**Multi-PDF Chatbot Documentation**

**1. How It Works**

The Multi-PDF Chatbot is a Streamlit-based web application that allows users to upload multiple PDF files, extract text from them, and ask questions about the content. The application processes the PDFs, converts the text into embeddings (numerical representations), stores them in a vector database, and uses a conversational AI model to generate detailed answers based on the provided context.

**Key Workflow**:

* **PDF Upload**: Users upload one or more PDF files via a sidebar interface.
* **Text Extraction**: The app extracts text from all pages of the uploaded PDFs.
* **Text Chunking**: The extracted text is split into smaller chunks to manage large documents efficiently.
* **Embedding Creation**: Each chunk is converted into a vector embedding using Google’s Generative AI embedding model.
* **Vector Storage**: Embeddings are stored in a FAISS vector database for fast similarity search.
* **Question Answering**: Users input a question, and the app retrieves relevant text chunks from the FAISS index, passing them to a conversational AI model to generate an answer.
* **Response Display**: The answer is displayed on the Streamlit interface.

**Security Considerations**:

* The FAISS index is stored locally and loaded with allow\_dangerous\_deserialization=True, which is safe since the index is created by the app itself.
* The app uses environment variables to securely manage API keys.

**User Interface**:

* A clean Streamlit interface with a header, text input for questions, and a sidebar for PDF uploads.
* A footer credits the developer with a GitHub link.

**2. What Is Used**

The application leverages several Python libraries and tools to handle PDF processing, text embedding, vector storage, and conversational AI. Below is a detailed list:

**Libraries and Frameworks**

* **Streamlit (streamlit)**:
  + Used for creating the web-based user interface.
  + Provides components like st.file\_uploader, st.text\_input, st.button, and st.markdown for interactive elements.
  + Version: Compatible with recent versions (e.g., 1.38.x as of April 2025).
* **PyPDF2 (PyPDF2)**:
  + Extracts text from PDF files.
  + Handles multi-page PDFs and provides robust text extraction for text-based PDFs.
* **LangChain (langchain, langchain\_community, langchain\_google\_genai)**:
  + **LangChain Core**: Provides utilities for text splitting (RecursiveCharacterTextSplitter) and question-answering chains (load\_qa\_chain).
  + **LangChain Community**: Supplies the FAISS vector store (FAISS) for efficient similarity search.
  + **LangChain Google GenAI**: Integrates Google’s embedding and chat models (GoogleGenerativeAIEmbeddings, ChatGoogleGenerativeAI).
* **FAISS (faiss-cpu)**:
  + A vector database for storing and searching text embeddings.
  + Used for fast similarity search to retrieve relevant text chunks based on user questions.
* **Google Generative AI (google-generativeai)**:
  + Provides access to Google’s embedding and chat models via API.
  + Used for creating embeddings and generating conversational responses.
* **Python Dotenv (python-dotenv)**:
  + Loads environment variables (e.g., GOOGLE\_API\_KEY) from a .env file for secure API key management.

**Environment Setup**

* **Python**: Version 3.8+ recommended for compatibility.
* **.env File**: Stores the Google API key securely (e.g., GOOGLE\_API\_KEY=your\_api\_key).

**Development Tools**

* **IDE**: Any Python IDE (e.g., VS Code, PyCharm) for coding and debugging.
* **Command Line**: For running the Streamlit app (streamlit run chatapp.py).

**3. APIs Used**

The application relies on the following API:

* **Google Generative AI API**:
  + **Purpose**: Provides text embedding and conversational capabilities.
  + **Models Used**:
    - **Embedding Model**: models/embedding-001
      * Converts text chunks into vector embeddings for storage in FAISS.
      * Used in get\_vector\_store and user\_input functions.
    - **Chat Model**: gemini-1.5-pro
      * Generates detailed answers based on retrieved text chunks.
      * Used in get\_conversational\_chain for question answering.
  + **Authentication**: Requires a GOOGLE\_API\_KEY configured in the .env file.
  + **Configuration**: Initialized with genai.configure(api\_key=os.getenv("GOOGLE\_API\_KEY")).
  + **Usage**: Accessed via langchain\_google\_genai wrappers (GoogleGenerativeAIEmbeddings, ChatGoogleGenerativeAI).

**Note**: Ensure you have a valid Google Cloud account and API key with access to the Generative AI API. Usage may incur costs depending on the API’s pricing model.