Intelligent Reasoning System Practice Module
Final Project Report

GuruDocs: ChatBot for Querying Documents using RAG and LLMs

<u>Submitted By:</u>

Alvin Wong Ann Ying

Brandon Chua Hong Huei

Ong Si Ci

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY

GuruDocs is a free chatbot designed to enhance the efficiency of information retrieval from lengthy policy and/or regulatory documents. Leveraging Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs), GuruDocs offers context-aware search results, thus improving user experience and productivity. This project addresses the limitations of keyword-based search systems and the challenges in fine-tuning LLMs for specific document queries.

2 PROBLEM STATEMENT AND MARKET RESEARCH

2.1 Problem Statement

In today's fast-paced work environment, employees frequently encounter challenges when trying to extract pertinent information from lengthy policy and regulatory documents. This difficulty often leads to frustration and a significant loss of productivity. The current reliance on keyword-based search systems falls short when dealing with complex documents that require nuanced understanding and context. Additionally, the process of fine-tuning LLMs for specific documents is highly resource-intensive, making it impractical for many organizations.

This gap in efficient document retrieval not only hampers productivity but also contributes to delays in decision-making processes. Employees may spend excessive time sifting through dense documents, leading to a drain on valuable resources and impacting overall organizational efficiency. Moreover, the lack of a user-friendly and intuitive document query solution further exacerbates these challenges, creating a pressing need for a more effective and accessible tool.

GuruDocs seeks to address these critical issues by offering a free, innovative solution that leverages RAGand LLM technologies. By enabling users to engage in natural language interactions with their documents, GuruDocs empowers individuals to quickly and accurately access the information they need, thereby enhancing productivity, reducing frustration, and facilitating informed decision-making.

2.2 Market Research

GuruDocs competes with OpenAI's ChatGPT, which offers document question answering in AskAI. In addition, most products in the market uses OpenAI's ChatGPT4 for knowledge reasoning, which is not free and often hosted on the cloud, posing data confidentiality concerns. GuruDocs, being free, targets corporate staff dealing with confidential documents, offering an on-premises solution for enhance data security.

	GuruDocs	ChatGPT's AskAI	CHATDOC	Hypotenuse AI	DocBots AI
Multiple Document Query	✓	✗	✓	✓	✓
Unlimited number of documents upload	✓	✗	✗	✗	✗
Unlimited number of queries	✓	✗	✗	✓	✗
Unlimited number of users	✓	✗	✓	✗	✗
Free	✓	✗	✗	✗	✗
Able to work with confidential data	✓	✗	✗	✗	✗

To include the references and analysis for each of the features analyse.

2.3 Business Value

Traditional FAQ systems or chatbots have several limitations that GuruDocs is able to address:

1. **Limited Scope**: Traditional FAQ systems are typically limited to predefined questions and answers, which may not cover the full range of user queries or complex document-related inquiries.
2. **Lack of Context**: Chatbots, especially rule-based ones, may struggle with understanding context and providing relevant responses, leading to frustration for users seeking specific information from documents.
3. **Scalability Challenges:** As the volume and complexity of documents increase, traditional systems and chatbots may struggle to scale effectively, resulting in diminishing returns on investment and reduced user satisfaction.
4. **Maintenance Overhead:** Maintaining and updating traditional FAQ systems and chatbots can be resource-intensive, requiring ongoing efforts to keep information up to date and ensure accurate responses, which can add to operational costs over time.

GuruDocs brings several key business values:

1. **Improved Productivity:** By streamlining the process of extracting information from lengthy documents, GuruDocs helps employees find the information they need quickly and efficiently, saving time and increasing productivity.
2. **Enhanced User Experience:** With its context-aware search results and user-friendly interface, GuruDocs provides a more pleasant and effective way for employees to interact with policy/regulatory documents, improving overall user experience.
3. **Cost Savings:** By reducing the time employees spend searching for information, GuruDocs can lead to cost savings for businesses, as employees can focus on more important tasks.
4. **Data Security:** GuruDocs offers an on-premise solution for document query, ensuring that confidential information remains secure within the company's network, addressing data confidentiality concerns.

5. **Competitive Advantage:** By adopting GuruDocs, businesses can gain a competitive advantage by improving their information retrieval processes, making them more efficient and effective compared to competitors using traditional keyword-based search systems.

Pointers to include:

- Compare with traditional FAQ systems, downsides to these traditional chatbots

3 SYSTEM ARCHITECTURE DESIGN

GuruDocs adopts a modular and scalable architecture, leveraging containerization technologies for both the frontend and backend components. The frontend is encapsulated within a Docker container, hosting a React-based user interface along with HTML and CSS for rendering interactive and responsive user experiences. Containerization of the frontend ensures portability, ease of deployment, and efficient resource utilization.

On the backend side, GuruDocs utilizes another Docker container to house the application logic, primarily developed in Python. This backend container orchestrates the RAG process through LangChain. Chroma serves as the vector database, facilitating efficient retrieval and matching of relevant information from the documents. We leverage Ollama run the LLMs locally. Figure 1 shows the architecture design for GuruDocs.

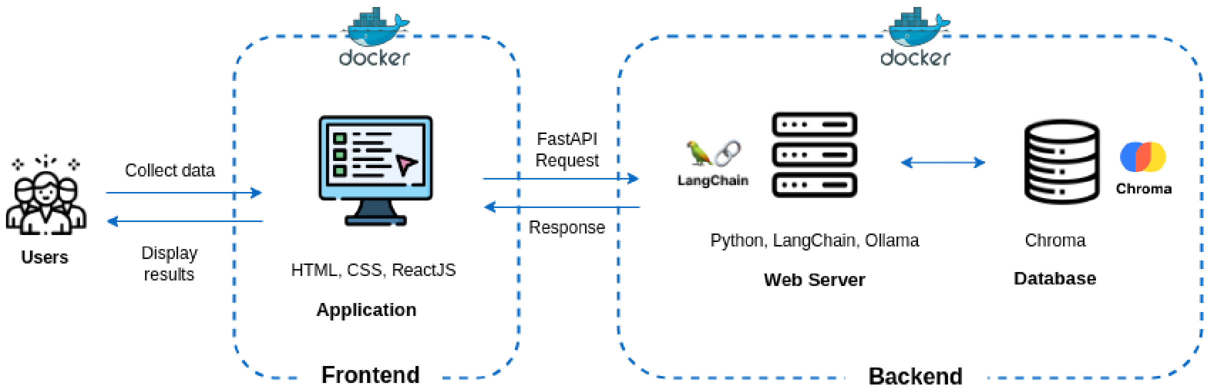


Figure 1 Architecture Design of GuruDocs

Within the backend, the process flow for GuruDocs' RAG is detailed in Figure 2. The document is first split using sentence chunking and HuggingFace Embedding Model is used to store the chunks as vectors in the vector database. When the user submits a query on the document, the same embedding model is used to extract the vector embeddings of the user query. The vector embeddings are then matched with the vectors in the vector database. Chunks with the highest similarity are returned as context that is parsed into the LLM together with the user query to generate a response.

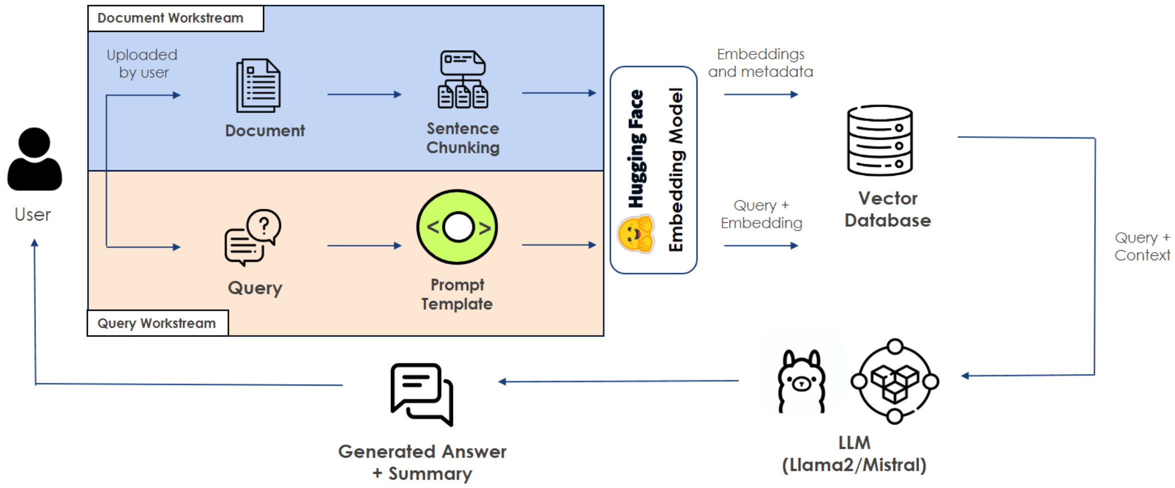


Figure 2 Process flow of RAG for GuruDocs

4 PRODUCT IMPLEMENTATION

4.1 Framework

After a thorough evaluation, the team concluded that LangChain is the optimal framework for GuruDocs due to several key factors. Firstly, LangChain's modular components allow the team to customize and switch out components such as LLM models, chunking strategy, types of retrievers and embedding models to fit GuruDocs' specific requirements (LangChain, 2024). This flexibility is crucial for GuruDocs, which focuses on providing free on-premises solution for handling confidential workloads. The ability to tailor LangChain to meet these requirements ensures that GuruDocs can maintain data security and confidentiality while providing efficient document query capabilities.

Secondly, LangChain's extensibility was a significant consideration for the team. The framework's ability to easily incorporate new features, such as optical character recognition (OCR) and speech-to-text translation from videos, makes it a future-proof choice for GuruDocs. These features are essential for enhancing GuruDocs' functionality and usability, especially as the project looks to expand its capabilities to meet the evolving needs of users.

In conclusion, LangChain's modular design, which enables customization and easy integration of new features, makes it the ideal framework for GuruDocs. By leveraging LangChain's capabilities, GuruDocs can deliver a secure, efficient, and user-friendly document query solution for corporate staff dealing with confidential documents.

4.1.1 Chunking Strategy

Chunking is a natural language processing technique that involves dividing a text into meaningful segments or "chunks" to simplify the complexity of text and extract important information for further analysis or processing. There are two main types of chunking methods: page chunking and sentence chunking.

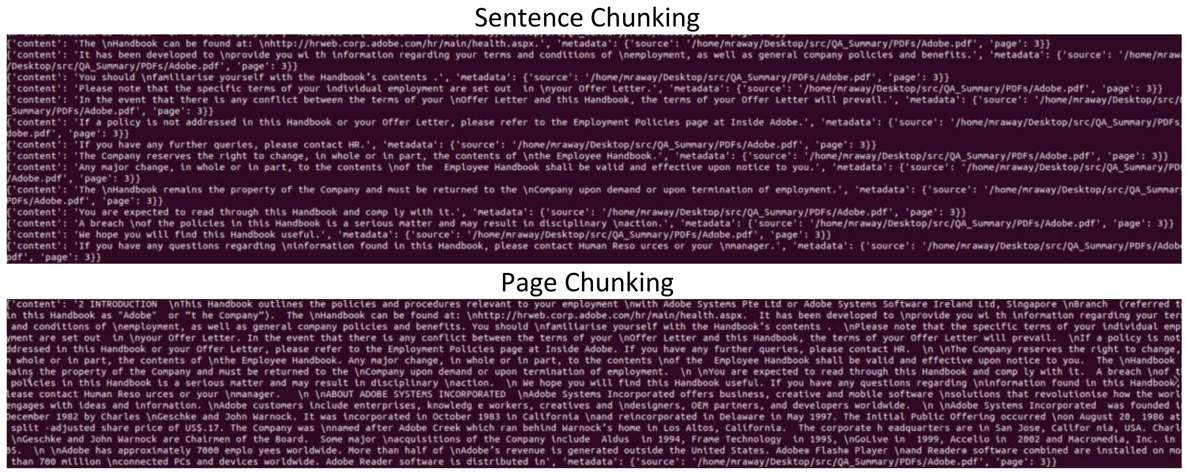


Figure 3 Output example for Sentence and Page Chunking on Adobe Employee Handbook

Page chunking involves dividing a document or text into larger segments, such as paragraphs or sections. This type of chunking is well-suited for summarization tasks, as it allows the summarizer to focus on the main ideas and key points of the text without getting bogged down by details.

On the other hand, sentence chunking involves dividing a text into smaller segments, typically sentences. Sentence chunking is particularly useful in scenarios where the questions are focused on specific details or facts within the text, as it helps to identify the relevant information more efficiently.

In GuruDocs, both page chunking and sentence chunking were used based on their respective strengths. Page chunking was utilized for summarization tasks to condense longer texts into key points and main ideas, while sentence chunking was employed for question answering tasks to extract relevant information from individual sentences to answer specific questions. This approach allowed GuruDocs to effectively leverage the strengths of both chunking techniques to provide better response to the user.

4.1.2 Embedding Model

To ensure that GuruDocs do not incur any cost i.e., no API key requirement, the team experimented with various options available on LangChain (LangChain, 2024) and narrowed down to 2 models namely Ollama and HuggingFace. To determine which embedding model would be used for GuruDocs, the team experimented with both embedding models with a series of simple human generated questions. For example, “What is the policy on annual leave?”, “What is the company name?”, etc.

After thorough experimentation, the team decided to use the Hugging Face embedding model for GuruDocs. Despite the potential need for an API key, which could incur costs, HuggingFace consistently produced better results compared to Ollama. Its wide range of pre-trained embeddings and models, along with its strong performance, made it the preferred choice for GuruDocs. An example of the result from GuruDocs using HuggingFace and Ollama embeddings when asked “What is the policy on annual leave?” is shown in the table below.

Table 1 Sample Results of Ollama and HuggingFace embedding models on Adobe Employee Handbook

Embedding Model	Generated Results
Ollama	Based on the provided context, Adobe's policy on annual leave appears to be as follows: "Employees are expected to maintain the highest standard of conduct and behavior at such events and must conduct themselves in a professional manner at all times. Failure to comply with this policy may result in disciplinary action, including but not limited to summary termination." There is no direct mention of annual leave in the provided context. However, it can be inferred that Adobe has a policy regarding annual leave based on the expectations for professional conduct and behavior at events. Without further information, I cannot provide a definitive answer to your question.
HuggingFace	Adobe's policy on annual leave provides eligible employees with up to 14 days of paid leave per year, in addition to any sick leave or other leaves of absence that may be available. Employees are entitled to use their annual leave at any time during the year, subject to advance notice and approval from management. Annual leave accrues at a rate of 14 days per year for employees who have completed their probationary period and have been employed by Adobe for at least six months. Employees who are covered by the Employment Act may be entitled to additional sick leave or other leaves of absence, as well as statutorily mandated benefits such as paid sick leave and hospitalization pay. Adobe's policy on annual leave is designed to provide employees with a reasonable amount of time off each year to relax, recharge, and pursue personal interests. By allowing employees to use their annual leave at any time during the year, Adobe aims to promote work-life balance and support employee well-being. To apply for annual leave, employees must submit a leave request form to their manager and HR representative in advance. The form can be found on Adobe's intranet or by contacting HR directly. Approval of annual leave requests will be based on business needs and availability of staff, and employees will be notified of the approval or denial of their request. It is important to note that annual leave does not accrue and will be refreshed to 14 days at the beginning of each calendar year. If an employee leaves Adobe for any reason, they will not be paid out for their unused annual leave.

By selecting the Hugging Face embedding model, the team is confident that GuruDocs will be able to deliver a robust and effective document query solution that meets the needs of corporate staff dealing with confidential documents.

4.1.3 Summary Technique

Summarization tasks in Natural Language Processing (NLP) involve condensing a piece of text into a shorter version while retaining the key information and main ideas. LangChain offers a range of tools and techniques for both extractive and abstractive summarization tasks (Sojasingarayar, 2024). Some of the key features and offerings of LangChain in this area include:

- Stuff Technique: LangChain's Stuff technique focuses on condensing text by removing redundant or irrelevant information while retaining the essential content. This technique is particularly useful for extractive summarization tasks where the goal is to identify and extract key sentences or phrases.
- Map_Reduce Technique: LangChain's map_reduce technique involves breaking down the text into smaller units and then mapping and reducing these units to extract the most important information. This technique is suitable for both extractive and abstractive summarization tasks, especially when dealing with large volumes of text or complex documents.
- Refine Technique: LangChain's refine technique focuses on refining and improving the quality of the summary generated by other techniques. This technique is beneficial for ensuring that the summary is accurate, coherent, and well-structured, especially in abstractive summarization tasks.

In the selection of the appropriate summarization technique for GuruDocs, several key considerations were carefully weighed, including processing speed and the accuracy of the summarization. The table below shows the analysis of the 3 summarization techniques. Summarization results can be found in Appendix 7.1.

Table 2 Analysis of Summarization Techniques

Summarization Technique	Stuff	Map_Reduce	Refine
Processing Time	4.9s	1m 48.9s	2m 59.7s
Details provided in Summary	Did not provide an overview of the document uploaded; Provided key sentences of certain sections of the document, missing out on most information.	Provided an overview of the document uploaded and pointed out the important sections deemed.	Provided an overview of the document uploaded along with short summaries of the certain sections of the document.

While the Stuff Technique offers the advantage of providing a summary in a relatively shorter timeframe, it was found to be inadequate in producing accurate summaries of the uploaded documents. This limitation arises from the technique's focus on condensing text by eliminating redundant or irrelevant information, which may sometimes result in the exclusion of essential details from the summary.

In contrast, the Map_Reduce Technique, while potentially less accurate than the refine technique, offers a significant advantage in terms of processing speed. In addition, as the summary feature is meant to provide users a broad idea of the document uploaded, the requirement on the level of detail in the summary is lower. In conclusion, the team prioritized the speed and efficiency of processing summaries while still maintaining an acceptable level of accuracy and opted to implement the Map_Reduce technique for its summarization tasks.

4.1.4 Retrievers

In LangChain, Retrievers are components responsible for retrieving relevant information from a knowledge source, such as a document or database, based on a given query. They play a crucial role in information retrieval tasks by narrowing down the search space and improving the accuracy of the final results.

RetrievalQA and ConversationalQA are two approaches used in language processing, each with its own focus and objectives (Yang, 2024). RetrievalQA is primarily concerned with retrieving specific information from a knowledge source in response to a query. It is commonly used in applications where users need precise and factual answers to their queries, such as question-answering systems. For example, a RetrievalQA system may be used to answer questions like "What is the name of the company?" or "What are the types of leave available?" by retrieving the information from a reliable source.

ConversationalQA, on the other hand, focuses on maintaining context and continuity in a conversation by providing relevant responses based on previous interactions. It is commonly used in chatbots and virtual assistants, where the system needs to understand the context of the conversation and provide relevant responses. For example, in a ConversationalQA scenario, a user might ask a chatbot "What is the name of the company?" and then follow up with "Who is the founder?" The chatbot needs to maintain the context of the conversation and provide a relevant response based on the previous interaction.

The key differences between RetrievalQA and ConversationalQA lie in their approach to context handling, response generation, user interaction, and application. ConversationalQA systems focus on maintaining context and continuity in a conversation, generating responses dynamically based on the conversation's context. In contrast, RetrievalQA systems retrieve pre-existing information from a knowledge source and are more straightforward in providing direct answers to queries.

GuruDocs has opted to use ConversationalQA as the preferred option as it outperformed RetrievalQA in both speed and accuracy of response. Furthermore, its ability to maintain context, provide dynamic responses, and handle follow-up questions enabled GuruDocs to be more natural, engaging, and contextually aware. This enhancement in user experience of information retrieval made it an easy decision.

4.1.5 Prompt Template

Prompt template in NLP is a predefined format or structure used to guide the generation of natural language text. It consists of placeholders or variables that are filled in with specific information to create a coherent and contextually relevant response. Prompt templates play a crucial role in NLP tasks such as text generation, question answering, and summarization, as they provide a structured approach to generating text and ensure that the output is tailored to the intended purpose.

Firstly, they provide a standardized format for generating text, which helps to ensure consistency and coherence in the output. This is particularly important in applications where the generated text needs to be clear, concise, and grammatically correct. Secondly, prompt templates enable the generation of contextually relevant responses by specifying the information that should be included in the output. This helps to ensure that the generated text is relevant to the user's query or input. Finally, prompt templates can also improve the efficiency of text generation by providing a structured framework that guides the generation process, reducing the need for manual intervention and editing.

For GuruDocs, we experimented between 2 frameworks: i) Basic question-answering prompt framework; and ii) CO-STAR prompt framework (Teo, 2024). The basic question-answering prompt template provided the LLM a persona with a task to do i.e., AI Assistant for question-answering tasks. For the CO-STAR prompt template framework, it considers key aspects such as Context, Objective, Style, Tone, Audience, and Response to influence the effectiveness and relevance of the LLM's response, leading to more optimal responses. Table below shows a comparison between the 2 prompt templates.

Table 3 Prompt Template for Basic question-answering framework and CO-STAR framework

Type of Framework	**Basic Question-answering Framework**	**CO-STAR Prompt Framework**
Prompt Template	You are an AI assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. If you don't know the answer, just say that you don't know. Use three sentences maximum and keep the answer concise.	# Context # You are an AI assistant tasked to efficiently query and retrieve information from staff documents, including employee handbooks, policies, and regulations # Objective # Retrieve relevant information and present it in a readable format, improving overall information retrieval efficiency for employees. # Style # The writing style should be clear, concise, and professional, aligning with the tone and style of our company's official documents. The language should be easily understandable by employees across all levels of the organization. # Tone # Informative and authoritative, reflecting the importance and seriousness of the information contained in the staff documents. It should also be user-friendly and helpful, ensuring employees feel

		supported in their information retrieval efforts. # Audience # The response is intended for internal employees of our company who need to access and understand staff documents, including new hires, current employees, and managers. The information should be tailored to the understanding and expertise of the audience. # Response # The output format should be user-friendly and accessible, providing information in a structured manner that is easy to read and navigate. Consider using bullet points, summaries, and links to relevant sections for quick reference.

Through our experimentation, we found that the performance varied between the 2 frameworks with the basic question-answering outperforming the CO-STAR framework for 1 document and the CO-STAR framework outperforming basic question-answering framework for another. However, we noted a significant difference in the speed of response between the 2 frameworks. As such, we decided to adopt the basic question-answering framework for GuruDocs.

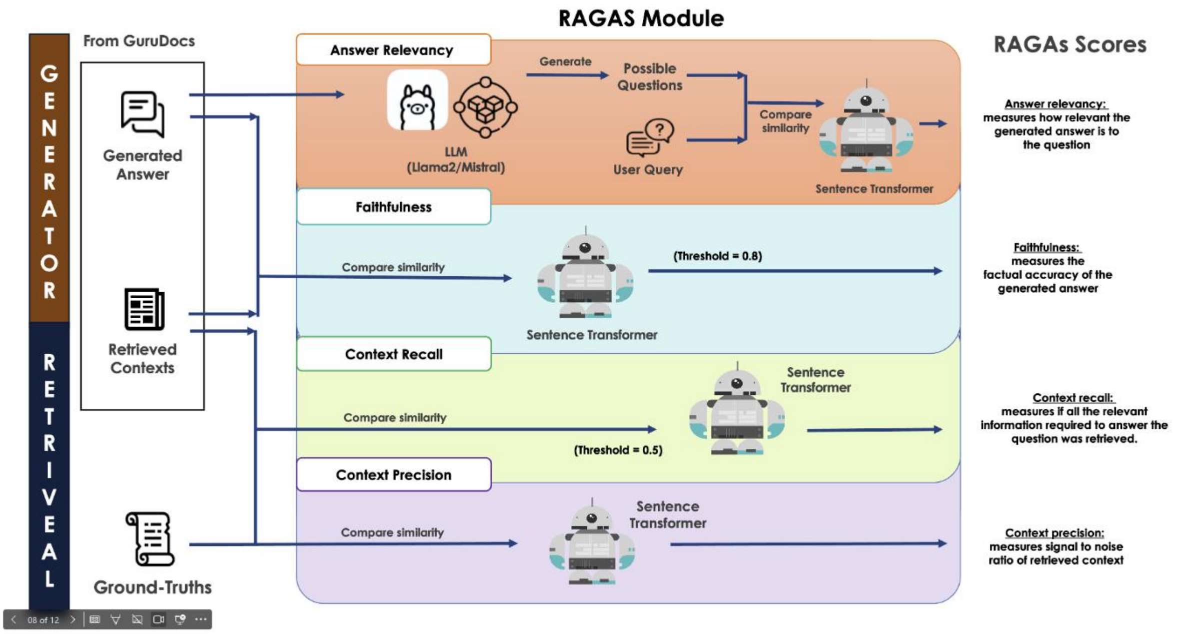
4.2 Evaluation of GuruDocs Performance

The assessment methodology and standards for Retrieval Augmented Generation Frameworks (RAGAs) are meticulously crafted to scrutinize both the retrieval and generation aspects of the framework. This comprehensive approach ensures the accuracy of retrieved context and the coherence of generated responses in alignment with provided context and prompts. Ground truths of question-and-answer pairs, each with predetermined context information, are leveraged to facilitate this evaluation process.

In evaluating the retrieval component, the primary aim is to validate the accuracy of retrieved context from the vector store, ensuring its alignment with the intended context based on predefined criteria. Evaluation metrics such as context precision and context recall gauge the relevance of retrieved context to the ground truth.

Conversely, in assessing the generator component, the focus shifts to ensuring coherence and consistency of generated answers with the retrieved context and prompts. Metrics like faithfulness and answer relevancy are utilized to measure the similarity and fluency of generated answers compared to the ground truth. Our evaluation workflow is demonstrated in Figure 4 below.

Figure 4 Evaluation Workflow with modified RAGAs



Experiments within this framework utilize two sets of documents: one from Adobe comprising 80 question-and-answer pairs, and another from NTUC consisting of 150 pairs. These documents serve as the foundation for evaluating both the retrieval and generation components of RAGAs.

The Text-to-Text Transfer Transformer (T5) Vanilla model serves as the primary generator, supplemented by Transformer-QG-on-SQuAD for comparative analysis, allowing for a comprehensive assessment of the RAGAs framework. To facilitate product evaluation, we developed a modified version of RAGAs to minimize costs associated with the open-source library's dependency on OpenAI's API key. Our modified version enables evaluation of both retriever and generator components for RAG solutions.

In our methodology, we meticulously generated ground truth question-and-answer pairs using the T5 vanilla transformer model, manually verifying each pair for accuracy and correctness.

Our experiments focused on three major components: the LLM model, chunking technique, and retrieval technique, across both documents. Notably, employing sentence-level chunking and conversational QA with minimal prompt engineering yielded optimal performance across both the Adobe and NTUC handbooks, as demonstrated in Tables 4 and 5 respectively.

Table 5 Evaluation Results on Adobe Employee Handbook

Mistral		**Faithfulness**	**Answer Relevancy**	**Context Precision**	**Context Recall**
Chunking					
Page Level		6.82%	55.62%	14.16%	3.26%
Sentence Level		**45.46%**	**63.49%**	**14.77%**	**3.81%**
Retrieval	**Prompt Engineering**				
Conversational QA	Minimal (provide basic instructions)	**49.13%**	**64.12%**	**15.18%**	**4.14%**
Conversational QA	COSTAR	49.33%	63.65%	15.02%	4.17%
Retrieval QA	COSTAR	30.97%	53.20%	16.68%	3.58%
Llama2		**Faithfulness**	**Answer Relevancy**	**Context Precision**	**Context Recall**
Chunking					
Page Level		15.61%	52.09%	14.37%	3.40%
Sentence Level		**59.14%**	**57.67%**	**14.60%**	**3.47%**
Retrieval	**Prompt Engineering**				
Conversational QA	Minimal (provide basic instructions)	**67.06%**	**60.53%**	**14.32%**	**3.78%**
Conversational QA	COSTAR	65.40%	58.53%	14.17%	3.77%
Retrieval QA	COSTAR	21.91%	41.42%	15.97%	2.10%

In Table 4, we looked at how Mistral and Llama2 models performed with different ways of organizing and finding information for the Adobe dataset. We noticed some big differences in how well they did.

For Mistral, when we switched from breaking down info by pages to breaking it down by sentences, there were some huge improvements in how truthful and relevant the answers were. For example, faithfulness went up from 6.82% to 45.46%, and answer relevancy increased from 55.62% to 63.49%. However, the improvements weren't as big for context precision and recall.

On the other hand, Llama2 did well no matter how we organized the information. It consistently did better than Mistral, especially when we looked at how truthful and relevant its answers were. For instance, at the sentence-level chunking, Llama2 achieved a faithfulness score of 59.14% and an answer relevancy score of 57.67%.

Both Mistral and Llama2 did best when we asked questions in a simpler, more conversational way using minimal prompt engineering. This method consistently gave us better scores for how truthful and relevant the answers were compared against COSTAR. So, asking questions in a simpler way seemed to work best for both models.

Overall, based on what we found, the Llama2 model with sentence-level chunking and asking questions in a conversational way using minimal prompt engineering seemed to be the best setup. It consistently gave us the most truthful and relevant answers, showing that it's the most effective way to use the RAGAs system for the Adobe dataset.

Table 6 Evaluation Results on NTUC Employee Handbook

Mistral		Faithfulness	Answer Relevancy	Context Precision	Context Recall
Chunking					
Page Level		30.18%	55.54%	13.28%	1.69%
Sentence Level		**45.93%**	**57.72%**	**12.58%**	**1.39%**
Retrieval	**Prompt Engineering**				
Conversational QA	Minimal (provide basic instructions)	45.32%	55.09%	13.66%	1.90%
	COSTAR	**45.45%**	**55.36%**	**13.63%**	**1.97%**
Retrieval QA		34.23%	50.31%	14.34%	1.60%

Llama2		Faithfulness	Answer Relevancy	Context Precision	Context Recall
Chunking					
Page Level		42.07%	52.97%	13.18%	1.98%
Sentence Level		**51.71%**	**59.04%**	**12.75%**	**1.43%**
Retrieval	**Prompt Engineering**				
Conversational QA	Minimal (provide basic instructions)	57.20%	59.05%	12.88%	1.59%
	COSTAR	**59.49%**	**57.69%**	**12.82%**	**1.59%**
Retrieval QA		23.60%	42.31%	13.20%	0.87%

Looking at Mistral and Llama2 models across different ways of breaking down information and finding it again, we see some big differences in how well they work, for the NTUC dataset in Table 5.

For Mistral, when we switched from breaking down info by page to breaking it down by sentence, there were some improvements in how truthful and relevant the answers were, but not as much as with Llama2. Llama2 did better overall, especially when we looked at how truthful and relevant its answers were. For example, Llama2 got a faithfulness score of 51.71%, while Mistral only got 45.93%.

Both models did better when we asked questions in a more conversational way, especially using COSTAR. Mistral got a faithfulness score of 45.45% and a relevancy score of 55.36% with this method, and Llama2 got even higher scores - 59.49% for faithfulness and 57.69% for relevancy. Asking questions in a simpler way didn't work as well for either model.

When we compare all the scores we got, it's clear that Llama2 did better than Mistral in pretty much every way, especially in being truthful and relevant. Also, breaking down info by sentence worked better than by page for both models.

In conclusion, based on our findings, the best way to set up the RAGAs system seems to be using Llama2, breaking down info by sentence, and asking questions in a conversational way using minimal Prompt Engineering. This setup consistently gave us the most truthful and relevant answers, showing that it's the most effective way to use the RAGAs system.

However, we did notice some problems with getting the right amount of detail in our answers. The answers we used to compare were often just short phrases, which didn't match up well with how the models were trained. This made it hard to get high scores for how well the answers matched up with each other.

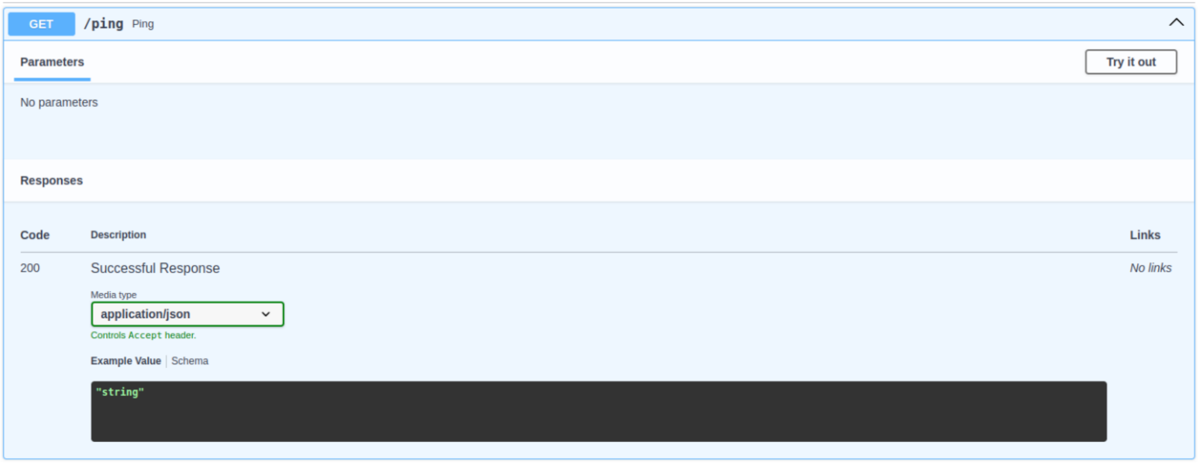
Overall, our evaluation method gives a good way to look at how well the RAGAs system works, but there are still some things we need to work on to make it even better.

4.3 Backend

In crafting the backend architecture for this project, several considerations were pivotal in the selection of FastAPI as the framework of choice. FastAPI's asynchronous capabilities were instrumental in handling concurrent requests efficiently, especially during intensive tasks such as document processing, embedding, and summarization. Its robust design and adherence to modern Python standards facilitated the development of a scalable and responsive API. Additionally, FastAPI's support for OpenAPI documentation streamlined endpoint management and ensured clear documentation of API functionalities, which will be crucial for future scaling and deployment of GuruDocs.

- While currently we are not using FastAPI asynchronously, will be needed for future implementations with higher workloads

The backend system comprises three key endpoints designed to handle distinct functionalities. These endpoints—embed, query, and summary—form the backbone of document embedding, user query handling, and document summarization within the application. Besides the three key functional endpoints, we have also implemented a ping endpoint to allow us to check whether the FastAPI service is healthy. Figure X shows the OpenAPI documentation of our ping endpoint.



4.3.1 Embed Endpoint

This endpoint facilitates the extraction and embedding of PDF documents received from the frontend. Utilizing FastAPI, it processes base64 strings representing PDFs that is passed from the frontend, storing them temporarily for efficient handling. Upon receiving the document, the backend employs llm_utils functions to dissect the content into sentences, chunk them appropriately, generate vector embeddings, and store these embeddings in Chroma vectorstore. The vectorstore object and pages are stored in different dictionaries, each with unique UUIDs as keys to access the objects. To manage data transfer efficiently, the API response returns the unique UUIDs of the vectorstore and pages, enabling subsequent operations without overwhelming the response payload. Upon completion of document embedding processes, the temporary PDF file is removed from local file storage and the “summary” endpoint will be automatically called. Figure X shows the OpenAPI documentation for embed endpoint.

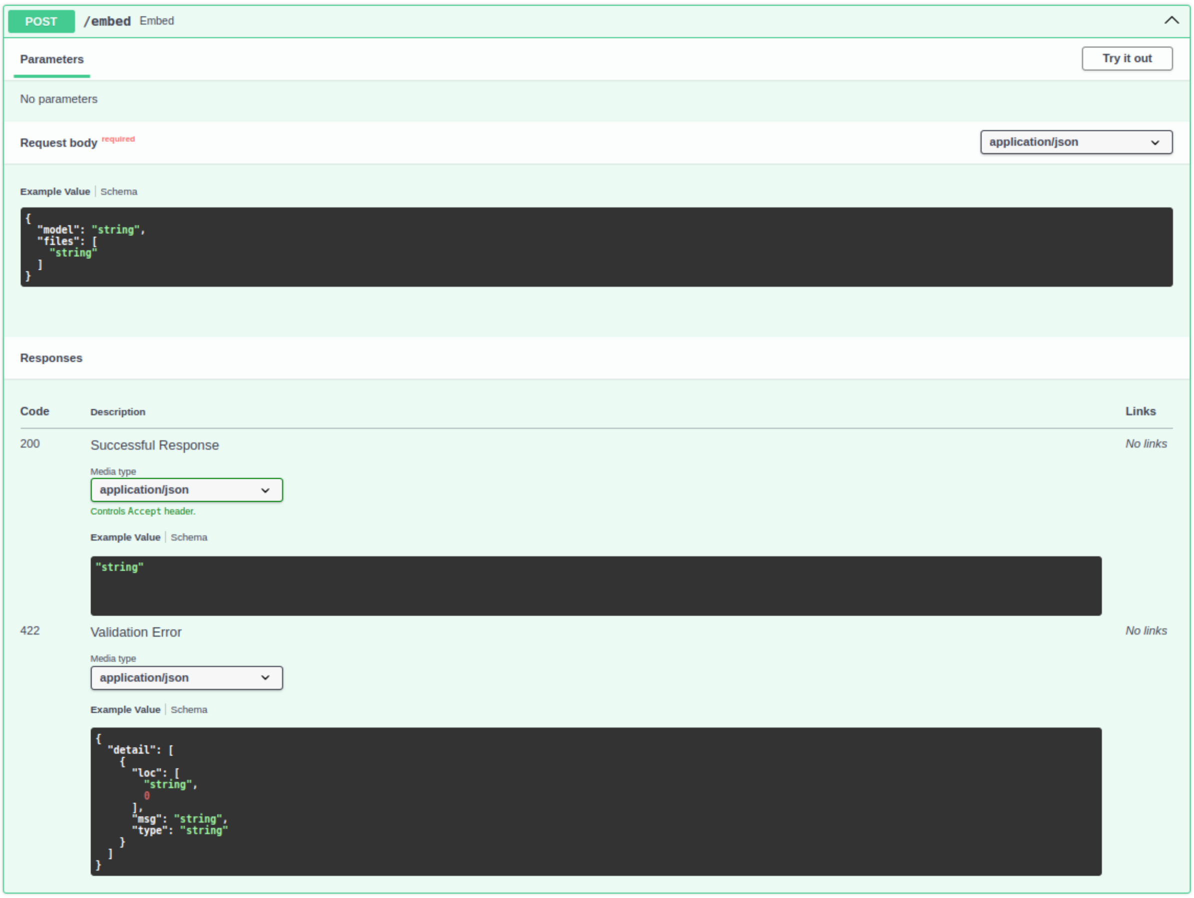


Figure X: OpenAPI documentation of embed endpoint

4.3.2 Summary Endpoint

This endpoint takes in the UUID of the pages chunks from the "embed" endpoint and the name of the LLM in the API body. The pages are then retrieved from the dictionary stored in memory and the relevant functions from llm_utils will then be called to generate a summary of the document as described in Section 4.1.3 above. This endpoint is automatically called when the document has been successfully chunked and embedded in the vectorstore. Figure Y shows the OpenAPI documentation of the "summary" endpoint.

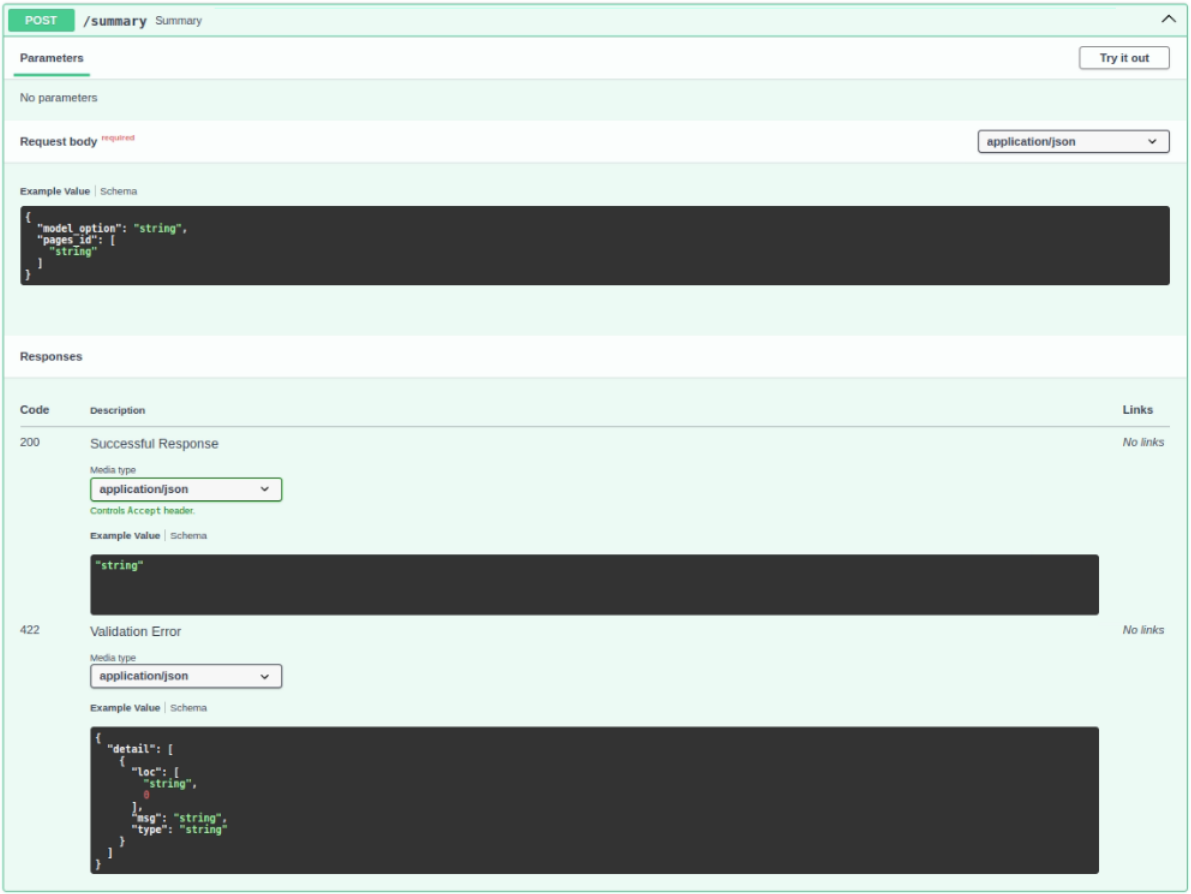


Figure Y: OpenAPI documentation of summary endpoint

4.3.3 Query Endpoint

This endpoint takes in the unique UUID of the vectorstore where the document chunks have been embedded, the model of choice and user's query in the API body. Similar to how the pages UUID was used to retrieve the document pages, the vectorstore UUID is used to retrieve the vectorstore object from the dictionary stored in memory. The relevant functions will then be called in llm_utils to generate vector embeddings of the user's query, match it with the document vector embeddings in the vectorstore to find those that are similar and retrieve it. The text of the retrieved vector embeddings are then passed into the LLM as context to generate a completion which is returned as the API response.Figure Z shows the OpenAPI documentation of the "query" endpoint.

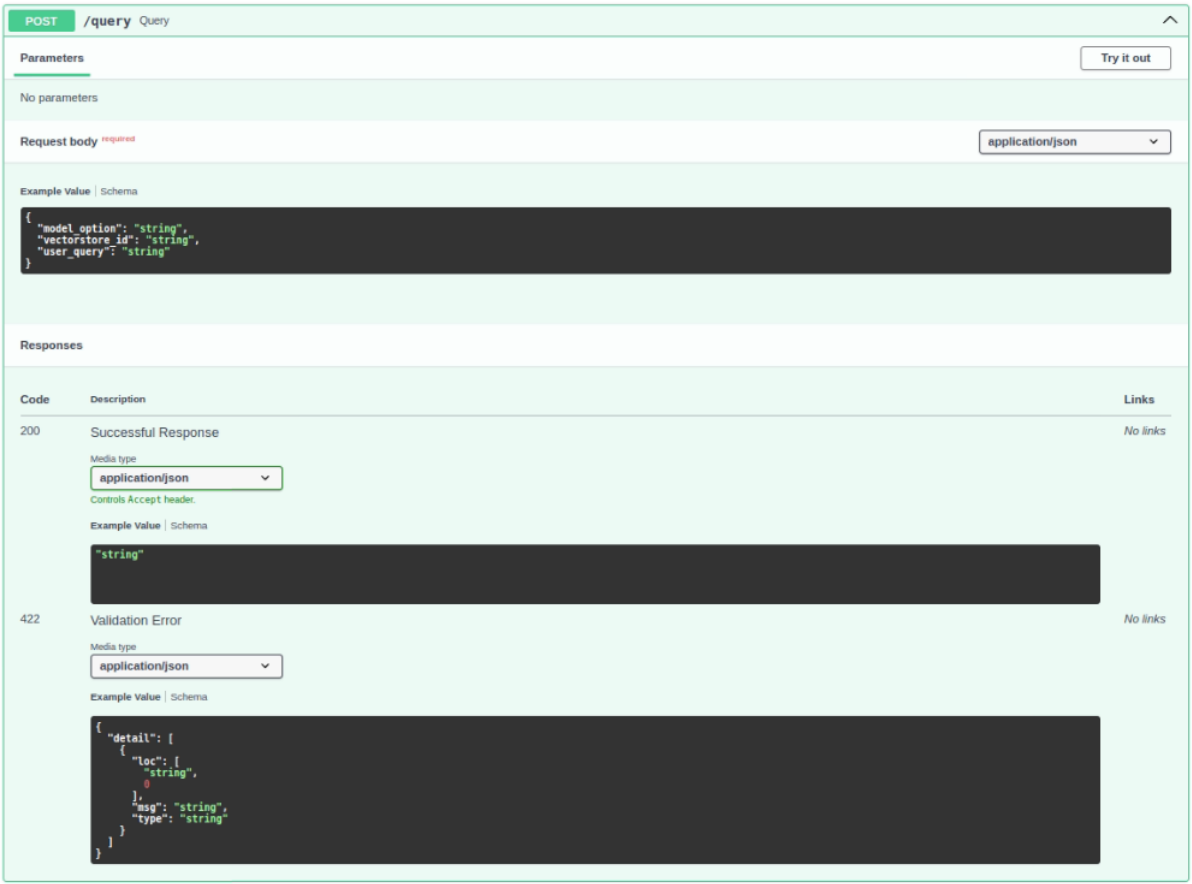


Figure Z: OpenAPI documentation of query endpoint

4.4 Frontend

GuruDocs' initial frontend development phase leveraged Streamlit for its rapid prototyping capabilities and ease of use in crafting interactive data-driven applications. However, as the project progressed, we encountered limitations within Streamlit. To address these challenges and ensure a more robust and customizable frontend experience, we transitioned to the React framework. React's component-based architecture, efficient virtual DOM rendering, and extensive ecosystem of libraries and tools provided us with the flexibility of our application.

4.4.1 Streamlit

In the initial phase of development using Streamlit, our primary focus was on building foundational features that would lay the groundwork for the application's functionality. The first feature we developed was the File Upload feature, allowing users to upload documents or files directly into the application. This feature was essential as it enabled users to interact with the application by providing input data for processing.

We then implemented a Chatbot feature that allowed users to engage with the application through a conversational interface. This Chatbot served as a user-friendly way for users to interact with the uploaded documents. These initial features were designed to showcase basic functionality and were not integrated with the backend. Figure 4 shows the first iteration of the Streamlit frontend.

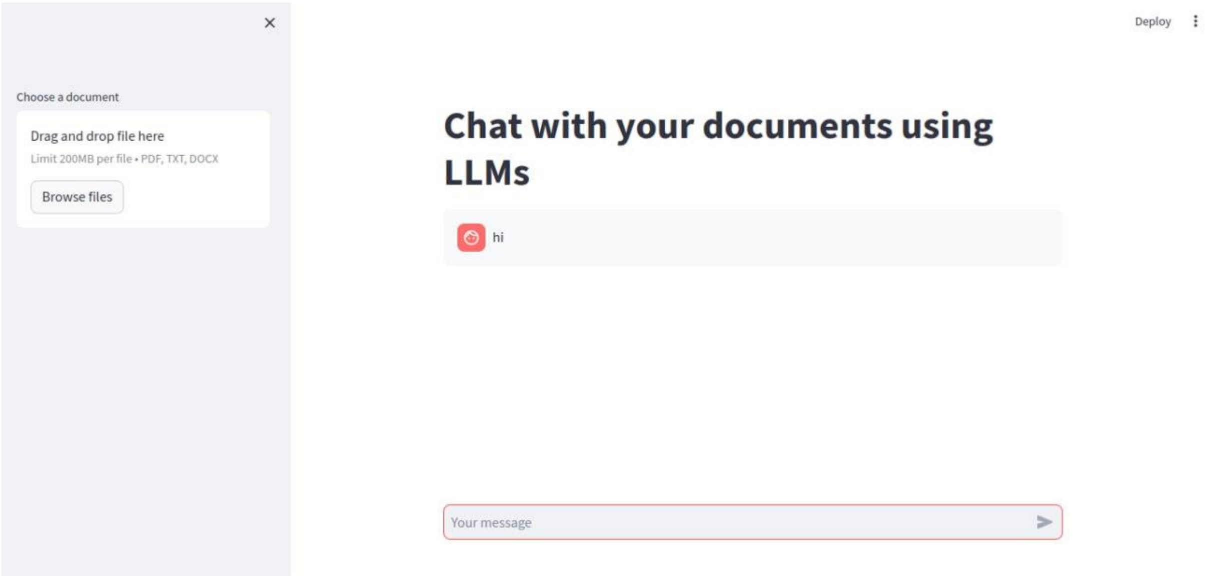


Figure 5: First iteration of GuruDocs frontend without integration with backend logic

In the second iteration, we expanded upon the initial features by integrating them with the backend RAG process logic. The File Upload feature was integrated with the backend processes to call the "embed" API to chunk and embed the document chunks in the vector database.

Simultaneously, the Chatbot feature was integrated with the backend where the vector embeddings of user's query was extracted using the same embedding model, and matched with the embeddings in the vector database. The generated output from the LLM based on extracted contexts were then presented to the user as a formatted response.

This iteration demonstrated the key features and functionalities of GuruDocs. Figure 5 shows the second iteration outcome of the Streamlit frontend.

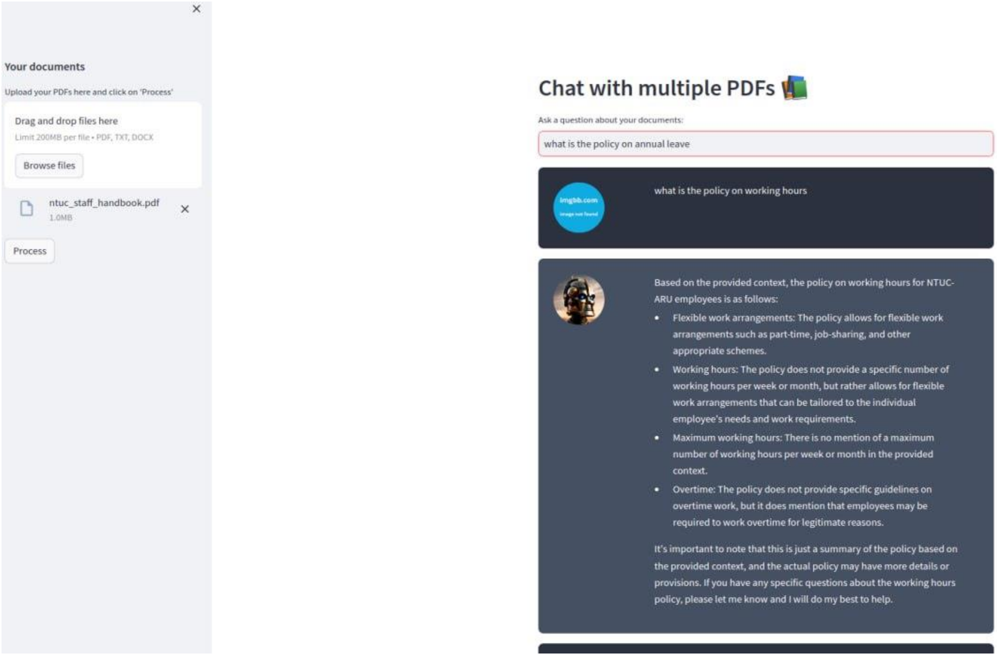


Figure 6: Second iteration of GuruDocs' frontend with integration to backend logic

The final iteration on Streamlit focused on implementing the remaining features of GuruDocs such as the Model Option feature and a placeholder for extracting document summaries. The Model Option feature provided users with the ability to choose from different LLMs to generate the response, offering flexibility and customization. This feature was also integrated with the backend to call the model that was selected by the user in the response generation step.

Furthermore, we implemented a placeholder for the document summary feature while waiting for backend developments. Figure 6 shows the final iteration of GuruDocs' frontend on Streamlit.

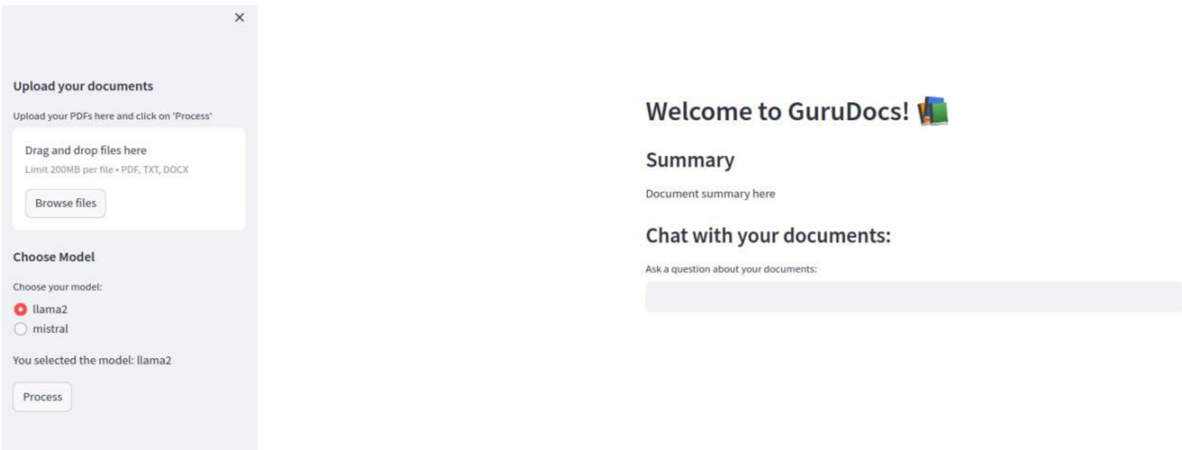


Figure 6: Final iteration of GuruDocs frontend on Streamlit

While Streamlit was chosen initially for its benefits of ease of use and pythonic nature, we also realised the limitations of Streamlit.

One notable limitation we encountered was the need for Streamlit to be run as a main application This meant that the whole application interface would be refreshed whenever there was a small update or when the response was returned from the backend. This led to performance inefficiencies, especially when the user poses a large number of questions.

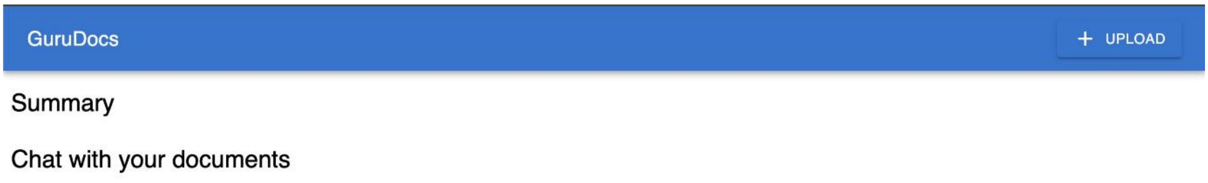
Moreover, Streamlit's limited customizability posed challenges for displaying informative loading interfaces. For instance, our application required notifying users about ongoing processes, such as document embedding or backend retrieval and generation tasks, which often required significant processing time. However, Streamlit's constraints made it difficult to implement dynamic loading indicators or progress notifications, hindering user feedback and engagement

We also considered a future full-fledged web deployment of GuruDocs, and Streamlit's scalability considerations emerged as a concern. Streamlit's architecture as the main application posed challenges for incorporating load balancers, gateway services, or server-side technologies like Nginx, which are crucial for managing high traffic and ensuring optimal performance in scalable web applications.

These limitations highlighted the need for a more flexible and scalable frontend framework, leading to our decision to transition to React. React's component-based architecture, robust state management capabilities, and support for server-side rendering and backend integrations addressed these challenges, allowing us to deliver a more seamless, customizable, and scalable user experience in our application's web deployment phase.

4.4.2 ReactJS

Due to the abovementioned limitations, the team decided to transition to ReactJS for the frontend framework despite not having any experience in React or frontend development. We focused on building the key features such as document upload, summary display and chatbot. Similar to the development approach taken for Streamlit, our first iteration of development on React did not include backend logic integration. To maximise real estate on the home page for users to interact with the chatbot and view the document summary, we decided to implement the document upload and model selection features in a Dialog component. Figure 7 shows the first iteration of GuruDocs on React.



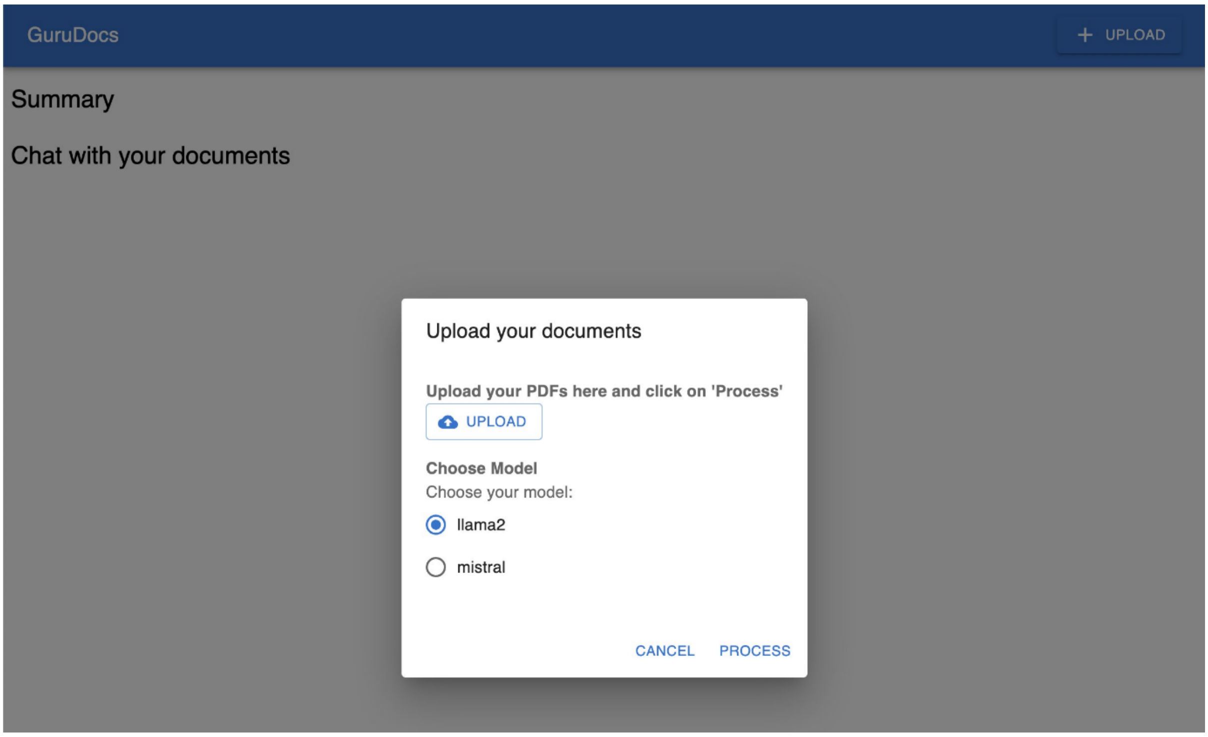


Figure 7: First iteration of GuruDocs homepage (above) and the upload Dialog component (below) on React.

In the second iteration, we integrated the features in the Dialog component with the backend logic. Once the document was uploaded and the user has selected their model of choice, the document will be sent as base64 string to the backend "embed" API to perform chunking and extract the vector embeddings to store in the vector database. As mentioned in Section 4.3.1, we also implemented the logic to automatically trigger the "summary" endpoint to be called once the vectors of the document chunks have been stored in the vector database. As such the document summaries will be displayed on the frontend once the "summary" endpoint returns the summary response. Figure 8 shows the second iteration of GuruDocs frontend.

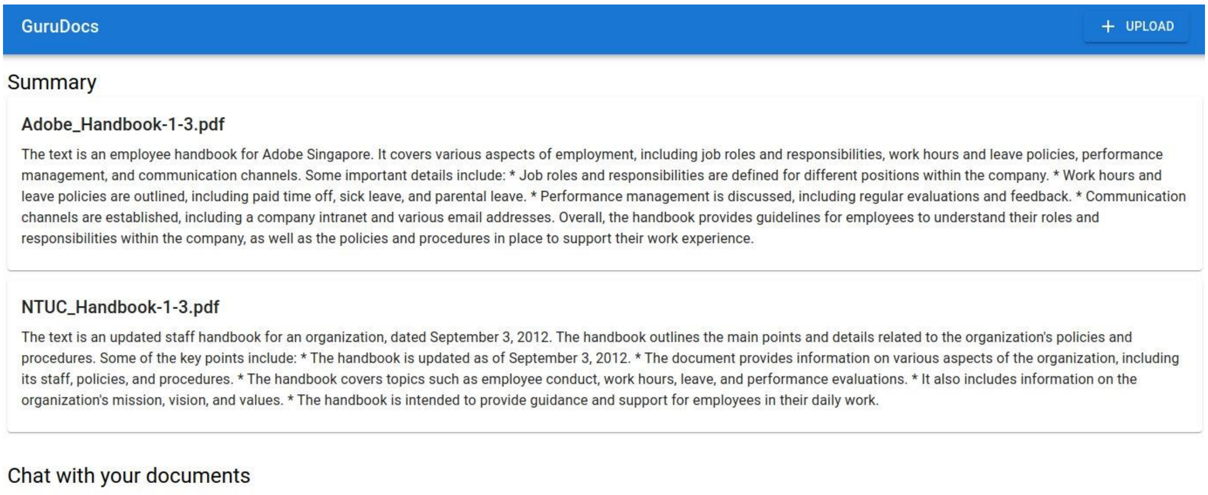


Figure 8: Second iteration of GuruDocs frontend where the document summaries will be automatically displayed once the documents have been embedded and summary has been generated.

In the next iteration, we implemented the Chatbot feature using React's *react-chatboot-toolkit* library and integrated it with the backend "query" API endpoint once the user submits a query. As the API response usually takes a couple of seconds to process and return a response, we return a default message on the chatbot, "Sending query, please wait..." to inform the user that the message has been sent to the backend. Once the "query" API endpoint returns a response, it will be rendered on the chatbot as a reply. We also added in icons to each of the section headers. Figure 9 shows the GuruDocs frontend implementation with Chatbot feature.

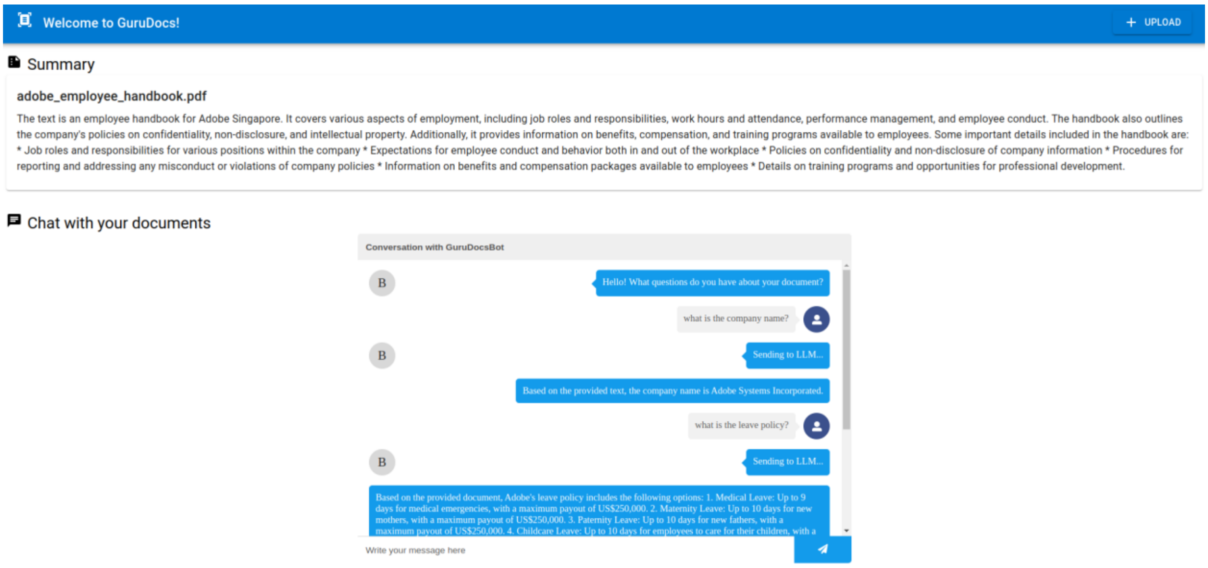


Figure 7: Third iteration of GuruDocs frontend with Chatbot implemented.

In the final round of iteration of GuruDocs frontend, we focused on the smaller features that were important for a good user experience. First, we added a display of the filename for the file that was selected by the user as shown in Figure 10. Second, we added the Spinner components with description when the documents are being chunked and embedded, and when the summary is being generated as shown in Figure 11. This allows the user to know which processes are running. We also added a toast message to inform the user when the document has been successfully embedded as seen in Figure 12. Finally, we implemented HTML wrapping of the texts as some of the summary or chatbot responses returned by the API consists of several lines with new lines or numbered lists.

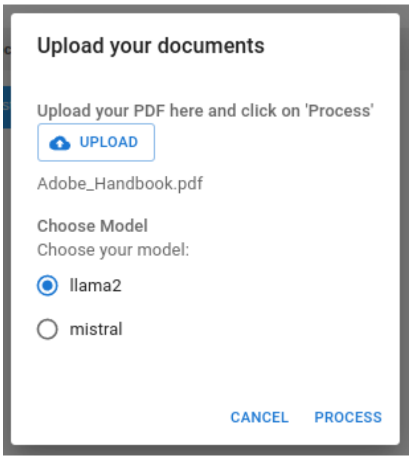


Figure 10: Display of filename of the file selected by the user in the upload Dialog component

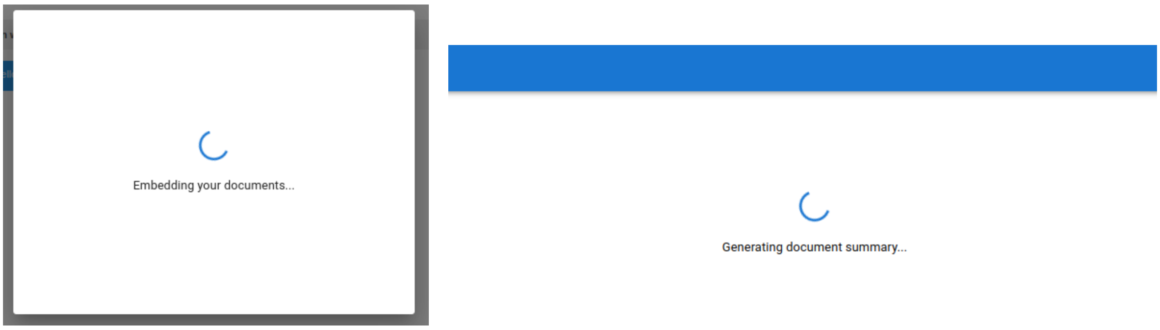


Figure 11: (Left) Spinner component in Dialog component when documents are being chunked and embedded, and (Right) Spinner component in Summary section when the document summary is being generated

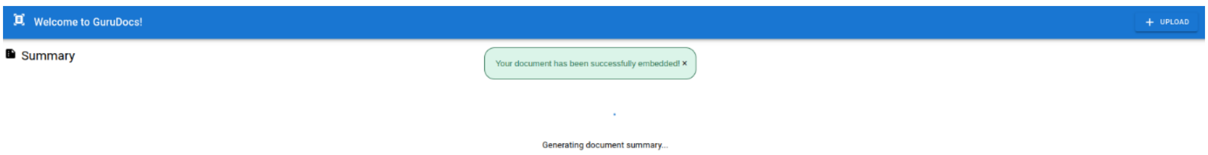


Figure 12: Toast message to inform the user that the document has been successfully embedded

While the current frontend UI may not be the most aesthetic or stylistic, it is functional and fulfils the key requirements of GuruDocs, to allow the user to upload a document, select their model of choice and pose queries that they have on their document.

5 CONCLUSION

5.1 Summary

In conclusion, GuruDocs presents a solution to a common workplace challenge faced by employees dealing with policy and/or regulatory documents. By leveraging RAG and LLMs, GuruDocs, provides a user-friendly and efficient platform for querying documents, ultimate enhancing the overall user experience and productivity.

5.2 Challenges and Limitations

- Compute (GPU and CPU) requirements

5.3 Future Recommendations

Multi-Modal Capabilities. For future implementations, we can explore expanding the capabilities of GuruDocs beyond text-based documents to incorporate multi-modal understanding. Our current implementation only ingests texts from documents, which creates limitations when documents contain images or tabular data as GuruDocs would lack data extraction accuracy and coverage of diverse content formats. By integrating technologies such as computer vision for image analysis, optical character recognition (OCR) for text extraction from images, and support for other document or image formats, GuruDocs can achieve a more comprehensive analysis of documents. This expansion not only enriches content analysis by recognizing visual elements like graphs, tabular data and identifying key information in images but also allows for a holistic view of the document's structure and insights.

User Authentication and Management. Second, we can introduce user authentication and management for future implementations. Our current implementation did not require user authentication as it was designed for local, on-premise installation on the user's laptop with no shared account access. However to cater for robustness in future implementations on Cloud, user authentication and management will be needed. This would be crucial for ensuring the security and privacy of user data. By incorporating features such as secure login, role-based access control, and encryption of sensitive data, the GuruDocs can prevent unauthorized access to confidential information and mitigate the risk of data leakage. This recommendation aims to establish trust and confidence among users by prioritizing data security and privacy.

Trust and Safety Guardrails. Third, future implementations of GuruDocs can include trust and safety guardrails which are important in protecting users from potential risks such as malicious content or misleading information. While RAG with LLM applications can have fewer instances of model hallucinations as the context is provided for the LLM to generate completions, GuruDocs can still proactively pre-empt against possible hallucinations or malicious users and intents.

Knowledge Graph Visualisation. Fourth, we can integrate GuruDocs with knowledge graph representation libraries such as RAGxplorer[1]. This would allow the user to better visualise the vector queries and understand which vector chunks were identified to be more similar to their user query. Users are also able to hover over the vector chunks to view the corresponding text of the chunks as well. Figure X below shows an example of RAGxplorer's visualization, extracted from their GitHub.

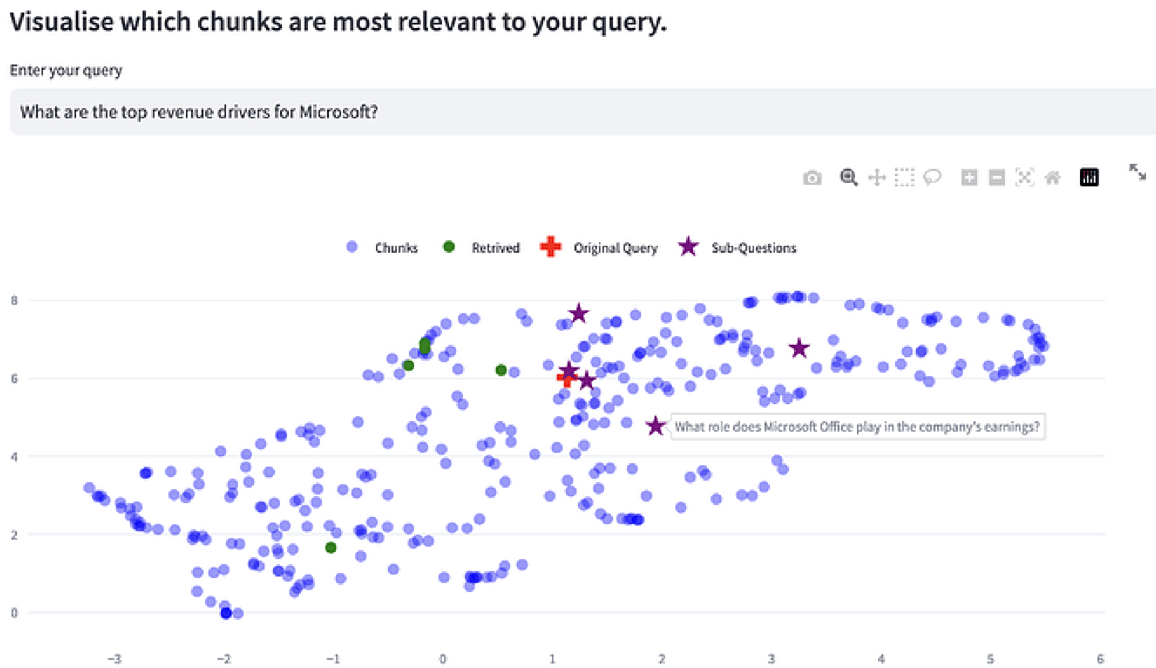


Figure X: Example of RAGxplorer's visualization. Users are able to view the vector representation of their query together with that of the vector chunks from the document, and easily identify which chunks are similar to their query.

Customisable Configurations. Finally, further customisation configurations can be provided for the user to allow them to have more options such as choosing the embedding model to use, chunk size, etc. Our current implementation of GuruDocs only has one configurable parameter (model option) as we designed it with the intention of allowing users without generative AI or RAG knowledge to be able to use our product. As users become increasingly familiar with generative AI and concepts of RAG, customisation of configurations would allow more savvy users to explore different parameters on their document question and answering results.

---

[1] https://github.com/gabrielchua/RAGxplorer

6 REFERENCES

DeepLearning.AI. (01 February, 2024). *LangChain for LLM Application Development*. Retrieved from DeepLearning.AI: https://www.deeplearning.ai/short-courses/langchain-for-llm-application-development/

DeepLearning.AI. (01 Feburary, 2024). *LangChain: Chat with Your Data*. Retrieved from DeepLearning.AI: https://www.deeplearning.ai/short-courses/langchain-chat-with-your-data/

LangChain. (01 Mar, 2024). *Embedding models*. Retrieved from LangChain: https://python.langchain.com/docs/integrations/text_embedding/

LangChain. (01 Feb, 2024). *Introduction*. Retrieved from LangChain: https://python.langchain.com/docs/get_started/introduction

Sojasingarayar, A. (01 Mar, 2024). *Summarization with LangChain*. Retrieved from Medium: https://medium.com/@abonia/summarization-with-langchain-b3d83c030889

Teo, S. (1 Apr, 2024). *How I Won Singapore's GPT-4 Prompt Engineering Competition*. Retrieved from Medium: https://medium.com/@mrumair384/how-i-won-singapores-gpt-4-prompt-engineering-competition-361253d3b14a

Yang, S. (01 Mar, 2024). *4 Ways to Do Question Answering in LangChain*. Retrieved from Medium: https://towardsdatascience.com/4-ways-of-question-answering-in-langchain-188c6707cc5a

7 APPENDICES

Appendix 7.1: Summary Results

Summarization Technique	Summary Results
Stuff	Personal information about employees, including address and phone number, is confidential and will not be shared outside the company without authorization. Employees are expected to dress appropriately for work, in smart business attire. The company does not accept responsibility for damage or loss of personal property on its premises. Loans or loan guarantees are not provided to employees. All employees must present for work sober and without any prohibited substances, including alcohol and non-prescription drugs. Outside committees/boards requests by employees must be approved by Adobe's Legal Department. The company is committed to providing opportunities for employee development and career progression.
Map_Reduce	The text is an employee handbook for Adobe Singapore. It covers various aspects of employment, including job roles and responsibilities, work hours and breaks, performance management, communication channels, and policies on confidentiality, intellectual property, and non-discrimination. The handbook also outlines the company's expectations for employees in terms of conduct and behavior, and provides information on benefits, compensation, and training opportunities. Some important details included in the handbook are: * Job roles and responsibilities for various positions within the company * Expectations for employee conduct and behavior both in and out of the workplace * Policies on confidentiality and intellectual property * Procedures for reporting and addressing any issues or concerns * Information on benefits, compensation, and training opportunities available to employees.
Refine	Based on the new context provided, here is a refined summary of Adobe's policies and benefits for its employees: Adobe prioritizes creating a healthy and productive work environment for all employees, with a focus on innovation, involvement, and people as the greatest asset. The company has established clear policies and procedures that are transparent and fair, including licenses, maternity/paternity leave, adoption leave, family leave, medical leave, and other benefits. Adobe also has a set of core values that are at the heart of its culture, such as authenticity, innovation, involvement, and people as the greatest asset. In addition to the existing summary, here are some additional points that can be added: * Adobe provides a medical check-up benefit for employees only, which includes a preventive health check-up (body screening) not for the medical check-up for the treatment of specific illness. Family members are not eligible for this medical check-up benefit. * Adobe also provides vision reimbursement for employees only, which covers preventive eye exam, prescribed glasses/lens and contact lens. Exclusions include cosmetic surgery, laser surgery, eye surgery that requires specialist care, contact lens solution, sunglasses, etc. Family members are not eligible for this vision reimbursement benefit.

	* Adobe has a business travel insurance policy which provides employees with three times their salary, up to a maximum of US$500,000. The program pays benefits if an employee dies or becomes disabled as a result of an accident while traveling on business. In terms of responsibilities, employees are responsible for immediately notifying their manager or HR if they experience harassing or discriminatory behavior and/or are aware of any such behavior by any individual. Managers are responsible for making employees aware of the policy and preventing harassment from occurring, while HR is responsible for undertaking effective, thorough, and objective investigations of any harassment allegations and handling any appropriate disciplinary or remedial action in respect of a complaint. Adobe takes harassment and discrimination seriously and has procedures in place to handle any incidents of such behavior. Any employee or contractor who engages in conduct contrary to this policy will be disciplined, and serious cases of discrimination, harassment or victimisation will result in termination of employment. There are no exceptions to this policy, and employees are encouraged to voice grievances connected with their work to obtain a response from Adobe. However, the abusive use of this policy, such as raising unfounded allegations with malicious intent, will be treated as a serious disciplinary matter. Overall, Adobe prioritizes creating a healthy and productive work environment for its employees while also providing a range of benefits and policies to support their well-being and professional growth. Please let me know if you would like me to refine the summary further or if there is anything else I can help with.

Appendix 7.2: Installation and User Guide

Note: GuruDocs was developed on <specs> as running the LLM models locally requires significant compute. Do note that there would be significant latency for lower specifications machines.

Installation Procedure:

1. Install Ollama, see download instructions at https://ollama.com/download
2. Clone GuruDocs repository

```
git clone
cd GuruDocs/
```

3. Download LLM models from Ollama

```
ollama pull mistral
ollama pull llama2
```

Once you have completed the above steps, you can choose to run the application using either Docker containers or run it locally.

A7.2.1 Running GuruDocs using Docker Container

Prerequisites:

1. Ensure that you have **Docker engine** installed. If not, please follow instructions from https://docs.docker.com/engine/install/
2. **Nvidia container toolkit** to run Docker images with GPU. Please follow instructions from https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html if you do not have it installed.

Once you have the above installed, you can proceed to build the Docker image

Building Docker image

1. You can build the Docker images using the shell script that we have prepared.

```
chmod +x ./docker_build.sh
./docker_build.sh
```

2. Once the docker images have been built, spin up the Docker containers using the shell script that we have provided as well. You can spin down the containers using the stop command as well.

```
chmod +x ./gurudocs.sh
./gurudocs.sh start
./gurudocs.sh stop
```

3. After you have started the containers with the shell script, open your browser and go to http://localhost:3001. The page will load once the containers are running.

A7.2.2 Running GuruDocs using Local Installation

Requirements:

- Python 3.10
- NodeJS v18.20.1 (recommend to use nvm install)

Installation and run:

1. Create your conda environment (or venv)

```
conda create -n gurudocs python=3.10 -y
conda activate gurudocs
pip install -r requirements.txt
```

2. Start the application, both frontend and backend

```
python main.py
cd frontend/
```

```
npm start
```

3. Once the frontend has started, your web browser should automatically open with the address http://localhost:3001

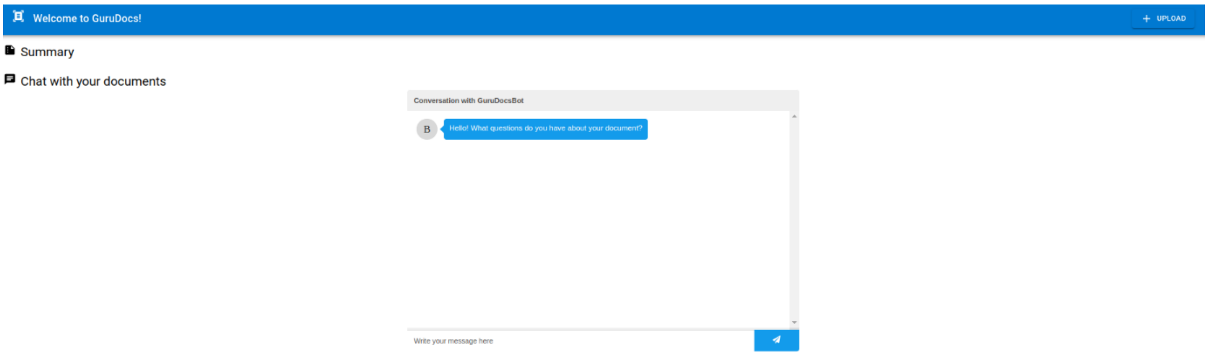
Troubleshoot

If you face issues with starting the frontend, we recommend removing all node modules and installing them again. You can do so using the following code. Once you have completed this, run the above steps again.

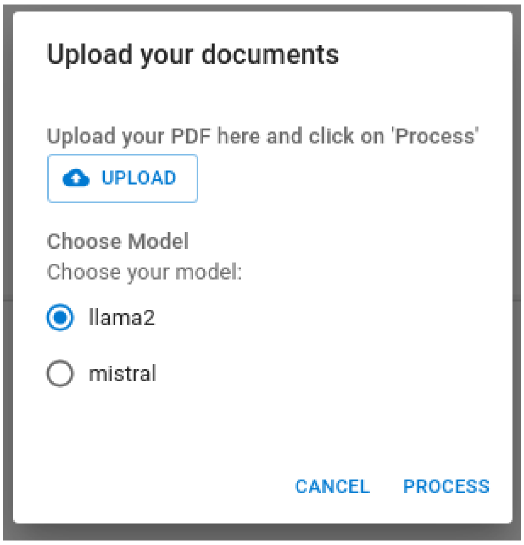
```
cd frontend/
rm -rf node_modules
npm install
```

A7.2.3 How to use GuruDocs

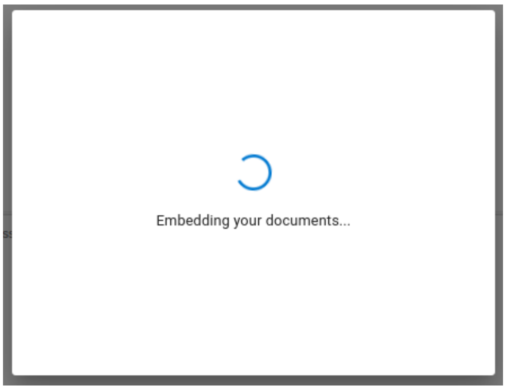
1. Once your application has started successfully, you should see the main page.



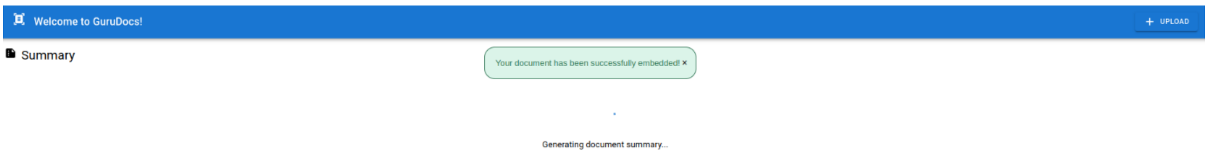
2. Click the “Upload” button on the top right hand corner and a Dialog will pop up
3. In the Dialog window, you can:
    a. Upload the document (PDF) that you want to query by clicking the “Upload” button. Once you have selected your document, the document name should appear below the “Upload” button.
    b. Select the LLM you would like to use in the RAG process. The default option is “llama2”
    c. Once ready, click on “Process”



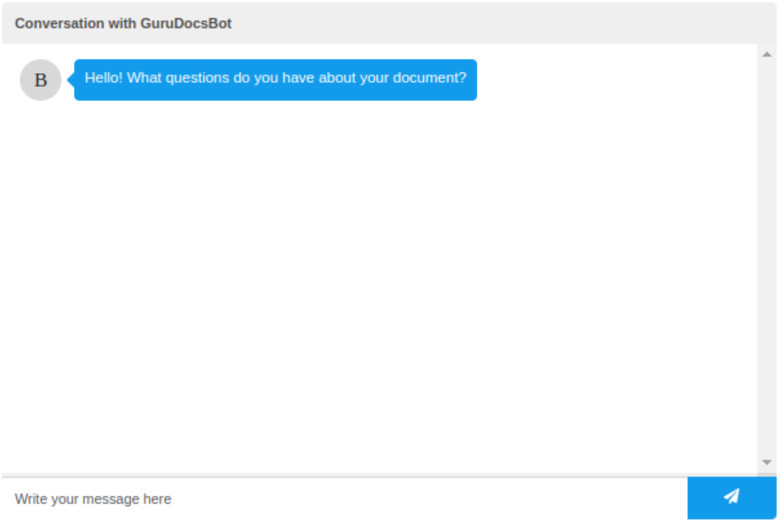
4. After you have clicked on "Process" the document chunking and embedding will begin. You will see a spinner window to let you know the process. You can either close this window by clicking anywhere outside of the Dialog box, or you can wait for the process to complete and it will automatically close when done.
   a. Do wait for the green toast message in Step 5 below before you start chatting with GuruDocs. Otherwise GuruDocs will reply "Please upload a document or wait for the document processing to be completed."



5. You will see a toast message in green once the document has been successfully embedded. This means that you can start chatting with GuruDocs on your queries! In the meantime, the document summary will be generated in the background and you will see the spinner in the "Summary" section.



6. To start chatting with GuruDocs, simply type your question into the chatbot and hit "Enter".



Appendix 7.3: Mapped System Functionalities to Intelligent Reasoning System Courses

Course	Mapped Functionalities
Machine Reasoning	• Knowledge Search: To find and retrieve relevant information sought by the user
Reasoning Systems	• Knowledge based reasoning: Understanding, interpreting and synthesizing information to provide meaningful responses to user queries
Cognitive Systems	• Natural Language Understanding: embedding of document text chunks • Natural Language Generation: generating coherent answers from extracted content based on user's semantic query

Appendix 7.4: Project Proposal

Refer to separate PDF attachment in ZIP file.

Appendix 7.5: Individual Project Report

Project Team Member 1: Alvin Wong Ann Ying

1) Personal Contribution
 	
• Conceptualised project idea with team
• Literature review for market research
• Data annotation for Ground truths datasets.
• End-to-end evaluation pipeline
    o Utilizing t5 model for generation of Q&A pairs
    o Implemented a modified RAGAs evaluation functionality
    o Evaluated all the performances of all experiment's permutations with multiple runs provided in the excel sheet.
    o Provided an automated script for running batch evaluation
 	
2) **Most valuable knowledge/skills acquired**
 	
3) How to apply knowledge/skills acquired in your workplaces

Project Team Member 2: Brandon Chua Hong Huei
1) **Personal Contribution** During the GuruDocs project, I assisted in the conceptualization of the project idea with Si Ci by developing a simple proof-of-concept to validate its feasibility. Upon the successful validation, I conducted in-depth research into various modules of the solution, including the database architecture, Large Language Models (LLMs), types of embeddings, document splitting methodologies (chunking techniques), retrieval techniques, summarization techniques, and prompt templates. Additionally, I took charge of developing the RAG model from scratch, ensuring its seamless integration with other components of the solution. Furthermore, I actively contributed to the evaluation of the solution, refining it based on feedback and performance metrics. **2) Most valuable knowledge/skills acquired** The most valuable takeaway being the knowledge of how to build a Retrieval-Augmented Generation (RAG) solution. This hands-on experience has not only deepened my understanding of NLP concepts but has also honed my skills in using the LangChain framework, which is instrumental in developing such solutions. Additionally, I have gained valuable insights into the complexities of document processing and information retrieval, which have broadened my skill set as an AI practitioner. **3) How to apply knowledge/skills acquired in your workplaces** The knowledge and skills acquired are highly applicable in my current workplace as an AI Engineer. The trend in AI is shifting towards large multi-modal models and intelligent agents, and my experience with LangChain and RAG solutions positions me well to contribute meaningfully to such projects. My understanding of these advanced concepts acts as a stepping stone for me to propose and explore new use cases in my work, thereby enhancing the capabilities of our AI systems and driving innovation in our organization.

Project Team Member 3: Ong Si Ci

1) **Personal Contribution**

- Conceptualised project idea
- Coordinated and delegated project tasks to team members
- Full stack architecture design
- Frontend (Streamlit and React) and backend API implementation
- Dockerisation of frontend and backend. Prepared docker compose and shell scripts for easy installation
- Initial implementation of RAG model with Brandon

2) **Most valuable knowledge/skills acquired**

Through the development of GuruDocs, I gained valuable knowledge on RAG

3) **How to apply knowledge/skills acquired in your workplaces**