RAG-Based Job Search System

Name: Anish Maharjan
Assignment: GenAI_Takeaway_Assignment

1. Overview

This project implements a Retrieval-Augmented Generation (RAG) system for searching job listings from a structured dataset.

The system consists of:

- Semantic Search via Vectors (Facebook AI Similarity Search (FAISS) +Embeddings)
- Chunk-level retrieval
- LLM-based grounded summarization

The goal is to return relevant job listings for a user query and provide a concise, human-readable explanation of why those jobs match.

2. High-Level Architecture

Architecture Summary

The system works in two phases:

- Offline indexing
    - Clean job descriptions
    - Split them into chunks
    - Generate embeddings
    - Store vectors in a FAISS index
- Online querying
    - Embed the user query
    - Retrieve top-K similar chunks

- Aggregate chunks into job-level results
- Use an LLM to generate a grounded summary

Architecture Diagram

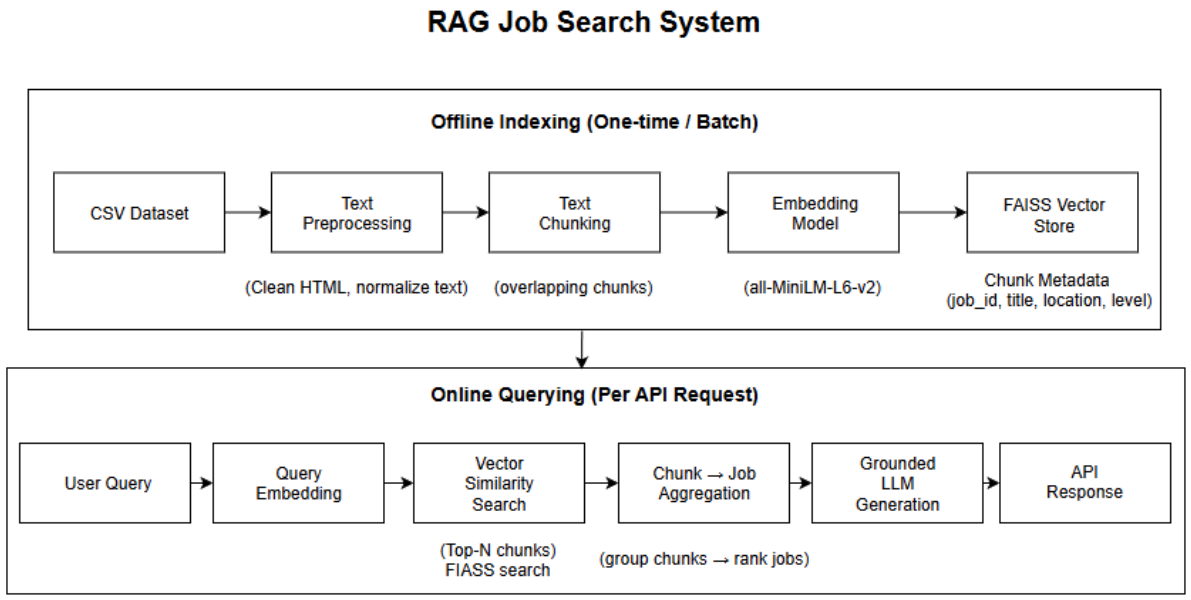


Figure 1: High-level RAG architecture used for job search

3. Engineering Decisions & Reasoning

3.1 Why Chunking?

Job descriptions are long and contain mixed information (requirements, benefits, responsibilities).

Chunking improves:

- Retrieval accuracy
- Semantic matching
- Grounded generation

Overlapping chunks ensure important context is not lost at chunk boundaries.

3.2 Why FAISS?

FAISS was chosen because:

- It is fast and efficient for vector similarity search
- It supports cosine similarity via normalized inner product
- It works well for offline indexing + online querying

This makes it suitable for production-style RAG pipelines.

3.3 Why Separate Indexing from Querying?

Embeddings are generated once, offline.

Benefits:

- Faster API responses
- Lower compute cost
- Cleaner architecture

This mirrors real-world systems where indexing and querying are decoupled.

3.4 Why LLM Only for Summarization?

The LLM is used only after retrieval, not for searching.

Reasons:

- Deterministic and reliable retrieval
- No hallucinated filters or invented skills
- Clear separation of responsibilitie

4. RAG Pipeline Components

Component	Implementation
Preprocessing	HTML cleaning, whitespace normalization
Chunking	Character-based overlapping chunks
Embeddings	SentenceTransformer (Hugging Face)
Vector Store	FAISS (IndexFlatIP)
Retriever	Top-K chunk retrieval + aggregation
LLM Integration (Google Gemini)	Grounded summary using retrieved chunks
API	FastAPI POST /api/query

Table 1: RAG pipeline Components

The `all-MiniLM-L6-v2` model was chosen because it provides a good balance between semantic quality and inference speed, making it suitable for real-time retrieval use cases.

4.1 Large Language Model (LLM)

Model Used: Google Gemini (`gemini-2.5-flash`)

The LLM is used only for grounded response generation, not for retrieval or ranking.
After relevant job chunks are retrieved via vector similarity search, the model combines the user query and retrieved content to produce a concise, human-readable summary.

To ensure reliability and prevent hallucination, the prompt explicitly restricts the model to using retrieved chunks only.

If the LLM is unavailable, the system returns a deterministic fallback summary built directly from retrieved job metadata.

5. API Design

Endpoint

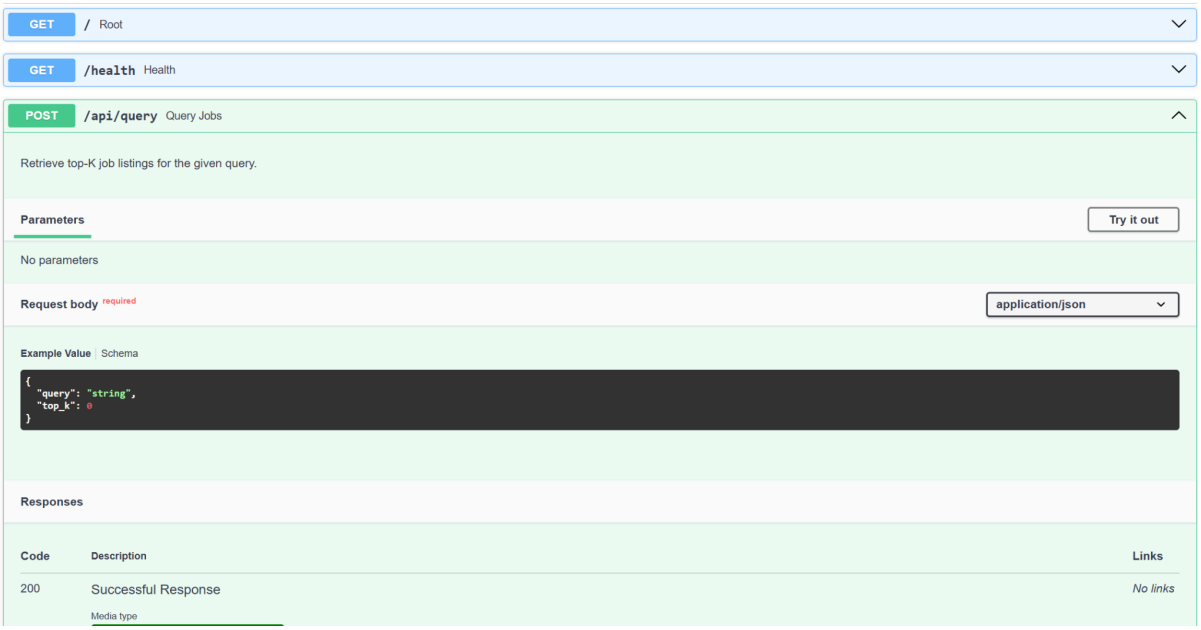
POST /api/query

API Behavior

The API returns both ranked job results and a grounded summary explaining why the retrieved jobs match the user query.
The summary is generated using retrieved job chunks only and does not rely on external knowledge.

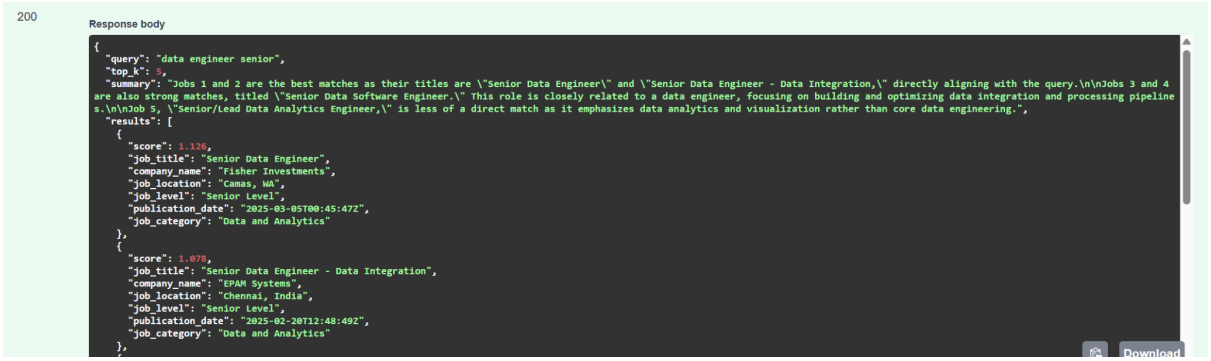
Example Request & Response


```
{

  "query": "junior data engineer remote",

  "top_k": 5,

  "summary": "Several junior-level data engineering roles match your query, particularly internships and entry-level positions with remote or hybrid options.",

  "results": [

    {

      "score": 0.81,

      "job_title": "Junior Data Engineer",

      "company_name": "Example Corp",

      "job_location": "Remote",

      "job_level": "Junior"

    }

  ]

}
```



Request Body

Response Body



6. Setup & Installation

6.1 Environment Setup

Create a .env file:

DATASET_PATH=path/to/jobs.csv

FAISS_INDEX_PATH=data/faiss.index

METADATA_PATH=data/metadata.json

EMBEDDING_MODEL=sentence-transformers/all-MiniLM-L6-v2

TOP_K=5

LLM_PROVIDER=gemini

LLM_MODEL=gemini-2.5-flash

6.2 Install Dependencies

pip install -r requirements.txt

6.3 Build Vector Index

python scripts/build_index.py

6.4 Run API

uvicorn app.main:app --reload

7. Assumptions Made

- Job descriptions may contain HTML
- Semantic similarity is sufficient for initial retrieval
- LLM availability is optional (fallback summary is provided)
- Dataset schema is consistent across rows

8. Drawbacks & Limitations

- No structured filters (location, level) at query time
- No cross-encoder reranking
- Chunk-level retrieval may surface partial information
- LLM output quality depends on retrieved chunks

9. Future Enhancements

- Add a cross-encoder reranker
- Add structured filtering (location, experience)
- Cache frequent queries
- Add pagination support
- Support multiple embedding models

10. Conclusion

This project demonstrates a clean, modular RAG pipeline for job search using modern NLP techniques.

The system:

- Separates retrieval from generation
- Avoids hallucination
- Is production-oriented
- Meets all assignment requirements