MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Private LLM for querying internal documentation and knowledge base

Bachelor's Thesis

JAKUB MEDOVIČ

Brno, Spring 2025

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Private LLM for querying internal documentation and knowledge base

Bachelor's Thesis

JAKUB MEDOVIČ

Advisor: doc. Bruno Rossi, PhD

Department of Computer Systems and Communication

Brno, Spring 2025


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

During the preparation of this thesis, I used the following AI tools:

- Grammarly for grammar check
- ChatGPT to improve my writing style

I declare that I used these tools in accordance with the principles of academic integrity. I checked the content and took full responsibility for it.

Jakub Medovič

Advisor: doc. Bruno Rossi, PhD

Acknowledgements

I would like to sincerely thank my thesis advisor, Doc. Bruno Rossi, PhD, and my thesis consultant, Štěpán Kuchař, for their time and quick responses when needed. Next, I would like to thank my family for their support when I needed it and for providing me with great experiences. And finally, thanks to all my friends, such as Tomáš, Michal, Jakub, Tomáš and others, who provided me with support and help, as well as motivation and great evenings together.

Abstract

This thesis presents the design, implementation, and evaluation of a private large language model (LLM) chatbot for the Adacta company, aimed at enabling secure and efficient querying of their internal documentation. To address data privacy concerns associated with public LLMs, a Retrieval-Augmented Generation (RAG) pipeline was implemented using locally hosted components, including LMStudio, ChromaDB, and transformer-based embedding models, with the understanding that a cloud-based solution will be used in practice.

The system processes Adacta's markdown documentation using contextual chunking, allowing users to ask natural language questions and receive grounded responses. A reranking module was also integrated to improve retrieval precision. While reranking improved contextual relevance, it increased latency, leading to a toggleable option in the user interface.

The final solution, evaluated by the RAGAS framework, demonstrates that private, RAG-based LLMs can effectively support enterprise knowledge access, offering a secure alternative to commercial AI services.

Keywords

Large Language Models (LLMs), Retrieval Augmented Generation (RAG), Embedding, Reranking, Secure Documentation Search, Internal Knowledge Retrieval

Contents

List of Tables

List of Figures

1 Introduction

Large Language Models (LLMs) have become a transformative innovation in artificial intelligence, particularly in the field of natural language processing (NLP). These advanced models, such as OpenAI's ChatGPT [1], Google's BERT [2], and Meta's LLaMA [3], have enabled computers to process and generate human language with remarkable precision and fluency. Their applications span various industries, including healthcare, customer support, and finance, where they assist with tasks such as answering questions, summarizing data, and generating meaningful content. LLMs derive their strength from their ability to handle large-scale data, understand complex contexts, and adapt to diverse language patterns. Despite their growing utility, their use in the processing of sensitive or proprietary data introduces significant challenges [4], especially when commercially available models are used in such scenarios.

The aim of this bachelor's thesis is to explore the use of private LLMs for secure and efficient information retrieval within an organizational setting. Public LLMs, while powerful, are not always suitable for tasks involving proprietary documentation due to concerns related to data privacy, security, and compliance with regulations, since many of these models use their prompts and data for further training. This thesis focuses on addressing these challenges by evaluating existing private LLM tools and selecting the most appropriate one for the needs of Adacta, a company that has extensive and varied internal documentation. The chosen tool will be used to train and implement a chatbot capable of accessing Adacta's knowledge base, which is stored in markdown format. This chatbot will allow employees to retrieve information efficiently using natural language queries, providing a more intuitive and effective alternative to traditional keyword-based search methods.

To achieve this goal, the thesis adopts a systematic approach. The first step involves analyzing the current state of private LLM technologies and evaluating their capabilities in terms of efficiency, information reliability, and security. Following this evaluation, one tool is selected for implementation. The practical phase of the work includes testing the selected model on Adacta's documentation, deciding the handling

and embedding of data with different strategies, and implementing it as a prototype chatbot tailored to the organization's specific needs.

The structure of this thesis is as follows. In Chapter 2, I present the theoretical foundation necessary for understanding the development and evaluation work. This includes an overview of Large Language Models, embedding techniques, reranking mechanisms, and the evaluation framework employed. Chapter 3 details the practical development process, including the initial experimentation, design decisions, and implementation strategies for the chatbot application. Chapter 4 presents the experimental evaluation of the system, comparing various model configurations and parameter choices based on quantitative performance metrics, while explaining my hardware limitations. Finally, Chapter 5 offers a summary of the findings and proposes directions for future improvements and applications of the system within Adacta and similar organizational settings.

In summary, this thesis bridges the gap between the potential of LLMs and their practical application in proprietary data environments. By focusing on private LLMs, the work highlights their value in overcoming limitations associated with public commercial models and provides a structured approach to their deployment within an organizational/private context. The outcomes of this research aim to contribute to the operational efficiency of Adacta and the broader applications of AI in private/enterprise environments.

2 Background

Firstly, I will lay out the terms and technologies that I will be using in this thesis. My aim is to evaluate possible models and ways in which to create a private chatbot to query Adacta's internal documentation; therefore, I will be using Large Language Models (LLMs), embeddings, chunking, and reranking, all in order to implement a functional Retrieval Augmented Generation (RAG) chatbot.

2.1 Large Language Models (LLMs)

Large Language Models (LLMs) are advanced machine learning models trained to understand and generate human language. They are based on the transformer architecture introduced by Vaswani; Shazeer; Parmar; Uszkoreit; Jones; Gomez; Kaiser; Polosukhin [5] in *Attention Is All You Need*, which relies heavily on self-attention mechanisms to process language in a non-sequential manner. This architectural innovation allows LLMs to grasp contextual relationships between words and even across long distances within a text. This is very relevant for any advanced natural language processing applications.

The emergence of models such as BERT [6], GPT-3 [7], and more recently LLaMA [8] has revolutionized the capabilities of natural language processing (NLP). These models are pre-trained on massive corpora that include books, web pages, encyclopedias, and more, making them highly versatile across tasks.

LLMs can perform a wide range of NLP tasks with minimal fine-tuning, including question answering, summarization, translation, and even code generation. They learn complex syntactic and semantic patterns, allowing them to generalize beyond their training data. This power, however, comes with privacy risks when deployed in organizational contexts: commercial LLMs may store or use user prompts for further training, posing privacy and compliance issues.

To address this, organizations increasingly turn to private, self-hosted LLMs. These include open-source models, such as LLaMA 2 [9], Deepseek [10], and Hermes [11], which can be deployed in secure environments, ensuring data never leaves the organization's infrastructure. While private models may require more resources to

deploy and maintain, they offer enhanced data governance, compliance with regulations (e.g., GDPR), and complete control over model behavior. The central aim is to determine the extent to which private LLMs can provide secure, accurate, and context-aware information retrieval for Adacta's internal documentation.

This involves a critical comparison of several open-source LLMs based on factors such as model performance, resource efficiency, deployment complexity, and integration potential with retrieval-augmented architectures. By deploying the chosen model in a controlled, self-hosted environment, the thesis demonstrates how organizations like Adacta can harness the capabilities of modern language models without compromising data privacy or operational integrity.

2.2 Embeddings

Embeddings are compact, dense vector representations of text that encode semantic meaning by mapping words, phrases, or entire documents into a continuous high-dimensional space. In this space, the proximity between vectors reflects the semantic similarity of the underlying text. Unlike traditional keyword-based representations, such as term frequency-inverse document frequency (TF-IDF) [12] or bag-of-words models, which only capture surface-level word frequency without understanding semantics, embeddings are capable of representing nuanced relationships such as analogies, synonymy, and contextual dependencies.

Initial breakthroughs in embeddings came from Word2Vec [13], which introduced shallow neural networks trained to predict context words (Skip-Gram) or center words (CBOW) in a local window. This approach enabled vectors to encode syntactic and semantic relationships, such as king $-$ man $+$ woman $\approx$ queen. GloVe [14], another early model, used global word-word co-occurrence statistics to build word vectors, capturing global corpus structure. However, both models generate static embeddings, meaning that each word has a single vector regardless of its use in different contexts, a significant limitation for polysemous words (e.g., "bank").

To address this, contextual embeddings emerged with the development of ELMo [15], which leverages bidirectional long-short-term

memory (LSTM) [16] trained on a language modeling objective. ELMo enables word representations to vary depending on the surrounding sentence, capturing rich contextual meaning. This idea was extended further with BERT [6], a transformer-based model pretrained using masked language modeling and next sentence prediction. BERT provides token-level embeddings that consider the full sentence context, significantly improving performance across various NLP benchmarks.

Building on BERT, Sentence-BERT (SBERT) [17] modified the architecture to produce sentence-level embeddings suitable for tasks like semantic similarity, clustering, and information retrieval. SBERT uses Siamese and triplet networks to fine-tune BERT on sentence-pair classification tasks, enabling efficient cosine similarity comparisons between fixed-size sentence vectors.

Recent advances have further refined embedding generation for large-scale retrieval. Models such as all-MiniLM-L6-v2 [18], MPNet [19], E5 [20] and BGE-M3 [21] combine efficient architectures with domain adaptation strategies. MiniLM and MPNet optimize for speed and small model size without sacrificing semantic quality, while the E5 (Embedding for Embedding-based Retrieval) model family uses weakly supervised contrastive learning from large web corpora, specifically aligning queries and passages for retrieval tasks. In contrast, BGE-M3 introduces task instructions during training (e.g., "Represent this sentence for searching relevant passages:"), which guides the model to generate embeddings tailored to specific downstream applications, improving generalization and robustness across diverse retrieval tasks.

2.3 Chunking

Chunking refers to the process of dividing a large body of text into smaller, manageable segments that can be independently processed by language models. This is particularly important when dealing with transformer-based models, such as LLMs, which have a limited input size and are typically capped at a few thousand tokens. When documents exceed this limit, they must be split into smaller pieces that retain semantic coherence to preserve the context and meaning during processing.

While the most elementary chunking algorithms simply link constituent parts based on elementary search patterns, approaches that use machine learning techniques (classifiers, topic modeling, etc.) can take contextual information into account and thus compose chunks in such a way that they better reflect the semantic relations between the fundamental constituents. That is, these more advanced methods get around the problem of combinations of elementary constituents having different higher-level meanings depending on the context of the sentence.

Effective chunking strategies are critical for applications such as retrieval-augmented generation (RAG) and semantic search. Poor chunking can lead to information fragmentation or loss of context, both of which degrade the performance of downstream NLP tasks. Standard techniques include splitting by headings, paragraphs, or sentences, often with a sliding window to maintain context overlap between adjacent chunks. The choice of chunk size and overlap are empirical, balancing the need for detailed information with chunk size. For example, with markdown files, splitting by headings and perhaps adding metadata based on the document structure and description into each chunk is good. This adds a lot of context to the entirety of the document chunks, but at the cost of adding tokens to all chunks.

2.4 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is a hybrid approach that combines the benefits of information retrieval with the language generation capabilities of LLMs. Instead of generating responses based solely on internal model weights, RAG retrieves relevant text passages from an external knowledge base to ground the model's outputs in factual content. This reduces hallucinations and increases the accuracy and relevance of responses, particularly in domain-specific or knowledge-intensive contexts. It modifies interactions with LLMs so that the model responds to user queries concerning a specified set of documents, using this information to supplement information from its pre-existing training data. The term RAG was first introduced in 2021 by Lewis; Perez; Piktus; Petroni; Karpukhin; Goyal; Küttler; Lewis; Yih;

Rocktäschel; Riedel; Kiela in their research paper *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* [22].

The RAG architecture typically involves three components: a retriever, an encoder-decoder model, and a generator. First, the retriever identifies top-k relevant documents or text chunks using vector similarity. Next, these documents are concatenated with the query and passed into the LLM, which generates an answer based on both the question and retrieved context. This architecture enables dynamic updates to the knowledge base without requiring model retraining, making it especially useful in enterprise settings.

In the context of this thesis, RAG is implemented using a vector search system to retrieve markdown content related to employee queries at Adacta. The retrieved chunks serve as grounding context for the private LLM, ensuring that responses reflect actual internal documentation. This improves trust in the AI assistant's output and ensures alignment with organizational knowledge.

2.5 Vector Storage

Vector storage is a crucial component in retrieval-augmented systems, enabling the efficient indexing and retrieval of dense vector representations of text, known as embeddings. Once documents or queries are transformed into embeddings using a sentence transformer or similar model, these vectors must be stored in a format that allows for rapid similarity comparison. Unlike traditional databases, which use exact key matching, vector stores perform approximate nearest neighbor (ANN) searches to locate embeddings, which makes them perfect for embedding retrieval.

Vector databases allow the system to retrieve semantically similar documents even when exact word matches are absent, making them fundamental to search systems that rely on meaning rather than keywords. They use distance metrics such as cosine similarity or Euclidean distance to rank and return results. These stores are designed to handle high-dimensional vectors at scale and are commonly used in applications such as semantic search, recommendation engines, and question-answering systems.

In this thesis, the chosen vector database was ChromaDB [23], a modern open-source vector storage solution designed for embedding persistence, similarity search, and metadata tagging. ChromaDB allows users to store high-dimensional vectors and retrieve the most semantically similar ones given a query embedding. It supports multiple indexing strategies under the hood and is optimized for local or lightweight production environments.

ChromaDB also integrates seamlessly with Python-based NLP pipelines, including LangChain and Hugging Face transformers. In the context of this project, each document chunk was stored in ChromaDB alongside metadata such as the source file name and section title. This metadata is beneficial not only for returning relevant answers but also for enabling traceability and explainability in the chatbot interface. As embeddings are updated or retrained, ChromaDB supports efficient reindexing and management, making it simple to change documents and metadata and handle changes made to documents.

2.6 Reranking

Reranking is the process of reordering the results produced by an initial retrieval step to improve their relevance. Unlike an embedding model, a reranker model uses question and document as input and directly outputs similarity instead of embedding. You can get a relevance score by inputting a query and a passage to the reranker. And the score can be mapped to a float value in [0,1] by a sigmoid function. In most RAG systems, the first retrieval stage relies on vector similarity, which may prioritize semantically similar content but lacks deeper contextual understanding. Rerankers use more computationally intensive models, such as cross-encoders or sequence-to-sequence transformers, to assess and sort the top-k candidates based on a more nuanced comparison between the query and the document.

Reranking is especially effective in reducing noise in the top-ranked results and refining answer quality. It often involves scoring retrieved passages with a BERT-based model trained to predict relevance labels. In large-scale search systems like ColBERT [24] or MonoT5 [25], reranking significantly boosts both recall and precision. It also sup-

ports interpretability, as scores can be used to justify or filter model outputs.

This thesis uses reranking to ensure that the most relevant markdown chunks are passed to the LLM. After initial retrieval, candidate documents are rescored using a lightweight FlagReranker [21] model fine-tuned from a study on RAG. This second-layer filter helps reduce irrelevant content and improves answer precision, especially when many documents share overlapping terminology.

2.7 Evaluation with RAGAS Framework

Evaluating the quality and reliability of responses in RAG systems requires a nuanced approach that goes beyond traditional metrics like the BLEU [26] score. The Retrieval-Augmented Generation Assessment Suite (RAGAS) [27] framework provides a structured methodology for evaluating various dimensions of performance in these hybrid systems. This thesis adopts RAGAS to assess the quality of the developed LLM-based chatbot using four primary metrics: answer relevancy, context recall, context precision, and faithfulness.

2.7.1 Answer Relevancy

Answer relevancy [28] measures how well the generated response addresses the user's original query. This is typically assessed through human annotation or semantic similarity scoring between the query and the model's response. In this case, this metric is calculated using the user_input and the response as follows:

1. Generate a set of artificial questions (default is 3) based on the response. These questions are designed to reflect the content of the response.

2. Compute the cosine similarity between the embedding of the user input (Eo) and the embedding of each generated question (Egi).

3. Take the average of these cosine similarity scores to get the Answer Relevancy:

$$\text{Answer Relevancy} = \frac{1}{N} \sum_{i=1}^{N} \text{cosine similarity}(E_{g_i}, E_o) \tag{2.1}$$

$$\text{Answer Relevancy} = \frac{1}{N}\sum_{i=1}^{N}\frac{E_{g_i}.E_o}{\|E_{g_i}\|\|E_o\|} \tag{2.2}$$

Where: - Egi: Embedding of the ith generated question. - Eo: Embedding of the user input. - N: Number of generated questions (default is 3).

2.7.2 Context Recall

Context recall [29] evaluates whether the retrieved passages include all necessary information to generate a complete and accurate response. LLMContextRecall is computed using user_input, reference, and the retrieved_contexts, and the values range between 0 and 1, with higher values indicating better performance. This metric uses reference as a proxy to reference_contexts, which also makes it easier to use, as annotating reference contexts can be very time-consuming. To estimate context recall from the reference, the reference is broken down into claims, and each claim in the reference answer is analyzed to determine whether it can be attributed to the retrieved context or not. In an ideal scenario, all claims in the reference answer should be attributable to the retrieved context. The formula for calculating context recall is as follows:

$$\text{Context Recall} = \frac{\text{Number of claims in the reference supported by the retrieved context}}{\text{Total number of claims in the reference}} \tag{2.3}$$

2.7.3 Context Precision

Context Precision [30] assesses how much of the retrieved information is actually used in the final answer. It is calculated as the mean of the precision@k for each chunk in the context. Precision@k is the ratio of the number of relevant chunks at rank k to the total number of chunks at rank k.

$$\text{Context Precision@K} = \frac{\sum_{k=1}^{K}(\text{Precision@k} \times v_k)}{\text{Total number of relevant items in the top K results}} \tag{2.4}$$

$$\text{Precision@k} = \frac{\text{true positives@k}}{\text{true positives@k}+\text{false positives@k}} \tag{2.5}$$

Where K is the total number of chunks in retrieved_contexts and $v_k \in \{0, 1\}$ is the relevance indicator at rank k. To estimate if a retrieved context is relevant or not, this method uses the LLM to compare each of the retrieved contexts or chunks present in retrieved_contexts with the reference. High precision implies that the system retrieves only relevant and helpful content, reducing cognitive noise and hallucinations. This is crucial for enterprise settings where irrelevant data can lead to confusion or misinformation.

2.7.4 Faithfulness

Faithfulness [31] (also referred to as groundedness or factual alignment) measures whether the retrieved context supports the content of the generated response. Faith-

fulness is particularly important in RAG systems because LLMs can hallucinate plausible, but incorrect information. Recent works, including "Survey of Hallucination in Natural Language Generation" [32] by Ji; Lee; Frieske; Yu; Su; Xu; Ishii; Bang; Madotto; Fung, emphasize faithfulness as a key metric for trustworthy language generation. A response is considered faithful if all its claims can be supported by the retrieved context. To calculate this:

1. Identify all the claims in the response.
2. Check each claim to see if it can be inferred from the retrieved context.
3. Compute the faithfulness score using the formula:

$$\text{Faithfulness Score} = \frac{\text{Number of claims in the response supported by the retrieved context}}{\text{Total number of claims in the response}} \tag{2.6}$$

Together, these metrics provide a comprehensive evaluation of both the retrieval and generation components of the system. They also guide iterative improvements by revealing where the pipeline may fail, whether due to poor retrieval, insufficient context usage, or hallucinated answers.

3 Process and Methodology

This section outlines the process and implementation methodology adopted during my development of the private LLM-based chatbot. It presents the progression from initial research through to deployment, highlighting the technical decisions, limitations, and tools explored across each phase of the project.

3.1 Evaluating Private LLM Options

The first step in the methodology involved researching viable methods for running private large language models. Initial considerations included cloud-based providers like Microsoft Azure, OpenAI, and Cohere, as well as on-premise solutions that could be run locally. Azure offers a secure, scalable option for deploying OpenAI models via its OpenAI Service, which integrates with GPT-3.5 and GPT-4 as well as updates their library to include new models as they come out, such as the new GPT-4.1. However, due to constraints in billing and authentication, particularly issues with payment method verification, my usage of these platforms was limited.

Cohere has a good selection of capable models for RAG and also has reranking and embedding models, with their privacy policy [33] being suitable for a private RAG application; however, using their Software as a Service (SaaS) platform directly could be limiting in terms of future use and model changes.

OpenAI, as one of the biggest and most widely known AI companies [34], would not be a bad choice. Their Application Programming Interface (API) has compatible data processing terms [35] with private AI solutions and has very performant models available [36] in both language models and embedding models. Their model list does not include any reranking-specific models, unlike Cohere or Azure's AI platform. It is also specific to OpenAI models only, which could be limiting in the future.

Azure AI Services satisfy all of the privacy requirements for working with private documentation [37], and also include a long list of different models available to deploy in various regions, from many providers, including the previously mentioned Cohere and OpenAI, or newer Deepseek. It has multiple options for all types of models, and its catalog [38] is being updated with the latest models as they come out, ready for deployment. This makes it a more flexible option than the others, since they include chat, embedding, and reranking models in the most significant variety. After a consultation with the Adacta consultant Štěpán Kuchař, we concluded that Azure AI Services would be the best suited for use in their office setting, since they also already use Azure services outside of this application.

Despite this, I was not able to properly test the application within the Azure environment, due to previously mentioned billing and card verification problems. This led to a strong focus on testing with self-hosted solutions, while understanding that the final product will be hosted on Azure infrastructure.

3.2 Running Local Models with LMStudio and Azure GPT-3.5

To circumvent commercial restrictions and my problems with card verification, I opted for local deployment using LMStudio [39], a GUI-based tool that allows users to download and run quantized transformer models (e.g., LLaMA 2, Hermes) on their own hardware. LMStudio supports models in GPT-Generated Unified Format (GGUF) [40] and includes an API interface, making it suitable for testing retrieval-augmented pipelines without relying on third-party services. While local models were helpful for offline experimentation, `gpt-3.5-turbo-instruct` on Azure (under a free account and limitations) was intermittently used for comparison and later LLM-based evaluation tasks. However, it suffered from limited availability due to the account constraints.

3.3 Developing a Flask API for Inference

To standardize access to models, whether local or cloud-based, I created a lightweight Python Flask API [41]. This API wraps around the model inference process, allowing requests to be made from the frontend or from testing scripts. The modular structure of the API makes it adaptable for switching between different models (e.g., LMStudio, OpenAI API), and it supports response streaming to enable real-time interaction, mimicking the responsiveness of modern chatbot User Interfaces (UIs).

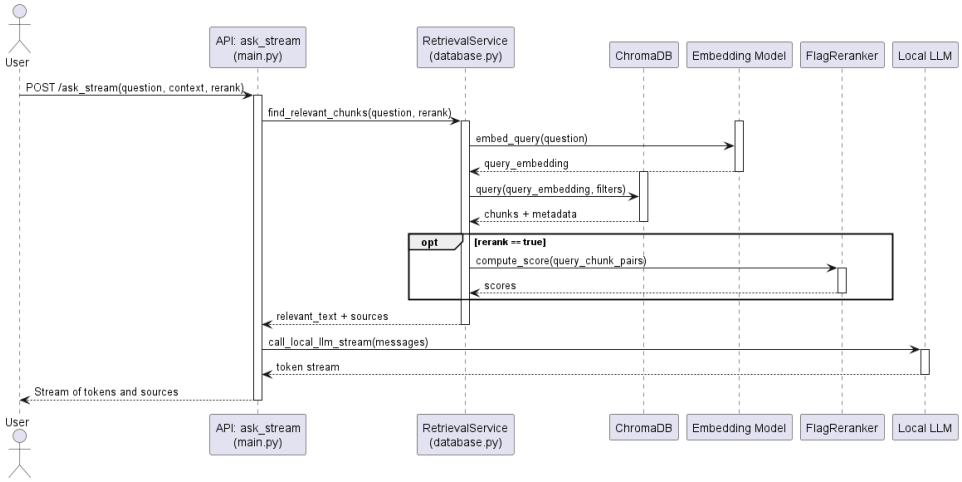

Figure 3.1: Sequence Diagram of my ask_stream API call

Additionally, I implemented two separate API endpoints: one for non-streaming queries and another for retrieval-only operations that return relevant document sources without invoking the language model. The non-streaming API was primarily used for testing and evaluation, where real-time output was unnecessary. The retrieval-only endpoint served similar purposes and was also integrated into the user interface to allow users to locate relevant files or documentation segments using natural language queries, without generating complete responses or manually searching the documentation.

3.4 Embedding Exploration and FlagEmbedding

For the retrieval component, I examined multiple embedding strategies. A particularly impactful discovery was FlagEmbedding, as proposed by Zhang; Xiao; Liu; Dou; Nie in *Retrieve Anything To Augment Large Language Models* [42]. This family of embeddings, trained via contrastive learning with both multilingual and domain-agnostic robustness, showed superior retrieval performance when benchmarked against more commonly used models like all-MiniLM or MPNet. FlagEmbedding models from the associated GitHub repository [21] were tested and compared in terms of speed and semantic alignment. Their open-source nature made them especially practical for integration into local private pipelines.

3.5 Chunking Strategies

Document chunking was a critical part of the retrieval pipeline. Initial attempts involved simple character-based chunking, where markdown documents were sliced into fixed-length segments. This approach, though easy to implement, often disrupted the semantic flow of the content. The next iteration employed semantic sentence chunking, using sentence tokenizers to ensure linguistic completeness in each chunk. However, specific files, like those mainly containing HTML, since the documentation is generated into a web format, proved challenging. Ultimately, I adopted a hybrid approach: contextual markdown chunking was used as the primary strategy, supported by sentence- and character-based chunking as fallback methods in cases of irregular formatting.

3.6 Embeddings Creation and Persistence with ChromaDB

I adopted ChromaDB as the vector database to store and retrieve embeddings efficiently. It supports persistent storage, similarity search, and metadata tagging-all essential for scalable retrieval. ChromaDB's simple API integration and ability to persist across sessions made it a reliable backend for embedding storage and lookups during both development and testing. Metadata such as file paths and

document titles were also stored alongside vectors to enhance traceability in chatbot responses.

Based on metadata, I also separated a part of the files concerning past versions of the application documents into a separate collection since they were affecting results by showing some information from older versions. In addition to that, I added a filter that ignores files that are composed of more than 50% HTML tags. This is due to the fact that some of the files in the database were generated to be directly interpreted as a webpage, but this kind of information does not provide information that can be well interpreted by LLMs and uses far too many tokens.

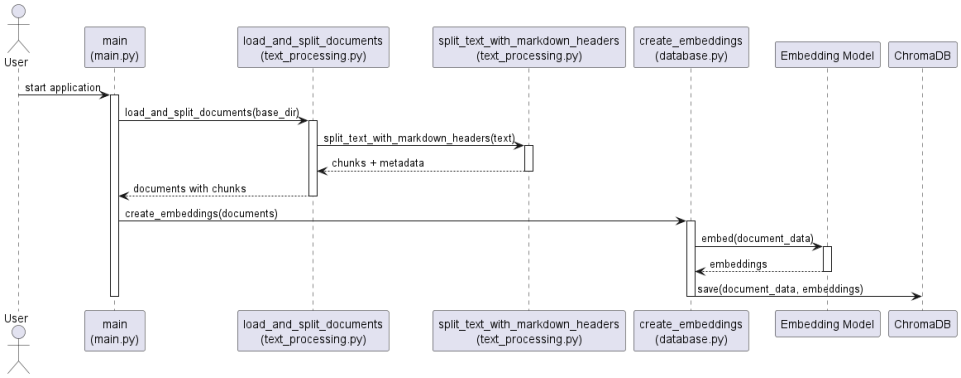

Figure 3.2: Sequence Diagram of my main function that creates embeddings

The application loads the documents on main start and creates embeddings for any files not found in the database or those that have changed since, as shown in the figure above(Figure 3.2).

3.7 UI Development and Streaming

I created a minimalistic user interface (UI) to support interaction with the system. Built with streamlit [43], the UI connects to the Flask backend and enables users to input natural language queries. The interface supports response streaming(Figure 3.1), allowing the user to see the model's output token by token, closely mimicking the behavior of commercial chat models like ChatGPT. This interactivity enhances usability and perceived responsiveness. To further enhance user experience, I added options that allow the user to control what they want to happen when they ask a question. These options include:

- including or excluding conversation history
- enabling or disabling reranking for context retrieval,
- another endpoint that only returns sources in context of the question without LLM input

These features enhance the user experience by giving the user a simple and easy way to control how the app handles their question and what they receive from it without added complexity. Disabling reranking may lower source quality, but it improves response times. Disabling conversation history saves on tokens and prevents the LLM from being confused by previous information sources, and the sources endpoint gives the user a quick way to search for the documentation pages relevant to their question in natural language without waiting for an LLM.

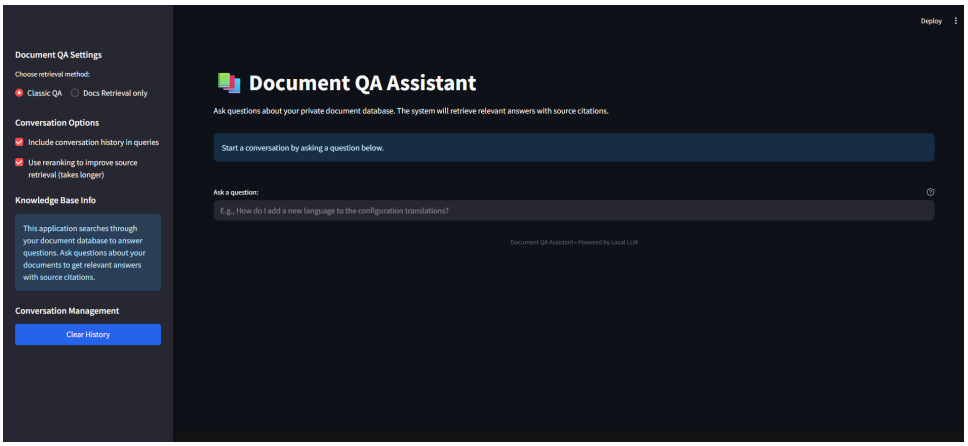

Figure 3.3: Screenshot of the frontend of the application

3.8 Evaluation with RAGAS

To evaluate the overall performance of the retrieval-augmented system, I used the RAGAS framework using `gpt-3.5-turbo-instruct` as the evaluator LLM, since it was the only compatible free model on Azure. This is not ideal, since it is a model from 2023 and its performance is inferior to newer ones, but it can still provide a consistent metric better than what my local models can do.

RAGAS provides a set of standardized metrics- Answer Relevancy, Context Recall, Context Precision, and Faithfulness- which collectively assess both the retrieval and generation phases. By comparing generated answers to the original user queries and source chunks, the evaluation helped identify where the system excelled and where it needed adjustments in retriever tuning or chunking strategies. In addition to these, I evaluated the completion token mean values to simulate the possible costs for questions.

For the evaluation, I gathered questions from the frequently asked questions sections of the documentation, as well as example questions and responses provided by the company consultant. In total, the number of evaluation questions was 63 for the RAGAS pipeline.

4 Experimental Evaluation and Results of the RAG Pipeline

This chapter presents a systematic evaluation of the Retrieval-Augmented Generation (RAG) pipeline developed in this thesis. The goal is to determine which combination of components—embedding models, reranking mechanisms, and local language models—yields the most accurate and efficient results for querying Adacta's internal documentation using natural language.

I conducted all experiments using a constrained local hardware setup: a laptop equipped with an NVIDIA GeForce RTX 3050 Ti GPU (4GB VRAM) and then evaluated by `gpt-35-turbo-instruct` via the RAGAS framework. These limitations significantly influenced the selection of models and parameter configurations, especially for tasks involving reranking and inference with larger LLMs.

The evaluation is based on these RAGAS framework metrics:

- Answer Relevancy(Section 2.7.1)
- Context Recall(Section 2.7.2)
- Context Precision(Section 2.7.3)
- Faithfulness(Section 2.7.4)

These metrics help isolate the effectiveness of each pipeline stage and measure its individual and combined contributions to system performance. The overall score metric was calculated by taking a weighted average of these metrics with weight of 0.3 for faithfulness and answer relevancy, and 0.2 for context recall and precision. I chose these weights in order for the scores to better reflect the usefulness of the given answer rather than the context, since the relevant parts of it might not be properly utilized.

4.1 Embedding models

The first important thing to determine was the embedding model to use. Without good embeddings, the retrieved contexts will not be relevant enough to the questions asked, lowering the overall effectiveness of the RAG application.

I ran tests with local model `hermes-3-llama-3.1-8b` [44] with the same parameters in 2 different combinations:

- 1. chunk size = 512
- 1. overlap = 50
- 1. top_k = 7

2. chunk size = 256
2. overlap = 64
2. top_k = 10

I ran this evaluation on five different embedding models:

- `BAAI-bge-m3` [45]
- `mxbai-embed-large-v1` [46]
- `all-minilm-l6-v2-embedding` [18]
- `nomic-embed-text-v1.5` [47]
- `snowflake-arctic-embed-m-v1.5` [48]

I chose to specifically showcase performance in the context recall metric in addition to the overall scores. This is because it is the more important metric when measuring the effectiveness of embedding models. Other metrics, such as answer relevance, are influenced more by the LLM's usage of the context provided.

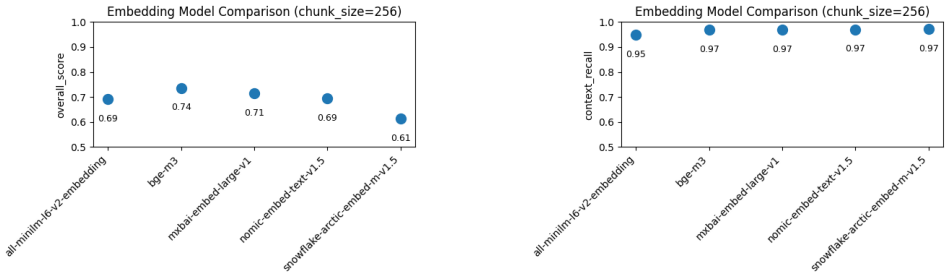

Figure 4.1: Comparison of embedding context recall and overall scores with the first set of parameters

As we can see from the above figure (Figure 4.1), the embedding model `bge-m3` performed the best out of the selected models in these specific conditions, even if the difference was only 0.01 in the overall score when rounded to 2 decimal places.

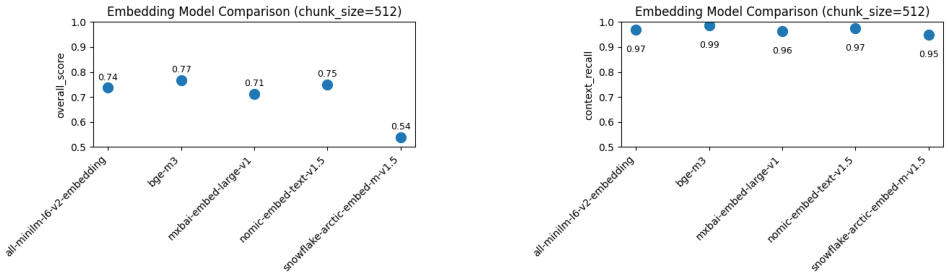

Figure 4.2: Comparison of embedding context recall and overall scores with the second set of parameters

In this second test in figure 4.2, we can see that again it was very close with these embedding models, this time even in the thousands. Still, the most consistent model out of them in marginally better performance was `bge-m3`. Therefore, this is the model I decided to include for local app usage.

4.2 Reranking

Initially, the system relied solely on vector similarity to retrieve the top-k relevant chunks from the knowledge base. While this approach offered fast response times and acceptable performance in general cases, it occasionally returned chunks that were only loosely related to the user's query. This became especially evident in scenarios involving subtle semantic distinctions, where a surface-level similarity failed to reflect the actual informational value of the document.

To improve the quality of retrieval, I integrated a reranker model. The one I chose was FlagReranker from the *Retrieve Anything To Augment Large Language Models* [42] study's GitHub repository. It is a lightweight, high-performance reranking model designed to refine top-k retrievals by more precisely evaluating query–chunk relevance. After embedding-based retrieval, the app retrieves more results from the database and then FlagReranker(`bge-reranker-large`) scores each candidate chunk and reorders the list based on semantic alignment with the original query, before returning the top-k chunks. I tested this approach on multiple embedding model bases with the same chunk size and overlaps to check that the improvements are not specific to the model from FlagEmbeddings. Here are the results:

Empirical testing using the RAGAS evaluation framework confirmed only a marginal performance improvement. For specific questions, however, context recall improved significantly in cases where multiple chunks were topically related but only one contained the exact answer. This was the case with questions such as "How can I dynamically hide UI elements?". There are multiple methods in which a user can hide UI elements, but the way to do it dynamically is in one file, specifically, which was not being returned before.

However, reranking came at a cost: increased latency, particularly in a local setup. Running FlagReranker on a consumer-grade GPU sometimes added up to 20 seconds per query, depending on the number of candidate chunks and system load. This introduced noticeable delays during interactive use, which, while acceptable in evaluation settings, could impact user experience in production.

To address this, I implemented a toggle feature in the UI, allowing users to enable or disable reranking as needed. When speed and responsiveness are prioritized, reranking can be turned off. Conversely, for high-precision needs, reranking can be activated to enhance answer quality. This flexible approach allows the system to adapt to varying user preferences and task criticality, balancing performance with usability.

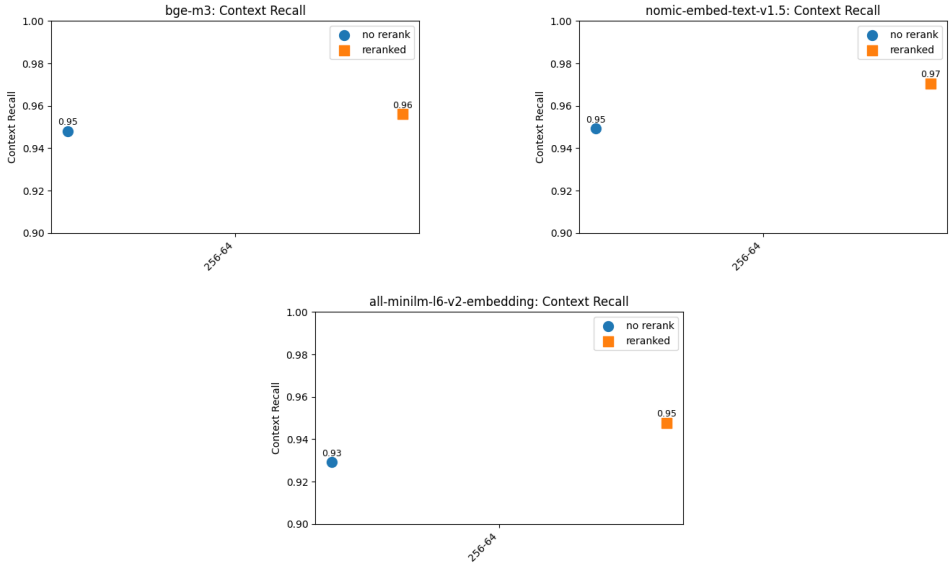

Figure 4.3: Comparison of the models context recall performance before and after reranking (*mind the small scale*)

4.3 Local Models

After selecting an appropriate embedding model and determining that incorporating a reranker improved overall answer quality despite its added computational cost, I turned my attention to evaluating local language models. The objective was to identify which open-source models would be most suitable for inference while delivering acceptable performance across RAGAS evaluation metrics such as Answer Relevancy, Context Precision, and Faithfulness.

To this end, I decided on five models for testing, all of which are quantized versions intended for efficient local inference:

- `deepseek-r1-distill-qwen-7b` [49]
- `dragon-mistral-7b-v0` [50]
- `gemma3-1b-it-qat` [51]
- `hermes-3-llama-3.1-8b` [44]
- `llama-3-8b-instruct-finance-rag` [52]

I selected these models based on recommendations by the LM Studio desktop application, which takes hardware capabilities into account. As such, I focused on 7B and 8B parameter models that had been quantized for GGUF-based inference and could deliver responses within acceptable timeframes for interactive use.

I again ran test on these models in 2 sets of conditions:

- 1. chunk size = 256
- 1. overlap = 64
- 1. top_k = 10

2. chunk size = 512
2. overlap = 50
2. top_k = 7

The purpose of this comparison was not only to assess which model produced the most accurate and contextually grounded responses, but also to measure the resource usage of selected models. All models were evaluated using the same system pipeline and chunking parameters, and their outputs were assessed using the RAGAS framework. The metrics obtained helped inform the default model choice for the local deployment variant of the application. The above configurations were chosen as a standardized baseline to ensure fairness across model comparisons. Here are the results:

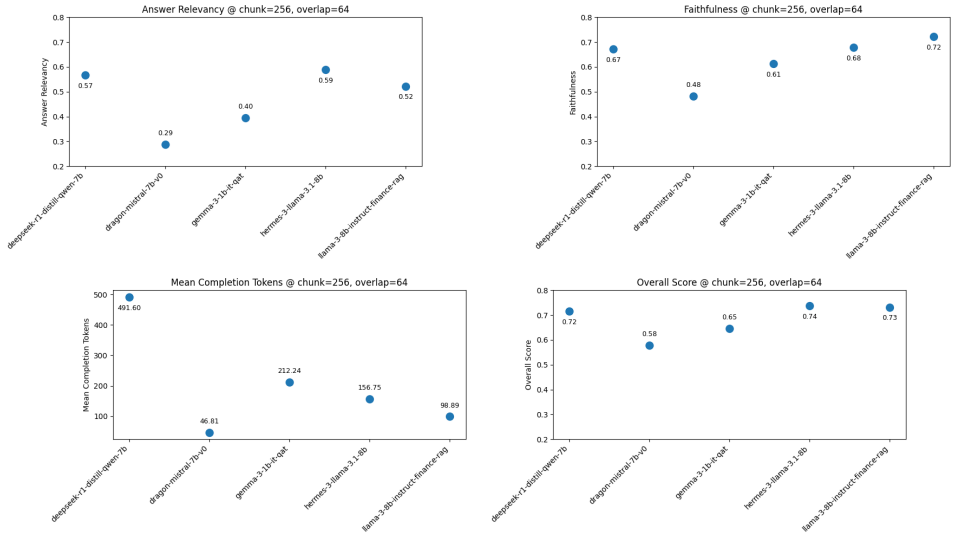

Figure 4.4: Comparison of the models metrics with chunk size 256(*mind the differing scale for tokens*)

The results presented in Figure 4.4 show a comparison of the five tested local models using the first set of fixed parameters.

The `hermes-3-llama-3.1-8b` model emerged as the most balanced and consistent performer across all RAGAS metrics. It scored the highest in both **Answer Relevancy** (0.59) and **Overall Score** (0.74), indicating that its responses were not only relevant to the queries but also structurally sound and well-supported by the provided context. It also showed competitive performance in **Faithfulness** (0.68), coming close to the best-performing model in that metric, which was `llama-3-8b-instruct-finance-rag` (0.72).

Interestingly, the `deepseek-r1-distill-qwen-7b` model showed strong performance in Faithfulness and Overall Score as well (0.67 and 0.72 respectively), but

had slightly lower Answer Relevancy. This suggests it tended to produce grounded but sometimes overly verbose or less directly relevant outputs as suggested by it's mean completion tokens.

Another important factor considered was the **Mean Completion Tokens** metric, which reflects the verbosity and conciseness of generated responses. `hermes-3-llama-3.1-8b` maintained a moderate average of 196.75 tokens per response—indicating that its answers were informative without excessive verbosity. In contrast, `deepseek-r1` had significantly higher token usage (401.60 tokens), suggesting it was prone to producing unnecessarily long completions, affecting latency and user readability.

On the lower end of performance, `dragon-mistral-7b-v0` scored the weakest across almost all categories, particularly in Answer Relevancy (0.29) and Faithfulness (0.48), making it an unsuitable candidate for precise local documentation-based retrieval in these conditions.

Overall, the results of the first set suggest that `hermes-3-llama-3.1-8b` provides the best trade-off between accuracy, faithfulness, and response length. It consistently produced relevant and grounded responses within a reasonable token budget, making it the most practical choice for local deployment under constrained hardware conditions.

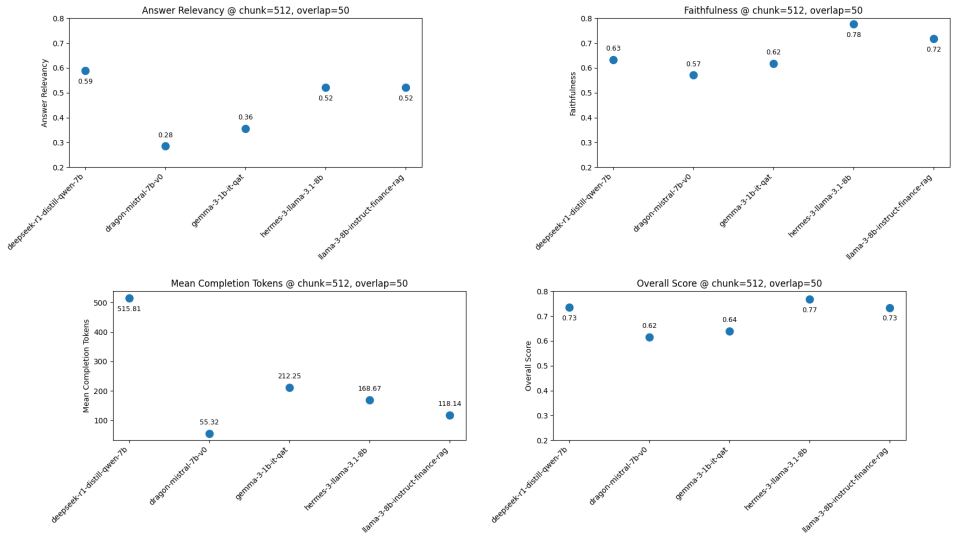

Figure 4.5: Comparison of the models metrics with chunk size 512(*mind the differing scale for tokens*)

The results in Figure 4.5 represent the same set of models evaluated under the second set of conditions. Compared to the previous configuration, this setup tested whether a broader context window would impact model retrieval accuracy and faithfulness.

The performance trends observed earlier remained mostly consistent. Once again, `hermes-3-llama-3.1-8b` and `llama-3-8b-instruct-finance-rag` led across all major metrics. However, their performance margins narrowed slightly compared to the previous test, particularly in Answer Relevancy. `Hermes` scored 0.57 in Answer Relevancy (down from 0.59) and 0.71 in Overall Score (down from 0.74), while `llama-finance-rag` held stable at 0.72 for Faithfulness and 0.73 for Overall Score.

A noteworthy change was seen in the performance of `gemma-3.1b-it-qat`, which improved its Answer Relevancy to 0.47 and lowered its average token usage from 212.24 to 188.88, indicating that larger chunk sizes benefited this model's ability to find and represent relevant information more concisely.

Token usage remained efficient overall, with no major outliers. `hermes` continued to balance performance and verbosity well, averaging 183.47 tokens, while `deepseek-r1-distill-qwen-7b` remained the most verbose at 398.11 tokens.

In conclusion, while the overall score differences were marginal between chunk sizes of 256 and 512, larger chunks introduced slight trade-offs in accuracy for some models. `hermes-3-Llama-3.1-8B` maintained its position as the top performer across both configurations.

Table 4.1: Table summary of model performance comparison across two chunk size configurations (256/512)

Model	Answer Relevancy	Faithfulness	Overall Score	Tokens
`deepseek`	0.57/0.56	0.67/0.66	0.72/0.71	401.60/398.11
`mistral`	0.29/0.26	0.48/0.49	0.58/0.56	46.81/51.23
`gemma`	0.40/0.47	0.61/0.60	0.65/0.68	212.24/188.88
`hermes`	0.59/0.57	0.68/0.67	0.74/0.71	196.75/183.47
`llama`	0.52/0.51	0.72/0.72	0.73/0.73	98.99/103.02

4.3.1 Best Model: `Hermes-3-Llama-3.1-8B`

Among the models evaluated, the `hermes-3-llama-3.1-8b` consistently delivered the highest performance across key evaluation metrics, including Answer Relevancy, Faithfulness, and overall RAGAS score. Its outputs were more contextually accurate, better grounded in the retrieved information, and semantically aligned with user queries than those generated by the other models tested. This made `hermes` the most suitable choice for local deployment within the constraints of my development environment.

Despite being an 8B parameter model, `hermes` maintained a relatively balanced profile in terms of inference latency and resource usage. It kept a relatively small completion token usage, but not bare bones, making it concise while still giving enough information.

The choice of `hermes` as the default local model was further justified by its performance under multiple chunking, overlap, and retrieval configurations (as shown in the following section).

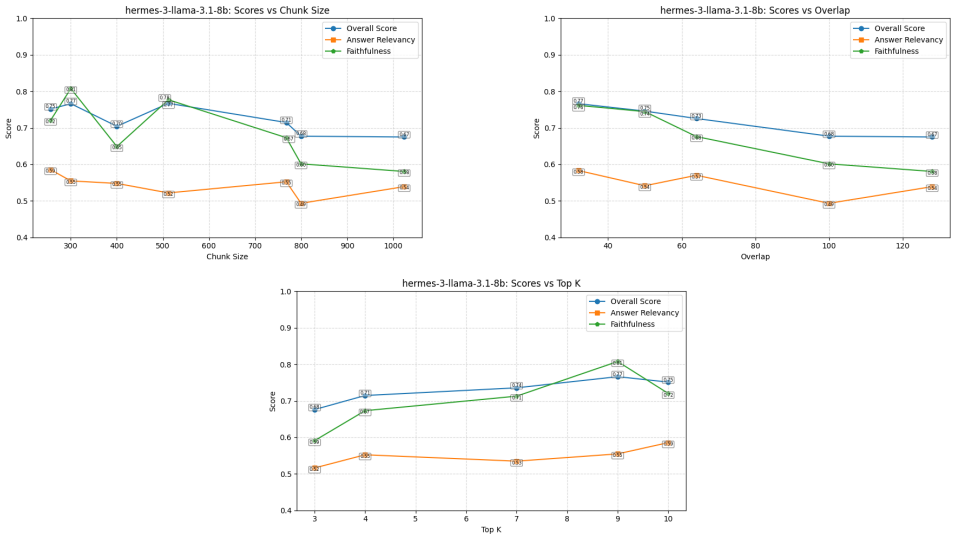

Figure 4.6: Comparison of the model's trends across different parameters

4.3.2 Effect of Chunk Size and Top-k

Next, I tested how varying *chunk size*, *overlap* and *top-k* values affected model performance. The results from the three scatter plots (Figure 4.6) suggest that:

- **Chunk size** peaked at around 300, where performance in terms of retrieval accuracy and context recall was at its highest.

- **Overlap** values showed a somewhat negative correlation with performance—the higher the overlap, the worse the retrieval results, if only marginally.

- **Top-k** values showed a nearly direct positive correlation with performance—the higher the top-k, the better the retrieval results across the board.

The findings from these plots highlight that `hermes` works better when tuned with higher top_k values and an optimal chunk size of around 300, but with it's overlap set to a lower size. The higher chunk size leads to better context coverage, without unnecessarily big clusters that can lead to confusion, significantly boosting the model's ability to retrieve relevant information. The lower overlap, likely means that more relevant information was passed and used into the queries, improving performance.

This difference was most clearly observed in the response to the previously mentioned question *"How can I dynamically hide UI elements?"*. The `hermes-3-llama-3.1-8b` model generated a comprehensive and well-structured answer. It not only correctly identified the key file and section within the documentation where dynamic visibility settings were discussed but also enumerated multiple options of how to achieve

it in different ways and circumstances. The formatting of the response also followed a clear list structure, which enhanced readability and interpretability for the user.

In contrast, the other models—particularly `dragon-mistral-7b-v0` and `gemma-3.1b-it-qat`, returned incomplete or underdeveloped responses. These often consisted of only a few vague lines, lacking specific implementation guidance and only providing minimal sources. Additionally, the formatting of their outputs was noticeably inferior, with minimal structure or use of bullet points, making it more difficult for users to extract useful information efficiently.

These qualitative differences reinforced the quantitative evaluation: while many models were capable of identifying related context, the `hermes` model consistently synthesized that context into clear, actionable, and user-friendly answers.

Given these results, `hermes-3-Llama-3.1-8B` was the best-performing model for local use, especially when configured with a chunk size around 300 and higher top-k values. These results also demonstrate that reranking plays a role in improving retrieval accuracy and context recall, making it a recommended feature when using these models, despite my use case showing only minimal differences.

5 Conclusion

This thesis explored the design, implementation, and evaluation of a private LLM-powered chatbot tailored for secured information retrieval from Adacta's internal documentation. The core goal was to assess the viability of deploying LLMs in a privacy-conscious organizational setting, using a Retrieval-Augmented Generation (RAG) architecture to deliver grounded, context-aware answers. To accomplish this, both local and cloud-based model options were considered, culminating in the deployment of a local prototype leveraging LMStudio, ChromaDB, and transformer-based embeddings.

Throughout the thesis, I conducted a comparative analysis of LLM deployment options such as Azure, Cohere, OpenAI, and local hosting. Due to practical limitations, particularly Azure's billing setup, I focused on running quantized models locally through LMStudio and interfacing them via a Python Flask API. I tested several embedding models for document retrieval and ultimately integrated FlagEmbedding for embeddings and FlagReranker for reranking due to its performance on domain-specific queries. Markdown documentation from Adacta was preprocessed using a documentation-specific chunking strategy that balanced structural, semantic, and contextual considerations. The resulting document vectors were indexed in ChromaDB for efficient vector-based retrieval and reranked using a reranker to improve context quality. A minimal UI built with Streamlit allowed users to interact with the system in real-time, with features like response streaming, reranking toggles, and query history management.

The chatbot's performance was evaluated using the RAGAS framework, which highlighted the pipeline's strengths in delivering relevant and sourced responses. The results validated that even lightweight local models can perform relatively competitively in enterprise use cases when combined with proper chunking and retrieval logic. The application demonstrated meaningful value to Adacta by allowing natural language access to complex documentation, improving usability, and reducing the need for manual searching.

However, there are several areas for improvement. First, while functional, the user interface and experience could benefit from further refinement in user experience and design consistency. For instance, the chatbot currently retrieves sources for every query, regardless of whether the context was part of an ongoing conversation. This can lead to repetitive or unnecessary results, which affects perceived intelligence and continuity. Enhancements such as first evaluating whether or not to retrieve sources are necessary, or if the message history is enough. Additionally, the reliance on markdown structure alone may not be sufficient in all cases, and future work could involve richer metadata or more documentation-specific strategies to retain the most information per chunk.

Finally, this thesis illustrates the growing utility and maturity of RAG-based systems for enterprise knowledge management, even at the local usage level. The modularity of the architecture makes it highly extensible, and as open-source models and embedding techniques continue to evolve, the performance gap between private

and cloud-based solutions is narrowing. As reranking techniques, model quantization, and overall model performance continue to improve, RAG-based applications are becoming increasingly useful in both enterprise and private environments. With AI getting better at persuasion and human communication, I believe RAG based applications will become more and more popular to provide proper sources and context to given information.

Bibliography

1. OPENAI. *ChatGPT: A Conversational AI Model* [online]. 2025. [visited on 2025-05-18]. Available from: https://openai.com/chatgpt/overview/.
2. GOOGLE. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* [online]. 2025. [visited on 2025-05-18]. Available from: https://github.com/google-research/bert.
3. META. *LLaMA: Open and Efficient Foundation Language Models* [online]. 2025. [visited on 2025-05-18]. Available from: https://github.com/facebookresearch/llama.
4. INTERTECH. *Risks to Proprietary Data During AI Implementation and How to Protect Your Data* [online]. 2025. [visited on 2025-05-18]. Available from: https://www.intertech.com/risks-to-proprietary-data-during-ai-implementation-and-how-to-protect-your-data/.
5. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. *Attention Is All You Need* [online]. 2023. [visited on 2025-05-13]. Available from arXiv: 1706.03762 [cs.CL].
6. DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: BURSTEIN, Jill; DORAN, Christy; SOLORIO, Thamar (eds.). *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* [online]. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 4171–4186 [visited on 2025-05-13]. Available from DOI: 10.18653/v1/N19-1423.
7. BROWN, Tom B.; MANN, Benjamin; RYDER, Nick; SUBBIAH, Melanie; KAPLAN, Jared; DHARIWAL, Prafulla; NEELAKANTAN, Arvind; SHYAM, Pranav; SASTRY, Girish; ASKELL, Amanda; AGARWAL, Sandhini; HERBERT-VOSS, Ariel; KRUEGER, Gretchen; HENIGHAN, Tom; CHILD, Rewon; RAMESH, Aditya; ZIEGLER, Daniel M.; WU, Jeffrey; WINTER, Clemens; HESSE, Christopher;

CHEN, Mark; SIGLER, Eric; LITWIN, Mateusz; GRAY, Scott; CHESS, Benjamin; CLARK, Jack; BERNER, Christopher; MCCANDLISH, Sam; RADFORD, Alec; SUTSKEVER, Ilya; AMODEI, Dario. *Language Models are Few-Shot Learners* [online]. 2020. [visited on 2025-05-13]. Available from arXiv: 2005.14165 [cs.CL].

8. TOUVRON, Hugo; LAVRIL, Thibaut; IZACARD, Gautier; MARTINET, Xavier; LACHAUX, Marie-Anne; LACROIX, Timothée; ROZIÈRE, Baptiste; GOYAL, Naman; HAMBRO, Eric; AZHAR, Faisal; RODRIGUEZ, Aurelien; JOULIN, Armand; GRAVE, Edouard; LAMPLE, Guillaume. *LLaMA: Open and Efficient Foundation Language Models* [online]. 2023. [visited on 2025-05-21]. Available from arXiv: 2302.13971 [cs.CL].

9. TOUVRON, Hugo; MARTIN, Louis; STONE, Kevin; ALBERT, Peter; ALMAHAIRI, Amjad; BABAEI, Yasmine; BASHLYKOV, Nikolay; BATRA, Soumya; BHARGAVA, Prajjwal; BHOSALE, Shruti; BIKEL, Dan; BLECHER, Lukas; FERRER, Cristian Canton; CHEN, Moya; CUCURULL, Guillem; ESIOBU, David; FERNANDES, Jude; FU, Jeremy; FU, Wenyin; FULLER, Brian; GAO, Cynthia; GOSWAMI, Vedanuj; GOYAL, Naman; HARTSHORN, Anthony; HOSSEINI, Saghar; HOU, Rui; INAN, Hakan; KARDAS, Marcin; KERKEZ, Viktor; KHABSA, Madian; KLOUMANN, Isabel; KORENEV, Artem; KOURA, Punit Singh; LACHAUX, Marie-Anne; LAVRIL, Thibaut; LEE, Jenya; LISKOVICH, Diana; LU, Yinghai; MAO, Yuning; MARTINET, Xavier; MIHAYLOV, Todor; MISHRA, Pushkar; MOLYBOG, Igor; NIE, Yixin; POULTON, Andrew; REIZENSTEIN, Jeremy; RUNGTA, Rashi; SALADI, Kalyan; SCHELTEN, Alan; SILVA, Ruan; SMITH, Eric Michael; SUBRAMANIAN, Ranjan; TAN, Xiaoqing Ellen; TANG, Binh; TAYLOR, Ross; WILLIAMS, Adina; KUAN, Jian Xiang; XU, Puxin; YAN, Zheng; ZAROV, Iliyan; ZHANG, Yuchen; FAN, Angela; KAMBADUR, Melanie; NARANG, Sharan; RODRIGUEZ, Aurelien; STOJNIC, Robert; EDUNOV, Sergey; SCIALOM, Thomas. *Llama 2: Open Foundation and Fine-Tuned Chat Models* [online]. 2023. [visited on 2025-05-13]. Available from arXiv: 2307.09288 [cs.CL].

10. DEEPSEEK-AI et al. *DeepSeek-V3 Technical Report* [online]. 2025. [visited on 2025-05-21]. Available from arXiv: 2412.19437 [cs.CL].

11. TEKNIUM, Ryan; QUESNELLE, Jeffrey; GUANG, Chen. *Hermes 3 Technical Report* [online]. 2024. [visited on 2025-05-21]. Available from arXiv: 2408.11857 [cs.CL].

12. MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHÜTZE, Hinrich. *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008. ISBN 9780521865715.

13. MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. *Efficient Estimation of Word Representations in Vector Space* [online]. 2013. [visited on 2025-05-13]. Available from arXiv: 1301.3781 [cs.CL].

14. PENNINGTON, Jeffrey; SOCHER, Richard; MANNING, Christopher. GloVe: Global Vectors for Word Representation. In: MOSCHITTI, Alessandro; PANG, Bo; DAELEMANS, Walter (eds.). *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* [online]. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1532–1543 [visited on 2025-05-13]. Available from DOI: 10.3115/v1/D14-1162.

15. PETERS, Matthew E.; NEUMANN, Mark; IYYER, Mohit; GARDNER, Matt; CLARK, Christopher; LEE, Kenton; ZETTLEMOYER, Luke. *Deep contextualized word representations* [online]. 2018. [visited on 2025-05-13]. Available from arXiv: 1802.05365 [cs.CL].

16. HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-Term Memory. *Neural Comput.* [online]. 1997, vol. 9, no. 8, pp. 1735–1780 [visited on 2025-05-21]. ISSN 0899-7667. Available from DOI: 10.1162/neco.1997.9.8.1735.

17. REIMERS, Nils; GUREVYCH, Iryna. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks* [online]. 2019. [visited on 2025-05-13]. Available from arXiv: 1908.10084 [cs.CL].

18. REIMERS, Nils; GUREVYCH, Iryna. *all-MiniLM-L6-v2: A Miniature Transformer-based Sentence Embedding Model* [online]. 2020. [visited on 2025-05-13]. Available from: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

19. SONG, Kaitao; TAN, Xu; QIN, Tao; LU, Jianfeng; LIU, Tie-Yan. *MPNet: Masked and Permuted Pre-training for Language Understanding* [online]. 2020. [visited on 2025-05-13]. Available from arXiv: 2004.09297 [cs.CL].

20. WANG, Liang; YANG, Nan; HUANG, Xiaolong; JIAO, Binxing; YANG, Linjun; JIANG, Daxin; MAJUMDER, Rangan; WEI, Furu. *Text Embeddings by Weakly-Supervised Contrastive Pre-training* [online]. 2024. [visited on 2025-05-13]. Available from arXiv: 2212.03533 [cs.CL].

21. FLAGOPEN. *FlagEmbedding: A Library for Embedding-based Models* [online]. 2025. [visited on 2025-05-13]. Available from: https://github.com/FlagOpen/FlagEmbedding/.

22. LEWIS, Patrick; PEREZ, Ethan; PIKTUS, Aleksandra; PETRONI, Fabio; KARPUKHIN, Vladimir; GOYAL, Naman; KÜTTLER, Heinrich; LEWIS, Mike; YIH, Wen-tau; ROCKTÄSCHEL, Tim; RIEDEL, Sebastian; KIELA, Douwe. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* [online]. 2021. [visited on 2025-05-13]. Available from arXiv: 2005.11401 [cs.CL].

23. CHROMA. *Chroma DB: A Database for Embeddings and Vector Search* [online]. 2025. [visited on 2025-05-14]. Available from: https://www.trychroma.com/.

24. KHATTAB, Omar; ZAHARIA, Matei. *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT* [online]. 2020. [visited on 2025-05-21]. Available from arXiv: 2004.12832 [cs.IR].

25. RAFFEL, Colin; SHAZEER, Noam; ROBERTS, Adam; LEE, Katherine; NARANG, Sharan; MATENA, Michael; ZHOU, Yanqi; LI, Wei; LIU, Peter J. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer* [online]. 2023. [visited on 2025-05-21]. Available from arXiv: 1910.10683 [cs.LG].

26. PAPINENI, Kishore; ROUKOS, Salim; WARD, Todd; ZHU, Wei-Jing. Bleu: a Method for Automatic Evaluation of Machine Translation. In: ISABELLE, Pierre; CHARNIAK, Eugene; LIN, Dekang (eds.). *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics* [online]. Philadelphia, Pennsylvania,

USA: Association for Computational Linguistics, 2002, pp. 311–318 [visited on 2025-05-21]. Available from DOI: 10.3115/1073083.1073135.

27. EXPLODINGGRADIENTS. *Ragas: Supercharge Your LLM Application Evaluations* [online]. 2024. [visited on 2025-05-14]. Available from: https://github.com/explodinggradients/ragas.

28. RAGAS. *Response Relevance Metric* [online]. 2025. [visited on 2025-05-14]. Available from: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/answer_relevance/.

29. RAGAS. *Context Recall Metric* [online]. 2025. [visited on 2025-05-14]. Available from: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/context_recall/.

30. RAGAS. *Context Precision Metric* [online]. 2025. [visited on 2025-05-14]. Available from: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/context_precision/.

31. RAGAS. *Faithfulness Metric* [online]. 2025. [visited on 2025-05-14]. Available from: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/faithfulness/.

32. JI, Ziwei; LEE, Nayeon; FRIESKE, Rita; YU, Tiezheng; SU, Dan; XU, Yan; ISHII, Etsuko; BANG, Ye Jin; MADOTTO, Andrea; FUNG, Pascale. Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys* [online]. 2023, vol. 55, no. 12, pp. 1–38 [visited on 2025-05-13]. ISSN 1557-7341. Available from DOI: 10.1145/3571730.

33. COHERE. *Enterprise Data Commitments* [online]. 2025. [visited on 2025-05-13]. Available from: https://cohere.com/enterprise-data-commitments.

34. ANALYTICS, IoT. *Leading Generative AI Companies* [online]. 2025. [visited on 2025-05-13]. Available from: https://iot-analytics.com/leading-generative-ai-companies/.

35. OPENAI. *Data Processing Addendum* [online]. 2025. [visited on 2025-05-13]. Available from: https://openai.com/policies/data-processing-addendum/.

36. OPENAI. *OpenAI Models Documentation* [online]. 2025. [visited on 2025-05-13]. Available from: https://platform.openai.com/docs/models.

37. MICROSOFT. *OpenAI Data Privacy* [online]. 2025. [visited on 2025-05-14]. Available from: https://learn.microsoft.com/en-us/legal/cognitive-services/openai/data-privacy?tabs=azure-portal.

38. MICROSOFT. *Azure AI Models* [online]. 2025. [visited on 2025-05-13]. Available from: https://ai.azure.com/explore/models.

39. STUDIO, LM. *LM Studio* [online]. 2025. [visited on 2025-05-13]. Available from: https://lmstudio.ai/.

40. GGML. *GGUF Specification* [online]. 2025. [visited on 2025-05-14]. Available from: https://github.com/ggml-org/ggml/blob/master/docs/gguf.md.

41. RONACHER, Armin. *Flask: A Microframework for Python* [online]. 2025. [visited on 2025-05-14]. Available from: https://flask.palletsprojects.com/.

42. ZHANG, Peitian; XIAO, Shitao; LIU, Zheng; DOU, Zhicheng; NIE, Jian-Yun. *Retrieve Anything To Augment Large Language Models* [online]. 2023. [visited on 2025-05-13]. Available from arXiv: 2310.07554 [cs.IR].

43. STREAMLIT. *Streamlit: The fastest way to build and share data apps* [online]. 2025. [visited on 2025-05-14]. Available from: https://streamlit.io/.

44. NOUSRESEARCH. *Hermes-3-Llama-3.1-8B: A Pretrained Language Model* [online]. 2025. [visited on 2025-05-14]. Available from: https://huggingface.co/NousResearch/Hermes-3-Llama-3.1-8B.

45. CHEN, Jianlv; XIAO, Shitao; ZHANG, Peitian; LUO, Kun; LIAN, Defu; LIU, Zheng. *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation* [online]. 2023. [visited on 2025-05-13]. Available from arXiv: 2309.07597 [cs.CL].

46. AI, MixedBread. *mxbai-embed-large-v1: A Pretrained Language Model* [online]. 2025. [visited on 2025-05-17]. Available from: https://huggingface.co/mixedbread-ai/mxbai-embed-large-v1.

47. AI, Nomic. *nomic-embed-text-v1.5: A Pretrained Text Embedding Model* [online]. 2025. [visited on 2025-05-17]. Available from: https://huggingface.co/nomic-ai/nomic-embed-text-v1.5.

48. SNOWFLAKE. *snowflake-arctic-embed-m-v1.5: A Pretrained Language Model for Embedding* [online]. 2025. [visited on 2025-05-17]. Available from: https://huggingface.co/Snowflake/snowflake-arctic-embed-m-v1.5.

49. AI, DeepSeek. *DeepSeek-R1-Distill-Qwen-7B: A Pretrained Language Model* [online]. 2025. [visited on 2025-05-17]. Available from: https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B.

50. THEBLOKE. *dragon-mistral-7B-v0-GGUF: A Pretrained Language Model* [online]. 2025. [visited on 2025-05-17]. Available from: https://huggingface.co/TheBloke/dragon-mistral-7B-v0-GGUF.

51. GOOGLE. *gemma-3-1b-it-qat-q4$_0$ − gguf : APretrainedLanguageModel* [online]. 2025. [visited on 2025-05-17]. Available from: https://huggingface.co/google/gemma-3-1b-it-qat-q4_0-gguf.

52. QUANTFACTORY. *Llama-3-8B-Instruct-Finance-RAG-GGUF: A Pretrained Language Model for Finance* [online]. 2025. [visited on 2025-05-17]. Available from: https://huggingface.co/QuantFactory/Llama-3-8B-Instruct-Finance-RAG-GGUF.

Index

A Appendix

The appendix intentionally does not include the dataset of questions used for RAGAS evaluation, due to the private nature of the documentation it concerns. Otherwise it includes:

- GitHub link to the project repository:
  `https://github.com/Kubik037/document-qa-assistant`
- project files compressed in .zip format:(main.py, database.py, text_processing.py, config.py, query_ui.py, tests.py, requirements.txt)
- README.md with information on installing and deployment