LangChain Hybrid RAG Pipeline: Step-by-Step Guide

Prerequisites

Before you begin, ensure you have the necessary packages installed:

- ✓ `langchain`
- ✓ `langchain-community`
- ✓ `langchain-google-genai`
- ✓ `faiss-cpu`

---

1. Load and Split the PDF Document

Purpose: Extract text from a PDF and divide it into manageable chunks for processing.

Process:

- **Load the PDF:** Utilize `PyPDFLoader` from `langchain_community.document_loaders` to read the PDF file.
- **Split the Text:** Use `RecursiveCharacterTextSplitter` to divide the text into chunks of 1000 characters with an overlap of 200 characters. This ensures that each chunk has sufficient context, which is beneficial for downstream processing.

---

2. Initialize Embeddings

Purpose: Convert text chunks into vector representations using Google's Generative AI Embeddings.

Process:

- **Set API Key:** Provide your Google API Key to authenticate requests.
- **Initialize Embeddings:** Use `GoogleGenerativeAIEmbeddings` with the model `"models/embedding-001"` to generate embeddings for each text chunk. These embeddings capture the semantic meaning of the text, facilitating effective similarity searches.

---

3. Create and Save FAISS Vector Store

Purpose: Store vector embeddings in a FAISS index for efficient similarity search.

Process:

- **Extract Text Content:** Retrieve the textual content from each split document.
- **Create FAISS Vector Store:** Use `FAISS.from_texts` to create a vector store from the text chunks and their corresponding embeddings.
- **Save the Vector Store:** Persist the vector store locally using `save_local`, allowing for reuse without recomputing embeddings.

---

4. Load the FAISS Vector Store

Purpose: Reload the saved FAISS vector store for retrieval operations.

Process:

- **Load Vector Store:** Use `FAISS.load_local` to load the previously saved vector store, enabling retrieval of relevant documents based on semantic similarity.

---

5. Set Up Dense Retriever

Purpose: Enable semantic search by converting the FAISS vector store into a retriever.

Process:

- **Convert to Retriever:** Use the `as_retriever` method on the FAISS vector store to create a retriever that can fetch documents based on their semantic similarity to a query.

---

6. Set Up BM25 Retriever (Sparse)

Purpose: Enable keyword-based search using the BM25 algorithm.

Process:

- **Create BM25 Retriever:** Use `BM25Retriever.from_documents` with the split documents to create a retriever that scores documents based on term frequency and inverse document frequency.
- **Set Retrieval Parameters:** Configure the retriever to return the top 5 documents for each query by setting `k = 5`.

---

7. Combine Dense and Sparse Retrievers into a Hybrid Retriever

Purpose: Leverage both semantic and keyword-based search by combining retrievers.

Process:

- **Initialize Ensemble Retriever:** Use `EnsembleRetriever` to combine the dense and sparse retrievers.
- **Assign Weights:** Provide weights `[0.3, 0.7]` to the sparse and dense retrievers, respectively, indicating the relative importance of each in the final retrieval results.
- **Retrieval Mechanism:** The `EnsembleRetriever` uses Reciprocal Rank Fusion (RRF) to merge and rerank the results from both retrievers, enhancing the overall retrieval performance.

---

8. Initialize the Gemini Language Model

Purpose: Set up the Gemini language model for generating answers.

Process:

- **Initialize LLM:** Use `ChatGoogleGenerativeAI` with the model `"gemini-2.0-flash"` and your Google API Key to create a language model instance capable of generating responses based on provided context.

---

9. Create a Prompt Template for the QA System

Purpose: Define how the retrieved context and user question are presented to the language model.

Process:

- **Create Prompt Template:** Use `ChatPromptTemplate.from_template` to define a template that structures the input to the language model. The template includes placeholders for the context and the user's question, guiding the model to generate appropriate answers.

---

10. Create a Chain to Combine Retrieved Documents

Purpose: Combine the retrieved documents into a single input for the language model.

Process:

- **Create Combine Documents Chain:** Use `create_stuff_documents_chain` with the language model and prompt template to create a chain that concatenates the retrieved documents and formats them according to the prompt template. This prepares the input for the language model.

---

11. Create the Retrieval Chain Using the Hybrid Retriever and Combine Docs Chain

Purpose: Assemble the retrieval chain that handles document retrieval and answer generation.

Process:

- **Create Retrieval Chain:** Use `create_retrieval_chain` with the hybrid retriever and the combine documents chain to create a retrieval-augmented generation pipeline. This chain first retrieves relevant documents using the hybrid retriever and then generates an answer using the language model.

---

12. Ask a Question and Retrieve the Answer

Purpose: Use the retrieval chain to answer a user question based on the PDF content.

Process:

- **Define the Question:** Specify the user's question, e.g., "What are the assumptions of regression?"
- **Invoke the Retrieval Chain:** Use the `invoke` method on the retrieval chain with the input question to retrieve relevant documents and generate an answer.
- **Display the Answer:** Extract and display the answer from the response returned by the retrieval chain.

Notes

- ✓ **BM25 and TF-IDF Relationship:** BM25 is an advanced ranking function that builds upon the TF-IDF model. While TF-IDF calculates term importance based on term frequency and inverse document frequency, BM25 introduces additional factors like term frequency saturation and document length normalization. This makes BM25 more effective in handling variations in document lengths and term distributions, leading to improved retrieval performance.

- ✓ **Deprecation Notice:** The `RetrievalQA` class has been deprecated in favor of the `create_retrieval_chain` function. It's recommended to use `create_retrieval_chain` for building retrieval-based QA systems.
- ✓ **Hybrid Retrieval:** Combining dense and sparse retrievers using the `EnsembleRetriever` allows leveraging the strengths of both semantic understanding and keyword matching, leading to more accurate and relevant results.
- ✓ **Prompt Template:** The `ChatPromptTemplate` is used to structure the input for the language model, ensuring that the context and question are presented clearly.
- ✓ **Gemini Model:** The `ChatGoogleGenerativeAI` class initializes the Gemini language model, which is used to generate answers based on the retrieved context.
- ✓ **FAISS Vector Store:** FAISS is used for efficient similarity search over dense vector embeddings, enabling fast and accurate retrieval of relevant documents.