EMA TAKE HOME CHALLENGE

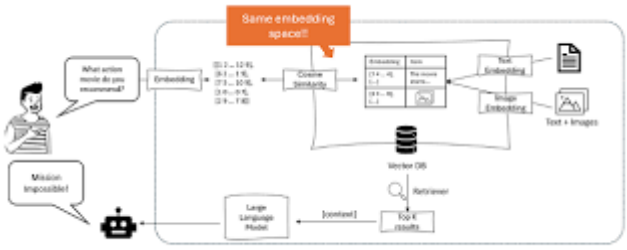
PROJECT OBJECTIVES:

The project implements a Natural Language Querying Agent using LangChain, OpenAI, and ChromaDB. The agent is designed to process and respond to natural language queries based on provided documents.

1. To build a natural language querying agent by utilizing LLMs, vector embeddings and vector databases.
2. Other than conversational answers the project should be able to extract details such as citations , tables etc

MULTIMODAL RAG FOR LLMS

Large Language Models (LLMs) have revolutionized how we interact with machines. They can generate text, translate languages, and answer questions with impressive fluency. However, LLMs have limitations, such as relying solely on text and lacking the ability to understand the broader context beyond their limited window of training data. This is where Multimodal Retrieval-Augmented Generation (Multimodal RAG) comes in.



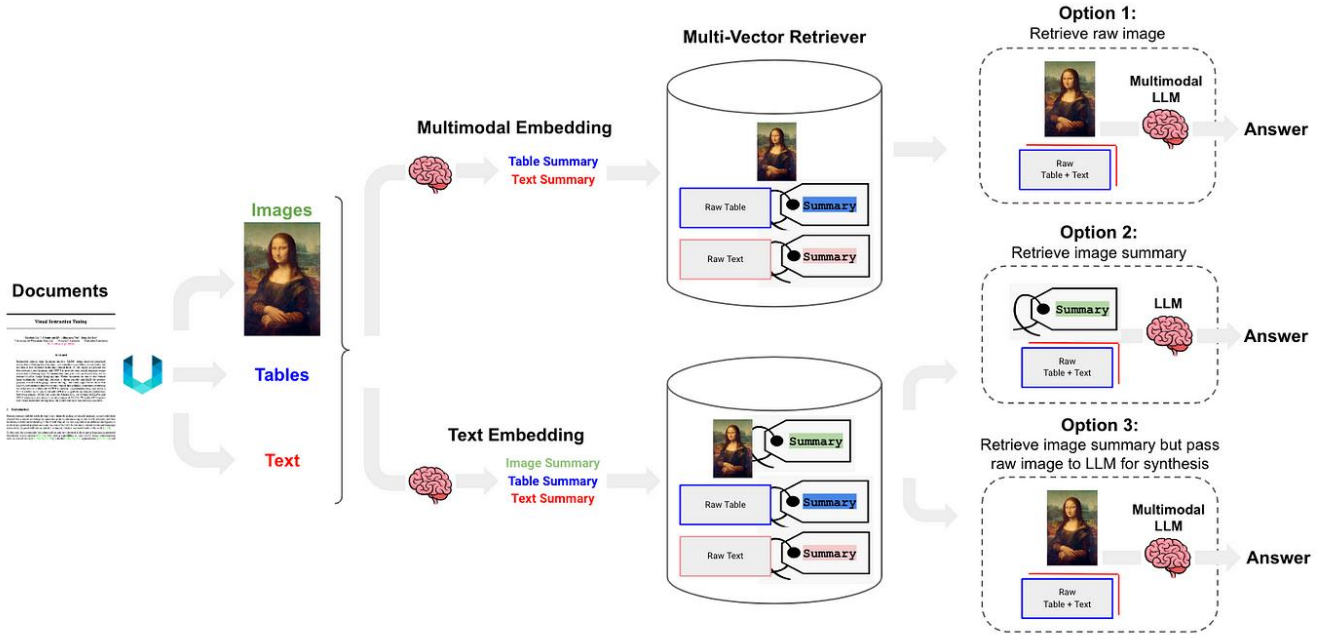
What is Multimodal RAG?

Multimodal RAG builds upon the foundation of standard RAG (Retrieval-Augmented Generation). In classic RAG, an LLM is "fed" relevant text snippets retrieved from a large corpus in response to a user query. This helps the LLM generate more accurate and informative responses compared to relying solely on its internal knowledge base. Multimodal RAG takes this a step further by integrating various data modalities beyond just text. This can include images, videos, audio, and potentially even tactile or olfactory information in the future. By incorporating these additional modalities, Multimodal RAG empowers LLMs to:

Gain a richer understanding of the user's intent and the context surrounding a query.

Generate more comprehensive and nuanced responses* that go beyond just text.

Better mimic human-like interaction* by processing information through multiple channels, similar to how we perceive the world.



How Does it Work?

The core principle of Multimodal RAG remains similar to its textual counterpart. Here's a simplified breakdown:

1. User Query: The user submits a question or request.

2. Retrieval: A multimodal retrieval system (often powered by vector databases) searches for relevant data across various modalities based on the query. This retrieved data can include text documents, images, audio clips, etc.

3. Generation: The retrieved data is presented to the LLM, which uses it as context to generate a response. Depending on the LLM capabilities, it might directly process the multimodal data or rely on text descriptions of the non-textual elements.

FEATURES

1. Quantized Model Integration: This app uses what are called "quantized models." These are special because they are designed to work well on regular consumer hardware, like the kind most of us have at home or in our offices. Normally, the original versions of these models are really big and need more powerful computers to run them. But quantized models are optimized to be smaller and more efficient, without losing much performance. This means you can use this app and its features without needing a super powerful computer. Quantized Models from TheBloke

2. Audio Chatting with Whisper AI: Leveraging Whisper AI's robust transcription capabilities, this app offers a sophisticated audio messaging experience. The integration of Whisper AI allows for accurate interpretation and response to voice inputs, enhancing the natural flow of conversations. Whisper models

3. Image Chatting with LLaVA: The app integrates LLaVA for image processing, which is essentially a fine-tuned LLaMA model equipped to understand image embeddings. These embeddings are generated using a CLIP model, making LLaVA function like a pipeline that brings together advanced text and image understanding. With LLaVA, the chat experience becomes more interactive and engaging, especially when it comes to handling and conversing about visual content. llama-cpp-python repo for Llava loading

4. PDF Chatting with Chroma DB: The app is tailored for both professional and academic uses, integrating Chroma DB as a vector database for efficient PDF interactions. This feature allows users to engage with their own PDF files locally on their device. Whether it's for reviewing business reports, academic papers, or any other PDF document, the app offers a seamless experience. It provides an effective way for users to interact with their PDFs, leveraging the power of AI to understand and respond to content within these documents. This makes it a valuable tool for personal use, where one can extract insights, summaries, and engage in a unique form of dialogue with the text in their PDF files.

GETTING STARTED

To get started with Local Multimodal AI Chat, clone the repository and follow these simple steps:

1. Create a Virtual Environment: I am using Python 3.10.12 currently
2. Upgrade pip: pip install --upgrade pip
3. Install Requirements: pip install -r requirements.txt
4. Windows Users: The installation might differ a bit for you, if you encounter errors you can't solve, please open an Issue here on github.

5. Setting Up Local Models: Download the models you want to implement. Here is the llava model I used for image chat (ggml-model-q5_k.gguf and mmproj-model-f16.gguf). And the quantized mistral model form TheBloke (mistral-7b-instruct-v0.1.Q5_K_M.gguf).
6. Customize config file: Check the config file and change accordingly to the models you downloaded.

Running the Project

1. Install dependencies: Make sure to install all required dependencies.
2. Run the script: Execute pdf_handler.py to process the PDF and add documents to the vector database.
3. Query the system: Use app.py to interact with the querying agent.

DEPENDENCIES

1. LangChain
2. OpenAI
3. ChromaDB
4. Additional dependencies as mentioned in the respective scripts

CODEBASE STRUCTURE

1. **app.py**
   This is the main entry point for the application. It handles the Streamlit web interface and user interactions.

   main(): The main function that initializes the application.

upload_pdf_file: Allows users to upload a PDF file.
add_docs_to_db(uploaded_pdf_file): Processes the uploaded PDF and adds its content to the vector database.
load_chain(chat_history): Loads the LLM chain for handling chat interactions.
run_llm_query(): Handles querying the LLM and displaying results.
llm_chain.py

2. **llm_chain.py**

This file handles the loading and management of the LLM chain. create_embeddings(): Initializes the embedding model.

load_vectordb(embeddings): Loads or creates a vector database.

load_pdf_chat_chain(chat_history): Sets up the LLM chain with the provided chat history.

pdfChatChain: A class that manages the interaction between the vector database and the LLM for generating responses.

3. **pdf_handler.py**

This file contains functions for processing PDF documents and adding them to the vector database.

get_pdf_elements(pdf_bytes, pdf_name): Extracts headers, footers, tables, and content from the PDF.

add_docs_to_db(pdf): Adds the extracted PDF elements to the vector database.

process_pdf(pdf_file): Reads a PDF file and processes it using the above functions.

4. **image_handler.py**

Handles image processing by converting image bytes to base64 format and generating descriptive text for images using a language model.

convert_bytes_to_base64: Converts image bytes to a base64-encoded string.

handle_image: Uses the Llama model to generate a chat-based description of the image, utilizing the Llava15ChatHandler for context.

5. **audio_handler.py**

Processes audio files by converting audio bytes to an array and transcribing the audio using the OpenAI Whisper model.

convert_bytes_to_array: Converts audio bytes into an array format using librosa.

transcribe_audio: Transcribes audio into text using the Whisper model from the Hugging Face transformers library.

6. **requirements.txt :** Contains all the dependencies required for installation

CONCLUSION

This codebase demonstrates how to build a Natural Language Querying Agent using LLMs, vector embeddings, and vector databases. The project includes functionalities for processing PDF documents, extracting content, and efficiently querying the extracted information using an LLM. The clear separation of concerns and modular design make it easy to extend and enhance the system in the future.