

# Job Recommendation System with RAG (LangChain + Groq/Gemma)

## 1. Problem Explanation

Finding relevant job opportunities efficiently is a challenge for many professionals. Job seekers often spend significant time filtering through irrelevant postings, manually searching on multiple platforms, and aligning opportunities with their specific skills, experience, and preferences.

Our objective was to **build an intelligent job recommendation system** that:

- Retrieves the most recent and relevant job postings from the web.
- Matches them to a candidate's skills, experience, and preferences.
- Allows input either **manually** or by **uploading a resume** for automated profile extraction.
- Provides ranked and contextually relevant recommendations.

The system integrates **Retrieval-Augmented Generation (RAG)** to ensure the recommendations are both **accurate** and **up-to-date**.

## 2. Use of RAG with LangChain + Gemini/Gemma

### RAG Workflow

1. **Retriever**: Uses Tavily API (and optionally Groq's browser search (openai model)) to fetch the latest job postings from multiple online sources.
2. **Retriever (LLM)**: Uses Groq's OpenAI model - "openai/gpt-oss-20b" which uses inbuilt tool of browser search for efficient web search.
3. **Vector Store**: Stores job postings in **FAISS** for semantic search and caching.

4. **Generator (LLM)**: Uses **Groq's Gemma model** to re-rank, summarize, and generate recommendations tailored to the candidate's profile.
5. **UI**: Built with **Streamlit** to allow interactive inputs and display results.

The RAG pipeline ensures:

- **Freshness** of information (latest jobs).
- **Contextual relevance** via embeddings and semantic search.
- **Natural language explanations** for each recommendation.

### 3. Folder / Code Walkthrough

```
project_root/
├── app/
│   ├── retriever.py    # Retrieves job postings (Tavily + Groq)
│   ├── recommender.py  # Ranks and summarizes recommendations
│   ├── resume_parser.py # Extracts skills, experience, location from PDF resumes
│   └── __init__.py
├── ui/
│   └── app.py          # Streamlit interface for user interaction
├── cache/              # Stores cached results
├── faiss_index/        # Stores persistent FAISS vector index
├── main.py             # CLI entry point (optional)
├── requirements.txt
├── README.md
└── .env                # API keys (Groq, Tavily)
```

## 4. Challenges Faced & Solutions

### Challenge 1: Inconsistent results from Groq browser tool

- **Issue:** Sometimes Groq API failed or returned incomplete data.
- **Solution:** Added **Tavily API fallback** to ensure we always retrieve relevant job postings.

### Challenge 2: Duplicate job listings

- **Issue:** Multiple sources returned the same postings.
- **Solution:** Added **URL-based deduplication** before storing in FAISS.

### Challenge 3: Resume parsing accuracy

- **Issue:** Regex-based parsing was brittle.
- **Solution:** Switched to **Groq LLM extraction** for robust and context-aware parsing.

### Challenge 4: Latency in recommendations

- **Issue:** Long processing times when fetching & re-ranking jobs.
- **Solution:** Implemented **result caching** using both JSON cache and FAISS index.

## 5. Summary of What I Learned


- **RAG Architecture:** How to combine retrieval (FAISS + API search) and generation (LLM re-ranking) for real-world applications.
- **LangChain Integration:** Efficiently chaining retrievers, vector stores, and LLMs.
- **Fallback Strategies:** Ensuring reliability by combining multiple data sources.

- **Resume Parsing with LLMs:** Using language models to extract structured data from unstructured documents.
- **Streamlit UI Development:** Building an intuitive, interactive frontend for non-technical users.

## 6. Additional Notes & Improvements for Future


- Add **job filtering** by salary range, company size, and remote/on-site preference.
- Integrate **real-time job alerts** via email/Slack.
- Use **FAISS search directly** for returning cached relevant results before hitting APIs.
- Implement **multi-language support** for parsing and recommendations.

## 7. Sample Input / Output


 **AI-Powered Job Finder**

Upload your resume or fill in your details to get personalized job recommendations.

Upload your Resume (PDF)

 Drag and drop file here  
Limit 200MB per file • PDF

Browse files

 Resume.pdf 1.0MB

×

✓ Resume parsed successfully!

Skills

Python, SQL, Machine Learning, Deep Learning, Power BI, Tableau, Flask, Streamlit, Docker, LangChain, LlamaIndex, TensorFlow, Keras

Experience


3 years


Preferred Location

Pune

Job Type

Full-time



 **Recommended Jobs**

Here are the most relevant job postings for your profile:

- Hire the best Artificial Intelligence Engineers in Pune, IN - Upwork**
  - **Why it's a good fit:** This posting specifically mentions deep learning, Flask, and Python, which align perfectly with your experience. The hourly rate is also competitive.
  - **URL:** <https://www.upwork.com/hire/artificial-intelligence-engineers/in/pune/>
- Machine Learning Jobs In Pune**
  - **Why it's a good fit:** This listing on Naukri.com is a broad search for machine learning roles in Pune. Your experience with Python, ML frameworks, and deployment tools makes you a strong candidate.
  - **URL:** <https://www.naukri.com/machine-learning-jobs-in-pune>
- Machine Learning Engineer - Python / Tensorflow - Pune**
  - **Why it's a good fit:** This job explicitly calls for Python and TensorFlow experience, two of your core skills. The 4-5 year experience requirement aligns with your 3 years of experience.
  - **URL:** <https://www.naukri.com/job-listings-machine-learning-engineer-python-tensorflow-vayuz-technologies-pune-4-to-5-years-240725929276>

**Important Note:** While the other postings may mention "machine learning," they lack the specific skills and experience requirements that match your profile.

---

Powered by Groq LLM + Tavily Search + FAISS Vector DB

## 8. Project Setup & Execution

### # 1. Clone the repository

```
git clone https://github.com/ish-war/AI-Job-Recommendation.git
cd AI-Job-Recommendation
```

### # 2. Create virtual environment

```
uv init
uv venv
source venv/bin/activate # Linux/Mac
venv\Scripts\activate    # Windows
```

### # 3. Install dependencies

```
uv add -r requirements.txt
```

### # 4. Add environment variables at .env

```
GROQ_API_KEY = "your_groq_key"
TAVILY_API_KEY = "your_tavily_key"
```

### # 5. Run Streamlit app

```
streamlit run ui/app.py
```