TUM SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY

TECHNICAL UNIVERSITY OF MUNICH

Technical Report

Retrieval Augmented Generation: Enhancing LLMs with internal knowledge

Mingxiao Guo, Florian Koller, Mohamed Nejjar, Ali Rabeh, Paul Reisenberg, Anton Sattler, Arnav Singh, Akshat Tandon

Contact: {firstname.lastname}@tum.de

Supervisor:	**Prof. Dr. Ingo Weber**
Advisor:	**Ph.D. Hans Weytjens**
Submission Date:	**11.07.2024**

Acknowledgments

We thank the Chair of IT Service Management, Development, and Operations for their ongoing support and valuable feedback. We also appreciate the people at Fraunhofer for their role as our customer.

Contents

Acknowledgments **i**

1. Introduction **1**

2. Data Preparation **2**

3. Retrieval Augmented Generation **3**
3.1. LangGraph Framework . . . 3
3.2. Data Ingestion . . . 3
3.3. Indexing . . . 4
3.4. Vector Databases . . . 5
3.5. Retrieval . . . 7
3.6. Reranking . . . 9
3.7. Generation . . . 9
3.8. Query Preprocessing: Routing and Enhancement . . . 14
3.9. Guardrails . . . 18
3.10. User Interface . . . 22

4. Pipeline Evaluation **23**
4.1. Metrics . . . 23
4.2. Evaluation Dashboard . . . 23
4.3. Results . . . 24

5. Future Outlook **27**

6. Reflection **28**

A. Pipeline Evaluation Results **29**
A.1. Context Precision and Recall for Indexing-Retrieval Combinations . . . 29
A.2. Context Precision and Recall for Chunking Methods . . . 29

Bibliography **30**

1. Introduction

Retrieval-Augmented Generation (RAG) is a powerful method for extending the capabilities of large language models (LLMs) by incorporating an external knowledge base for reference. This approach allows LLMs to access and utilize vast amounts of information that they may not have been exposed to during their initial training. By using RAG, LLMs can effectively address specific domains or leverage an organization's internal knowledge without having to retrain the model, making it a versatile and economical solution for various applications.
We propose a modular pipeline accompanied by an evaluation dashboard. Each module of the pipeline was tested individually. For end-to-end testing, the RAG was configured using Fraunhofer websites data and the entire pipeline was evaluated using the dashboard. In addition, the evaluation dashboard facilitates the optimization of the pipeline with any chosen data set.

We first show how the corpus used by the RAG is created in chapter 2. In chapter 3, we present a modular pipeline, detailing each component and its evaluation process. Given the interdependence of some modules on user data and their interconnected nature, chapter 4 provides a comprehensive evaluation of the pipeline. After that, a future outlook is given in chapter 5. Finally, we conclude the report with a personal reflection in chapter 6.

2. Data Preparation

Although our RAG pipeline is designed to be general-purpose and able to handle different datasets, our primary customer, the Fraunhofer Institute, requires a customized approach. Since we lacked access to the internal data, we worked with proxy data by collecting relevant public information from the official websites of all Fraunhofer institutes.

1. **Fraunhofer institutes scraping**: The first step is to obtain the list of starting URLs containing the 76 Fraunhofer institutes' main website links. This list is temporarily stored locally for the following steps.

2. **Recursive crawling**: The main crawling spider recursively crawls all links starting from the previously saved 76 links. It is restrained to the Fraunhofer domain to avoid infinite crawling extending to external web domains.

3. **Text data parsing** All available text data formats, including HTML, PDFs, DOCs, and TXTs, are considered within the scope of the project.

   BeautifulSoup [1] was used as one of the industry standards to parse the HTML content. For PDF files, additional challenges such as intricate layouts, embedded images or tables, etc. complicate accurate text extraction during the crawling process. Advanced frameworks capable of utilizing computer vision and Optical Character Recognition (OCR) need to be leveraged to ensure case-independent handling. For this purpose, we utilized the Unstructured framework [2], which provides a unified approach to extracting key elements from PDF files and outperforms other considered frameworks in the dimensions of accuracy, performance, and ease of use. Other text formats can be read directly without requiring additional processing.

4. **Data cleaning**: We designed our pipeline to ingest clean text data without dependency on the specific data formats and chunking strategies based on the structure of the file structure. The preprocessing step includes removing format-specific characters or tags, removing unnecessary whitespace characters (including extra spaces, tabs, and newline characters), stripping special characters and unnecessary punctuations, sanitizing file names, standardizing text encoding to utf-8, etc. These measures aim to enhance data quality, usability, and readability for subsequent retrieval and analysis tasks.

5. **Data storage**: Finally, the processed text data of each file is serialized into JSON format, together with useful metadata such as the file's title, retrieval timestamp, language, file format, etc. The JSON files are then stored locally or in a cloud storage service available for the pipeline evaluation.

3. Retrieval Augmented Generation

The RAG pipeline consists of various components organized into distinct modules. Before a question can be asked, the data corpus needs to be loaded, chunked, and embedded. These embeddings are then stored in a vector database. Once this setup is complete, the pipeline can be initiated, and questions can be posed. Initially, each question is checked for information leakage or inappropriate content. After any necessary reformulation, a router determines if the question requires internal data. If it does, the relevant documents are retrieved from the database. Ideally, these documents will contain useful information to generate a comprehensive answer. This pipeline is displayed in figure 3.1.

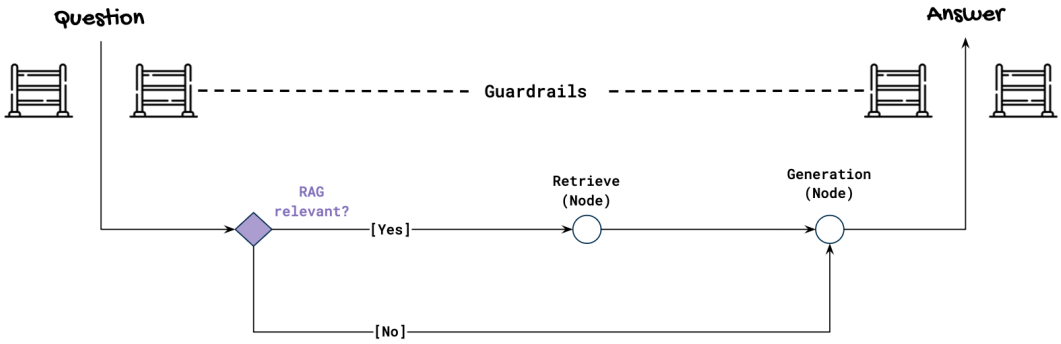

Figure 3.1.: Illustration of the Retrieval-Augmented Generation (RAG) Pipeline.

3.1. LangGraph Framework

A key feature of the RAG pipeline is its modularity, facilitated by the integration of LangGraph, a flexible framework that allows nodes to be easily added and manipulated. Rather than a linear sequence, the pipeline is designed as a graph, allowing for the junctions and loops necessary for routing and reranking. It consists of two primary nodes: Retrieval and Generation, with a state that carries the query, conversation, retrieved documents, and result between them. This modular structure ensures that each component can be developed, tested, and replaced independently without impacting the overall system, providing the flexibility needed to adapt to different use cases and improve performance over time [3].

3.2. Data Ingestion

The initial stage of the pipeline involves ingesting the data provided by the user. For now, the pipeline is designed to process any text data. To support versatile data ingestion capabilities, the pipeline allows the integration of data from diverse sources, including local filesystems

and industry-standard cloud storage services such as AWS S3 and Azure Blob Storage. Upon starting the pipeline, the content in the data storage is checked to identify any updates to trigger the indexing process, thereby reducing unnecessary overhead.

3.3. Indexing

To retrieve relevant document passages for the generation step, the collected text data must be divided into chunks and converted into embeddings. These embeddings are stored in a database capable of performing fast similarity searches.

3.3.1. Chunking

To handle large documents, all documents are broken into smaller chunks. This approach decreases the LLM prompt length and speeds up retrieval. However, there's a trade-off: smaller chunks contain less context, while larger chunks provide more information to the LLM but increase costs. Two different text splitting techniques are offered: Recursive Character Text Splitter and Semantic Chunking.

Recursive Character Text Splitter: The splitting into characters is done recursively. First, the text is split into paragraphs (denoted by "\n\n"), then into sentences (denoted by "\n"), then into words, and finally into single characters. The text is split recursively in the middle of the largest group and progressively split into smaller groups until it is short enough [4]. The maximum chunk size can be configured.
This can be illustrated by the following paragraph: "The current president of Fraunhofer is Prof. Holger Hanselka. He took office on August 15, 2023". Now we want to split it into a maximum of 4 words. First, the paragraphs are split into the sentences "The current president of Fraunhofer is Prof. Holger Hanselka" and "He took office on August 15, 2023. Then they are split into words, in the middle of the sentence: "The current president of Fraunhofer", "is Prof. Holger Hanselka", "He took office on", and "August 15, 2023". Finally, only the first part needs to be split again: "The current president" and "of Fraunhofer".

Semantic Chunking: Semantic chunking splits text based on meaning. The text is split into sentences that are then embedded. They can then be merged based on similarity [5]. Three adjustable thresholds determine which sentences belong together:

- **Percentile**: Sentences whose difference is less than a percentage are grouped together.
- **Standard Deviation**: The difference is based on the standard deviation.
- **Interquartile**: The difference is based on the interquartile range.

Specialized chunking: Many document-specific splitting methods exist, e.g. for PDFs and HTML files [5]. Since the crawling already returns documents in raw text format, they are not implemented.

3.3.2. Embedding

To store the chunks in a vector database, the chunks must be embedded. The chunks are transformed into vectors using an embedding model. These vectors are then stored in a vector database. To search the database for chunks similar to a user query, the query is also embedded. The vector database can then quickly return chunks similar to the query using vector similarity. Other methods are shown in 3.5. There are several offline and online embedding models available. A complete list of all embedding models that can be configured is shown in the table 3.1.

Model	Dimension	Provider	Connection	Cost
text-embedding-ada-002	1536	OpenAI	Online	US$0.1/M. Token
	1536	Azure OpenAI	Online	US$0.02/M. Token
text-embedding-3-small	1536	OpenAI	Online	US$0.02/M. Token
	1536	Azure OpenAI	Online	US$0.02/M. Token
text-embedding-3-large	3072	OpenAI	Online	US$s0.13/M. Token
	3072	Azure OpenAI	Online	US$0.13/M. Token
all-mpnet-base-v2	768	Huggingface	Offline	Operation Cost
Hosted Embedding Models	-	Ollama	Online/Offline	Operation Cost

Table 3.1.: List of available Embedding Models [6][7][8][9][10].

3.3.3. Multi-Representation Indexing

The concept behind multi-representation indexing is to store documents in multiple formats and use different ones for searching and retrieval. A parent document consists of two parts: a smaller chunk for searching and a larger chunk that is returned when retrieving [11].
This is implemented using summaries. The documents are chunked as before, but each chunk is summarized using an LLM. The summaries are much shorter than the chunks because they condense the content and remove redundant formatting often present in web scrapes. Each chunk is stored with its summary in a database. A user query is then matched against the summaries. Because the summaries are shorter, the search is faster, and fewer irrelevant documents are returned. The full chunks are then retrieved, allowing the LLM to still have access to the complete information without errors introduced during summarization.

3.3.4. Evaluation

Evaluating different indexing methods in isolation is challenging due to the lack of ground truth and useful metrics for assessing index performance. Therefore, different chunking and embedding models have been tested using an evaluation tool that evaluates the entire pipeline. The evaluation results are presented in chapter 4. Due to the high cost of summarizing chunks in the multi-representation indexing, it was not tested.

3.4. Vector Databases

The pipeline supports four prominent vector database options, namely ChromaDB, Milvus, Neo4j, and Postgres, due to their unique focus and strengths in handling text embeddings at scale.

3.4.1. Comparison

- **ChromaDB**: ChromaDB is an open-source vector database optimized for managing high-dimensional vector data and offers efficient similarity search capabilities. It supports various indexing algorithms that enhance the speed and accuracy of vector searches, making it a strong candidate for large-scale, real-time applications. It is probably the most popular option in LLM-related projects due to its simplicity of setup.
- **Milvus**: Milvus is another open-source option designed specifically for handling massive amounts of vector data, making it an ideal choice for RAG applications. It utilizes advanced indexing and partitioning strategies to ensure high performance and scalability, which is critical for applications requiring real-time search and retrieval of vectors.
- **Neo4j**: Neo4j is primarily known as a graph database but also supports vector embeddings. While it offers robust graph-based query capabilities, its performance in handling pure vector data might not be as optimized as specialized vector databases. However, it provides unique advantages when combining vector search with graph-based relationships.
- **Postgres**: Postgres is a relational database that has been extended to support vector embeddings through various extensions. It offers the familiarity and reliability of SQL databases, but its performance may lag behind dedicated vector databases in terms of handling large-scale vector data and real-time queries.

3.4.2. Evaluation

The first component that directly impacts the performance of the RAG pipeline is the vector database used for handling large-scale text embeddings. Therefore, evaluating the performance of different vector databases is essential to identify the most suitable option that can enhance the overall efficiency and effectiveness of the pipeline.

Metrics

We have considered the following key ways in which the vector database choice influences the RAG pipeline performance.

- **Query Throughput (Queries per Second)**: A high query throughput means that the database can handle more queries in a given time, which is essential to ensure the pipeline's responsiveness and scalability.
- **Recall Percentage**: Recall measures the accuracy of the database in retrieving relevant vectors. A higher recall percentage indicates better performance in finding the most relevant documents or data points, which is crucial for the quality of the generated responses in the RAG.
- **Load Duration**: The time taken to load data into the vector database affects the initialization and update times of the RAG pipeline. A shorter load means that the pipeline can be ready to process queries faster upon setup and can update its data more efficiently.

- **Latency**: Latency is the time taken to retrieve a response from the database after a query is made. Lower latency results in faster response times, which is critical for real-time user interactions.

Results and Conclusion

The results as indicated in the table 3.2 demonstrate that Milvus significantly outperforms the other databases across most metrics. It achieved the highest QPS at 520.1, indicating exceptional throughput. Milvus also led in recall percentage at 0.97, showcasing its superior accuracy in retrieving relevant vectors. Additionally, it had the shortest load duration at 2.7 ms, emphasizing its efficiency in data ingestion. However, Milvus exhibited a slightly higher latency (12 ms) compared to ChromaDB (8.7 ms), though this is a minor trade-off considering its overall performance.

	ChromaDB	Milvus	Neo4j	Postgres
Queries Throughput [qps]	5.3	**520.1**	46.2	10.63
Recall [%]	0.85	**0.97**	0.89	0.88
Load duration [ms]	9.7	**2.7**	15.3	10.1
Latency [ms]	**8.7**	12	43	633

Table 3.2.: Evaluation for the Vector Databases.

3.5. Retrieval

This section explains all available retrievers. We distinguish between basic retrievers (Nearest Neighbor and Support Vector Machines) and advanced retrievers (Multi-Query Retriever, Contextual Compression, Ensemble). Advanced retrievers still rely on a base retriever to return documents after applying various operations.

3.5.1. Nearest Neighbor

One of the easiest ways to retrieve documents with similar topics to a query is to use similarity search with Hierarchical Navigable Small World (HNSW) graphs. This approximate nearest neighbor search method is a state-of-the-art technique for handling high-dimensional vectors. By constructing a hierarchical graph where each node represents a document and edges connect nearby points, HNSW allows for efficient and accurate searches. When a query is vectorized and entered into the graph, the algorithm navigates through multiple levels, refining the search at each level to quickly find the most relevant documents [12].

3.5.2. Support Vector Machine

A support vector machine (SVM) information retrieval system ranks documents by relevance to a query using a trained SVM model. There are two main steps: model training and document scoring. During training, the embeddings of all documents in the corpus are combined with the embedding of the query. A one-dimensional label vector is created, where the query

is labeled as relevant (1) and all other documents are labeled as irrelevant (0). Despite this highly unbalanced data, the SVM learns to separate relevant from irrelevant documents by creating an optimal hyperplane. To retrieve the top relevant documents, the SVM scores each document based on confidence and returns the top k documents. Unlike nearest neighbor methods, this approach considers the entire dataset to compute the hyperplane, resulting in better performance [13].

3.5.3. Multi-Query Retriever

The Multi-Query Retriever uses a large language model to generate multiple queries from different angles for a given user input. For each of these queries, it retrieves a collection of relevant documents and returns the unique union over all documents to form a larger, more comprehensive set of potentially relevant documents. By considering multiple perspectives on the same question, the Multi-Query Retriever can overcome some of the limitations of traditional distance-based retrieval methods, resulting in a more diverse set of retrieved documents. The Multi-Query Retriever requires a base retriever to retrieve documents for each query [14].

3.5.4. Contextual compression

Contextual compression involves reducing the length of text data while preserving its core meaning and relevance, specifically tailored to a given query. Instead of returning the documents retrieved by a base retriever, this technique uses an LLM to compress them by focusing on the query context to ensure that only relevant information is returned. This process involves both compressing the content of individual documents and filtering out entire documents that are not relevant [14].

3.5.5. Ensemble Retriever

The Ensemble Retriever combines the results of different retrieval algorithms using the Reciprocal Rank Fusion (RRF) algorithm to improve performance. This approach uses BM25 as a sparse retriever and a Nearest Neighbor Retriever as a dense retriever. Sparse retrievers are effective for keyword-based searches, while dense retrievers excel at finding documents through semantic similarity. By combining these methods, the Ensemble Retriever leverages their complementary strengths to provide more accurate and relevant search results than either algorithm alone [14][15].

3.5.6. Retrieval Evaluation

Since the optimal retrieval parameters depend on the indexing parameters and the data, we tested different combinations of indexing and retrieval using an evaluation tool that evaluates the entire pipeline. The results are presented in chapter 4.

3.6. Reranking

Recent work has shown that the position of the retrieved documents influences the quality of the answer. The research discovered a U-shaped performance curve in large language models. They perform better when relevant information is at the beginning (primacy bias) or the end (recency bias) of the input. However, their performance drops significantly when the information is in the middle of the input. Ranking and reordering the retrieved documents aims to move the most relevant and valuable documents to the top or bottom of the returned list in order to improve the quality of the LLM answer [16].

3.6.1. Long Context Reordering

The Long Context Reordering reorders documents to improve their relevance to a given context. It uses the "Lost in the Middle" strategy implemented into a Long Context Reorder document transformer to move important documents at the beginning or the end and the irrelevant documents in the middle [14][16].

3.6.2. Cohere Reranking

Cohere Rerank improves search results by applying machine learning techniques to understand the semantic meaning of queries and potential results. When a user submits a query, Cohere Rerank first examines a set of candidate documents retrieved by a base retriever. Using deep learning models, it assesses the relevance of each document to the query, taking into account context and semantic relationships. It then reorders the results based on this semantic analysis, ensuring that the most contextually relevant documents appear at the top. This method requires an internet connection and Cohere access [17].

3.6.3. Cross Encoder Reranking

A cross-encoder predicts the similarity between the query and each document by processing them pairwise using the transformer network "BAAI/bge-reranker-base" from HuggingFace. It outputs a score between 0 and 1 indicating how similar the sentences are, with higher scores indicating greater similarity. After sorting, a list of reordered documents is returned [18][19].

3.6.4. Reranking Evaluation

Our goal was to evaluate the various reranking options. However, we encountered difficulties due to repeatedly hitting the API rate limit as we had limited OpenAI resources. Despite this, we believe reranking would be particularly beneficial, especially in scenarios where k>10 and more than 10 documents are retrieved. Therefore, we still want to offer Fraunhofer the opportunity to use these options.

3.7. Generation

In a Retrieval-Augmented Generation (RAG) pipeline, the generation phase is responsible for producing the final output based on the retrieved documents and the user query. This section

delves into the techniques employed to ensure that the generated responses are accurate, relevant, and contextually appropriate.

3.7.1. Large Language Models

For the generation and the routing inside the pipeline, various Large Language Models can be selected. A full list of the available models is displayed in Table 3.3.

Model	Provider	Connection	Cost
GPT-3.5 turbo	OpenAI	Online	US$0.50 / 1M input tokens, US$1.50 / 1M output tokens
	Azure OpenAI	Online	US$0.50 / 1M input tokens, US$1.50 / 1M output tokens
GPT-4	OpenAI	Online	US$30.0 / 1M input tokens, US$60.0 / 1M output tokens
	Azure OpenAI	Online	US$30.0 / 1M input tokens, US$60.0 / 1M output tokens
GPT-4o	OpenAI	Online	US$5.0 / 1M input tokens, US$15.0 / 1M output tokens
	Azure OpenAI	Online	US$5.0 / 1M input tokens, US$15.0 / 1M output tokens
Llama3 7B	Ollama	Offline	Operation Cost
Phi3 3.8B	Ollama	Offline	Operation Cost
Mixtral 8x7B	Ollama	Offline	Operation Cost

Table 3.3.: List of available Large Language Models [6][8].

3.7.2. Prompt Engineering

One of the ways in which one can guide the LLM through the internal knowledge in RAG is prompt engineering.

Effective prompt engineering can significantly improve the performance of LLMs on specific tasks. It is done by providing instructions and contextual information that help guide the model's output. By carefully designing prompts, we can guide the LLM's attention toward the most relevant information for a given task, leading to more accurate and reliable outputs [20].

The following use cases illustrate how prompt engineering plays a vital role in making the generated answers more relevant and of higher quality.

- User Query: *Question about a fabricated faculty at TUM*
  - 👎 Hallucinated answer
  - 👍 *"I don't know"*
- User Query: *Query about a simple fact inside the organization*
  - 👎 Long answer with extraneous context about the fact
  - 👍 Concise answer that is directly relevant to the query
- User Query: *Query that needs internal context to be understood by the LLM e.g. technical jargon used inside the organization*
  - 👎 Either a hallucinated answer or incapacity to answer the query
  - 👍 Personalized answer which can also follow preset formatting rules, tailor-made for the organization's employees

Multiple elements could be taken into consideration when crafting a prompt that maximizes an LLM's capabilities:

- **Context**: The introductory text snippet sets the context for the prompt. It dictates the behavior of the LLM according to the circumstances. It can be defined in the context of an organization (e.g. "You are an assistant of employees at the *company-name* ...")
- **Conciseness**: This point emphasizes the importance of brevity and relevance in the responses. Being concise ensures that the responses are efficient, easy to understand, and without unnecessary information, while relevance ensures that the information provided is useful to the user.
- **Avoid hallucinations**: To maintain the credibility of the assistant, it is important to avoid providing incorrect or fabricated information. If the answer cannot be found in the provided documents, the assistant should explicitly state "I don't know". Paired with the temperature setting, the ground is set for the hallucinations to be minimized.
- **Task types**: Understanding these task types helps the assistant prepare for the different types of queries it may encounter, ensuring that it can handle a range of questions effectively.
- **Answer format**: The assistant can be asked to write answers in a specific format (e.g. in bullet points or numbered lists), provide links for context or maintain a professional tone in its answers for example.
- **Synthesizing Information**: To maintain the completeness of the answer, it is advised to give the LLM instructions on how to deal with multiple documents and documents with complex structures.
- **Chain of thought prompting**: The assistant can either be provided by examples or by a thought process that would help it deal better with a wide range of user queries. In other terms, the user could provide the LLM with a use case, a logical chain of thought, and an expected output. The LLM then attempts to apply this use case to other situations and mimics the reasoning process followed by the user. An example logical chain of thought was passed onto an LLM in the following example 3.1.

Listing 3.1: Example snippet of prompt engineering - chain of thought

```
]
"User_query": What are the projects handled by the FDM
and their impacts on healthcare?
"Retrieved_documents":
- Doc A: FDM works on developing digital tools for medical imaging.
- Doc B: The projects led by FDM improve diagnostic accuracy and reduce
surgery times.
- Doc C: FDM also conducts research on climate change (unrelated to the
question).
- Doc D: the FDM is software for medical diagnostics.

"Desired_output":
```

```
 *** The FDM handles the following projects: ***
- Developing digital tools for medical imaging
- Creating software for medical diagnostics
 *** These projects have the following impacts on healthcare: ***
- Improving diagnostic accuracy
- Reducing surgery times

Refer to Doc A,B and D for further details as Doc C was not related to
the question.
```

3.7.3. Prompt Engineering Evaluation

Evaluating the performance of the LLM is crucial for assessing the quality of its generated outputs. The chosen metrics aim to measure various aspects of the LLM's ability to produce accurate and informative responses. The testing will assess the impact of prompt engineering (PE) and the LLM's proficiency in generating answers using internal knowledge.

A total of 30 use cases per metric were generated by gpt-3.5-turbo according to the testing framework described in 3.7.3 and will be assessed based on selected criteria described in 3.7.3. The results will then be discussed in section 3.7.3.

Testing Framework

We have established a testing framework that simulates a straightforward interaction between the user and the RAG-enhanced LLM. Dummy documents with references are provided to the LLM within this framework, enabling the evaluation of generation quality using a manageable sample size. This approach simplifies testing procedures and facilitates easy modifications.

The testing template itself is structured as follows:

- **Test name and description**: Name of the test (e.g. context fidelity), the metric to be tested, and how it should be evaluated.
- **Test cases**:
  - Question: User query
  - Documents: Dummy internal knowledge supplied to the LLM.
    - page content: dummy content of the document
    - source: dummy link or source

Metrics

To evaluate the quality of the pipeline's output, we have chosen key metrics that are essential for measuring performance. These metrics are crucial for determining the effectiveness and reliability of the results.

- **Correctness**: Measures the accuracy of factual information provided by the LLM in response to queries, assessing performance on single data point retrieval, multiple data point retrieval, and complex fact-based reasoning requiring chains of thought, aiming to quantify how reliably the LLM can access and apply its knowledge base to provide accurate information across various levels of complexity.
- **Precision and Conciseness**: Evaluates how well the LLM's responses directly address the query without extraneous information. An average answer length of all of the outputs in this test case has been calculated to visualize the PE's role in shortening the output of the LLM.
- **Completeness**: Assesses if the output covers all of the facets of the query.
- **Ambiguity Handling**: Evaluates the LLM's ability to manage ambiguities in queries effectively. It measures its reliability against purposefully faulty queries and its ability to find relevant data in documents with possibly contradictory or outdated data.

Results

In our study of generation quality, we compared the behavior of the LLM both with and without PE as guiding principles. Table 3.4 presents the results of our analysis. Since both methods are proficient in retrieving single facts, this sub-criterion is excluded from the overall rating. A star is awarded when 90% or more of the questions in a metric's test set are answered correctly.

Metric		No PE	With PE
Correctness	Single-fact	✓	✓
	Multi-fact or multi-step	✗∗	✓
Precision and Conciseness	Average answer length (tokens)	157.3	87.4
	Succinct output	✗	✓
Completeness	Query satisfaction	✓	✓
Ambiguity Handling	Ambiguity-proof answers	✗	✗∗∗
	Hallucinated answers	46.7%	16.7%
Overall Rating		⋆⋆	⋆⋆⋆⋆

Table 3.4.: Results for the quality of the Generation, comparing Performance with and without Prompt Engineering. Based on different Criteria, we provide an overall qualitative Rating from one ($\star$) to five Stars ($\star\star\star\star\star$).

Conclusion

The implementation of prompt engineering (PE) significantly improves the generation quality across various criteria, as evidenced by the following key points:

- **Correctness**: While not using PE does well enough in this aspect, enhancing the LLM with it improves its performance when requiring a chain of thought to generate an answer.
- **Precision and Conciseness**: The average answer length is reduced from 147.3 to 87.4 tokens with PE, indicating that PE enables the LLM to generate more concise and precise responses. This is also synonymous with cost reduction when calling an API, since the consumption of usage tokens is greatly reduced.
- **Completeness**: Query satisfaction remains consistent with and without PE, suggesting that all of the questions raised by the query are answered regardless of the usage of PE.
- **Ambiguity Handling**: The use of PE significantly enhances the LLM's ability to deal with faulty or hallucination-prone queries. It predominantly refrains from generating information when the answer is unknown, rather than fabricating responses. Nonetheless, hallucinations still occur in some scenarios, suggesting the need for further refinement. Notably, better LLM models exhibit fewer hallucinations. For instance, GPT-4 demonstrated an impressive hallucination rate of 3.4 % when tested on ambiguity handling.

In summary, while PE greatly enhances the system's capabilities, particularly in handling ambiguous queries or providing a concise output, it is not entirely foolproof and requires ongoing improvements to address issues like hallucinations. Furthermore, the modularity of the pipeline enables the use of different LLMs depending on the task at hand.

3.8. Query Preprocessing: Routing and Enhancement

The standard pipeline can be further optimized by implementing routers. These routers preprocess queries to improve comprehension and reflect on retrieved documents and generated responses. They can potentially increase accuracy, reduce costs by determining RAG necessity and provide safeguards against misuse.

3.8.1. Routing

Routers serve two primary functions: they refine the state by modifying queries or retrieved documents, or they act as decision nodes, allowing certain pipeline stages to be skipped or repeated. Figure 3.2 illustrates an overview of the routing process. Routers primarily utilize LLMs for decision-making. Smaller and cheaper LLMs can be used for the queries because of their simplicity.

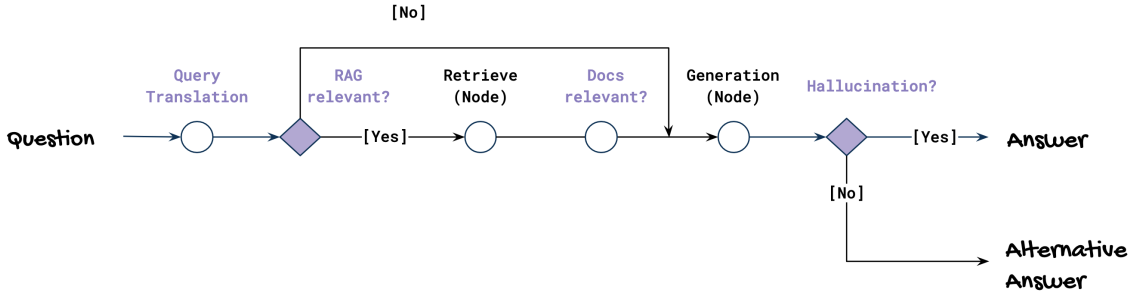

Figure 3.2.: Overview of Routers used in the pipeline. Routers are shown in purple font.

3.8.2. Query Translation

Misunderstandings are a common occurence between the user and the LLM since the quality of a user prompt has a direct impact on the generation quality of the LLM [21]. Such disparities can lead to unexpected outputs, which makes query translation a vital part of this project.

In the RAG pipeline, query translation is aimed at refining user inputs to optimize their clarity and precision. This process involves transforming user queries to correct any spelling or grammatical errors and removing ambiguities, thus ensuring the queries are concise yet comprehensive. The translated query retains the original intent while being formatted for optimal processing by subsequent components of the RAG pipeline.

Techniques and tools: It is common knowledge that LLMs are proficient at correcting texts [22]. Consequently, query translation was done using a *translator* LLM that takes a non-refined user query and a system prompt as inputs and generates a typo- and ambiguity-free query, then it is passed on to the subsequent components of the pipeline. One of the methods that can be used to make this possible is the Rephrase and Respond (RaR) method [23]. This method involves rewording, elaborating on the query and correcting it for typos so that it's more understandable for an LLM. This method proved to be very efficient and had a positive impact on the quality of the answers generated by the LLM.

An example of a typical query translation workflow and the potential misinterpretations that could happen due to an ambiguous user query is displayed in figure 3.3.

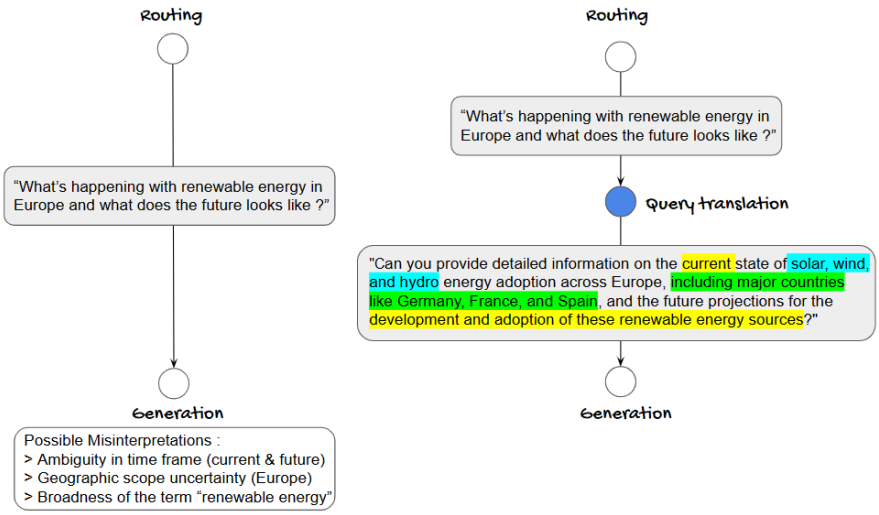

Figure 3.3.: Typical Workflow With and Without Query Translation

Further examples of Query Translation:

- User Query: *"Plase show me the latests news about teh weather."*
  - Translated Query: "Please show me the latest news about the weather."
- User Query: *"Find stuff about AI"*
  - Translated Query: "Show me recent advancements in artificial intelligence."
- User Query: *"what does fraunhofer do in bio engineering "*
  - Translated Query: "What is the Fraunhofer Society's involvement in the field of bioengineering?"

By refining user queries in this manner, the RAG pipeline can retrieve more relevant documents and generate more accurate responses, thus improving the quality of the generated answer.

3.8.3. RAG Relevance

Many user queries do not necessitate internal knowledge. Employing RAG for every question increases response times and computational costs. An initial router evaluates whether a query requires internal knowledge or if the LLM can provide an answer independently. If internal knowledge is considered to be unnecessary, the RAG pipeline is bypassed entirely. For example, a question about Fraunhofer's applications of lasers will use RAG but a question about GitHub won't require any internal documents from Fraunhofer and will therefore skip the rest of the pipeline.

3.8.4. Document Relevance

This router assesses the relevancy of retrieved documents to the query. As this evaluation is performed by an LLM rather than retrieval techniques, it may yield different results and improve document selection. Documents considered irrelevant can be discarded, saving

computing resources and streamlining the result sources. For example, the question "What laser applications does Fraunhofer have in Germany?" might retrieve documents about climate change, since they also reference Germany, even though they are unrelated to the question. The router will then determine if there are any documents related to the question, e.g. about lasers, and if not, send the question without documents to the LLM.

3.8.5. Hallucination Detection

LLMs are prone to hallucination [24]. Given that the RAG pipeline provides source documents, the LLM should stick to the facts presented and avoid relying on its own knowledge or fabricating information. After response generation, this router verifies that all information is stated in the provided documents. This can be done with an LLM, using the prompt from the listing 3.2. If hallucinations are detected, the response can be discarded and an alternative answer can be returned or the pipeline can be re-executed with different documents or a rephrased query. Typical hallucinations include dates and addresses. When asked for the address of Fraunhofer's headquarters in Bhutan, the LLM might return a realistic answer, despite there not being a Fraunhofer office in Bhutan. The router will detect that no documents passed to the LLM contain this address and tell the user that a hallucination has been detected.

Listing 3.2: Example Prompt: Hallucination Detection Router

```
You are an expert at determining if a question is based on facts and is not
hallucinated. You will receive a question, potentially some documents and a
response.
Answer "True" if: if the answer to the question is grounded in documents or the
answer was correctly answered without the documents
Answer "False" if the answer conflicts with the information given in the
document or is impossible to be answered correctly
```

3.8.6. Router Evaluation

The routers are tested against a set of questions to determine their correctness, which helps us understand which routers are effective. They also provide insight into whether the use of a router can lead to quality degradation. Each router has individual questions and their expected answers, e.g. "What was the excel formula for conditionals?" - no RAG needed. They are then asked the test questions and their answer is compared to the expected answer. The results are evaluated based on key metrics.

Metrics

Two metrics are used to evaluate the effectiveness of the pipeline. They are used to determine whether a router is responding correctly to requests.

- **TP-Rate**: Represents the percentage of correctly classified *positive* instances.
- **TN-Rate**: Represents the percentage of correctly classified *negative* instances.

LLM	RAG Relevance		Document Relevance		Hallucination	
	TP-Rate	TN-Rate	TP-Rate	TN-Rate	TP-Rate	TN-Rate
GPT-3.5 turbo	100%	100%	100%	57%	17%	100%
LLaMA3 7B	34%	100%	100%	17%	50%	86%
Phi3 3.8B	100%	100%	100%	57%	34%	86%
Mixtral 8x7B	100%	100%	100%	57%	0%	86%

Table 3.5.: Results of Testing the Routers.

Conclusion

The statistics of the evaluation results can be seen in the table 3.5. What can be clearly seen is the high probability of choosing the more conservative answers. The RAG Relevance router models have a very high chance of correctly identifying positive and negative responses, except for Llama3. The Document Relevance routers all have very high chance of correctly identifying documents that are relevant to the question but a TN-Rate of less then 60% in the test cases. While many unnecessary documents are not recognized, all relevant documents are recognized, which does not lead to a decrease in RAG quality. The same applies to the hallucination router. If there is no hallucination, the models detect it with an accuracy of 86% or higher in the tests. If there is hallucination, the router detects it no more than half the time. This results in only a small loss of quality, since factual information is rarely classified as a hallucination. However, the effectiveness of the router is low because it has difficulty detecting hallucinated information.

3.9. Guardrails

Guardrails are critical mechanisms ensuring the safe, ethical, and appropriate use of Large Language Models (LLMs), particularly in enterprise environments. This section outlines a comprehensive guardrail system designed to mitigate risks associated with LLM interactions. The guardrail system acts as the entry and exit point of the LLM pipeline, independent of the specific implementation of the pipeline. Figure 3.4 provides a schematic overview of the guardrail system's architecture.

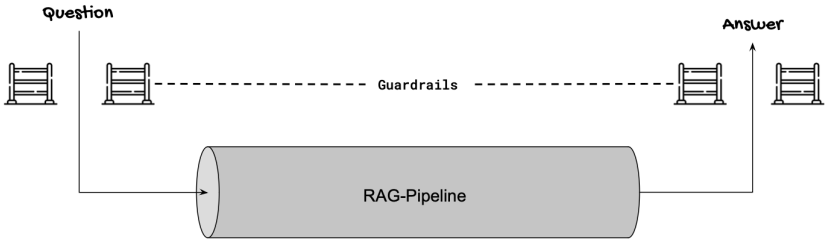

Figure 3.4.: Guardrails Example

3.9.1. Guardrail Blockers

The first use case of guardrails is to prevent unwanted queries from entering the RAG pipeline. We implemented two types of LLM-based blockers to achieve this:

1. **PII Blocker**: Identifies and stops queries containing Personally Identifiable Information.
2. **Non-Work Blocker**: Filters out queries unrelated to work tasks.

These blockers are based on LLMs that take the user query as input together with the instructions to classify this input as either *True* or *False,* depending on the type of content this guardrail should block. The following text is an example of the prompt that is used for the PII Blocker.

Listing 3.3: Example Prompt: PII Blocker

```
You are an expert at determining if a question for a Fraunhofer employee
contains personal identifiable information.
Answer: "True" if the question contains names, phone numbers, email adresses
or information of any kind related to a private person.
Answer: "False" if the question does not contain any personal information
related to a private person.

EXAMPLES:

TEXT:
Can you please draft an email for my boss telling him that I will be late?
RESPONSE:
False

TEXT:
Can you please draft an email for my boss Peter Lanz?
RESPONSE:
True

TEXT:
Is Professor Dr. Holger Hanselka still the President of Fraunhofer?
RESPONSE:
False

TEXT:
Can you send a meeting invite to j.schmidt@partner-company.com for our
collaboration?
RESPONSE:
True

OUTPUT INSTRUCTIONS:
Only answer "True" or "False" for the following case
```

```
TEXT:
{{User Query}}
RESPONSE:
```

3.9.2. Guardrail Blocker Evaluation

To evaluate the effectiveness of the implemented guardrail blockers, we conducted comprehensive tests using different LLMs to assess their capability in detecting PII and non-work related content. Our evaluation employed two key performance metrics: True Positive Rate (TP-Rate) and True Negative Rate (TN-Rate) for both PII and Non-Work detection. TP-Rate indicates the model's ability to correctly identify instances containing PII or non-work related content, while TN-Rate represents the model's accuracy in identifying instances without such content. Table 3.6 summarizes our findings, revealing consistently high accuracy across most models. Notably, open-source models such as Phi-3 3.8B demonstrate performance comparable to proprietary models like GPT-3.5 turbo. Our practical experience with the pipeline aligns with these results, indicating that the blockers exhibit high accuracy in real-world scenarios. Misclassifications, when they occur, are typically limited to edge cases where the content's nature is inherently ambiguous, and multiple interpretations could be justified.

LLM	**PII Detection**		**Non-Work Detection**	
	TP-Rate	TN-Rate	TP-Rate	TN-Rate
GPT-3.5 turbo	100%	92%	100%	94%
LLaMA3 7B	89%	92%	100%	89%
Phi-3 3.8B	95%	92%	90%	100%
Mixtral 8x7B	79%	100%	90%	100%

Table 3.6.: Performance of Various LLMs in Detecting PII and Non-Work Related Content

3.9.3. Anonymization

The anonymization component handles sensitive information within queries and responses:

1. **Anonymization Process**: Replaces PII (names, phone numbers, email addresses, credit card numbers) with fake but realistic values. Two strategies are available for the anonymization process: Presidio [25] and LLM-based anonymization.

2. **De-anonymization Process**: Restores original PII in the LLM's output. Two strategies are available: combined exact-fuzzy matching, and LLM-based de-anonymization.

A schematic representation of the anonymization and subsequent deanonymization process is outlined in Figure 3.5.

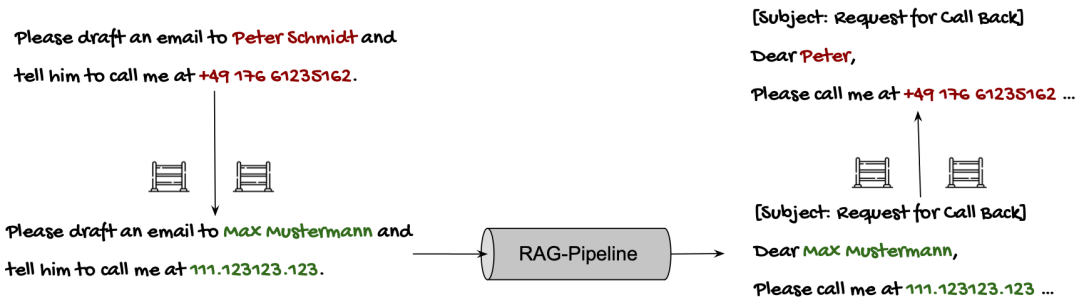

Figure 3.5.: Example of Anonymization and Deanonymization Process.

3.9.4. Anonymization Evaluation

To evaluate the performance of the anonymization and subsequent de-anonymization processes, we tested multiple combinations of methods, as outlined in Table 3.7. Our findings indicate that LLM-based methods consistently outperform other non-LLM-based approaches for both anonymization and de-anonymization tasks. This superior performance can be attributed to several factors:

1. **Flexibility in Name Recognition**: Presidio, while effective for American conventions, struggles with names and phone numbers from other cultures. LLM-based methods demonstrate greater adaptability to diverse naming conventions.

2. **Contextual Understanding**: Unlike Presidio, which indiscriminately anonymizes every name it encounters (including entities like "Fraunhofer" or public figures like "Barack Obama"), LLM-based methods can differentiate between sensitive personal information and widely known names. This distinction is crucial for maintaining the context and answering ability of the system.

3. **Robustness to Name Variations**: Non-LLM de-anonymization methods often falter when names are presented in different forms (e.g., "Peter Schmidt" becoming "Mr. Schmidt"). LLM-based approaches show greater resilience to such variations.

4. **Contextual De-anonymization**: LLM-based methods excel in understanding the context of anonymized text, leading to more accurate restoration of original information.

Anonymization Method	De-Anonymization Method	Accuracy
Presidio	Presidio	55%
Presidio	GPT-3.5 turbo	73%
Presidio	LLaMA3 7B	73%
Presidio	Phi-3 3.8B	73%
Presidio	Mixtral 8x7B	73%
GPT-3.5 turbo	GPT-3.5 turbo	**91%**
Llama3 7B	Llama3 7B	82%
Phi-3 3.8B	Phi-3 3.8B	**91%**
Mixtral 8x7B	Mixtral 8x7B	82%

Table 3.7.: Accuracy Comparison of Various Anonymization and De-Anonymization Method Combinations

3.9.5. Limitations

While these guardrails significantly enhance data privacy and appropriate use, they are not infallible. They should be used in conjunction with other security measures and not relied upon as the sole means of protecting sensitive information or ensuring appropriate LLM use.

3.10. User Interface

A user interface is provided to interact with the pipeline. This can be used for a demo or as a standalone application. The user can ask questions and a conversation history is displayed. Sources are shown as cards below the answers with a title, date of last access, and the content provided to the LLM. The user interface and its features can be seen in figure 3.6. The backend is implemented in Flask and the frontend doesn't rely on further dependencies.

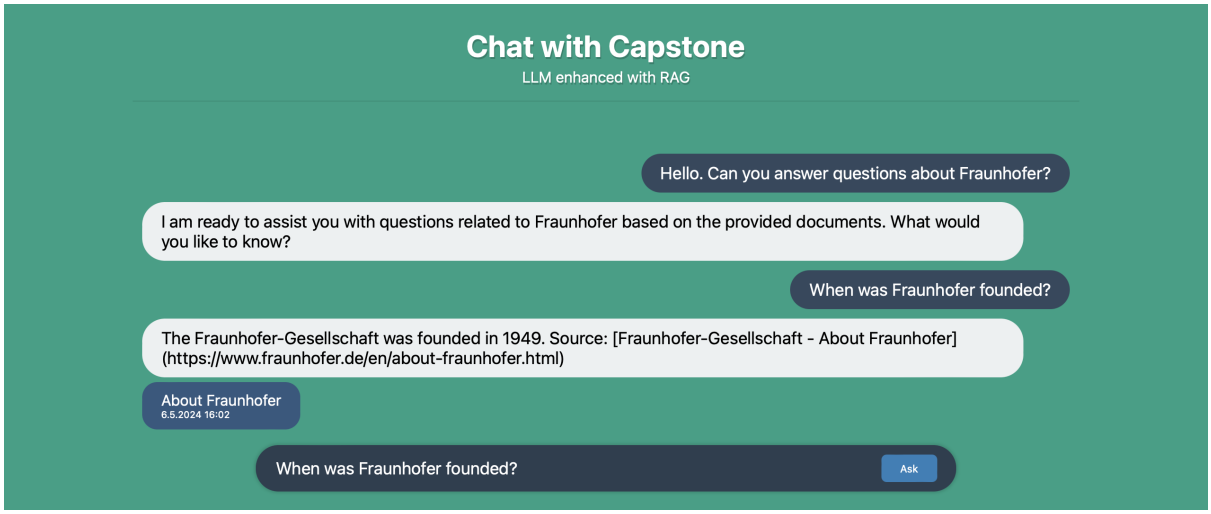

Figure 3.6.: User Interface to communicate with data.

4. Pipeline Evaluation

To evaluate the indexing and retrieval modules, we use a dataset designed to answer predefined questions. As described in Chapter 2, the corpus for the RAG pipeline is obtained from the Fraunhofer Main Page. The Q&A dataset is divided into two categories: single fact questions and multi-fact questions. This classification helps us test how well the RAG pipeline handles questions that require either a single piece of information or multiple pieces of information. Different combinations of indexing and retrieval methods result in different context chunks and thus different answers. These contexts and answers are evaluated using several metrics that involve a Large Language Model that compares the contexts or generated answers to the ground truth.

4.1. Metrics

To evaluate the performance of the entire pipeline, the pipeline responses were assessed on four main metrics:

- **Context Recall**: Evaluates the extent to which the retrieved context aligns with the ground truth for every sentence in the ground truth [26].
- **Context Precision**: Measures how much of the provided context is useful for arriving at the ground truth answer. Chunks similar to the ground truth should ideally appear at the top of the retrieved context [27].
- **Faithfulness**: Determines if the generated answer can be derived from the retrieved context. Significant deviations and inconsistencies would suggest hallucination [28].
- **Answer Relevancy**: Assesses how effectively the generated answer answers the initial question [29].

These metrics provide a more comprehensive and holistic overview of pipeline performance for both indexing and retrieval, as well as the subsequent response generation from the LLM. It complements the detailed metrics for different sections of the pipeline.

4.2. Evaluation Dashboard

A dashboard was implemented to enhance the visualization and monitoring of pipeline evaluation results. It enables users to upload an input.csv file containing the Q&A dataset, select the configuration of the different indexing and retrieval parameters, run the pipeline, and view metrics presented using heatmap or as a table for detailed results. Additionally, the results can be stored and retrieved for later access and comparison.

The dashboard was developed using the Streamlit framework. We added user authentication using Firebase, ensuring a secure access to the dashboard. Furthermore, MongoDB is used to store the results of every pipeline run, providing a scalable and efficient solution for managing the evaluation data. The data is sorted based on average evaluation score, allowing users to easily view the top performing configurations or search for a specific one.

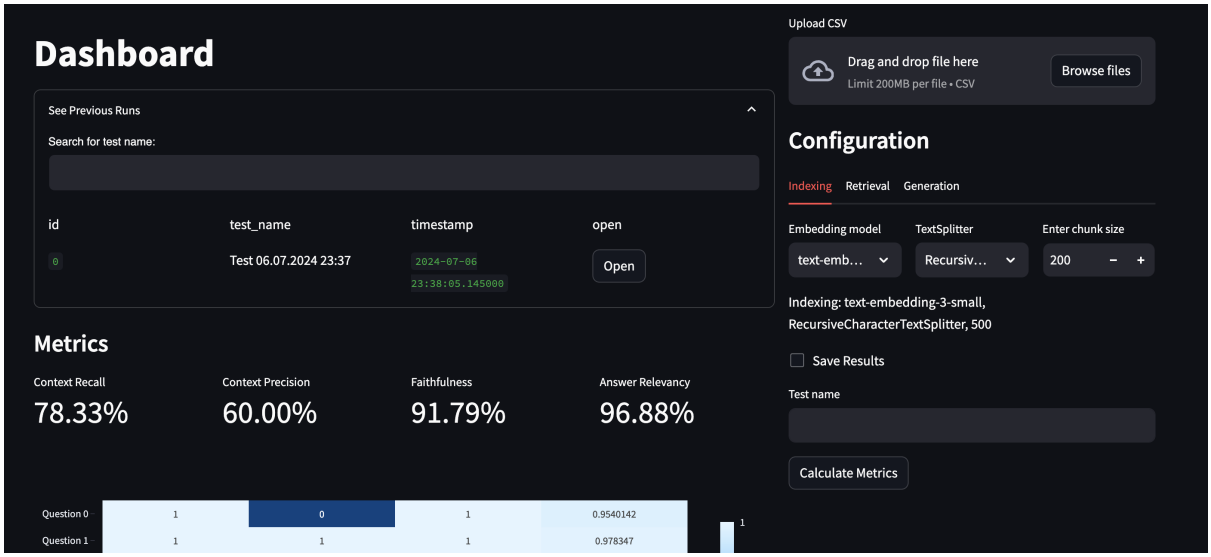

Figure 4.1.: Streamlit Dashboard for evaluation of different configurations.

4.3. Results

In this section, we present evaluations of various indexing and retrieval combinations to provide an overview of which methods perform better than others.

Embeddings

Table 4.1 shows the results of testing the embedding models discussed in section 3.3.2. To generate these results, we only changed the embedding model between runs and froze all other parameters. We split the text using the Recursive Character Text Splitter with a chunk size of 1000 characters. Then we used the Nearest Neighbor Retriever to find the top 5 similar documents. We tested different embedding models for our RAG pipeline to determine which model best embeds text to retrieve the most relevant context. The primary metrics used for evaluation were context recall and precision. By focusing on these metrics, we aimed to improve the quality of the retrieved context to ensure more accurate and relevant contexts to answer the questions. The all-mpnet-base-v2 model from Huggingface had the highest context precision. The text-embedding-ada-002 model from OpenAI had the highest context recall. Overall, considering both recall and precision, the text-embedding-ada-002 model delivered the best performance, with the highest averaged precision and recall scores.

Model	Dimension	Context Precision	Context Recall	Average
text-embedding-ada-002	1536	62%	**73%**	**67.5%**
text-embedding-3-small	1536	50%	55%	52.5%
text-embedding-3-large	3072	57%	50%	53.5%
all-mpnet-base-v2	768	**64%**	66%	65%

Table 4.1.: Results of testing the different Embedding Models presented in section 3.3.2. Indexing and retrieval parameters were frozen, only the embedding model was changed.

Indexing and Retrieval

After identifying the optimal embedding model, we evaluated different indexing and retrieval combinations using context precision and recall metrics. The results of this evaluation are shown in table 4.2. In particular, for the ensemble retriever with a chunk size of 2000, the OpenAI token limit was exceeded, resulting in no obtainable results for this test. The lack of an upward trend indicates that increasing the chunk size or the number of documents returned does not generally improve the metrics. Among the basic retrievers (Nearest Neighbor and SVM), the SVM outperforms the Nearest Neighbor. However, the setup of the SVM (training an SVM for each query) causes it to take about 20 times longer to retrieve the data, making it less practical. For the advanced retrievers, the ensemble significantly outperformed the other retrievers, regardless of number of documents returned. This supports the hypothesis that combining a keyword-based retriever with a context-based retriever yields better results than using either approach alone. By leveraging the strengths of both methods, we can improve the accuracy and relevance of retrieval results.

	Recursive, Chunk size= 500	Recursive, Chunk size=1000	Recursive, Chunk size=2000
Retriever	K=5	K=10	K=5
Nearest Neighbor / Base	49%	33.5%	48.5%
Support Vector Machine	66.5%	73.5%	**58%**
Contextual Compression	46%	40.5%	44%
Multi-Query	55%	54%	52.5%
Ensemble	**74.5%**	**74%**	-

Table 4.2.: Average of Context Precision and Recall for different Indexing-Retrieval Combinations. Tables for each metric are shown in the Appendix A.1.

Chunking

We also tried to evaluate the different chunking methods described in section 3.3.1, but ran into OpenAI's token limitations during database creation and evaluation. The results we were able to obtain are shown in the appendix A.2. Without a complete set of results, we cannot effectively evaluate whether Semantic Chunking is better than or equivalent to Recursive Character Text Splitting.

Conclusion

For the Fraunhofer Main Page, we recommend using the text-embedding-ada-002 embedding model. For chunking the documents, we suggest the Recursive Character Text Splitter with a

chunk size of 1000. For document retrieval, an ensemble retriever that returns 5 documents has shown the best results.
Finally, these parameters are guidelines and may vary depending on the specific data used. It's important to re-evaluate them with your own data using the provided evaluation dashboard.

5. Future Outlook

To improve the RAG pipeline in the future, we have gathered several ideas to focus on:

- **Expansion to multilingual support** Currently, our RAG pipeline is fine-tuned to handle English text data, with all evaluation metrics based on English textual inputs. As a potential direction for future development, multilingual support with special attention on German data is considered.
- **Integration of multimodal data** Another area for exploration is the integration of multimodal capabilities, enabling the pipeline to process text, images, and other types of data inputs seamlessly.
- **Hallucinations** While efficient prompt engineering, query manipulation, and the application of low-temperature settings (if applicable) significantly lessen hallucinations, they do not entirely eradicate them. Further measures can be employed to address the residual occurrences. Two promising approaches are the integration of an additional LLM to verify output accuracy, and the incorporation of knowledge graphs, which is a structured representation of information that captures relationships between entities in a graph format. Integrating these methods allows the LLM to cross-reference its generated content with the graph's verified data, significantly reducing the chances of hallucinations. While this approach may introduce additional overhead and complexity, the trade-off is a net improvement in the accuracy and reliability of the LLM's outputs, making it a worthwhile consideration for potential additions to the pipeline.
- **User interface** The user interface is kept simple for demonstration purposes only. Many features are still missing for a production environment, such as the ability to select an LLM, having a different conversation for each user and the ability to save them, as well as file support. The pipeline could also be integrated into an existing user interface or application.
- **Evaluation Framework** The evaluation metrics used in the dashboard can be extended, which currently primarily assess the performance for indexing and retrieval. Further metrics from other parts of the pipeline, such as generation, can also be integrated to give a fuller picture through a single platform.

6. Reflection

In retrospect, the project was a great learning opportunity for all of us. We closely collaborated with the client, assuming industry roles and a professional work setting. Incorporating Scrum and Sprints into our workflow established a structured routine that significantly improved the team's efficiency and organization. During the Sprint Review, we received beneficial feedback from the client that helped us refine our approach and we were able to prioritize tasks important to them.

Throughout the project, we faced several unforeseen challenges, each with multiple solutions and each carrying its own set of advantages and disadvantages. This necessitated us to make decisions collectively while carefully considering all factors. For example, during testing with different chunking options, the OpenAI token limit per minute often made it difficult to obtain useful results. Errors frequently occurred towards the end of the testing, failing to generate results and using a significant amount of money.

Additionally, managing the dependencies and different frameworks used for modularity in our pipeline was challenging, especially with eight team members working simultaneously. This often led to significant merge conflicts. Through this experience, we learned that effective communication between team members is critical to preventing major merge conflicts and avoiding duplication of work. This often resulted in conflicts within the dependency trees that needed to be resolved.

Finally, making the pipeline modular and offering various options increased the implementation complexity and resulted in dead code antipatterns. This occurred because we typically used only the best configuration for our data, leaving other options unused but still in the codebase, requiring maintenance.

In the future, enhancing the organization and timely access to OpenAPI would be advantageous, as the delay in access posed a challenge, necessitating reliance on our own resources initially. Additionally, having clearer requirements from the client from the start would significantly improve project outcomes. Early in the project, we faced uncertainty and had limited direction, which made it difficult to revise major decisions, such as language support, later on.

Overall, we believe we have gained valuable knowledge and experience. This encompasses both technical skills and our roles as Product Manager, Scrum Master, Cloud Architect, and more. For some team members, it was a new experience, while others deepened their understanding of specific aspects. As a team, we have improved our decision-making, communication, and prioritization strategies. We are now more proficient at handling and resolving issues effectively and we have developed a strong understanding of how to create a robust RAG pipeline.

A. Pipeline Evaluation Results

A.1. Context Precision and Recall for Indexing-Retrieval Combinations

Retriever	Recursive, Chunk size= 500 K=5	Recursive, Chunk size=1000 K=10	Recursive, Chunk size=2000 K=5
Nearest Neighbor / Base	48%	35%	47%
Support Vector Machine	72%	**80%**	**60%**
Contextual Compression	48%	36%	50%
Multi-Query	59%	56%	52%
Ensemble	**83%**	67%	-

Table A.1.: Context Precision for different Indexing-Retrieval Combinations using text-embedding-3-small.

Retriever	Recursive, Chunk size= 500 K=5	Recursive, Chunk size=1000 K=10	Recursive, Chunk size=2000 K=5
Nearest Neighbor / Base	50%	32%	50%
Support Vector Machine	61%	67%	**56%**
Contextual Compression	44%	45%	38%
Multi-Query	51%	48%	53%
Ensemble	**66%**	**81%**	-

Table A.2.: Context Recall for different Indexing-Retrieval Combinations using text-embedding-3-small.

A.2. Context Precision and Recall for Chunking Methods

Chunking	Nearest Neighbor, K=5	Ensemble, K=5	Ensemble, K=10
Recursive, Chunk size= 500	58%	**72%**	63%
Recursive, Chunk size=1000	**73%**	65%	**70%**
Recursive, Chunk size=1500	56%	66%	58%
Semantic, percentile	64%	65%	-

Table A.3.: Context Precision for different Chunking Methods using text-embedding-ada-002

Chunking	Nearest Neighbor, K=5	Ensemble, K=5	Ensemble, K=10
Recursive, Chunk size= 500	53%	**69%**	72%
Recursive, Chunk size=1000	**62%**	66%	**79%**
Recursive, Chunk size=1500	47%	60%	-
Semantic, percentile	-	-	-

Table A.4.: Context Recall for different Chunking Methods using text-embedding-ada-002.

Bibliography

[1] L. Richardson. "Beautiful soup documentation". In: *April* (2007).

[2] Unstructured. *Unstructured Documentation*. https://docs.unstructured.io/welcome. (Accessed on 07/07/2024).

[3] LangChain. *LangGraph*. https://langchain-ai.github.io/langgraph/. LangGraph Documentation, Accessed: 2024-07-04.

[4] LangChain. *Recursively split by character*. https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/recursive_text_splitter/. Accessed: 2024-07-08.

[5] G. Kamradt. *Text Splitting*. https://retrieval-tutorials.vercel.app/document-loaders/text-splitting/. Accessed: 2024-07-08.

[6] OpenAI. *Pricing*. https://openai.com/api/pricing/. Accessed: 2024-07-04.

[7] OpenAI. *Embeddings*. https://platform.openai.com/docs/guides/embeddings/what-are-embeddings. Accessed: 2024-07-04.

[8] Microsoft Azure. *Azure OpenAI-Dienst – Preise*. https://azure.microsoft.com/de-de/pricing/details/cognitive-services/openai-service/. Accessed: 2024-07-04.

[9] Hugging Face. *sentence-transformers/all-MiniLM-L6-v2*. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. Accessed: 2024-07-04.

[10] Hugging Face. *sentence-transformers/all-mpnet-base-v2*. https://huggingface.co/sentence-transformers/all-mpnet-base-v2. Accessed: 2024-07-04.

[11] D. Dixit. *Advanced RAG series: Indexing*. https://div.beehiiv.com/p/advanced-rag-series-indexing. Accessed: 2024-07-08.

[12] Labelbox. *How vector similarity search works*. https://labelbox.com/blog/how-vector-similarity-search-works/. Accessed on July 06, 2024.

[13] A. Karpathy. *kNN vs. SVM*. https://github.com/karpathy/randomfun/blob/master/knn_vs_svm.ipynb. GitHub repository, Accessed: 2024-07-04.

[14] H. Chase. *LangChain*. Oct. 2022. URL: https://github.com/langchain-ai/langchain.

[15] A. Trotman, A. Puurula, and B. Burgess. "Improvements to BM25 and Language Models Examined". In: *Proceedings of the 19th Australasian Document Computing Symposium*. ADCS '14. Melbourne, VIC, Australia: Association for Computing Machinery, 2014, pp. 58–65. ISBN: 9781450330008. DOI: 10.1145/2682862.2682863. URL: https://doi.org/10.1145/2682862.2682863.

[16] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. *Lost in the Middle: How Language Models Use Long Contexts*. 2023. arXiv: 2307.03172 [cs.CL]. URL: https://arxiv.org/abs/2307.03172.

[17] Cohere. *Rerank on LangChain*. https://docs.cohere.com/docs/rerank-on-langchain. Accessed on July 05, 2024.

[18] S. Xiao, Z. Liu, P. Zhang, and N. Muennighoff. *C-Pack: Packaged Resources To Advance General Chinese Embedding*. 2023. arXiv: 2309.07597 [cs.CL].

[19] N. Reimers and I. Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. arXiv: 1908.10084 [cs.CL]. URL: https://arxiv.org/abs/1908.10084.

[20] Microsoft. *Prompt Engineering*. https://learn.microsoft.com/en-us/ai/playbook/technology-guidance/generative-ai/working-with-llms/prompt-engineering. Microsoft Learn Documentation, Accessed: 2024-07-04.

[21] OpenAI. *Best practices for prompt engineering with the OpenAI API*. https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api. Accessed: 2024-07-04.

[22] Xiaowu Zhang, Xiaotian Zhang, Cheng Yang, Hang Yan, Xipeng Qiu. *Does Correction Remain A Problem For Large Language Models?* https://arxiv.org/pdf/2308.01776. Accessed: 2024-07-04.

[23] Yihe Deng, Weitong Zhang, Zixiang Chen, Quanquan Gu. *Rephrase and Respond: Let Large Language Models Ask Better Questions for Themselves*. https://arxiv.org/abs/2311.04205. Accessed: 2024-07-04.

[24] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung. "Survey of hallucination in natural language generation". In: *ACM Computing Surveys* 55.12 (2023), pp. 1–38.

[25] Microsoft. *Presidio: Data Protection and De-identification*. https://github.com/microsoft/presidio. 2024.

[26] Ragas. *Context Recall*. https://docs.ragas.io/en/stable/concepts/metrics/context_recall.html. Accessed: 2024-07-07.

[27] Ragas. *Context Precision*. https://docs.ragas.io/en/stable/concepts/metrics/context_precision.html. Accessed: 2024-07-07.

[28] Ragas. *Faithfulness*. https://docs.ragas.io/en/stable/concepts/metrics/faithfulness.html. Accessed: 2024-07-07.

[29] Ragas. *Answer Relevance*. https://docs.ragas.io/en/stable/concepts/metrics/answer_relevance.html. Accessed: 2024-07-07.