**AI Data Assistant – Technical Documentation**

**1. Overview**

The **AI Data Assistant** is an interactive document-based chatbot application that enables users to **upload and query multiple file types** (PDF, DOCX, TXT, PPTX). It leverages **LangChain**, **FAISS**, and **OpenAI’s GPT models** to process, embed, and retrieve contextual information, allowing users to “chat” with their data.

The frontend is built with **Streamlit**, providing an intuitive web interface for document upload, chat interaction, and result visualization.

**2. System Architecture**

**2.1 Components**

* **Frontend:** Streamlit-based user interface for file uploads and chat input.
* **Backend Logic:** Handles document ingestion, text extraction, vector storage, and question answering.
* **AI Engine:** Combines LangChain’s RetrievalQA with OpenAI’s ChatOpenAI model for intelligent querying.
* **Vector Database:** FAISS index used for similarity search and efficient document retrieval.

**2.2 Workflow Diagram**

User → Upload Files → Text Extraction → Chunking & Embedding → FAISS Indexing

↓

Chat Query → Contextual Retrieval → GPT-4 Response Generation → Display

**3. Environment Setup**

**3.1 Dependencies**

The following Python packages are used:

* streamlit – UI framework
* langchain – LLM orchestration and retrieval pipeline
* faiss – Vector similarity search
* openai – Embedding and chat completion APIs
* pdfplumber, pytesseract, PIL – Text and OCR extraction
* python-docx, python-pptx – Document parsing
* dotenv – Environment variable management

**3.2 Configuration**

The .env file stores the OPENAI\_API\_KEY, which is loaded at runtime.  
If not found, the app halts with a clear error message via Streamlit.

**4. Core Functionalities**

**4.1 Text Extraction**

The extract\_text\_from\_file() function supports multiple file types:

* **PDFs:** Extracts text using pdfplumber; falls back to **OCR** with pytesseract for scanned documents.
* **Word (DOCX):** Extracts all paragraph text.
* **Text (TXT):** Reads and decodes UTF-8 text.
* **PowerPoint (PPTX):** Retrieves text from slide shapes.

**4.2 Summarization**

Long texts exceeding the defined SUMMARY\_THRESHOLD (1500 characters) are summarized using a GPT-based summarization chain for performance optimization and context clarity.

**4.3 Vector Store Creation**

The build\_vectorstore\_from\_files() function:

1. Extracts text from uploaded files.
2. Splits text into overlapping chunks using RecursiveCharacterTextSplitter.
3. Embeds chunks using OpenAIEmbeddings.
4. Builds a FAISS index for vector similarity search.
5. Stores indexed chunks in InMemoryDocstore.

This ensures fast semantic retrieval during user queries.

**4.4 Context Management**

The function build\_context\_from\_history() maintains conversation memory.  
If chat history exceeds a character limit (default: 3000), it is summarized to preserve efficiency.

**4.5 Question Answering**

The answer\_question() function:

1. Retrieves the most relevant chunks from FAISS (k=2).
2. Injects retrieved context into a prompt template.
3. Sends the prompt to GPT-4 for a natural-language response.
4. Returns the generated answer to the user.

**5. Streamlit User Interface**

**5.1 Sidebar – Document Upload**

* Users upload multiple files (pdf, docx, txt, pptx).
* On clicking “⚡ Process Files,” the system:
  + Extracts and indexes documents.
  + Confirms success with a visual notification.

**5.2 Chat Interface**

* Messages are displayed in a conversational format with role indicators:
  + 👤 **User messages**
  + 🤖 **Assistant responses**
* User inputs trigger retrieval + GPT response generation.
* Errors (e.g., invalid files or missing keys) are caught and displayed.

**6. Key Design Considerations**

* **Scalability:** The modular architecture allows adding more file types or vector databases easily.
* **Robustness:** Automatic OCR fallback ensures handling of scanned PDFs.
* **Efficiency:** Summarization and context management prevent memory overload.
* **Security:** API keys are managed securely through .env and not hardcoded.

**7. Future Enhancements**

* Integration with **persistent vector databases** (e.g., Pinecone, Chroma).
* Addition of **multi-user session support**.
* Implementation of **document re-indexing** and **metadata filtering**.
* Support for **image and table extraction** from documents.

**8. Conclusion**

The AI Data Assistant provides a complete, extensible framework for **document-based conversational AI**.  
By combining Streamlit’s interactivity, LangChain’s orchestration, and OpenAI’s language models, it delivers a powerful yet simple solution for data-driven dialogue and information retrieval.