📄 RAG-based PDF Q&A System (LangGraph + OpenAI)

This project implements an advanced Retrieval-Augmented Generation (RAG) system using LangGraph, FAISS, and OpenAI GPT models. It allows you to upload PDF documents (text + tables) and ask questions. The system extracts relevant chunks from PDFs, embeds them, retrieves context, and generates accurate, structured answers using LLMs.

🚀 Features

Text & Table Extraction: Extracts text and tables row-by-row from PDFs using pdfplumber.

Embeddings: Uses OpenAI's text-embedding-3-small for vectorization.

Vector Store: FAISS for fast similarity search.

LLM Integration: OpenAI GPT models (gpt-4o-mini) for generation.

Workflow Orchestration: LangGraph for modular graph-based RAG (Embed → Retrieve → Generate).

Streamlit UI: Interactive web interface for PDF upload and question answering.

Multi-PDF Support: Handles multiple PDFs simultaneously.

📂 Project Structure

```
Rag_Model_ML/
│── .env # OpenAI API Key
│── app.py # Streamlit UI
├── src/
│ ├── extractor.py # Extracts text & tables from PDFs
│ ├── embedder.py # Embeddings + FAISS index
│ ├── retriever.py # Retrieves top-k relevant chunks
│ ├── graph_builder.py # LangGraph workflow (Embed → Retrieve → Generate)
│ └── generator.py # OpenAI GPT response generation
├── data/ # Uploaded PDF files
└── requirements.txt
```

🔑 Configuration

Create a .env file in the project root:

OPENAI_API_KEY=sk-your-real-api-key-here

⚠️ No quotes, no spaces.

📝 How It Works (Module Overview)

1️⃣ Extractor (src/extractor.py)

Uses pdfplumber to extract:

Text from each page.

Tables row-by-row (converted to plain text with | separators).

Returns chunks: [Page 1] text, [Page 1 | Table 0] row1 | row2 | ....

2 Embedder (src/embedder.py)

Embeds chunks using OpenAIEmbeddings (text-embedding-3-small).

Builds a FAISS index for similarity search.

Provides embedding for queries to retrieve relevant chunks.

3 Retriever (src/retriever.py)

Performs top-k search on FAISS index.

Returns the most relevant chunks based on query embedding.

4 Generator (src/generator.py)

Uses OpenAI GPT (gpt-4o-mini) via langchain_openai.ChatOpenAI.

Combines retrieved context + user query to generate:

Text answers: summarized in sentences.

Table answers: key rows/columns explained briefly.

If information is missing: responds "Not found in the provided document".

5 LangGraph Workflow (src/graph_builder.py)

Defines a graph-based RAG pipeline:

Embed Node: Embeds user query.

Retrieve Node: Fetches top-k chunks from FAISS.

Generate Node: Calls GPT for final answer.

Modular and extensible for additional nodes/pipelines.

6 Streamlit UI (app.py)

Allows uploading PDFs and asking questions.

Displays:

Answer generated by LLM.

Retrieved Context (expandable).

Automatically builds FAISS index and runs LangGraph workflow.

🧩 Tech Stack

Component	Technology / Model
PDF Extraction	pdfplumber
Embeddings	OpenAI text-embedding-3-small
Vector Store	FAISS
LLM	OpenAI GPT gpt-4o-mini
Workflow	LangGraph (StateGraph)
Interface	Streamlit
Environment Config	.env (OpenAI API Key)

🔄 Workflow

Extract: Parse PDFs → text + table chunks.

Embed: Convert chunks into vector embeddings.

Store: Save embeddings in FAISS.

Retrieve: Search top-k relevant chunks for query.

Generate: LLM generates structured answer using retrieved context.

UI: User sees answer + context in Streamlit app.

📖 Example

Question:
"What is the total revenue mentioned in Report.pdf?"

Retrieved Context:
Rows extracted from PDF financial table.

Answer:
"The total revenue reported is 12.4M USD (FY2022)."

☑ Running the App

From project root:

streamlit run app.py

Upload PDFs in .pdf format.

Type your question in the input box.

Get AI-generated answers instantly.

🛠 Future Improvements

☑ OCR support for scanned PDFs.

☑ Table-aware embeddings for better accuracy.

☑ Multi-modal RAG (text + images).

☑ Optional Ollama local LLM support.

☑ Advanced LangGraph multi-step reasoning pipelines.

🤝 Contributing

Fork → branch → feature → PR.

Pull requests and suggestions are welcome!

📜 License

MIT License