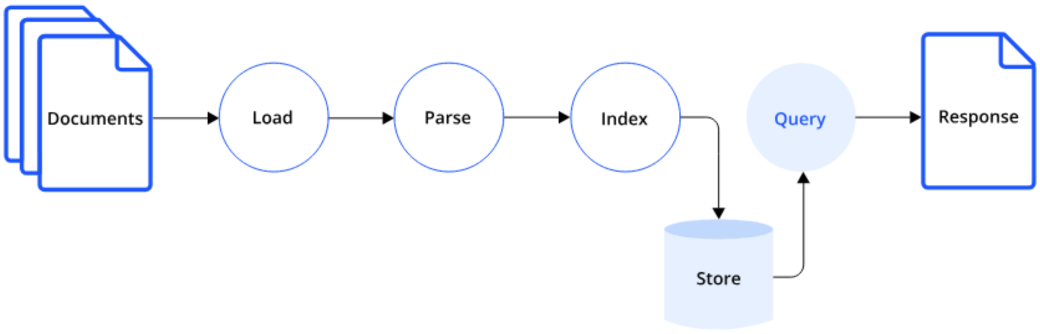
Documentation of Semantic_Spotter

Contributor: Ayushi Pitchika, ACP Gen AI C10, UpGrad

Objective

The goal of the project will be to build a robust generative search system capable of effectively and accurately answering questions from various policy documents. This chatbot must effectively understand user queries, retrieve relevant product information, and generate comprehensive explanations. The solution involves preprocessing the dataset, creating a vector-based index for efficient querying, and implementing Retrieval-Augmented Generation (RAG) using LlamaIndex to generate detailed responses. The chatbot should also manage conversational context and memory across interactions.

Choice of LlamaIndex:
LlamaIndex is an ideal framework for this project because it provides robust tools for creating vectorbased search indices and implementing Retrieval-Augmented Generation. It supports efficient querying and retrieval of documents, which is crucial for generating detailed and contextually relevant responses by combining retrieval with generated content.

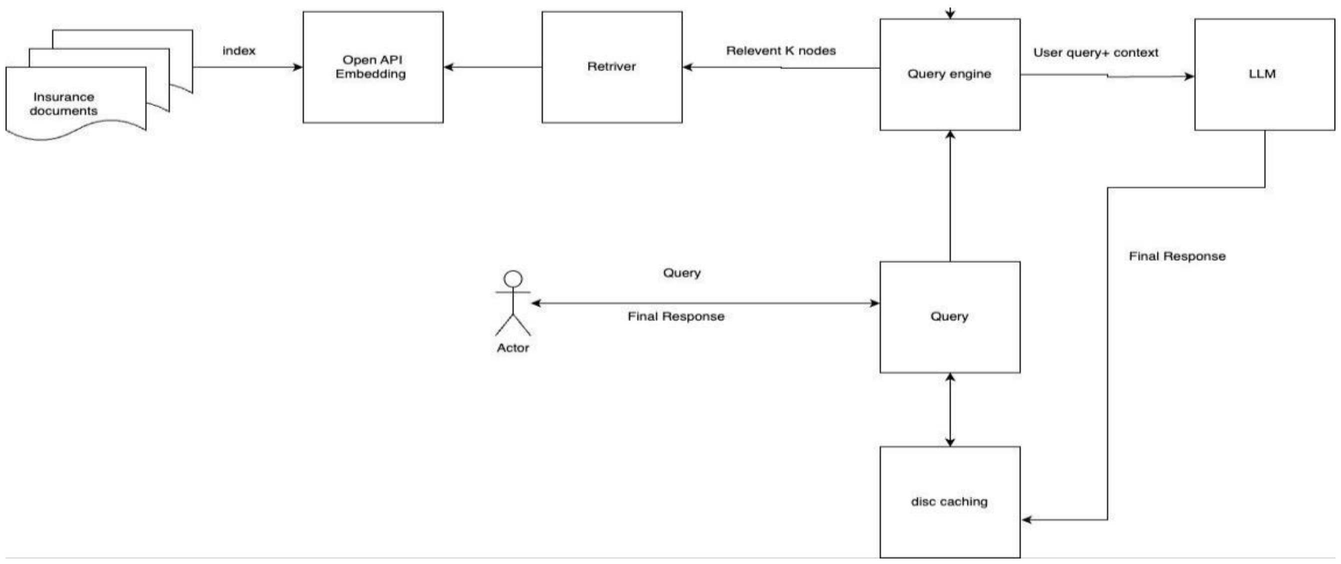


System Design

Dataset: A single long life insurance policy document in .pdf format named “**Group Member Life Insurance Policy**”.

Project Goals

1. Build Vector-Based Index: Create a vector store index using LlamaIndex for efficient querying and retrieval of product information.
2. Implement RAG: Build Retrieval-Augmented Generation using LlamaIndex to provide detailed, context-aware responses by combining retrieval with generated content.
3. Develop Conversational Interface: Build a chatbot interface that manages conversational history and provides relevant responses based on user input.
4. Feedback Mechanism: Implement a feedback system to evaluate the quality of responses and improve the chatbot’s performance over time.



Design Choices

1. Framework Choice: LlamaIndex is chosen for its capability to handle document parsing, indexing, and querying efficiently. It integrates well with various vector storage solutions and provides a robust querying engine.
2. Document Parsing: The PDF is parsed into structured data using LlamaIndex’s `SimpleNodeParser`. This choice allows us to handle complex documents with multiple sections and formats.
3. Vector Storage: For simplicity and to maintain local control, the vector storage is handled directly within the LlamaIndex framework without external databases.
4. Query Processing: The system uses LlamaIndex’s querying capabilities to search through the indexed document and retrieve relevant information based on user queries.
5. Integration with LLM: While not required in the current implementation, integration with a Language Model like OpenAI's GPT could be considered for enhanced interaction and natural language understanding.

Implementation

1. Loading and Creating Data/Documents:
   1.1. In this phase, the pre-processed data is loaded and organized into a format that can be indexed.
   1.2. Loading the dataset, creating documents for indexing.

2. Indexing the Documents Using LlamaIndex:
   2.1. This step involves creating an index of the documents, which allows for efficient querying.
   2.2. Parsing the documents into nodes and creating an index.

3. Build Query Engine:
   3.1. Here, a query engine is constructed to handle and process user queries.
   3.2. Setting up the query engine based on the indexed data.

4. Initialize Conversation:
   4.1. This step involves setting up the initial state for the chatbot conversation.
   4.2. Initializing variables to store conversation history and other relevant data.

5. Query Response:
   5.1. The query engine processes user inputs and generates responses based on the indexed data, this data is then passed in API calls and the relevant response is generated.

   5.2. Handling user queries, generating detailed responses, and updating conversation history.

6. Feedback:
   6.1. Collecting feedback from users about the responses provided by the chatbot to improve future interactions.
   6.2. Asking users for feedback and storing their responses.

7. End Conversation:
   7.1. This step concludes the interaction with the user.
   7.2. Providing final outputs, saving conversation logs, and exiting the interaction.

Challenges

1. Chunking Strategy: Loading the entire dataset without chunking posed a challenge due to memory constraints. However, chunking was not implemented due to time constraints and the manageable size of the dataset.
2. Context Management: Ensuring that the chatbot maintains conversational context over multiple interactions.
3. RAG by LlamaIndex: Effectively crafting retrieval and generation in LlamaIndex to produce detailed and relevant responses.
4. User Feedback: Collecting and analyzing feedback to continuously improve the chatbot’s performance.

Lessons Learned

1. Importance of Data Quality: High-quality data is crucial for the accuracy of the system.
2. Modularity of Llama Index: The modular nature of Llama Index significantly simplifies the development process.
3. Continuous Improvement: Regular updates and fine-tuning of the model are necessary to maintain high performance.
4. User Feedback: Incorporating user feedback helps in refining the system and improving user

Further Improvement

1. The model’s performance can be further improved by using a cleaner dataset or undertaking some additional pre-processing techniques.
2. Custom nodes and LLMs will give better results.
3. Caching can be implemented for frequent queries for faster retrieval of responses.