DocuBot: A Local RAG-based Intelligent Document Assistant using Ollama + LLaMA 3.2

By: Rishitha

Table of Contents

1. Project Overview

DocuBot is a local Retrieval-Augmented Generation (RAG) system that allows users to upload .pdf or .txt documents and ask natural language questions about them. It uses Ollama's LLaMA 3.2 model locally for answer generation and leverages vector search to retrieve relevant context chunks before answering.

2. Objective

To create a private, offline, intelligent chatbot that can:
- Understand and summarize uploaded documents
- Answer questions based on real content
- Work fully locally without the internet or cloud services

3. System Architecture

A simplified flow:

PDF/TXT Files → Chunking & Embedding → FAISS Vector DB → Context Retrieval → LLaMA 3.2 → Response

4. Technologies Used

- Ollama: Run LLaMA 3.2 locally
- LLaMA 3.2: Language model for Q&A
- LangChain: RAG pipeline
- FAISS: Vector DB
- HuggingFace Embeddings: Sentence vectors
- PyPDF2/TextLoader: Document parsing

5. Techniques & Concepts Explained

Retrieval-Augmented Generation (RAG), Vector Embeddings, LLaMA 3.2 (local model).

6. Project Folder Structure

```
rag_doc_chat/
├── app.py              # Main chat
├── ingest.py           # Vector DB creation
├── requirements.txt    # Dependencies
├── docs/               # Your input files
├── vectorstore/        # Auto-created FAISS DB
```

7. Implementation Steps

1. Load PDFs/TXTs from `docs/`
2. Split and embed chunks
3. Save FAISS index
4. Query user input
5. Retrieve context and run via Ollama + LLaMA 3.2

8. How to Run the Project

- pip install -r requirements.txt
- ollama pull llama3
- python ingest.py
- python app.py

9. Sample Use Cases

- Ask policy-based questions
- Summarize academic PDFs
- Extract specific info from long docs

10. Future Improvements

- Streamlit or Gradio UI
- OCR for images
- Support DOCX, XLSX
- Multi-file comparison

11. Project Code Files

ingest.py

```
import os
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.document_loaders import PyPDFLoader, TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

def load_docs(folder_path):
    documents = []
    for file in os.listdir(folder_path):
        path = os.path.join(folder_path, file)
        if file.endswith('.pdf'):
            loader = PyPDFLoader(path)
        elif file.endswith('.txt'):
            loader = TextLoader(path)
        else:
            continue
        documents.extend(loader.load())
    return documents

def split_docs(documents):
    splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=100)
    return splitter.split_documents(documents)

def save_vectorstore(docs, persist_path="vectorstore"):
    embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
    vectorstore = FAISS.from_documents(docs, embedding=embeddings)
```

```
    vectorstore.save_local(persist_path)

if __name__ == "__main__":
    docs = load_docs("docs")
    chunks = split_docs(docs)
    save_vectorstore(chunks)
```

app.py

```
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.chains import RetrievalQA
from langchain.llms import Ollama

def load_vectorstore(path="vectorstore"):
    embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
    return FAISS.load_local(path, embeddings)

def build_qa_chain(vectorstore):
    retriever = vectorstore.as_retriever(search_kwargs={"k": 3})
    llm = Ollama(model="llama3")
    chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever)
    return chain

if __name__ == "__main__":
    vectorstore = load_vectorstore()
    qa_chain = build_qa_chain(vectorstore)
    print("Ask your question (type 'exit' to quit):")
    while True:
        user_input = input(" Ask: ")
        if user_input.lower() in ["exit", "quit"]:
            break
        response = qa_chain.run(user_input)
        print(" Answer:", response)
```

12. Conclusion

The DocuBot project successfully demonstrates the power of Retrieval-Augmented Generation (RAG) combined with a locally running large language model (LLaMA 3.2 via Ollama). It provides a practical, private, and efficient solution for querying large and unstructured documents without requiring internet access or cloud services.

By integrating LangChain, FAISS, and HuggingFace embeddings, the system is capable of understanding and answering user queries based on contextual knowledge extracted from uploaded PDF and text files. The modular design ensures ease of use, extensibility, and support for multiple domains like education, law, HR, and more.

This project lays the foundation for future enhancements, including UI development, support for additional file formats, document summarization, and even conversational memory, making it a powerful base for any domain-specific intelligent document assistant.