AI-Powered Job Recommendation System using LangChain & Gemini

Problem Explanation - Job hunting is often a big challenge for students looking for a first opportunity, working professionals looking for a switch and people doing a career transition.A lot of challenges need to be faced due to jobs across multiple platforms, varying formats and description styles.Traditional searches relying on keywords only leading to irrelevant results sometimes.Manual shortlisting being time consuming at recruiters end and inefficient at candidates end.

This project solves the problem by automatic resume job matching using RAG technique and touch of Gemini model suggesting reasons for best fit for a particular job or application saving a lot of time reading job descriptions and analysing whether we are best fit for the application or not.Here LLM provides human readable match justification.

Use of RAG with LangChain + Gemini

Flow of the project:

1. Ingestion and Vectorisation

- Extracting skills, experience,location etc from candidate resume using langchain prompt and Gemini LLM.
- Fetching real-time job posting via RapidAPI (Google jobs / LinkedIn Jobs).
- Converting job description and resume extraction into embeddings using sentence-transformers

2. **Retrieval**

- Storing job vectors in FAISS vectorstore.
- Performing similarity search between the candidates resume vector and job vectors.

3. **Context Aware Generation**

- Passing the retrieved top matching-jobs + parsed resume to Gemini model via Langchain prompt.
- Gemini Model output gives a job match with valid justification.

4. **Presentation**

- Display job recommendation in Streamlit with company name , job title ,similarity score, apply link and justification.

Folder/Code Walkthrough

project-root/

```
│
├── app.py                          # Streamlit app entry point
├── .env                            # API keys and environment variables
├── requirements.txt
├── README.md
│
├── faiss_job_index_miniLM/         # Saved FAISS index (generated)
│
├── notebooks/                      # Colab/experiments and debugging
│   └── check
│
├── App_Snaps/                      # Screenshots of the app interface
│   ├── home.png
│   └── results.png
│
└── src/                            # All core application modules
    ├── settings.py                 # Config, API setup, model/embeddings init
    ├── resume_parser.py            # Resume extraction & text processing
    ├── jobs_scrapping.py           # Job scraping via RapidAPI
```

```
├── embeddings_model.py          # Embeddings + FAISS utilities
├── similarity_checks.py          # Vector similarity logic
├── chain.py                      # LLM prompt for match reasoning
├── final_chain.py                # Runnable pipeline assembly
└── main.py                       # Local testing and scripting
```

Challenges Faced and Solution

- LLM hallucinating skills not in resume - strict prompt rules were given to use only given terms for matching.
- Job Title search and flexibility - Implemented dynamic keyword-based OR-search functionality, mapping user inputs into API query strings. As per RapidAPI restrictions, the logical or operator must be explicitly placed between multiple job titles in the input string to ensure proper API handling.
- Token limitation with LLaMA model when processing FSSAI data - Faced token limiting while sending large chunks of data retrieved from FSSAi to LLaMA model , switched to gemini for longer text handling.
- Scraping jobs listings from LinkedIn/Google blocked - Blocked while scrapping data , switched to third party job aggregator API with free tier plan, accepting limitations.

Summary of what I learned

- How to design a retrieval augmented LLM pipelines with LangChain runnables.
- Practical implementation of LlaMA model and Gemini model in an end to end app.
- Building and querying FAISS vector stores form semantic similarity.
- Implementing robust prompt engineering to avoid hallucination of LLM model.
- Structuring scalable and modular python project for production deployment.
- Deploying an interactive streamlit app on streamlit cloud.