Case Study: RAG System for HDFC Insurance Policy Documents

Introduction

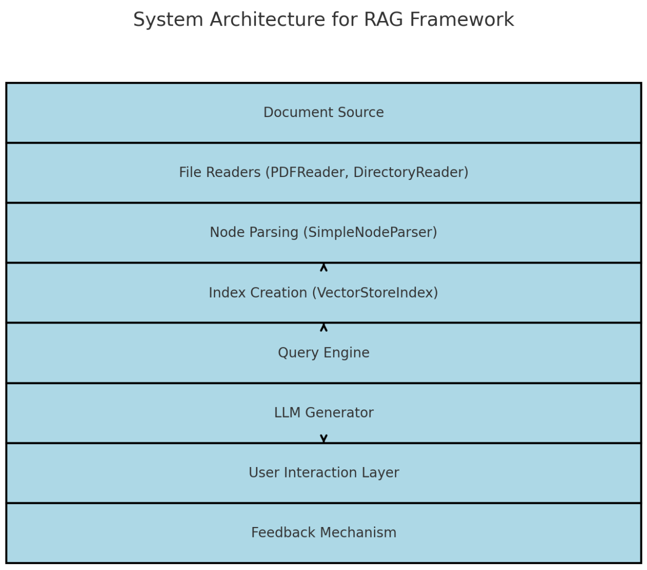
In this case study, we developed a Retrieval-Augmented Generation (RAG) framework to extract and respond to queries regarding seven types of HDFC insurance policy documents. This system leverages LlamaIndex for efficient document processing and semantic search. The RAG system comprises two main components: an information retriever and a Large Language Model (LLM) generator. This document outlines the system's architecture, implementation, improvements, challenges, and lessons learned.

Objectives

1. To design a system capable of retrieving relevant information from HDFC insurance policy documents.
2. To enhance the accuracy and relevance of responses using a combination of semantic retrieval and LLM generation.
3. To create a user-friendly interface for querying policy details and obtaining reliable responses.

System Architecture Diagram

Below is the flow of the system architecture for the RAG framework:


System Architecture for RAG Framework
Document Source
File Readers (PDFReader, DirectoryReader)
Node Parsing (SimpleNodeParser)
Index Creation (VectorStoreIndex)
Query Engine
LLM Generator
User Interaction Layer
Feedback Mechanism

The RAG framework was designed as follows:

1. **Document Processing and Indexing:**
    - **LlamaIndex:** Chosen for its ability to read files of various formats and create a robust vector database.
    - **Readers:**
        - `SimpleDirectoryReader`: Utilized for reading documents stored in a directory.
        - `PDFReader`: Employed for parsing insurance policy PDF documents.
    - **Parsing:**
        - `SimpleNodeParser`: Used to parse documents and generate nodes.
        - Nodes created from parsed content were indexed using `VectorStoreIndex` from LlamaIndex.
2. **Query Engine:**
    - A query engine was constructed using the indexed database. It retrieves semantically similar documents based on user queries.
    - The retrieved documents, along with the user query, were fed to an LLM generator to produce detailed and accurate responses.
3. **User Interaction:**
    - Defined `query_response` and `initialize_conv` functions to allow users to interact with the system in a chat-like format.
    - Integrated a feedback mechanism where users could rate each response provided by the model.
    - Constructed a pandas DataFrame to store:
        - **Question**: User's query.
        - **Response**: Model's answer.
        - **Page**: Page number of the retrieved document.
        - **Review**: User's feedback on the response.

System Architecture Diagram

Below is the flow of the system architecture for the RAG framework:

1. **Document Source**: Input policy documents in PDF or other formats.
2. **File Readers**: PDFReader and DirectoryReader parse and prepare the documents.
3. **Node Parsing**: SimpleNodeParser creates structured nodes from parsed content.
4. **Index Creation**: VectorStoreIndex organizes and stores the nodes for semantic retrieval.
5. **Query Engine**: Retrieves relevant documents based on user queries.
6. **LLM Generator**: Generates detailed responses based on retrieved documents and user queries.
7. **User Interaction Layer**: Provides a chat-like interface for users to interact with the system.

8. **Feedback Mechanism**: Collects user ratings and reviews for continuous improvement.

Improvements

To enhance the system's performance, the following modifications were implemented:

1. **Custom Prompting:**
   - Integrated OpenAI's GPT-3.5 Turbo and Microsoft's Phi-3 Mini 4k LLM (via HuggingFace) for response generation.
   - Tailored prompts were crafted to improve the clarity and relevance of the responses.
2. **Embedding Models:**
   - Experimented with advanced embedding models:
     - `all-mpnet-base-v2`
     - `bge-small-en-v1.5`
   - Improved the semantic search accuracy and retrieval quality.
3. **Sub-Question Query Structure:**
   - Developed a sub-questioning mechanism to break down complex queries into smaller, manageable parts.
   - This led to faster query processing and smoother interaction.

Challenges

1. **Document Complexity:**
   - Insurance policy documents are often lengthy and contain legal jargon, making it challenging to parse and retrieve information effectively.
2. **Semantic Retrieval:**
   - Ensuring high accuracy in retrieving the most relevant sections of the documents was difficult, especially for nuanced queries.
3. **LLM Integration:**
   - Fine-tuning prompts and ensuring consistency in LLM-generated responses required iterative testing and optimization.
4. **User Feedback Integration:**
   - Designing a robust feedback mechanism and leveraging user reviews to iteratively improve the system.

Lessons Learned

1. **Importance of Modular Design:**
    - Breaking down the system into modular components (retriever, parser, indexer, query engine) made it easier to troubleshoot and implement improvements.
2. **Embedding Model Selection Matters:**
    - The choice of embedding model significantly impacts retrieval performance.
3. **User-Centric Design:**
    - Incorporating user feedback directly into the system workflow improved the accuracy and relevance of the responses over time.
4. **Iterative Optimization:**
    - Continuous testing and fine-tuning were crucial to overcoming challenges related to document complexity and response

Conclusion

The RAG system for HDFC insurance policy documents demonstrates the potential of combining semantic retrieval with advanced LLMs to address complex information retrieval tasks. By leveraging LlamaIndex and experimenting with various LLMs and embedding models, we developed a robust pipeline capable of delivering accurate and user-friendly responses.