**AI RAG Chatbot**

This application allows a user to:

1. Enter and set their Google Gemini API key.
2. Upload a PDF document.
3. Process the PDF to extract text, split it into manageable chunks, create numerical representations (embedding’s) of these chunks, and store them in a searchable database (vector store).
4. Ask questions related to the content of the uploaded PDF. The chatbot will then retrieve relevant information from the PDF and use a Large Language Model (LLM) to generate an answer based on that information.

This entire process is known as Retrieval-Augmented Generation (RAG).

**a. Retrieval-Augmented Generation (RAG)**

RAG is a technique that enhances the capabilities of Large Language Models (LLMs) by giving them access to external, up-to-date, and specific information.

**- Without RAG:** An LLM answers questions based only on the data it was trained on. It might not know about recent events, specific documents, or internal company knowledge.

**- With RAG:** When a user asks a question, the system first *retrieves* relevant information from a knowledge base (like your uploaded PDF). This retrieved information is then provided to the LLM as *context* along with the user's question. The LLM then *generates* an answer using this specific context, making its response more accurate, relevant, and grounded in facts.

**b. Large Language Models (LLMs)**

LLMs are advanced artificial intelligence models capable of understanding and generating human-like text. They are trained on vast amounts of text data, allowing them to perform tasks like answering questions, summarizing text, translating languages, and writing creative content. In this project, a Google Gemini model (specifically gemini-1.5-flash) is used.

**c. Embedding**

Embedding’s are numerical representations of text (words, sentences, or entire documents). They convert human language into a format that computers can understand and process mathematically. Texts with similar meanings will have similar embedding’s, meaning they will be numerically "close" to each other in a multi-dimensional space. This allows for efficient searching and comparison of text.

**d. Vector Stores (FAISS)**

A vector store (or vector database) is a database designed to store and efficiently search through vector embedding’s. When you create embedding’s from your PDF text, you store them in a vector store. When a user asks a question, the question is also converted into an embedding. The vector store then quickly finds the most similar (closest) text chunks from the PDF based on their embedding’s.

**- FAISS (Facebook AI Similarity Search):** This is a library for efficient similarity search and clustering of dense vectors. It is used here as the vector store to index and search the embedding’s created from your PDF.

**e. LangChain**

LangChain is a framework designed to simplify the development of applications powered by LLMs. It provides tools and components to chain together different LLM functionalities, such as:

* Connecting to various LLMs (like Google Gemini).
* Integrating with different embedding models and vector stores.
* Building complex workflows like RAG pipelines, where multiple steps (retrieval, generation) are orchestrated.

**f. Gradio**

Gradio is a Python library that allows you to quickly create customizable user interfaces for your machine learning models. It automatically generates a web interface for your Python functions, making it easy to share and interact with your AI applications.

**g. PyPDF2**

PyPDF2 is a Python library used for working with PDF files. In this code, it is specifically used to extract text content from the uploaded PDF document.

**Imported Libraries**

* **os:** For interacting with the operating system, specifically to set the Google API key as an environment variable.
* **gradio as gr:** The Gradio library for building the user interface.
* **PyPDF2.PdfReader:** For reading PDF files.
* **langchain.text\_splitter.RecursiveCharacterTextSplitter:** A tool from LangChain to break down large texts into smaller, manageable chunks.
* **langchain\_community.embeddings.HuggingFaceEmbeddings**: For generating embeddings using models from Hugging Face.
* **langchain\_community.vectorstores.FAISS:** The FAISS vector store implementation for LangChain.
* **langchain\_google\_genai.ChatGoogleGenerativeAI**: To integrate with Google's Gemini LLM.
* **langchain.chains.create\_retrieval\_chain:** A LangChain utility to create the RAG retrieval chain.
* **langchain.chains.combine\_documents.create\_stuff\_documents\_chain**: A LangChain utility to combine retrieved documents with the prompt for the LLM.
* **langchain\_core.prompts.ChatPromptTemplate:** For defining the structure of the prompt sent to the LLM.

**FUNCTIONS**

* set\_api\_key(api\_key): Stores the Google API key as an environment variable.
* extract\_text\_from\_pdf(file): Extracts raw text from a PDF file.
* split\_text(text): Divides long text into small, overlapping chunks.
* create\_vector\_store(chunks): Converts text chunks into embeddings and stores them in FAISS.
* build\_rag\_chain(vectorstore): Sets up the retrieval-augmented generation pipeline using LangChain.
* chat\_with\_pdf(file, question, api\_key): Full pipeline to accept a PDF, extract content, run RAG, and answer the question.
* launch\_interface(): Builds and launches the Gradio web UI.