Advanced Retrieval-Augmented Generation (RAG) System

Overview

This project implements a modular Retrieval-Augmented Generation (RAG) system designed to ingest various document formats, retrieve relevant information, generate grounded answers using large language models (LLMs), and evaluate the system's performance comprehensively.

Features

- Supports loading PDFs and DOCX files with format-specific loaders.
- Preprocesses documents with recursive chunking to maintain semantic coherence.
- Implements dense vector retrieval using ChromaDB with persistent storage.
- Supports embeddings from HuggingFace and OpenAI.
- Combines retrieval with LLM-based generation (OpenAI GPT-4o, Groq llama-3.1-8b-instant).
- Provides a Streamlit frontend for interactive querying and model selection.
- Includes an evaluation framework scoring accuracy, coverage, hallucination, relevance, and fluency.
- Dockerized for deployment on Google Cloud Run with persistent vector storage.

Installation

1. Clone the repository.
2. Create and activate a Python virtual environment.
3. Install dependencies from `requirements.txt`.
4. Set up API keys for OpenAI and Groq as environment variables.

Usage

- Place your documents in the `Data/` directory.
- Run the Streamlit app to interact with the system.
- Use the evaluation script `Ragtester.py` to assess system performance.

Answering Assignment Questions

1. What documents and formats are supported?

- PDFs via `PyMuPDFLoader`.
- DOCX files via `UnstructuredWordDocumentLoader`.

2. How are documents preprocessed?

- Using `RecursiveCharacterTextSplitter` with chunk size 1000 and overlap 200 to preserve context.

3. What retrieval methods are implemented?

- Dense vector retrieval with cosine similarity.
- `ChromaDB` vector store with metadata for source tracking.

4. Which embeddings are used?

- HuggingFace `all-MiniLM-L6-v2` and OpenAI `text-embedding-3-small`.

5. What LLMs are supported?

- OpenAI GPT-4o.
- Groq llama-3.1-8b-instant.

6. How is generation prompted?

- Prompts include context, question, and instructions to avoid hallucination.

7. How is evaluation performed?

- Metrics include accuracy, coverage, hallucination score, relevance, and fluency.
- Implemented in `Ragtester.py`.

8. How is the system deployed?

- Dockerized with multi-stage builds.
- Designed for Google Cloud Run.
- Persistent vector store mounted via volume or GCS.

9. What trade-offs were made?

- Chose ChromaDB over FAISS for persistence and ease of use.
- Recursive chunking balances context size and coherence.
- Support for multiple LLMs for flexibility.
- Streamlit UI for rapid prototyping.

10. What future improvements are planned?

- Hybrid retrieval (BM25 + dense).
- Reranking with models like `bge-reranker`.
- User feedback loop.
- Document upload via UI.
- Citation highlighting.
- Export evaluation results to CSV or dashboard.

Testing Results Summary

The testing phase of the Advanced Retrieval-Augmented Generation (RAG) system focused on evaluating accuracy, coverage, hallucination rate, relevance, and fluency of generated answers.

Using the evaluation framework implemented in `Ragtester.py`, multiple test queries ran against diverse PDF and DOCX documents.

Key findings include:

- **Accuracy:** The system consistently retrieved and generated correct answers grounded in source documents, achieving high accuracy.
- **Coverage:** Recursive chunking ensured inclusion of relevant context, improving answer completeness.
- **Hallucination:** Prompt engineering and grounding techniques effectively minimized hallucinated content, yielding low hallucination scores.
- **Relevance:** Dense vector retrieval with ChromaDB provided highly relevant document chunks.
- **Fluency:** The LLMs (OpenAI GPT-4o and Groq llama-3.1-8b-instant) produced fluent and coherent responses.

Overall, testing results demonstrate robustness in generating accurate, relevant, and fluent answers with minimal hallucination, validating design choices in retrieval, embedding, and prompting.

Further testing with hybrid retrieval and reranking methods is planned to enhance performance.

Deployment

Access the deployed web application at: https://rag-based-system.streamlit.app/

For detailed instructions, usage examples, or contributions, please refer to the source repository.