Retrieval-Augmented Generation (RAG) Model for QA Bot

Overview

This documentation outlines the process of building a Retrieval-Augmented Generation (RAG) Model for a Question Answering (QA) Bot. The bot is designed to answer questions by retrieving relevant information from documents (such as PDFs) and generating coherent responses. It uses Chroma for vector storage and retrieval of document embeddings, and Google's Gemini API for both embedding generation and language modeling. The system is built using Langchain libraries for seamless integration of various components.

Model Architecture

The RAG model is built using the following components:

1. **Data Loader and Embedding Creation:**
    - The documents (e.g., PDF files) are loaded using the Langchain document loaders, such as PyPDFLoader.
    - The document content is split into manageable chunks using the RecursiveCharacterTextSplitter from Langchain.
    - The document chunks are then converted into vector embeddings using Google's Generative AI Embeddings through the Gemini API.
2. **Vector Database (Chroma):**
    - These document embeddings are stored in Chroma, a vector database that allows efficient similarity-based retrieval.
    - A retriever object is created using Langchain's VectorStoreRetriever, which handles the process of fetching relevant document chunks based on user queries.
3. **Query Handling and Answer Generation:**
    - For a given user query, relevant document chunks are retrieved from Chroma based on the similarity of the query to the stored embeddings.
    - The context from the retrieved documents and the user query is passed to Google's Gemini Model for generating a final coherent response.
    - A structured prompt template ensures that the response is clear, concise, and informative.

Step-by-Step Implementation

1. Data Loading and Text Extraction from PDF

To begin, we use the Langchain PyPDFLoader to load and extract text from a PDF document. The text is then split into smaller chunks using a text splitter, making it easier to store and process in the embedding and retrieval stages.

2. Text Chunking and Embedding Creation

To handle large documents, we use the RecursiveCharacterTextSplitter from Langchain to break the document into chunks of a specified size (500 characters with a 20-character overlap). Each chunk is then embedded using Google Generative AI Embeddings via the Gemini API.

3. Setting Up the Vector Database (Chroma)

The generated embeddings from the document chunks are stored in Chroma, which acts as the vector database. VectorStoreRetriever is then used to handle the retrieval of the most relevant document chunks when a user query is received.

4. Creating a Chat Prompt Template

A prompt template is defined to structure the input provided to the generative model. This template includes the context (retrieved documents) and user query, and instructs the model to generate a clear, concise, and detailed response.

5. Querying the Document and Retrieving Relevant Chunks

The user's query is processed by first retrieving relevant document chunks from Chroma using the retriever. These chunks are then used as context for the generative model to produce an answer.

Testing the QA Bot

To test the system, a set of questions related to DC motors is processed through the retrieval chain. The results are evaluated based on the relevance and coherence of the generated answers.

Deliverables

1. **Code Implementation:**
    - The full implementation is provided in a Google Colab notebook, which demonstrates the data loading, embedding creation, document retrieval, and generative answer production pipeline.
2. **Documentation:**
    - This document explains the architecture and methodology used to implement the RAG model for the QA bot, including key components such as document embedding, vector database retrieval, and generative response generation.
3. **Test Queries:**

- A set of sample queries has been provided along with the corresponding responses generated by the system, demonstrating the QA bot's ability to handle various questions about DC motors.

Conclusion

The RAG-based QA bot successfully integrates Langchain tools, Chroma for retrieval, and Google's Gemini API for both embeddings and language generation. This approach enables efficient retrieval of relevant document sections and the generation of accurate, well-formed answers to user queries. The bot can be further fine-tuned by adjusting chunk size, refining the prompt template, or experimenting with different embedding and generative models.

Future Enhancements

- Fine-tuning embeddings: Consider fine-tuning embedding models for domain-specific documents.
- Optimizing query handling: Implement techniques for better handling of ambiguous or vague queries.
- Scalability: As more documents are added, optimizing vector search mechanisms in Chroma can improve retrieval performance.