**DeepSeek AI (DocuMind) - Project Documentation**

**Table of Contents**

1. Introduction
2. Architecture and Explanation
3. Features
4. Technology Stack
5. Components Deep Dive
6. Setting Up the project
7. How It Works
8. References

**Introduction**

**DocuMind AI** is an intelligent document analysis application that leverages Retrieval-Augmented Generation (RAG) technology to enable users to interact with PDF documents through natural language queries. The application allows users to upload PDF documents and ask questions about their content, receiving accurate, context-aware responses.

**Architecture and Explanation**

The system operates through the following stages:

1. **User Input (Frontend):** The user interacts with the system via a Streamlit web interface. They upload a PDF document and enter their query.
2. **Streamlit Web Interface:**
   * **Upload PDF:** The uploaded PDF is saved to disk in a designated directory (document\_store/pdfs/).
   * **Chat Input:** The user's query is captured for processing.
3. **Save Uploaded File (Disk):** The PDF file is stored locally for subsequent processing.
4. **PDF Parsing (PDFPlumberLoader):** The PDF document is parsed using PDFPlumberLoader to extract raw text chunks. This converts the PDF into a machine-readable format.
5. **Chunking (RecursiveCharacterTextSplitter):** The extracted text is split into smaller, manageable chunks using RecursiveCharacterTextSplitter.
   * **Chunk Size:** 1000 characters
   * **Overlap:** 200 characters (to maintain context between chunks)
6. **Text Embedding (OllamaEmbeddings):** Each text chunk is converted into a vector embedding using OllamaEmbeddings.
   * **Model:** deepseek-r1:1.5b (running via Ollama)
   * This process transforms the text into a numerical representation that captures its semantic meaning.
7. **Vector Store (InMemoryVectorDB):** The vector embeddings are stored in an InMemoryVectorDB. This allows for efficient similarity searching.
8. **User Query Input:** The user's query is the starting point for generating an answer.
9. **Similarity Search:** The query is embedded using the same OllamaEmbeddings model, and a similarity search is performed against the vector embeddings stored in the InMemoryVectorDB. The top-N most relevant chunks are retrieved.
10. **Answer Generation (LangChain + LLM):**
    * **PromptTemplate:** A prompt is constructed using the user's query and the retrieved context (relevant chunks).
    * **LLM:** The prompt is fed into the deepseek-r1:1.5b LLM (via Ollama) to generate an answer. LangChain is used to orchestrate this process.
11. **Answer Displayed to User:** The generated answer is displayed to the user via the Streamlit chat interface.

**Features**

**Core Features**

* **PDF Document Upload**: Support for single PDF file uploads
* **Intelligent Q&A**: Natural language questioning about document content
* **Context-Aware Responses**: Answers based on relevant document sections
* **Real-time Processing**: Immediate document analysis and indexing
* **Modern UI**: Dark-themed, chat-based interface

**User Experience Features**

* **Chat Interface**: Familiar messaging-style interaction
* **Visual Feedback**: Loading spinners and success messages
* **Responsive Design**: Clean, modern interface with custom styling
* **File Upload Validation**: PDF-only file acceptance

**Technology Stack**

**Frontend**

* **Streamlit**: Web application framework for Python
* **Custom CSS**: Dark theme styling and chat interface design

**Backend Processing**

* **LangChain**: Framework for LLM application development
* **PDFPlumber**: PDF text extraction library
* **Ollama**: Local LLM inference engine

**AI/ML Components**

* **DeepSeek-R1 (1.5B)**: Language model for embeddings and text generation
* **RecursiveCharacterTextSplitter**: Text chunking algorithm
* **InMemoryVectorStore**: Vector storage for document embeddings

**Data Processing**

* **Vector Embeddings**: Document similarity search
* **RAG Pipeline**: Retrieval-Augmented Generation workflow

**Components Deep Dive**

**1. Document Processing Pipeline**

**PDFPlumberLoader**

document\_loader = PDFPlumberLoader(file\_path)

* **Purpose**: Extracts text content from PDF files
* **Advantages**: Better text extraction than standard PDF libraries
* **Handles**: Tables, complex layouts, and multi-column text

**RecursiveCharacterTextSplitter**

text\_processor = RecursiveCharacterTextSplitter(

chunk\_size=1000,

chunk\_overlap=200,

add\_start\_index=True

)

* **Chunk Size**: 1000 characters per chunk (optimal for embedding models)
* **Overlap**: 200 characters overlap to maintain context continuity
* **Start Index**: Tracks original position in document

**2. Vector Storage System**

**InMemoryVectorStore**

* **Type**: Ephemeral storage (data lost on restart)
* **Performance**: Fast similarity searches
* **Scalability**: Suitable for single-session document analysis
* **Embedding Model**: DeepSeek-R1 1.5B parameter model

**3. Language Model Integration**

**OllamaLLM Configuration**

* **Model**: DeepSeek-R1:1.5b
* **Local Deployment**: Runs on local machine
* **Privacy**: No data sent to external APIs
* **Performance**: Optimized for document Q&A tasks

**4. User Interface Components**

**Streamlit Components**

* **File Uploader**: st.file\_uploader() with PDF validation
* **Chat Interface**: st.chat\_message() for conversation display
* **Input Handler**: st.chat\_input() for user queries
* **Status Indicators**: Progress spinners and success messages

**Custom Styling**

* **Dark Theme**: Professional appearance
* **Chat Bubbles**: Distinct styling for user vs assistant messages
* **Responsive Layout**: Adapts to different screen sizes

**Setting Up the Project**

**Install Ollama Locally**

Ollama allows you to run powerful open-source LLMs on your local system without the need for internet-based APIs.

Step-by-step Instructions:

**a. Download Ollama**

* Visit: [ollama.com/download](https://ollama.com/download)
* Download and install based on your OS (Windows / Mac / Linux)

**b. Start Ollama Service**

After installation, run the following command in your terminal or command prompt:

- ollama serve

**c. Pull the Required Model**

We'll use deepseek-r1:1.5b for both chat and embedding:

- ollama pull deepseek-r1:1.5b

**d. Run the Required Model**

We'll run deepseek-r1:1.5b:

- ollama run deepseek-r1:1.5b

Create a new project folder:

-mkdir deepseek-r1

-cd deepseek-r1

Install the dependencies:

-pip install -r requirements.txt

Make sure ollama serve is running, then:

-streamlit run rag\_deep.py

**How It Works**

**Document Processing Workflow**

1. **File Upload**
2. uploaded\_pdf = st.file\_uploader("Upload Research Document (PDF)", type="pdf")
3. **File Storage**
4. saved\_path = save\_uploaded\_file(uploaded\_pdf)
5. **Content Extraction**
6. raw\_docs = load\_pdf\_documents(saved\_path)
7. **Text Chunking**
8. processed\_chunks = chunk\_documents(raw\_docs)
9. **Vector Indexing**
10. index\_documents(processed\_chunks)

**Query Processing Workflow**

1. **User Input**: Natural language question entered
2. **Similarity Search**: Find relevant document chunks
3. **Context Assembly**: Combine relevant chunks
4. **Prompt Construction**: Create structured prompt with context
5. **LLM Generation**: Generate response using language model
6. **Response Display**: Show answer in chat interface

**References**

Github Link :- [DeepSeek R1](https://github.com/indiragothi/BlackCoffer-Work/tree/main/DeepSeek-R1)