the requisition, the notification needs to be opened. Navigate to the menu and then Notifications. (2) View Notification: Click the notification icon in the menu to open the corresponding notification. (3) Check the Details: In the opened approval notification, you can:

- View all details regarding the requested goods/services and delivery data.

- Open any attachments or read the comments enclosed with the requisition. (4)

Reopen Notification: If you need to, reopen the notification to check the requisition on the notifications page which includes: Details, Requisition Lines, Comments, Attachments, and History of approvals. Optional: If you want to open the requisition details, you can click the link "View Requisition Details" in the section Requisition Lines. (5) For detailed steps and confirmation, you may refer to the [source](https://promatis.horus.bis/check Requisition for Approval) (7)

→ Compared to chunk size = 500 this output is much more complete. But lacks information on decision and further proceeding (item number 6). The output is still complete enough to derive the full process of it. The output took around 2 seconds longer to generate compared to chunk size=500.

k=5, chunk size 5000, 7,69 seconds

To check a requisition for approval, follow these steps:

Open Notification: Navigate to the Menu > Notifications. (2) Click the notification icon in the menu. Click the link of the corresponding notification in the notification overview to open it. (3)

Check Requisition: You can view all details regarding the requested goods/services and delivery data in the opened notification. Optionally, open any attachments or read comments enclosed with the requisition by clicking the link. (4) "View Requisition Details" in the section Requisition Lines: If needed, you can also view the document history by clicking the "View Document History" link. (5)

Decision Making: Once you have reviewed the details, you can decide on how to proceed with the requisition. If you need to escalate the requisition or make any other decisions such as routing it ad-hoc or reassigning the task, use the respective options under the Actions menu on the notifications page. (6) Source: [Check Requisition for Approval](https://promatis.horus.biz/check Requisition for Approval) (7)

→ This version lacks the precondition of a submitted requisition for approval (information item number 1). Other than that, it contains every piece of information required.

Evaluation:

Relevance was given as the output always gave information on the correct topic. This is further proven by the consistently correctly retrieved link. However, none of the outputs were fully complete as they all lacked at least one piece of information. The outputs took 6,78 seconds on average where the initial test with poor results took significantly less time compared to chunk size 1500 and 5000.

Unlike the *variation of k test*, where quality differences were relatively minor, the results here show more pronounced differences in output quality, highlighting the importance of appropriate chunking strategies. The poor performance observed in the initial output suggests that it is preferable to provide the system with more contextual information rather than too little.

6.3.3 Rejection testing

Question: How are behavioral issues with employees resolved?

The criteria to determine relevance and completeness:
Express that the database does not contain any information regarding that subject.

k=5, chunk size 1500, attempt 1, 5,64 seconds

I don't have specific information in the provided context on how behavioral issues with employees are resolved. Typically, this might involve a process of assessment, feedback, and intervention, possibly involving HR policies and procedures, but I recommend consulting your organization's specific guidelines or HR department for accurate information.

k=5, chunk size 1500, attempt 2, 4,83 seconds

I'm sorry, but I don't have information on how behavioral issues with employees are resolved.

k=10, chunk size 1500, attempt 1, 4,77 seconds

I don't have specific information from the context provided about how behavioral issues with employees are resolved. Generally, resolving such issues often involves identifying and addressing the behavior through direct communication, implementing performance improvement plans, offering coaching or counseling, and, if necessary, taking disciplinary actions. However, specific procedures can vary between organizations.

k=10, chunk size 1500, attempt 2, 4,89 seconds

I don't have the information on how behavioral issues with employees are resolved from the provided context. For specific procedures, you may need to refer to your organization's human resources policies or consult a human resources professional.

k=20, chunk size 1500, attempt 1, 5,23 seconds

I don't know. The provided context does not include information on resolving behavioral issues with employees.

k=20, chunk size 1500, attempt 2, 5,76 seconds

Based on the available context, I don't have specific information on resolving behavioral issues with employees. It's generally advisable to contact your HR department for guidance on managing behavioral issues in the workplace according to your organization's policies.

Evaluation:

All tests successfully rejected the presence of the required information. However, unlike required by the prompt to simply reject the availability the system oftentimes gave information that belongs to the LLM and exceeded the data found in the vector database. Upon repeated testing the results varied. *k*=5 and *k*=20 both included one attempt in which the question was rejected without adding additional information while *k*=10 added supplementary information beyond the information available in the vector database. The generation of the outputs took 5,19 seconds on average.

6.3.4 Latency testing

The data for generating output has been noted for each aforementioned test. For the given queries no major difference in time for the generation of the output could be detected. Inconsistencies were detected where the output for a given question took slightly longer although the chosen number or retrieved chunks *k* was lower. In one case the generation of an output took exactly 11 seconds for *k*=20 while the generation of the output for the same question with *k*=10 took 13.8 seconds.

So, no direct advice can be given as no direct correlation between the utilized chunk size/number of retrieved chunks and the time to generate an output could be detected. However, in cases where the quality was seriously influenced by the choice of k and chunk size (*variation of chunk size test* with chunk size=500) quality should be prioritized over latency.

However, the time to upload the test data to the vector database displayed significant differences depending on the chosen chunk size as seen in Table 4.

Table 4: Time to upload by chunk size

Chunk size	**Number of Chunks**	**Time to upload**
500	490	4 min 36s
1500	159	1 min 10s
5000	48	29 s

Source: Own work.

A longer upload time is observed for smaller chunk sizes. This can be explained by the fact that the data is divided into a greater number of smaller segments, each containing less information from the original dataset. As a result, more chunks are generated, and since each chunk must be individually converted into a vector, the overall upload time increases accordingly.

6.3.5 Language testing

Question: How is a purchase order modification performed?

The criteria to determine relevance and completeness:

1. Navigate to Actions > Edit purchase order
2. Notification for change order > Click yes > Edit on change order page
3. Number indicates change orders > Enter reason for change
4. Change all editable fields if necessary > Changed fields are marked
5. Save all changes > Optional: Add attachments
6. Send for approval
7. If frozen or hold > Order must be released > Resubmit
8. Link

k=5, chunk size 1500, 7,43 seconds

A purchase order modification is performed by first navigating to Actions > Edit in the purchase order. (1) Here are the steps involved:

A notification will inform you about the creation of a change order. Click the Yes button to proceed. On the Edit Change Order page, you can edit the change order. (2) The number indicates the amount of existing change orders for the document. Enter the reason for the change order in the Description field. (3) You can change all editable fields, such as the buyer in the purchase order header data or the amount in the purchase order line. Changed fields are marked, and changed amounts are indicated in the change order data (Amount Changed). (4) Save all changes. Additionally, you can add attachments by clicking the Add-Icon in the field Attachments at the header or line level. (5) After editing, the change order is submitted for approval. (6) If a purchase order is frozen or on hold, it may need to be unfrozen or released before changes can be made. Then, after modification, it needs to be resubmitted for approval. (7) Source: [Purchase Order Modification](https://promatis.horus.bis/Purchase Order Modification) (8)

→ This output contains all relevant information as well as the link to the respective Horus model. There are no further issues.

k=5, chunk size 1500, 8,24 seconds

The same question was then asked in German language:

Question: Wie wird eine Bestelländerung durchgeführt?

Um eine Bestelländerung durchzuführen, navigieren Sie zu "Aktionen" und wählen Sie "Bearbeiten" in der Bestellung. (1) Sie werden über die Erstellung einer Änderungsbestellung benachrichtigt. Klicken Sie auf "Ja", um fortzufahren. Auf der Ändern Seite können Sie die Änderungsbestellung bearbeiten. (2) Geben Sie den Grund für die Änderungsbestellung im Feld "Beschreibung" ein. (3) Alle bearbeitbaren Felder können geändert werden, beispielsweise der Käufer in den Bestelldaten oder der Betrag in der Bestellposition. Geänderte Felder werden markiert und geänderte Beträge in den Änderungsdaten angezeigt. (4) Speichern Sie alle Änderungen abschließend. Zusätzlich können Sie Informationen durch Anhängen von Dateien, Texten oder URLs an den Bestellkopf oder die Bestellposition hinzufügen. (5) Quelle: Kaufauftrag URL: [Link nicht bereitgestellt]

→ The output exhibits deficiencies regarding completeness as information regarding item number 8 “sending for approval” and item number 7 “actions for frozen purchase orders” are completely absent in the German output. In addition to that difficulties in the retrieval of the link were detected while the correct link was provided in the English query.

Evaluation:

As the information within the vector storage is saved in English an increased performance compared to the query in German was expected The results demonstrate that the system is capable of handling output in German, even when the stored information is in English. However, some information got lost when querying in German. The system was also not able to retrieve the proper link. If feasible a vector store for each required language should be built. However, working with one English vector storage also provides a solid output in this case. For use cases where high precision is needed creating a database with information in the respective language is advised. The time to create each output also did not differ much from each other with a difference of 0,81 seconds.

7 DISCUSSION

7.1 Lessons learned

The evaluation of the RAG system revealed key insights regarding retrieval accuracy, response generation, system latency, and multilingual capabilities. These findings highlight both strengths and limitations that can inform future optimizations and refinements.

7.1.1 Retrieval and Response Accuracy

The experiments demonstrated that the system successfully retrieved and synthesized relevant information in most cases. However, certain critical information, such as step 9 in the *variation of k test*, was inconsistently included in the responses. This suggests that the chunking strategy and document structure influence retrieval performance. Additionally, while increasing *k* generally improved completeness, it also introduced occasional noise, such as incorrect links, which appeared in some responses when *k=20*. This suggests that a trade-off exists between retrieval precision and the amount of retrieved context. The *variation of chunk size test* further reinforced this, as responses generated with smaller chunks (500 tokens) were significantly less complete, indicating that under-chunking leads to information fragmentation. These findings suggest that optimizing chunking strategies and fine-tuning retrieval parameters are essential for maximizing system accuracy.

7.1.2 Fluency and Output Structure

Variations in *k* also affected output fluency. Lower values of *k* (e.g., 5) generally resulted in more natural, coherent responses, whereas higher values (e.g., 10) tended to produce list-like outputs. This suggests that the LLM adapts to the structure of retrieved documents, meaning that retrieval strategies influence not only information completeness but also readability. This insight is crucial for applications where user experience and clarity are important factors.

7.1.3 Handling of Missing Information and Hallucinations

The *rejection testing* demonstrated that the system correctly identified when requested information was unavailable in the vector database. However, while some responses strictly rejected the query, others supplemented their answer with general knowledge from the LLM's pre-trained data. This inconsistency suggests that while the system successfully prevents hallucinations to an extent, it does not fully suppress generative tendencies when no relevant retrieved information is found. Future improvements should consider stricter prompt engineering or confidence thresholds to ensure that responses adhere strictly to retrieved data.

7.1.4 Latency and System Efficiency

The l*atency testing* showed no direct correlation between retrieval parameters (chunk size, *k*) and response time. While *k=20* was expected to take longer than *k=10*, this was not

consistently the case. However, the data ingestion step was significantly affected by chunk size. Smaller chunk sizes required more processing time due to a higher number of vectorized chunks, with chunk size 500 taking 4 minutes and 36 seconds to upload, while chunk size 5000 only took 29 seconds. This suggests that while larger chunks improve ingestion efficiency, they must be balanced with retrieval effectiveness to avoid missing critical information.

7.1.5 Multilingual Capabilities and Language Adaptability

The l*anguage testing* confirmed that the system can process queries in multiple languages despite the vector database being stored in English. However, the German response exhibited omissions in key process steps and failed to retrieve the correct link, whereas the English query produced a complete output. This indicates that while cross-lingual retrieval is possible, accuracy is reduced when querying in a different language than the stored information. If high precision is required, building separate vector databases for each language may be necessary. However, for general-purpose use, a single English-language vector storage provides a sufficiently accurate baseline.

Overall, the RAG system demonstrated strong retrieval accuracy, adaptability, and multilingual functionality but exhibited inconsistencies in completeness, hallucination control, and retrieval efficiency. Key areas for future improvement include optimizing chunking strategies to ensure all relevant information is consistently retrieved, refining prompt engineering to strictly limit outputs to retrieved content, and enhancing multilingual support to ensure equal performance across different languages. Further iterations should also evaluate hybrid retrieval techniques that integrate vector-based and keyword-based search to maximize precision while maintaining fluency in generated responses.

7.2 Challenges and potential solutions

The implementation of the RAG system within Business Process Management has demonstrated strong potential for improving process-related information retrieval and automated response generation. However, several challenges persist that affect retrieval accuracy, response completeness, system efficiency, and overall usability. This section outlines potential optimizations to enhance the system's performance while addressing key challenges and their respective solutions.

When researching RAG technology, one is most likely to be confronted with LangChain. LangChain is a powerful open-source framework designed to enhance the development of applications that integrate Large Language Models with structured data retrieval mechanisms. By providing modular components for document loading, retrieval,

augmentation, and response generation, LangChain simplifies the implementation of complex RAG workflows. The framework supports multiple vector databases, hybrid search techniques, and fine-tuned control over how queries interact with retrieved data. Its flexibility makes it a good tool for optimizing various aspects of the RAG pipeline, addressing core challenges such as retrieval precision, chunking efficiency and hallucination mitigation. As compatibility issues with versioning of LangChain and Chroma DB were encountered during the development phase, the current RAG prototype does not make use of the LangChain library. Future adaptations could resolve this issue and uncover the potential in dormant in LangChain. It is still worth considering the benefits that LangChain could provide.

7.2.1 Optimization of Chunking Strategies

Challenge: The evaluation has shown that retrieval inconsistencies can stem from ineffective chunking strategies. Smaller chunk sizes can lead to fragmented information retrieval, while excessively large chunks may introduce irrelevant content or redundancy.

Potential Optimization: A dynamic chunking approach could be implemented, where chunk size is adjusted based on document structure, semantic boundaries, or predefined business process sections. Techniques such as recursive text splitting, where chunk sizes vary depending on content density, could enhance retrieval efficiency. Furthermore, overlapping chunking can be refined to ensure that crucial information is not lost at chunk boundaries. A more structured data format and additional data cleansing are potential options for improvement as well.

LangChain provides various document loaders that facilitate structured chunking strategies based on document format and content type. Using its Recursive Character Text Splitter, documents can be split dynamically, ensuring that chunks preserve semantic integrity. This chunking optimization might help mitigate retrieval inconsistencies and improve response fluency by ensuring that LLMs receive contextually complete and well-structured data.

7.2.2 Retrieval Precision and Re-Ranking Mechanisms

Challenge: The system occasionally retrieves incomplete information, reducing the accuracy of responses. This issue is exacerbated when the number of retrieved chunks (k) is too high, introducing noise into the generation process.

Potential Optimization: One of LangChain's key strengths lies in its retriever architecture, which enables the system to access and retrieve relevant information more effectively.

Unlike traditional single-method retrieval, LangChain supports multi-retriever strategies, allowing developers to implement hybrid retrieval mechanisms that combine:

- Vector similarity search for semantically relevant document retrieval.
- BM25 keyword-based retrieval, which improves recall for exact keyword matches. It is particularly useful when dealing with structured process documentation.
- Multi-vector store querying, which enables seamless access to multiple databases, solving the challenge of handling queries that require information from distinct sources.
- Context-aware re-ranking mechanisms, which prioritize retrieved documents based on their alignment with the user query, reducing noise and improving response accuracy.

By integrating these retrieval techniques, LangChain ensures that the system consistently retrieves the most relevant and high-quality information, minimizing the chances of incomplete or irrelevant results. (Sawakar et al., 2024)

7.2.3 Mitigating Hallucinations and Ensuring Answer Reliability

Challenge: While the system effectively retrieves information from the vector database, some tests revealed that when no relevant information is found, the language model attempts to supplement responses using its pre-trained knowledge. This can lead to hallucinations or the generation of misleading information.

Potential Optimization: Implementing a stricter confidence threshold for retrieval can help prevent hallucinations. If retrieved chunks do not meet a minimum similarity score, the system should default to a controlled rejection mechanism, clearly stating that no relevant information is available. Additionally, negative prompting techniques can be employed to explicitly instruct the model to avoid generating speculative content beyond retrieved knowledge. Feeding the system with additional examples within the prompt might be an easy solution.

Improvements of this nature were made during the development of the prototype, yet there is room for further improvement. Especially when adding new types of data to the database, the current behavior of the system cannot be predicted, leading to the need for further adjustments. Prompt engineering being a rather simple task makes it an easy point for quick fixes and adjustments without major changes within the system.

7.2.4 Multilingual Retrieval Optimization

Challenge: While the system has demonstrated the ability to process queries in multiple languages, tests indicated minor losses of information when querying in a different language than the stored content (e.g., German queries retrieving from an English vector database).

Potential Optimization: Creating separate vector databases for different languages can improve accuracy for non-English queries.

7.2.5 Reducing Overhead for Indexing

Challenge: The indexing process introduces computational overhead, particularly when uploading datasets with small chunk sizes.

Potential Optimization: Improve indexing efficiency optimizing the trade-off between output quality and chunk size.

7.3 Limitations

Despite the successful implementation of a RAG-based approach for integrating company-internal process data into AI-driven business process modeling, several limitations were identified throughout the research. One of the primary challenges was data availability and quality. The effectiveness of the RAG system heavily depends on the comprehensiveness and accuracy of the indexed process data. In cases where process documentation was incomplete, outdated, or inconsistently formatted, retrieval performance was compromised, leading to potential gaps in AI-generated outputs. Additionally, semi-structured process data required significant preprocessing to be effectively integrated into the retrieval pipeline, increasing the complexity of data ingestion.

Another significant limitation was the handling of hierarchical and interdependent process structures. Business processes often involve complex relationships between subprocesses, decision points, and dependencies. While the implemented RAG system improved retrieval accuracy, it sometimes struggled with maintaining contextual continuity across multiple layers of process documentation. The chunking and embedding techniques used to index process data had inherent constraints, particularly when managing large-scale process models with deeply nested structures. As a result, AI-generated responses occasionally lacked holistic insights into multi-level workflows, which could impact their practical applicability for business analysts.

The currently developed system operates on a one-shot prompting method rendering advanced generation tactics such as chain prompting to iteratively (Shao et al., 2023; Tang

& Yang, 2024) improve the output invalid. While plans are made to allow users to refine their generated models iteratively, this functionality has not been implemented yet. Once this functionality has been implemented, additional research on how to incorporate the RAG pipeline into the new generation process might be worth investing.

The evaluation methodology also presented certain limitations. While retrieval accuracy and response quality were assessed through similarity metrics and expert reviews, the subjectivity of human evaluations introduced variability in assessing the effectiveness of AI-generated process models. Furthermore, the system's real-world performance was tested in a controlled environment with sample datasets, meaning its scalability and adaptability to diverse organizational contexts require further validation. The computational costs associated with indexing and retrieval operations, particularly when handling large process repositories, were also a consideration for real-world deployment.

Finally, the study focused primarily on optimizing retrieval and response generation but did not fully address integration with existing business process management tools beyond a proof-of-concept level. While the LangChain framework could facilitate modular implementation, further work is needed to ensure seamless interoperability with enterprise-grade BPM platforms and databases. Additionally, long-term maintenance and updating of indexed process data remain an open challenge, as business processes evolve over time, requiring continuous re-indexing to maintain accuracy.

Despite these limitations, the findings demonstrate that RAG is a promising approach for enhancing AI-driven business process modeling. However, future research should focus on refining retrieval mechanisms, improving hierarchical data handling, optimizing context management for large-scale workflows, and conducting real-world deployments to validate scalability and operational feasibility in business environments. The current setup is mostly useful for short-form factoid question answering (QA). Thus long-form QA which involves open-domain summarization and reasoning, cannot be sufficiently serviced as these have higher information needs that exceed the input (Beltagy et al. 2020; Chen et. al.,2023).

7.4 Implications for practice

This thesis investigated the use of Retrieval-Augmented Generation (RAG) to enhance AI-driven business process modeling by incorporating internal company knowledge into Horus AI. By implementing a RAG pipeline, the study addressed several practical challenges, including indexing (semi-)structured data, referencing document origins, and enriching the context through related process documents such as predecessor and successor activities. The prototype demonstrated that uploading process documentation into a vector store enables more accurate, contextually grounded outputs and reduces common generative model pitfalls such as hallucination and inaccuracies.

The implications for PROMATIS are twofold. First, the results provide a foundation for future decisions regarding the commercial implementation of RAG within Horus AI. Second, the experiment illustrates the potential of RAG to close the gap between generative AI capabilities and domain-specific knowledge requirements, an essential consideration for BPM tools operating in complex enterprise environments.

Moreover, the results suggest that enhancing the retrieval process by adapting chunk size and k can improve accuracy. While this prototype focuses on a standalone system, future adoption in enterprise settings will require careful attention to scalability, long-term maintenance, and security requirements. Commercial providers such as Weaviate or Pinecone offer managed solutions with built-in hybrid retrieval and performance optimizations that may become attractive options for organizations seeking to scale beyond the limitations of local deployments, albeit with additional financial and data governance considerations.

8 CONCLUSION AND OUTLOOK

8.1 Conclusion

This research has demonstrated the feasibility and value of integrating Retrieval-Augmented Generation into a business process modeling context. By enhancing Horus AI with a RAG-based approach, the system can dynamically access internal knowledge bases, thereby improving the accuracy, contextual relevance, and traceability of generated process models. The study highlighted several technical and methodological challenges. Data availability and quality emerged as crucial factors influencing retrieval effectiveness, emphasizing the need for well-maintained and consistently structured process documentation. The hierarchical nature of business processes posed difficulties in maintaining contextual continuity, requiring advanced indexing strategies and improved chunking techniques. While limitations remain, this prototype lays the groundwork for further development and adoption.

The evaluation confirmed that RAG enhances the quality of AI-generated content when configured properly. It also revealed the conditions under which RAG systems perform suboptimal, offering PROMATIS actionable insights for refining their AI capabilities. Although the implementation does not yet integrate deeply into Horus AI's architecture, the findings support the concept that RAG can serve as a powerful augmentation to model generation pipelines.

Overall, this thesis contributes to the evolving body of knowledge on the intersection of large language models, business process management, and enterprise data integration, an area that remains underexplored but holds considerable promise.

8.2 Outlook

Looking to the future, several promising directions emerge for advancing the use of Retrieval-Augmented Generation within business process modeling environments such as Horus AI. A key area of improvement lies in optimizing retrieval strategies to better handle the hierarchical and often complex nature of business processes. This includes exploring adaptive chunking methods and reranking mechanisms that take both semantic relevance and structural metadata into account, ensuring that multi-level workflows are accurately represented and understood by the system.

Moreover, tighter integration of RAG components into existing business process management platforms will be essential for real-world applicability. Moving beyond standalone prototypes toward fully embedded solutions would require robust API integration, secure data pipelines, and scalable vector infrastructure that can meet enterprise-grade demands. As organizations increasingly operate across multiple business domains and languages, extending RAG systems to support multilingual retrieval and generation will also become critical. Doing so would ensure that AI-driven outputs are accessible and contextually relevant in diverse operational settings.

Scalability remains another important consideration. Future implementations should be evaluated on larger and more diverse datasets to assess performance, latency, and resource efficiency at scale. Real-world testing in live business environments would provide valuable insights into long-term maintainability and user experience. Hybrid retrieval methods, which combine vector-based and symbolic search strategies, also offer an exciting frontier for improving both the precision and flexibility of information access.

Incorporating LangChain into the RAG system can address its current limitations such as retrieval precision. Future iterations should explore the deeper integration of LangChain's retrievers and document loaders to improve system performance further.

Looking ahead, one promising direction for future research and development is the integration of agentic behavior into Retrieval-Augmented Generation (RAG) systems. Traditional RAG setups follow a relatively static pipeline of retrieval followed by generation. In contrast, Agentic RAG, as surveyed by Singh et al. (2025), introduces a more dynamic and autonomous framework in which AI agents can plan, reflect, and iteratively refine their outputs based on feedback or additional information needs. These agents are capable of breaking down complex tasks, querying multiple sources, and chaining retrieval and generation steps in a goal-directed manner. For enterprise applications like Horus, the adoption of agentic RAG could enable more interactive and adaptive AI assistants that not only retrieve relevant process data but also proactively navigate information spaces and ask clarifying questions. This would mark a significant shift from simple document retrieval toward collaborative, reasoning-capable AI systems, opening new possibilities for intelligent process modeling and decision support.

In conclusion, this thesis has established a proof of concept for the use of RAG within Horus AI and, by extension, within BPM systems more broadly. While further development is required to reach production-readiness, the findings provide a solid foundation for PROMATIS and other organizations aiming to enhance their process modeling capabilities through intelligent, knowledge-aware AI systems. As the field evolves, the integration of structured enterprise data with large language models through RAG represents a promising path toward more intelligent, responsive process management.

REFERENCE LIST

1. Akter, S. (2024). Ethical AI Development for Sustainable Enterprises. *Journal of Artificial Intelligence General Science (JAIGS)*, *6*(1), 95–124. https://doi.org/60087
2. Amati, G. (2009). BM25. Encyclopedia of Database Systems. 257–260.
3. Amazon Web Services, Inc. (2024). *What is a Vector Database?* https://aws.amazon.com/what-is/vector-databases/.
4. Amazon Web Services, Inc. (2024*). What is RAG? – Retrieval Augmented Generation explained – AWS*. https://aws.amazon.com/what-is/retrieval-augmented-generation/.
5. Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2019). *Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI*. http://arxiv.org/pdf/1910.10045v2
6. Bansal, L. (2024). *Advance RAG-Improve RAG performance*. https://luv-bansal.medium.com/advance-rag-improve-rag-performance-208ffad5bb6a
7. Barnett, S., Kurniawan, S., Thudumu, S., Brannelly, Z., & Abdelrazek, M. (2024). Seven Failure Points When Engineering a Retrieval Augmented Generation System. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI,* 194−199. http://arxiv.org/pdf/2401.05856v1
8. Beltagy, I., Peters, M. E., & Cohan, A. (2020). *Longformer: The Long-Document Transformer*. http://arxiv.org/pdf/2004.05150v2
9. Bernardi, M. L., Casciani, A., Cimitile, M., & Marrella, A. (2024). Conversing with business process-aware large language models: the BPLLM framework. *Journal of Intelligent Information Systems*, *62*(6), 1607–1629. https://doi.org/10.1007/s10844-024-00898-1
10. Brockhaus, T. (2024). RAG Does Not Work for Enterprises.
11. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelkantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Voss, A.-H., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J [Jeffrey], Winter, C., . . . Amodei, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901.
12. Cai, J., & Li, Y. (2024). *Semi-Structured Data Parsing and Vectorization Using Retrieval-Augmented Generation*.
13. Chen, H.-T., Xu, F., Arora, S., & Choi, E. (2023). *Understanding Retrieval Augmentation for Long-Form Question Answering*. http://arxiv.org/pdf/2310.12150v1
14. Chen, J., Lin, H., Han, X., & Le, S. (2024). Benchmarking Large Language Models in Retrieval-Augmented Generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, *38*(16), 17754–17762. https://doi.org/10.1609/aaai.v38i16.29728
15. Chroma. (2025). *Chroma Documentation*. https://www.trychroma.com/
16. Dascalescu, D., & Hasan, Z. (2023). *Vector Embeddings Explained*. https://weaviate.io/blog/vector-embeddings-explained
17. Doshi-Velez, F., & Kim, B. (2017). *Towards A Rigorous Science of Interpretable Machine Learning*. http://arxiv.org/pdf/1702.08608v2
18. Eldin, A. N., Assy, N., Anesini, O., Dalmas, B., & Gaaloul, W. (2024). A Decomposed Hybrid Approach to Business Process Modeling with LLMs. *International Conference on Cooperative Information Systems*, 243–260.

19. Espejel, O. (2022). *Getting Started With Embeddings*. https://huggingface.co/blog/getting-started-with-embeddings
20. Forell, M. (2024). *Behind the Scences: How to teach OpenAI to create beautiful Horus Models.* Promatis Software GmbH.
21. Friedrich, F. (2010). Automated Generation of Business Process Models from Natural Language Input. *M. Sc., School of Business and Economics. Humboldt-Universität zu Berlin*.
22. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2023). *Retrieval-Augmented Generation for Large Language Models: A Survey*. http://arxiv.org/pdf/2312.10997v5
23. Gartner (2024). *Optimize AI Investments for Long-Term Business Growth*. With assistance of Michelle Carlsen. https://www.gartner.com/en/articles/optimize-ai-investments-for-long-term-business-growth.
24. Ggaliwango, M., Nakayiza, H., Daudi, J., & Joyce, N.-N. Prompt Engineering in Large Language Models. In *Data Intelligence and Cognitive Informatics*, 387–402. https://doi.org/10.1007/978-981-99-7962-2_30
25. Ghauri, P., Grønhaug, K., & Strange, R. (2020). *Research Methods in Business Studies*. Cambridge University Press.
26. Grohs, M., Abb, L., Elsayed, N., & Rehse, J.-R. (2023). Large Language Models can accomplish Business Process Management Tasks. In *International Conference on Business Process Management*, 453−465.
27. Guu, K., Lee, K., Tung, Z., Pasupat, P., & Ming, W. (2020). REALM: Retrieval-Augmented Language Model Pre-Training. In *International conference on machine learning*, 3929–3938. PMLR.
28. Han, Y., Liu, C., & Wang, P. (2023). *A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge*. arXiv preprint arXiv:2310.11703.
29. Hevner, A. (2007). A Three Cycle View of Design Science Reserach. *Scandinavian Journal of Information Systems*, *19* (2), 87–92.
30. Hevner, J. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105.
31. Honroth, T., Siebert, J., & Kelbert, P. (2024). *Retrieval Augmented Generation (RAG): Chatten mit den eigenen Daten*. https://www.iese.fraunhofer.de/blog/retrieval-augmented-generation-rag/
32. Horus Web Application (2025). *BP4-Apps*.
33. Hu, Y., Lei, Z., Zhang, Z., Pan, B., Ling, C., & Zhao, L. (2024). *GRAG: Graph Retrieval-Augmented Generation*. http://arxiv.org/pdf/2405.16506v2
34. Huang, Y. & Huang, J. (2024). *A Survey on Retrieval-Augmented Text Generation for Large Language Models*. arXiv preprint arXiv:2404.10981.
35. IBM. (2023*). What Is LangChain?* IBM. https://www.ibm.com/topics/langchain
36. Izacard, G., & Grave, E. (2020). *Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering*. http://arxiv.org/pdf/2007.01282
37. Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W. (2020). Dense Passage Retrieval for Open-Domain Question Answering. In *EMNLP, (1)*, 6769−6781. http://arxiv.org/pdf/2004.04906v3
38. Kau, A. (2024*). Understanding Retrieval Pitfalls: Challenges Faced by Retrieval Augmented Generation (RAG) models*. https://doi.org/10.59350/wek8b-sbj84

39. LangChain. (2025). *Introduction*. https://python.langchain.com/docs/introduction/
40. Lee, H., & Kim, S. (2024*). Bring Retrieval Augmented Generation to Google Gemini via External API: An Evaluation with BIG-Bench Dataset*. Research Square Platform LLC. https://doi.org/10.21203/rs.3.rs-4394715/v1
41. Lewis, P., Ethan, P., Aleksandra, P., Fabio, P., Vladimir, K., Naman, G., Heinrich, K., Mike, L., Wen-tau, Y., Tim, R., Sebastian, R., & Douwe, K. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Proceedings of the 34th International Conference on Neural Information Processing Systems*, *33*, 9459–9474.
42. Martineau, K. (2024). *What's an LLM context window and why is it getting larger?* https://research.ibm.com/blog/larger-context-window
43. Minor, M., & Kaucher, E. (2024). Retrieval Augmented Generation with LLMs for Explaining Business Process Models. In *International Conference on Case-Based Reasoning*, 175–190. Springer Nature Switzerland.
44. Mishra, A., & Gupta, A. (2025). Retrieval Augmented Generation (RAG) Model. *International Journal of Research Publication and Reviews*, *6*(6), 4690–4693. https://doi.org/10.55248/gengpi.6.0125.0635
45. Muennighoff, N., Tazi, N., Magne, L., & Reimers, N. (2022). *MTEB: Massive Text Embedding Benchmark*. http://arxiv.org/pdf/2210.07316v3
46. Nie, P., Zhang, Y., Ramamurthy, A., & Song, L. (Eds.) (2021). *Answering Any-hop Open-domain Questions with Iterative Document Reranking*. arXiv preprint arXiv:2009.07465.
47. OpenAI. (2021). *GPT-3 powers the next generation of apps*. https://openai.com/index/gpt-3-apps
48. OpenAI Platform. (2025). *OpenAI Documentation*. https://platform.openai.com/docs/
49. Oracle. (2025). *Oracle Cloud Applications*. https://www.oracle.com/applications/
50. Peffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Vitranen, V., & Bragge, J. (2006). *The Design Science Research Process: A Model for Producing and Presenting Information Systems Research*. In *1st International Conference on Design Science in Information Systems and Technology (DESRIST)*, 24 (3), 83−106.
51. Peng, B., Zhu, Y., Liu, Y., Bo, X., Shi, H., Hong, C., Zhang, Y., & Tang, S. (2024). *Graph Retrieval-Augmented Generation: A Survey*.
52. Pi, W. (2024). *Efficient Information Retrieval and Response Generation with Retrieval-Augmented Generation (RAG)*. https://doi.org/10.59350/q2pq3-0fv85
53. Pinecone. (2025). *Pinecone Documentation*. https://www.pinecone.io/
54. Promatis. (2025). *Company internal documentation*.
55. Promatis. (2025). *Horus Business Process Management Suite* [Computer software]. Promatis.
56. Quinn, L., Kasey, U., & Alkis, P. (2024). *Best Practices for LLM Evaluation. Databricks Blog*. https://www.databricks.com/blog/LLM-auto-eval-best-practices-RAG
57. Radeva, I., Popchev, I., Doukovska, L., & Dimitrova, M. (2024). Web Application for Retrieval-Augmented Generation: Implementation and Testing. *Electronics*, *13*(7), 1361. https://doi.org/10.3390/electronics13071361
58. Ravi, T. (2023). *Evaluating the Ideal Chunk Size for RAG System using LlamaIndex*. https://www.llamaindex.ai/blog/evaluating-the-ideal-chunk-size-for-a-rag-system-using-llamaindex-6207e5d3fec5.
59. Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. http://arxiv.org/pdf/1908.10084v1

60. Sawarkar, K., Mangal, A., & Solanki, S. R. (2024). Blended RAG: Improving RAG (Retriever-Augmented Generation) Accuracy with Semantic Search and Hybrid Query-Based Retrievers. In *2024 IEEE 7th International Conference on Multimedia Information Processing and Retrieval (MIPR)*, 155–161. https://doi.org/10.1109/MIPR62202.2024.00031
61. Shao, Z., Gong, Y., Shen, Y., Huang, M., Duan, N., & Chen, W. (2023). *Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy*. http://arxiv.org/pdf/2305.15294v2
62. Sharma, S., Yoon, D. S., Dernoncourt, F., Sultania, D., Bagga, K., Zhang, M., Bui, T., & Kotte, V. (2024). Retrieval Augmented Generation for Domain-specific Question Answering. *Proceedings of Workshop on Scientific Document Understanding*, 20−27. http://arxiv.org/pdf/2404.14760v2
63. Shen, M., Umar, M., Maeng, K., Suh, G. E., & Gupta, U. (2024). *Towards Understanding Systems Trade-offs in Retrieval-Augmented Generation Model Inference*. http://arxiv.org/pdf/2412.11854v1
64. Shuster, K., Poff, S., Chen, M., Kiela, D., & Weston, J. (2021). *Retrieval Augmentation Reduces Hallucination in Conversation*. http://arxiv.org/pdf/2104.07567v1
65. Singh, A., Ehtesham, A., Kumar, S., & Khoei, T. T. (2025). *Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG*. http://arxiv.org/pdf/2501.09136v3
66. Siriwardhana, S., Weerasekera, R., Wen, E., Kaluarachchi, T., Rana, R., & Nanayakkara, S. (2023). Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering. *Transactions of the Association for Computational Linguistics*, *11*, 1–17. https://doi.org/10.1162/tacl_a_00530
67. Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson.
68. Soudani, H., Kanoulas, E., & Hasibi, F. (2024). Fine Tuning vs. Retrieval Augmented Generation for Less Popular Knowledge. In *Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, 12–22. https://doi.org/10.1145/3673791.3698415
69. Suzuoki, S., & Hatano, K. (2024). *Reducing Hallucinations in LLM: A Consensus Voting Apporach Using Mixture of Experts*. https://doi.org/10.36227/techrxiv.171925057.75949684/v1
70. Tang, Y., & Yang, Y. (2024). *MultiHop-RAG: Benchmarking Retrieval-Augmented Generation for Multi-Hop Queries*. http://arxiv.org/pdf/2401.15391v1
71. Tu, Y., Su, W., Zhou, Y., Liu, Y., & Ai, Q. (2018). *RbFT: Robust Fine-tuning for Retrieval-Augmented Generation against Retrieval Defects*. http://arxiv.org/pdf/2501.18365v1
72. van der Aalst, W.M.P. (2002). *Making Work Flow: On the Application of Petri nets to Business Process Management*. Springer Berlin Heidelberg.
73. van der Aalst, W.M.P. (2013). Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, 1–37. https://doi.org/10.1155/2013/507984
74. Venkatesh, V., Brown, S. A., & Bala, H. (2013). Bridging the Qualitative-Quantitative Divide: Guidelines for Conducting Mixed Methods Research in Information Systems. *MIS Quarterly*, *37*(1), 21–54. https://doi.org/10.25300/MISQ/2013/37.1.02
75. Vidgof, M., Bachhofner, S., & Mendling, J. (2023). Large Language Models for Business Process Management: Opportunities and Challenges. In *International*

Conference on Business Process Management, 107−123. http://arxiv.org/pdf/2304.04309v1
76. Volynets, N. (2024). *Choosing the right vector embedding model for your generative AI use case*. https://www.datarobot.com/blog/choosing-the-right-vector-embedding-model-for-your-generative-ai-use-case/
77. Wang, Z., Ng, P., Ma, X., Nallapati, R. & Xiang, B. (2019). Multi-passage BERT: A globally normalized bert model for open-domain question answering. In *Emperical Methods in Natural Language Processing (EMNLP)*.
78. Weaviate. (2025*). Weaviate Documentation*. https://weaviate.io/
79. Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-43839-8
80. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., & Russel, N. (2006). On the Suitability of BPMN for Business Process Modellinig. *Proceedings 4th Interntational Conference on Business Process Management*, 161–176.
81. Wu, S., Xiong, Y., Cui, Y., Wu, H., Chen, C., Yuan, Y., Huang, L., Liu, X., Kuo, T.-W., Guan, N., & Xue, C. J. (2024). *Retrieval-Augmented Generation for Natural Language Processing: A Survey*. http://arxiv.org/pdf/2407.13193
82. Yu, H., Gan, A., Zhang, K., Tong, S., Liu, Q., & Liu, Z. (2024). Evaluation of Retrieval-Augmented Generation: A Survey. In *CCF Conference on Big Data*, 102–120. Singapore: Springer Nature Singapore.
83. Yu, W., Zhang, H., Pan, X. Ma, K., Wang, H. & Yu, D. (2023). *Chain-of-note: Enhancing robustness in retrieval-augmented language models*. arXiv preprint arXiv:2311.09210.
84. Zhao, S., Yang, Y., Wang, Z., He, Z., Qiu, L. K., & Qiu, L. (2024). *Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make your LLMs use External Data More Wisely*. http://arxiv.org/pdf/2409.14924v1

APPENDICES

Appendix 1: Povzetek (Summary in Slovene Language)

Magistrsko delo raziskuje uporabo pridobivanjem podprtega generiranja (RAG – Retrieval-Augmented Generation) za izboljšanje z umetno inteligenco podprtega modeliranja poslovnih procesov znotraj programske platforme Horus AI, ki jo razvija podjetje PROMATIS Software GmbH. Čeprav Horus AI že omogoča generiranje procesnih modelov iz besedilnega vnosa, nima dostopa do internega znanja podjetja, kar vodi do splošnih in pogosto kontekstualno neustreznih rezultatov. Osnovni cilj te raziskave je bil implementirati in ovrednotiti prototip sistema RAG, ki Horus AI omogoča iskanje, sklicevanje in razumevanje internih procesnih dokumentov.

Delo temelji na metodologiji raziskovanja z oblikovanjem (Design Science Research), ki vključuje načrtovanje, razvoj in vrednotenje dokaza koncepta sistema RAG. Arhitektura sistema vključuje ključne komponente, kot so Chroma DB za shranjevanje vektorjev, model OpenAI text-embedding-3-small za generiranje vdelav in GPT-4o za generiranje odgovorov na podlagi konteksta. Sistem podpira več jezikov, zavračanje nerelevantnih vprašanj in večnivojsko iskanje za bolj zapletena vprašanja.

Za vrednotenje sistema sem izvedel serijo strukturiranih eksperimentov, kjer so bile preizkušene spremenljivke, kot so velikost odsekov (chunk size), število pridobljenih dokumentov (k), struktura pozivov (promptov) in kompleksnost vprašanj. Uporabljeni so bili kazalniki, kot so ustreznost, popolnost, doslednost, tekočnost in odzivni čas. Rezultati kažejo, da ustrezna nastavitev velikosti odsekov in števila dokumentov pomembno vpliva na kakovost pridobivanja ter da zasidranost generiranih vsebin v internih podatkih bistveno izboljša natančnost in sledljivost.

Kljub nekaterim omejitvam, kot so odsotnost iterativnih pozivov, kompleksnost hierarhičnih modelov procesov in odvisnost od dobro strukturiranih podatkov, študija dokazuje, da je RAG izvedljiva in uporabna razširitev orodij za management poslovnih procesov. Vključevanje internega znanja preko RAG zmanjša halucinacije, poveča razložljivost in omogoča domensko prilagoditev. Prototip pozicionira Horus AI kot primerno orodje za prihodnjo rabo v podjetjih, ki iščejo inteligentne, prilagodljive in uporabniku prijazne rešitve za avtomatizacijo. Priporočila za nadaljnje delo vključujejo globljo integracijo v platforme za avtomatizacijo poslovnih procesov, boljšo podporo za hierarhične procese in uporabo komercialnih rešitev za hibridno iskanje za boljšo razširljivost.

Appendix 2: Supplementary Horus AI Prompt Instructions

Process:

1.**Concurrency Detection**:

- I will quickly spot signs of concurrency.

- I need to recognize transitions that depict parallel activities.

- I will apply concurrency as necessary.

- I will ensure that branching occurs at a transition and merging concludes at a subsequent transition.

- If concurrency is detected, intermediate steps are allowed and can be added.

2.**Cycle Detection**:

- If a cycle is detected, I must ensure that no deadlock occurs.

- I will carefully analyze the process structure to identify feedback loops or repeated paths.

- Upon detecting a cycle, I will check if appropriate control mechanisms are in place to prevent infinite repetition or blocking.

- I will ensure that there are clear conditions for entering and exiting the cycle to maintain process integrity.

3.**Title Identification**:

- I will deduce an appropriate title for the Petri net based on its process representation.

4.**Place Identification**:

- I will detail all places and label them in past participle format with succinct descriptions.

- Places with multiple outgoing edges will hint at process alternatives for me.

- If a place (Object Store) has multiple outgoing edges, it signifies alternatives representing a XOR-Split for me. I must ensure these paths reconverge at a subsequent place, representing an XOR-Join.

5.**Transition Identification**:

- I will chronicle all transitions, naming them and providing concise details.

- I need to highlight transitions with multiple outgoing edges, as these indicate concurrency.

- If a transition (Activity) has multiple outgoing edges, it suggests concurrency representing a AND-Split to me. I must confirm that these concurrent paths merge at a subsequent transition, denoting an AND-Join.

- I will ensure that transition is not the end of the process, as this would result in a deadlock.

6.**Edge Analysis**:

- I will delve into all edges, expounding on their direction and the nodes they interlink.

- It's vital for me to ensure the Petri net remains strongly connected, integrating concurrency from step 1.

- The same XOR-Join and AND-Join considerations as earlier apply here.

7.**Additional Constraints**:

- I will adhere to the rules that branching transitions due to concurrency should merge at another transition (AND-Join), and branching paths from a place should converge at another place (XOR-join).

Appendix 3: Code Screenshots

Utilized libraries

```
import os
from dotenv import load_dotenv
import chromadb
from openai import OpenAI
from chromadb.utils import embedding_functions
```

Function to load environment variables including API key and embedding model

```
load_dotenv()

openai_key = os.getenv("OPENAI_API_KEY")

openai_ef = embedding_functions.OpenAIEmbeddingFunction(
    api_key=openai_key, model_name="text-embedding-3-small"
)
```

Function to create or reference a Chroma DB

```
chroma_client = chromadb.PersistentClient(path="Horus_RAG_1500")
collection_name = "Horus_RAG"
collection = chroma_client.get_or_create_collection(
    name=collection_name, embedding_function=openai_ef
)

client = OpenAI(api_key=openai_key)
```

RAG Prompt

```
prompt = (
    "You are an assistant for question-answering tasks. Use the following pieces of "
    "retrieved context to answer the question. If you don't know the answer, say that you "
    "don't know. If possible, retrieve the link (URL) to the source of the information."
    "it is typically found at the beginning of paragraph."
    "Example 1: 1.13    BP - PO2RC - L2 - Quality Check URL:https://promatis.horus.biz/editor?open=
    "Open  Quality Parameters"
    "To perform the quality check, it is necessary to open the predefined quality parameters."
    "Check Quality of Goods"
    "To secure the quality of the received goods and services, they need to be compared to the qual
    "Capture Results"
    "Depending on the result of the quality check, the received goods are accepted or rejected."
    "\n\nContext:\n" + context + "\n\nQuestion:\n" + question
)
```

Appendix 4: Chroma Compatibility Table

	Python	Typescript
OpenAI	✓	✓
Google Generative AI	✓	✓
Cohere	✓	✓
Hugging Face	✓	-
Instructor	✓	-
Hugging Face Embedding Server	✓	✓
Jina AI	✓	✓

Appendix 5: RAG Architecture

