Chat with Multiple PDFs using Ollama

Date: 15th September 2025

□ Task Description

An **AI-powered document assistant** that allows users to chat with multiple PDFs. It combines **RAG (Retrieval-Augmented Generation)** with **Ollama's Gemma3:4b LLM** to make document understanding more interactive and user-friendly.

⚡ Key Features Implemented

- Uploading multiple PDFs
- Extracting and chunking text
- Creating embeddings using **Sentence Transformers (all-MiniLM-L6-v2)**
- Storing/retrieving data with **ChromaDB**
- Querying the knowledge base with **semantic search**
- Generating contextual answers via **Ollama (Gemma3:4b)**
- Building a **Streamlit UI** for document Q&A

□ Activities (Timeline)

- **10:30 – 11:30** → R&D and installation of Gemma model.
- **11:30 – 1:00** → Extracted text from PDFs and generated answers using Gemma3:4b.
- **1:30 – 2:30** → Lunch break.
- **2:30 – 3:30** → Learned FAISS indexing and Qdrant vector DB concepts.
- **3:30 – 5:30** → Developed the RAG model integrated with Ollama; implemented Streamlit UI for multi-document chat. Encountered bugs that require fine-tuning.
- **5:30 – 6:30** → Discussion with Rajasekar Anna about project scope and AI trends.
- **6:45** → Left for the day.

□ Skills & Learnings

From Today’s Work:

- Handling structured/unstructured text extraction from PDFs with **PyPDF2**.
- Understanding embeddings and semantic similarity using **Sentence Transformers**.
- Exploring **ChromaDB** collections, embedding storage, and query operations.
- Implementing a complete **RAG pipeline with LangChain + Ollama**.
- Building a user-friendly **Streamlit interface** with sidebar file upload & chat input.
- API integration with **Ollama’s LLM** for contextual responses.
- Troubleshooting **Python version issues** and database refresh logic.

Key Takeaways:

- RAG systems are only as good as the **quality of embeddings and retrieval**.
- Choosing the right **vector DB (FAISS, Qdrant, Chroma)** impacts scalability and efficiency.
- **UI matters**: an interactive frontend makes backend functionality much more accessible.

□ Challenges & Solutions

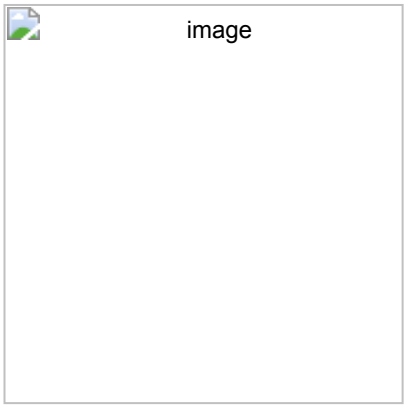
Obstacles Faced:

- Python version conflicts and dependency mismatches.
- Initial implementation only worked for **single documents**.
- Errors when scaling to **multi-document setup**.

Strategies Adopted:

- Did **step-by-step R&D** with single-document pipeline before moving to DB-based multi-document setup.
- Switched to **ChromaDB** for embedding storage.
- Implemented **checks for empty/duplicate files** and state refresh in Streamlit.

Output :



□ Self-Reflection

Observations:

Today gave me a **solid hands-on understanding** of how RAG pipelines work end-to-end. While theory (embeddings, retrieval) is clear, the real challenge was implementation—dealing with **bugs, DB handling, and Python compatibility**. The discussion with **Rajasekar Anna** gave useful clarity on scaling this project.

Future Action:

- Implement **FAISS and Qdrant** hands-on for comparison.
- Debug **multi-document retrieval accuracy** and Ollama integration.
- Enhance **Streamlit UI** with expandable retrieved context.
- Explore **prompt engineering tweaks** for more precise answers.