Retrieval-Augmented Generation (RAG) API

Author: Tauhid Hasan
Email: tauhidhasan811@gmail.com

Overview

The Retrieval-Augmented Generation (RAG) API is an intelligent system built to handle multimodal input (text and images) and retrieve contextually relevant information before generating responses. It integrates two Large Language Models (LLMs):

- **GPT (gpt-oss-120b)** – Optimized for high-quality text understanding and generation.
- **Gemini (gemini-1.5-flash)** – A multimodal model capable of processing both image and text inputs.

Data Preparation

The system also includes a pre-prepared dataset sourced via data scraping from ExcelBD. Approximately 30 brand datasets were collected.
[Noted: You can use any data for this RAG application]

Data processing steps included:

1. Extracting Brand, Title, and Description to generate structured PDF documents.
2. Creating a .db SQLite database with the following schema:
   Brand TEXT, Title TEXT, Price TEXT, SKU TEXT, Categories TEXT, Tags TEXT
3. Preparing a CSV file containing all product links along with the columns:
   [**'Brand'**, **'Title'**, **'Link'**, **'Image'**, **'Price'**, **'SKU'**, **'Categories'**, **'Tags'**]
4. Also take some snap of the product from the website.

This dataset can be used for indexing, retrieval, and RAG-based querying.
[Data will be provided via email in zip.]

System Architecture

The application is organized into the following primary directories:

1. **Frontend**
   - Contains the user interface files.
   - The left-side navigation bar includes:
     - **File Upload** option.
     - **Mode Selection** drops down.
     - **Delete All** button allows deletion of all uploaded files for a specific folder.
2. **Backend**
   - Built with **FastAPI** for handling API endpoints.
   - Contains subdirectories for various functionalities:
     - **app/** – Main FastAPI application code, including basic API key configurations.
     - **service/** – Contains the **chunk creation** module and file readers for PDFs, databases, text, images, and DOCX files.
     - **vector_store/** – FAISS-based vector indexing implementation (replaceable with ChromaDB by swapping this module).
     - **storage/** – Stores metadata files generated during ingestion.
     - **utility/** – Contains file_utility.py for saving metadata and other helper functions.
     - **rag/** –
       - embedding.py – Handles embedding creation.
       - llm_client.py – Processes user queries and interfaces with the LLMs.
       - retriever.py – Retrieves relevant metadata and context for responses.

Environment setup:

1. **Hugging Face:**
   - In ***backend/core/config.py*** put the Hugging Face API key.

- For less complexity, add it to your environment variables:
  Go to your ***desktop → Edit Environment Variables → Add:***
  Name: HF_TOKEN
  Value: your_access_token

2. **Gemini:**
   - Go to https://console.cloud.google.com/
   - Sign in with your Google account and create/select a project.
   - Enable the API you need (e.g., Vertex AI / Gemini API).
   - Go to "IAM & Admin" → "Service Accounts" → Create a new service account.
   - Generate a new key (JSON format) and download it.
   - In config.py, set the absolute path to that JSON:
     os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "D:/RAG-API/rag-api/your-key.json"

   [An additional key will be provided via mail]

3. **OCR Setup:**

   - To enable text extraction from images on Windows, set the default path for Tesseract OCR executable in the file ***backend/services/reader.py***.
   - The default path is usually: ***C:\Program Files\Tesseract-OCR\tesseract.exe***.
   - Ensure Tesseract OCR is installed on your system; you can download it from Download Tesseract.

Start:

1. **Backend:**
   pip install -r backend/requirements.txt
   uvicorn backend.app.fast_api:app –reload

2. **Frontend:**
   pip install -r frontend/requirements.txt
   streamlit run frontend/app.py

Key Features

- Multi-Model LLM Integration: Leverages GPT for text queries and Gemini for multimodal (text + image) queries.
- Extensible Vector Storage: Built-in FAISS integration.
- Multi-Format Document Support: Processes PDFs, DOCX, TXT, images, and database files.
- Metadata Management: Organized storage and retrieval for improved search performance.

****Codes are available on GitHub**

[Additional Note: Still, this application works primarily via APIs, but some backend processes like finding related data and embedding user queries may take longer if you upload large or many documents. Using a GPU can improve performance for these tasks.]