RAG-Based PDF Question Answering System (Mistral + Ollama + FAISS)

Abstract:

A lightweight, fully local Retrieval-Augmented Generation (RAG) system built using **Mistral (Ollama)** for embeddings and text generation, **FAISS** for vector search, and **Python** for PDF processing.
This project allows users to upload a PDF, generate embeddings, store them in a vector database, and then ask natural language questions related to the document.

🚀 What is RAG?

Retrieval-Augmented Generation (RAG) combines a Language Model (LLM) with an external knowledge source.
Instead of the LLM trying to "remember" everything, it **retrieves relevant chunks** from the PDF and uses them to generate precise, contextual answers.

In simple words:
👉 *The LLM becomes smarter by reading your PDF before answering.*

🛠️ Tech Stack

Component	**Technology**
Embeddings	**Mistral via Ollama**
LLM for answering	**Mistral via Ollama (chat mode)**
Vector Database	**FAISS (IndexFlatL2)**
PDF Extraction	**PyPDF**
Storage	vectors.index + chunks.pkl
Programming Language	**Python**

🧩 How the System Works

1️⃣ PDF → Text

The system extracts text from every page using **PyPDF**.

2️⃣ Text → Chunks

The text is split into **500-character chunks**.
This makes embedding + search efficient.

3️⃣ Chunk Embeddings (via Ollama Mistral)

Each chunk is converted into a vector using:

ollama.embeddings(model="mistral", prompt=chunk)

All vectors are stored in **FAISS** and saved as:

- vectors.index → FAISS index
- chunks.pkl → text chunks + metadata + total pages
-

4️⃣ Asking a Question

When the user asks a question:

1. The question is embedded (same Mistral embedding model)
2. FAISS retrieves the **top 3** most relevant chunks
3. These chunks are combined into a context
4. Mistral (chat mode) generates the final answer using:

ollama.chat(model="mistral", messages=[...])

5️⃣ Final Answer

The user gets an answer **grounded in the PDF**, not hallucinated.

📁 File Structure

rag/

|— pdf-vector.py # Convert PDF → chunks → embeddings → FAISS

|— question-vector.py # RAG querying system

|— vectors.index # FAISS index file

|— chunks.pkl # Chunk text + metadata

|— ragmist/ # Python virtual environment

✔ Step-by-Step Usage

1.Generate Vector Database from PDF

Set your PDF name inside pdf-vector.py, then run:

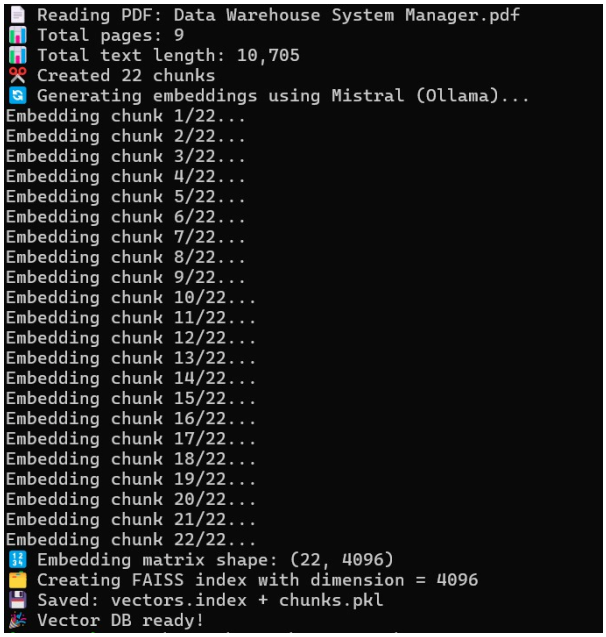
python pdf-vector.py

This creates:

- vectors.index
- chunks.pkl

[create the embedding for the pdf by splitting into chucks and store the metadata in the chunk.pkl, and store the vector embedding in the FAISS database.]

Snapshot for the embedding:



2. Ask Questions from the PDF

python question-vector.py

Example prompts:

"Explain page 4"

"What is data warehousing?"

"Define OLAP operations"

[creates embedding for the question and retrieves 3 most relevant chuncks from the vectorDB and send it to the Ollama mistral LLM to get the relevant data]

Snapshot of the output:

⚡ Challenges Faced

- Mistral embeddings occasionally fail for empty chunks
- Folder paths must be accurate when loading vectors.index and chunks.pkl
- Chunk/page estimation isn't perfect, but works well for general PDFs
- Ollama must be running locally for embeddings + chat generation

---

🧠 Why Mistral + Ollama?

- Fully offline
- Fast embedding generation
- Great quality for answering
- No API keys, no cloud billing
- Perfect for student projects & local experimentation

🎯 Outcome

You now have a complete, local, efficient **PDF Question Answering System** using:

- **Mistral for embeddings & LLM answers**
- **FAISS for similarity search**
- **Python for preprocessing**

A simple but powerful RAG pipeline that works end-to-end.