Bajaj AMC Factsheet Chatbot

Overview

This project is an AI-powered **Retrieval-Augmented Generation (RAG)** chatbot built using **LangChain**, **Groq LLM**, **Hugging Face embeddings**, and **FAISS vector database**.
It allows users to upload a **PDF** and ask questions about it.
The chatbot retrieves relevant information from the uploaded document and provides concise, factual answers with page citations.

Key Features

- Upload and process PDF documents dynamically.
- Extract and chunk the PDF text for better retrieval.
- Generate text embeddings using **Hugging Face MiniLM model**.
- Store embeddings in a **FAISS vector store** for semantic search.
- Use **Groq's LLM (Qwen 32B)** to generate grounded answers.
- Display results and citations in an interactive **Streamlit chat interface**.

Technology Stack

Component	Technology Used
Frontend / UI	Streamlit
Language Model	Groq LLM (qwen/qwen3-32b)
Embeddings	Hugging Face (sentence-transformers/all-MiniLM-L6-v2)
Vector Database	FAISS
Framework	LangChain
Language	Python

Working Of RAG Chatbot

1. Upload PDF

The user uploads the Bajaj AMC Factsheet through the Streamlit sidebar.

2. Data Ingestion and Processing

- The PDF is read using **PyPDFLoader**.
- The text is split into smaller chunks using **RecursiveCharacterTextSplitter**.
- Each chunk is converted into embeddings using **Hugging Face MiniLM model**.
- The embeddings are stored in a **FAISS index** for quick semantic search.

3. Retrieval-Augmented Generation (RAG)

When a user asks a question:

1. The query is embedded and matched with the most relevant chunks from FAISS VectorDB.
2. The top relevant text snippets (context) are combined with the user's question.
3. This context is sent to the **Groq LLM** which generates a factual response based only on that context.
4. The chatbot displays the answer along with citations (page numbers and text snippets).

4. Display

The conversation (user query, AI response, and sources) is shown in a chat-style interface using Streamlit.

Project Structure

```
├── app.py              # Main Streamlit application
├── .env                # Environment file storing API keys
├── faiss_index/        # Directory for FAISS index storage
└── requirements.txt    # Required Python dependencies
```

Environment Variables

Create a .env file in the root directory and add:

GROQ_API_KEY= "Groq API Key"

HF_TOKEN= "HuggingFaceToken"

Installation and Setup

Step 1. Install Dependencies

- streamlit
- langchain-Groq
- langchain-community
- langchain-text-splitters
- Sentence-transformers
- faiss-cpu
- pymupdf
- python-dotenv
- tqdm

Step 2. Add Environment Variables

Create a .env file as shown above.

Step 3. Run the App

streamlit run app.py

How to Use:

1. Open the app in your browser.
2. Upload the **Bajaj AMC Factsheet PDF** using the sidebar.
3. Wait for the document to be indexed.
4. Type your question in the chat input (for example, “What is the 3-year return of Bajaj Flexi Cap Fund?”).
5. The chatbot will display the answer along with citations from the document.

Explanation of Key Parameters

Parameter	Description
chunk_size = 800	Splits text into 800-character chunks for efficient retrieval.
chunk_overlap = 100	Adds slight overlap to preserve sentence continuity.
temperature = 0.0	Ensures factual, deterministic answers (no random outputs).
vectordb.save_local("faiss_index")	Saves FAISS index locally for reuse.
st.session_state.chat_history	Maintains conversation history during the Streamlit session.

Example Workflow

1. User uploads PDF → extracted and chunked.
2. Chunks converted into embeddings → stored in FAISS.
3. User asks a question → system retrieves top relevant chunks.
4. Groq LLM reads retrieved context → generates precise answer.
5. Answer + citations displayed in Streamlit chat.

Summary/Conclusion

This chatbot demonstrates how **RAG pipelines** can be used to build document-specific AI assistants. It ensures accurate, source-grounded answers using FAISS retrieval and Groq's LLM power, wrapped in a simple and user-friendly Streamlit interface.