Semantic Spotter Project - Building a RAG System for Insurance Domain

Project Goal

The aim of this project is to develop a **robust generative search system** capable of accurately and efficiently answering questions derived from a collection of **insurance policy documents**. The system will leverage **LlamaIndex** to construct the generative search application.

Data Source

The system is powered by seven **HDFC insurance policy documents** stored in PDF format, provided within a single folder:

1. **HDFC-Life-Easy-Health-101N110V03-Policy-Bond-Single-Pay.pdf**
2. **HDFC-Life-Group-Poorna-Suraksha-101N137V02-Policy-Document.pdf**
3. **HDFC-Life-Group-Term-Life-Policy.pdf**
4. **HDFC-Life-Sampoorna-Jeevan-101N158V04-Policy-Document.pdf**
5. **HDFC-Life-Sanchay-Plus-Life-Long-Income-Option-101N134V19-Policy-Document.pdf**
6. **HDFC-Life-Smart-Pension-Plan-Policy-Document-Online.pdf**
7. **HDFC-Surgicare-Plan-101N043V01.pdf**

System Architecture

Core Components

1. **Documents**
   - The seven policy documents are stored in a folder and ingested using `SimpleDirectoryReader`.
2. **Embedding & Indexing**
   - **OpenAI Embeddings** are used to convert the content of the documents into vector representations.
   - A **VectorStoreIndex** is created for efficient semantic search and retrieval.
3. **Query Engine**

    - Built using the **LlamaIndex QueryEngine** module.
    - Performs **semantic search** by using a **Retriever** and the **SentenceTransformerRerank** model (`cross-encoder/ms-marco-MiniLM-L-2-v2`) to retrieve and rerank the top-k relevant document chunks.

4. **Large Language Model (LLM)**
    - The top-k documents, along with the user's query, are passed to the **GPT-4** model to generate accurate responses.
5. **Caching**
    - **DiskCache** is used to improve performance by storing recent queries and their results.
    - When a similar query is received, it is served from the cache. If not found, the query is forwarded to the Query Engine and LLM.
6. **Metadata**
    - Along with the response, the system provides metadata:
        - **Document references**
        - **Semantic scores** for the retrieved documents
    - This improves user trust and transparency.

Tools and Technologies

- **LlamaIndex**: Provides the framework for document indexing, query execution, and response generation.
- **OpenAI Embeddings**: For converting document text into embeddings.
- **GPT-4**: For generating human-like, contextually relevant responses.
- **SentenceTransformerRerank**: Used to rerank retrieved documents using a semantic model (`cross-encoder/ms-marco-MiniLM-L-2-v2`).
- **DiskCache**: Lightweight caching solution to store frequently queried responses.
- **PDFplumber**: Used to extract and read data from PDF documents.

Why Choose LlamaIndex?

LlamaIndex is designed to support the development of **RAG-based applications**, providing:

- **Flexible Data Ingestion**: Supports a variety of formats (PDF, DOCX, databases, etc.).
- **Seamless Integration with LLMs**: Easily connects large language models to diverse data sources.
- **Query Interface**: Offers efficient query handling, re-ranking, and response generation.

Key Features:

- Data connectors for various formats
- Ability to synthesize data from multiple sources
- Integrations with vector stores, LangChain, and more

Evaluation Strategy

The system is evaluated using **GPT-4** based on the following metrics:

- **Relevancy**: How relevant the returned documents are to the user's query.
- **Faithfulness**: How accurately the generated response reflects the content of the source documents.
- **Correctness**: How factually accurate the response is.

We leverage the following evaluators from **LlamaIndex**:

```
from llama_index.core.evaluation import (

    CorrectnessEvaluator,

    FaithfulnessEvaluator,

    RelevancyEvaluator,

)
```

Challenges Faced

- **Compatibility Issues**: Encountered difficulties while importing **RAGAS** for evaluation and working with **gptcache**.
- **Performance Bottlenecks**: The system faced slow performance for larger document sets.
- **Dependency Conflicts**: Issues arose while resolving conflicting library dependencies.

Solutions Implemented

- **DiskCache** was used instead of **gptcache** to avoid performance issues.
- Imported evaluation modules directly from **LlamaIndex** instead of using **RAGAS**.
- For reranking, an **alternative** could be using **Cohere Rerank**.

Example Queries

The system answers queries from the insurance policy documents with high accuracy, such as:

- “What does the **Easy Health Plan** cover?”
- “How do I file a claim under the **Sanchay Plus Life** policy?”
- “What are the benefits of the **Smart Pension Plan**?”

Each response includes **source references** and **semantic similarity scores**, boosting transparency and user trust.

Conclusion

This project demonstrates the effectiveness of **RAG (Retrieval-Augmented Generation)** systems in handling complex queries from domain-specific documents, like insurance policies. By utilizing **LlamaIndex**, **OpenAI embeddings**, and **GPT-4**, we provide an efficient and accurate solution for users seeking information from a large corpus of text.