Project Workflow Document: RAG-Based Contract Query System

Project Title:

Development and Deployment of a RAG-Based Contract Query System

Project Duration:

16th August 2024 – 20th September 2024

Team Members:

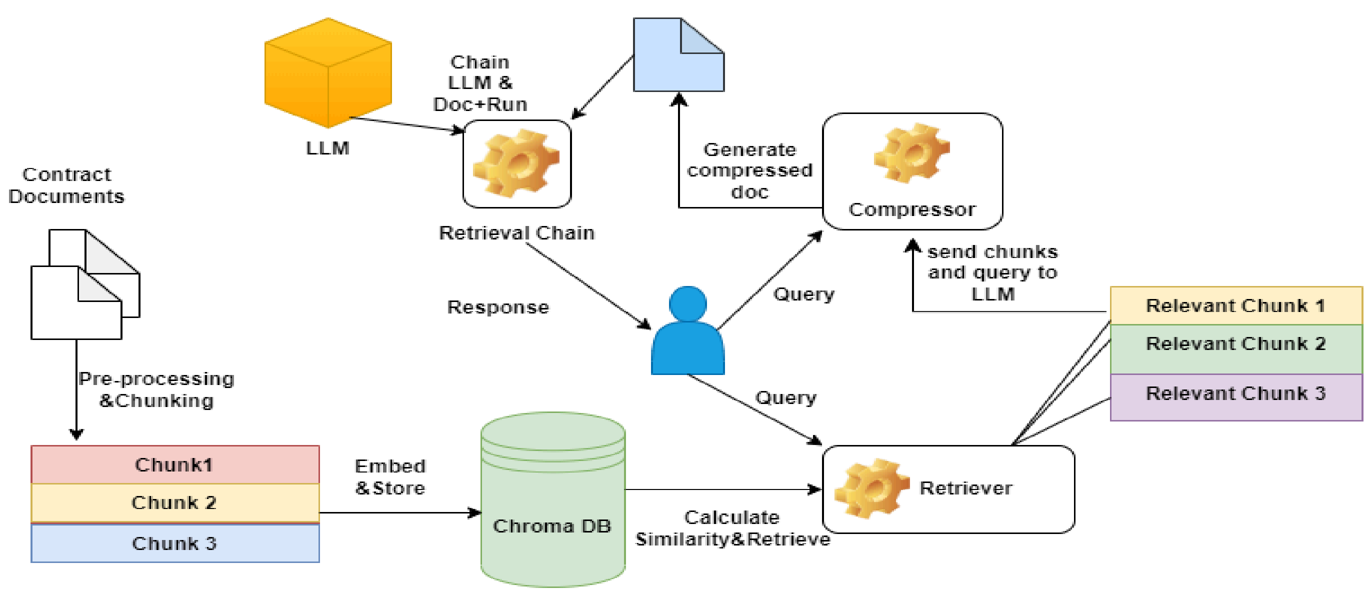
Intern 1: Vedant Joshi
Intern 2: Shivani Gangarapollu

Project Guides:

Prasanth Gudiwada , Dipti Pawar, Sakshi Dhokrat, Barbara Marques, Liam Jackson, Samuel Davila

1. Project Overview

The objective of this project is to develop and deploy a Retrieval-Augmented Generation (RAG) based Contract Query System that utilizes document embeddings for accurate answers to contract-related questions. The system will be built using Python, LangChain, and ChromaDB



2. Project Phases - Learning and Development

The project is divided into four key phases, with collaborative efforts between Intern 1 and Intern 2.

Phase 1: Learning and Requirement Gathering

Duration: 16th August 2024 – 22nd August 2024

Objective:

Familiarize the key concepts and technologies involved in the project. Gather detailed requirements for the system and create a comprehensive project plan.

Activities:

• Technology Familiarization:
  - Jointly explore LangChain, ChromaDB, PyMuPDF, PyPDF2 and Streamlit.
  - Understand the principles of Retrieval-Augmented Generation (RAG).
• Requirement Gathering:
  - Collaboratively engage with stakeholders to understand specific requirements related to the contract query system.
  - Document the requirements and draft the initial project specifications.
• Initial Setup:
  - work together to set up the development environment and tools.
  - Access and review necessary documentation, datasets, and resources.

Deliverables:

• Documented requirements and project specifications.
• Fully set up and configured development environment with all necessary tools installed.

2. Phase 2: Initial Implementation and Learning

Duration: 23rd August 2024 – 8th September 2024

Objective:

Begin the initial development of the RAG-based system, focusing on collaborative learning and application of the technologies. Implement the basic memory builder and RAG chatbot components together.

Activities:

• Memory Builder Development:
  - Extract, chunk, and preprocess text from provided PDF contracts.
  - Collaboratively create embeddings using OpenAI's embedding models.
  - Store the embeddings in ChromaDB for retrieval.
• RAG Chatbot Development:
  - Implement a simple question-answering system using LangChain.
  - Test the retrieval of relevant chunks based on user queries.

Implementation:

• Our first task was to read the data from 50 pdf files accurately for which we used PyPDF2 and directly passed it for next steps . This led us to a problem where we were only able to fetch information of 11 pages from each of 50 files .

• To fix the problem , we came up with an approach of creating a function to load each file in directory individually and passed it to another function which is responsible for extracting text from file page-wise and appending the text to a string variable and after the end of all pages , writing it to a file and storing in a separate directory .

• When we observed discrepencies /errors in the original pdf text , we realized that we need to address this issue as data quality plays an important role in terms of fetching and generating a good response . Certain words like PROGRAM and ARTICLE were not correctly spelled in a few pdf files .

• Since we are asked to develop a generic approach , we thought of implementing spell-check functionality. We used Textblob library to do this and realized that it has given some problems like :-

Chewy Doc

Sheamus Toal /s/ John J. Coane - Original

Sheamus Coal /s/ John J. One-After spell-check was applied

- Instances where names were involved was completely mishandled by spell checker

Costco

Law - Original

Saw-After spell-check was applied

- Replaced certain instances without any relation with context

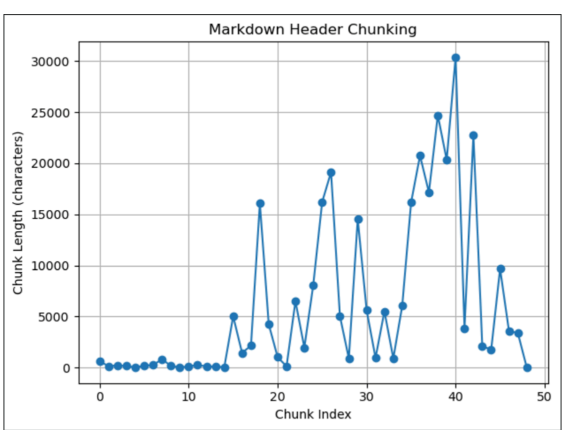
We decided to scrap this idea and move ahead with manually correcting a few words .

• The next step was to decide the Chunking Strategy for our use case . We experimented with different strategies like RecursiveCharacterTextSplitter, Markdown Header Chunking , Semantic chunking using semantic_router.splitters module .

Chunking Strategies

1. Markdown Header Chunking

Markdown Header Chunking splits text data based on predefined headers in the Markdown format. This technique identifies specific headers such as 'SECTION', 'Section', 'Articles', and 'ARTICLE' and uses these as points to split the text into chunks.



Pros

1. Logical Separation: Chunks are created based on meaningful sections of the text, preserving the semantic integrity of the content.

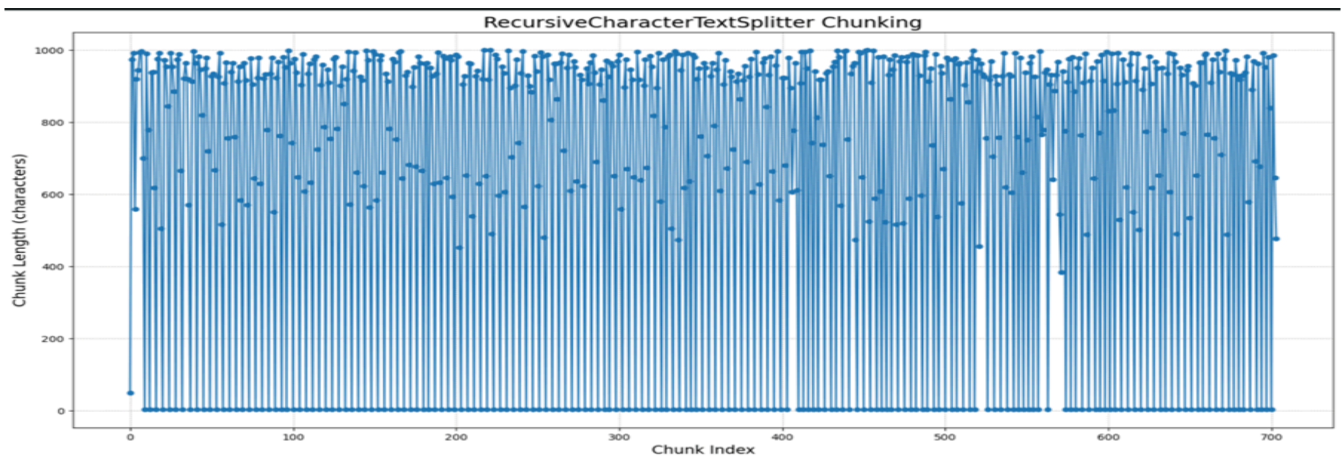
2. Customizable: You can define headers to split on, allowing flexibility based on the document's structure.

Cons

1. Dependency on Headers: If the headers are inconsistent or not well-defined in the text, this method might miss potential chunking points.
2. Variable Chunk Sizes: Chunks may vary significantly in size, which could affect downstream processing tasks like text analysis or model input.

2. Recursive Character Chunking

Recursive Character Chunking splits the text recursively based on character length, with specified chunk size and overlap. This technique aims to maintain chunk sizes within a certain range while allowing for some overlap between chunks.



Pros

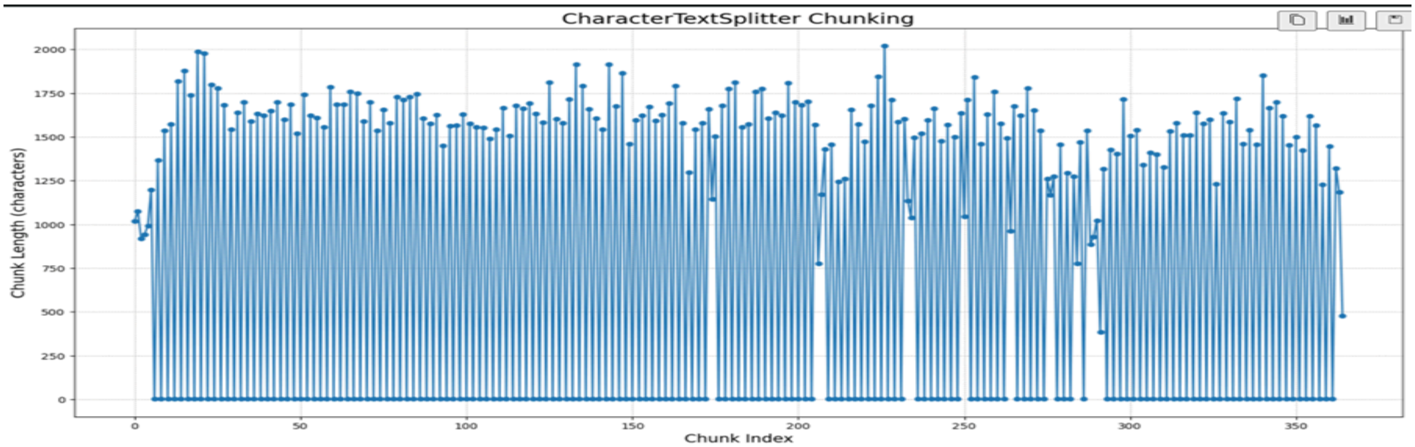
1. Controlled Chunk Size: Provides better control over the length of each chunk, making it more suitable for tasks requiring uniform input sizes.
2. Overlap Handling: Overlap ensures that no information is lost between chunks, which can be beneficial for tasks like text generation.

Cons

1. Complexity: Recursion adds complexity to the chunking process, which might increase computational overhead.
2. Less Semantically Aligned: Chunks may split content mid-sentence or paragraph, leading to less coherent chunks.

3. Character Chunking

Character Chunking is similar to Recursive Character Chunking but without recursion. It splits text into chunks based on a fixed character size with a defined overlap between chunks.



Pros

1. Simplicity: Straightforward and easy to implement with predictable results.
2. Uniformity: Produces chunks of relatively uniform size, which can be beneficial for tasks like batching in machine learning models.

Cons

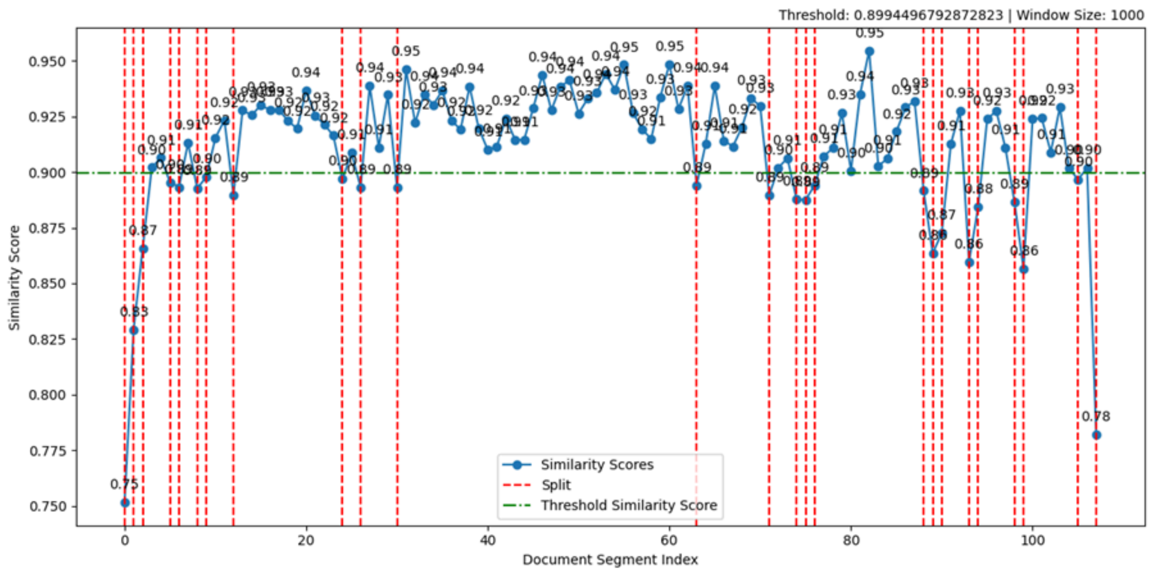
1. Potential for Information Loss: Overlaps might not be sufficient to capture the full context, leading to information loss between chunks.
2. Lack of Contextual Awareness: Splits text purely based on character count, without regard to the logical or semantic structure of the content.

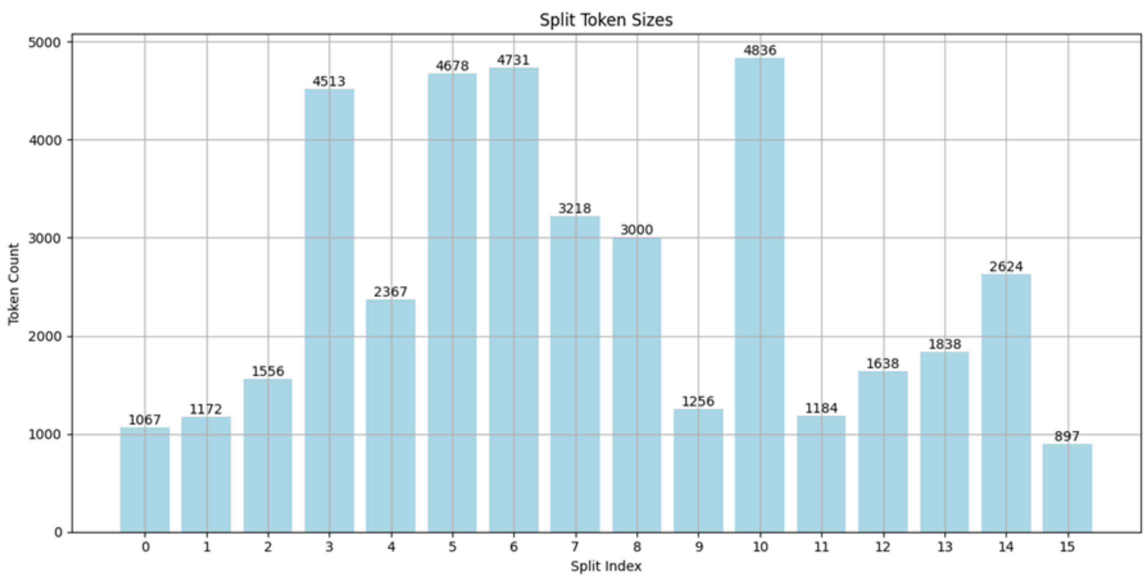
Semantic Chunking:

The semantic_router.splitters module is part of a toolkit designed for advanced text processing, particularly in the context of natural language understanding and information retrieval tasks. The splitters in this module are designed to intelligently segment text into chunks that preserve semantic meaning. This is particularly useful in ensuring that the split chunks do not lose context, which is crucial when feeding text into models for tasks like summarization, question answering, or retrieval-augmented generation (RAG).

- **Rolling window Splitter :** This is a specific splitter provided in the module that uses a "rolling window" approach to ensure overlapping content between chunks. This overlap helps maintain context across chunks, which is vital for tasks where understanding across chunk boundaries is necessary.

```
splitter = RollingWindowSplitter(
    encoder=encoder,
    dynamic_threshold=True,
    min_split_tokens=1000,
    max_split_tokens=5000,
    window_size=1000, plot_splits=True,enable_statistics=True)
```





General Guidelines for Chunk Size:

- **Model Token Limitations**:
  - **OpenAI's GPT Models**: These models have a token limit ranging from 4,000 to 8,000 tokens per input, depending on the model version. For efficient processing and context retention, it's recommended to keep chunks well below this limit, ideally around 1,000 to 2,000 tokens per chunk.
- **Balancing Context and Performance**:
  - **Context Preservation**: Larger chunks (e.g., 1,500–2,000 tokens) preserve more context, which can be beneficial for tasks where understanding larger sections of text is important (e.g., summarization or QA tasks).

- **Performance**: Smaller chunks (e.g., 500–1,000 tokens) are easier to handle for retrieval-based models and allow for more granular control over the content but might lose some context.

•The decision whether chunking has to be done page-wise /file-wise depends on data queried . If page-wise chunking is performed , only the chunks within the page will have context continuity but in our case since the whole document holds information about a Company's contract , it is essential that chunks have continuity throughout the file - Hence ,file-wise chunking.
• Created an Excel sheet consisting of PDF file names , associated questions , actual answers and the answers generated by using all these chunking strategies.

Conclusion

We concluded to go ahead with RecursiveCharacterSplitter as the chunking strategy and since we are chunking and overlapping chunks at file-level *chunk_size*=1200, *chunk_overlap*=500 worked well after testing the responses.

Deliverables:

• Decided and implemented appropriate chunking strategy
• A basic RAG Chatbot capable of answering contract-related queries for testing.

3. Phase 3: Creating/Loading Vectorstore and Implementation of Retrieval Chain

Duration: 9th September 2024 – 15th September 2024

Objective:

Store the chunks into ChromaDB Vector Database by embedding them using OpenAI Embeddings and using Retrieval Chain for retrieving the query-relevant chunks (top 5) and passing it as Context for the LLM to generate an answer.

Activities:

- Check whether vectorstore already exists in the working directory and based on that create/load existing vectorstore.
- Enhance the conversational retrieval system for better contextual understanding and response generation.

Implementation:

• Used langchain.vectorstores - Chroma to store the chunks and create a vectorstore. For embedding function we used OpenAI Embeddings . We created a condition to check whether vectorstore already exists in the working directory and based on that create/load existing vectorstore inside the save_to_chroma() function.
• Used vectorstore.as_retriever(*search_type*="similarity", *search_k*=5) to retrieve the top 5 chunks and using the langchain.prompts import PromptTemplate to design a prompt for achieving accurate and structured response .
• Used from langchain.chains.llm import LLMChain and from langchain.chains.combine_documents.stuff import StuffDocumentsChain to initialize the LLM and context retrieved by retriever(top 5 relevant chunks) .
• Used from langchain.chains import RetrievalQA to chain LLM and retrieved chunks inorder to generate the final response .
• Integrating the whole flow with Streamlit front-end.

Deliverables:

• A fully functional RAG based Contract Bot.

4. Phase 4:Fine-tuning , Documentation and Final Testing

Duration: 16th September 2024 – 20th September 2024

Objective:

Fine-tuning the query bot by tweaking the chunk sizes and retriever parameters .Finalize all documentation and prepare for the project handover. Conduct rigorous testing of the system and prepare for the final presentation.

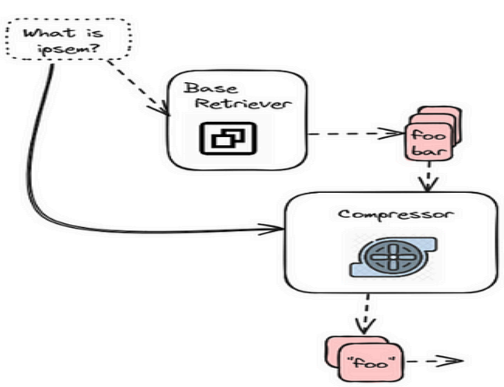
Activities:

• Final Documentation:
  - Document the entire development process, including code documentation and user manuals.
  - Prepare deployment guides and maintenance documentation collaboratively.

• Fine-tuning ,Testing and QA:
  - Conduct comprehensive testing together, including edge cases and performance testing.
  - Implement any necessary tunings, bug fixes or optimizations.

• Presentation Preparation:
  - Work together to prepare a detailed presentation covering project objectives, development process, and final outcomes.
  - Create demos and visualizations to showcase the functionality of the RAG system.

Implementation:



• Created a Contextual Compression Retriever and LLMExtractor Chain to get better responses. This method is known as Contextual Compression.

• Contextual Compression refers to the process of extracting information from your retrieved documents(chunks) that are relevant . It's all about increasing the signal-to-noise ratio.

• Contextual compression works by iterating over the retrieved documents, then passing them through a LLM which will extract information according to context specified in the query.

• Tweaked retriever parameters - search methodology as "mmr" instead of "similarity" and relevance_score_threshold .MMR-Maximal Marginal relevance is used in cases where diversity is also important . It tries to avoid redundancy by ensuring the selected results are not too similar to each other, promoting diverse information in the output along with picking relevant chunks . “mmr-threshold “ parameter is the ratio of relevance to diversity which ranges from 0-1.

- A **high MMR threshold** (λ\lambda closer to 1) prioritizes relevance, which is useful when the user wants the most relevant documents without much concern for redundancy.
- A **low MMR threshold** (λ\lambda closer to 0) prioritizes diversity, which can be helpful when the user needs to cover a broad range of information without much overlap.
- **λ between 0 and 1**: MMR balances relevance and diversity, retrieving results that are both relevant and diverse. The closer λ\lambda is to 1, the more importance is given to relevance.

• We also observed that when streamlit session was running and questions were prompted to the bot , each time vectorstore was getting loaded and generating inconsistent replies . To fix the issue we initialized a condition using st.session_state checking if ‘qa_chain’ exists. If it has already loaded once for the session ,then it continues with initialized vectorstore.

Conclusion

We found that responses generated after the implementation of Context Compression were more accurate and well-framed than before .Even Though it requires an extra layer of calling LLM , the method improved the bot’s performance. We decided to go ahead with” similarity” instead of “mmr” as computationally it is a little complex and we already decided to move with Context Compression which would use LLM for understanding and generating better answers from chunks retrieved.

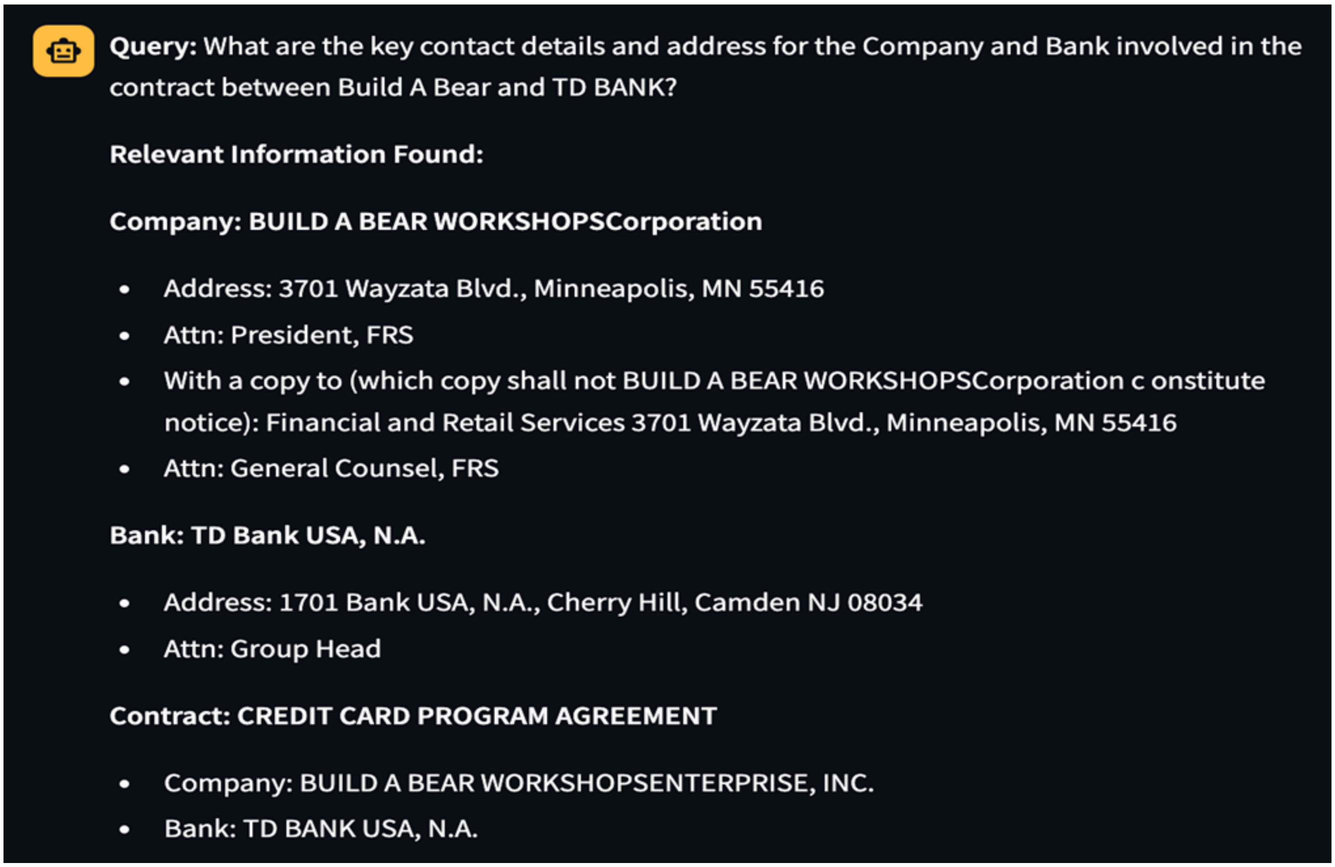
Deliverables:

• Complete project documentation and user manuals.
• A comprehensive test report.
• Final presentation and demo materials.

Testing:

1. What are the key contact details and address for the Company and Bank involved in the contract between Build A Bear and TD BANK?

   Answer:



2. What happens in the event of a default by the Company or the Bank in contract between Dicks Sporting Goods and Bank?

Answer:

In the event of a default by either the Company or the Bank in the contract between Dicks Sporting Goods and Bank, the following actions may occur:

- If the default is by the Company, the Bank may terminate the contract upon written notice if DICKS SPORTING GOODSCorporation does not have a senior unsecured debt rating of at least A or its equivalent from at least two of Moody's, Standard & Poor's and Fitch Ratings.
- If the default is by the Bank, the Company may terminate the contract upon notice upon the occurrence of a Company Event of Default.
- The occurrence of any one or more of the following events (regardless of the reason therefore) shall constitute an event of default by Bank:
  - Bank fails to settle in accordance with Section 7.2, any amount that is not disputed in good faith, within two (2) Business Days after Company shall have given notice thereof by 5 p.m. Eastern.
  - Bank shall no longer be solvent or shall fail generally to pay its debts as they become due or there shall be a substantial cessation of Bank's regular course of business.
- If the Company fails to perform, satisfy or comply with any obligation, condition, covenant or other provision contained in this Agreement for a period of not less than sixty (60) consecutive days due to a Force Majeure Event and such failure shall either have a material and adverse effect on the Program, the Bank Licensed Marks or the Company Licensed Marks, or materially diminish the economic value of the Program to Bank, the Bank may terminate the contract upon notice.
- If the Company or the Bank terminates the contract, the following rights and obligations apply:
  - If the Company terminates the contract, the Company shall have the rights and obligations of a defaulting party under Section 13.4(b).
  - If the Bank terminates the contract, the Company shall have the rights and obligations of a non-defaulting party under Section 13.4(b).
- If the Company or the Bank terminates the contract prior to the Closing Date, the following actions may occur:
  - Company may terminate this Agreement upon notice to Bank on or prior to the Closing Date if a Bank Event of Default shall have occurred.

3. Explain what happens if the Bank defaults in its obligations in contract between Cato Fashions and Bank?

Relevant Information Found:

If the Bank defaults in its obligations in the contract between Cato Fashions and Bank, it will constitute an event of default by the Bank. This means that the Bank has failed to fulfill its obligations as outlined in the contract and has breached the terms of the agreement. The contract specifically states that the occurrence of any one or more of the following events (regardless of the reason therefore) shall constitute an event of default by Bank:

- Bank fails to settle in accordance with Section 7.2, any amount that is not disputed in good faith, within two (2) Business Days after Company shall have given notice thereof by 5 p.m. Eastern.
- Bank shall no longer be solvent or shall fail generally to pay its debts as they become due or there shall be a substantial cessation of Bank’s regular course of business.
- The OCC or any other regulatory authority having jurisdiction over Bank shall order the appointment of a custodian, receiver, liquidator, assignee, trustee or sequestrator (or similar official) of Bank or of any substantial part of its properties, or order the winding-up or liquidation of the affairs of Bank, and such order shall not be vacated, discharged, stayed or bonded within sixty (60) days from the date of entry thereof.

In summary, if the Bank defaults in its obligations, it will be considered a breach of contract and may result in legal action being taken against the Bank by Cato Fashions. The contract also outlines specific actions that may be taken by regulatory authorities in the event of a default by the Bank.