Understanding of the project

By Golam Kaderye

This initiative, named "Generative AI & Chatbot Development," aims to develop an intelligent, document-aware question-answering system utilizing Python and FastAPI. The primary concept is to establish a Retrieval-Augmented Generation (RAG) API that does not depend solely on the pre-training of a language model to respond to user inquiries but rather enhances the model's answers with grounded, pertinent information sourced directly from documents uploaded by users. This necessitates that the system manage document ingestion, text and image processing, vector search, and integration with a language model to provide precise and contextual responses.

Central to the system is its capability to accept a diverse range of file types from users. These may encompass textual documents such as PDFs, Word files, and plain text files, in addition to tabular data in CSV format, or even structured data contained within SQLite database files. Visual formats like JPEG, PNG, and scanned PDF documents are also accommodated. Given that each of these formats necessitates distinct processing techniques, the system must integrate specialized parsing tools: for example, pdfplumber or PyMuPDF for extracting content from PDFs, python-docx for DOCX files, and pytesseract for performing OCR on images or scanned documents. After a file is uploaded, the system should clean and preprocess the content to derive meaningful, structured information. To enhance accuracy during retrieval, the content ought to be segmented into chunks, potentially with some overlapping text to maintain context and coherence across segments.

After the raw data is processed, the system transitions to the embedding phase, during which each segment of text is converted into a high-dimensional numerical vector. These embeddings encapsulate the semantic meaning of the text and facilitate similarity assessments. The project necessitates the use of either OpenAI's embedding models (such as text-embedding-ada-002) or Hugging Face models (like all-MiniLM-L6-v2). Subsequently, these embeddings are stored in a vector database, such as FAISS or ChromaDB, which enables efficient similarity searches. Additionally, the system must retain metadata for each segment, including the source filename, page number, and position within the document, to assist in tracing the origin of any response.

The system provides an HTTP POST endpoint query where users can submit a question, optionally including an image encoded in base64. If an image is provided, the system executes OCR to extract text from it. The user's question is also embedded, and a similarity search is conducted against the stored document segments to identify those that are most pertinent. The retrieved segments are then compiled into a prompt that encompasses both the context and the user's inquiry. This prompt is subsequently forwarded to a large language model such as GPT-3.5, GPT-4, Claude, or similar through an API call. The LLM utilizes the provided context to generate a grounded, precise, and human-readable response. The final output should not only present this answer but also include supporting evidence such as the filename and location (e.g., page number or chunk index) from which the information was sourced.

This system is crafted to emulate the manner in which a knowledgeable assistant would analyze documents and respond to inquiries by utilizing both its comprehension and direct citations from the document content. For instance, a user may upload an invoice and inquire, "What are the payment terms?" The system would then scan the embedded document, extract the pertinent section, and request the LLM to interpret and respond to the question using that section as context.

To further enhance the system, optional advanced features are available. These features include support for multimodal queries, where both image and text inputs are interpreted concurrently using models such as GPT-4 Vision. Additionally, it can accommodate multi-document querying, allowing questions to be answered based on context derived from several uploaded documents. A file-upload endpoint can be established to manage documents independently and assign unique file IDs for improved management and tracking. The orchestration of the entire pipeline, from ingestion to vector search to LLM response, can be executed more effectively using LangChain, a well-regarded framework designed for linking LLM-based applications. For deployment and sharing purposes, the system could be containerized with Docker, and optionally, a Streamlit frontend could be developed to provide users with a straightforward interface for uploading documents and visually submitting questions.

In conclusion, this project entails the creation of a comprehensive intelligent API system that integrates document parsing, optical character recognition, semantic embeddings, similarity search, and generative AI. It transcends merely answering questions; it aims to do so with accuracy and justification derived from actual document content, simulating a grounded AI assistant. The system will empower users to engage with their own data using natural language, thereby making document comprehension and question answering significantly more accessible, automated, and intelligent.