RAG + LangChain + Groq Gemini Chatbot — Project Report

1. Problem Explanation

AI chatbots generally provide answers based on pre-trained knowledge, lacking the ability to incorporate user-specific documents or datasets dynamically. Also, job seekers often require systems that suggest relevant jobs based on their resumes.

This project builds a hybrid chatbot that:

- Uses Groq's Gemini LLM (`llama3-8b-8192`) for advanced language understanding.
- Answers both general and document-specific queries by combining retrieval from user-uploaded PDFs (RAG approach).
- Includes a job recommendation feature where resumes are matched to a job dataset CSV.
- Provides a unified UI to upload PDFs, resumes, and job datasets, and ask questions or get job suggestions.

2. Use of RAG with LangChain + Groq Gemini

- **Groq Gemini LLM** is accessed via `langchain_groq` with a secure API key from `.env`.
- User PDFs are processed by:
    - Loading with `PyPDFLoader`.
    - Chunking with `RecursiveCharacterTextSplitter`.
    - Embedding chunks with HuggingFace's `all-MiniLM-L6-v2` embedding model (on CPU).

    - Indexed into FAISS vector store for fast similarity retrieval.
- For each user query:
    - Relevant document chunks are retrieved.
    - Conversation history and retrieved context are combined into a prompt.
    - Groq Gemini generates a response based on this enriched context.
- For job matching:
    - Jobs are loaded from a CSV dataset.
    - Skills are extracted from uploaded resumes.
    - Matching jobs are suggested based on shared skills.

---

3. Folder and Code Walkthrough

```
your_project/
├── app.py                     # Main Streamlit app with UI and flow control
├── .env                      # Stores GROQ_API_KEY securely
├── requirements.txt          # Python dependencies
└── _modules/                 # Modular helper code for maintainability
    ├── llm.py                # Loads Groq Gemini LLM securely
    ├── pdf_processing.py     # PDF loading, splitting, embedding, FAISS indexing
    ├── prompt.py             # Builds prompts with history and context
    ├── chat.py               # Defines LangChain chain and queries LLM
    └── job_matching.py       # Loads job CSV, extracts skills, matches jobs
```

---

4. Challenges Faced and Solutions

- **Embedding model loading error:** Solved by forcing HuggingFace embeddings to use CPU (`model_kwargs={"device": "cpu"}`).

- **UI clutter from multiple uploads:** Unified file uploader with user selection for file type.

- **Combining conversation history with retrieved context:** Created a prompt builder that appends history and retrieved docs before LLM query.

- **Skill extraction accuracy:** Extracted skills from resumes by comparing with the job dataset's skill list.

- **Secure key management:** Used `.env` and `python-dotenv` to safely load API key without hardcoding.

---

5. Summary of Learnings

- Successfully integrated Groq Gemini LLM into a RAG pipeline with LangChain and FAISS.

- Learned document chunking, embedding, and indexing for retrieval-based Q&A.

- Handled multi-file uploads and user-friendly UI with Streamlit.

- Solved PyTorch and embedding model loading issues related to device placement.

- Built a job recommendation by skill matching from resumes and a job dataset.

- Developed modular, maintainable Python code with clear separation of concerns.

- Gained practical skills in a secure environment and API key management.
