Job Search RAG System

# 1. Architecture Overview

This system is designed to provide an intelligent job search experience using a Retrieval-Augmented Generation (RAG) pipeline. It combines keyword search and semantic search, re-ranks results for better accuracy, and uses an LLM to validate and enrich the final output.  
  
Key Design Principles:  
- Hybrid retrieval ensures both precision (semantic) and recall (keyword).  
- Reranking boosts the most relevant results to the top.  
- LLM verification and explanation adds human-like interpretability to the results.  
- Separation of ingestion and query pipelines allows preprocessing-heavy work to be done once and reused efficiently.

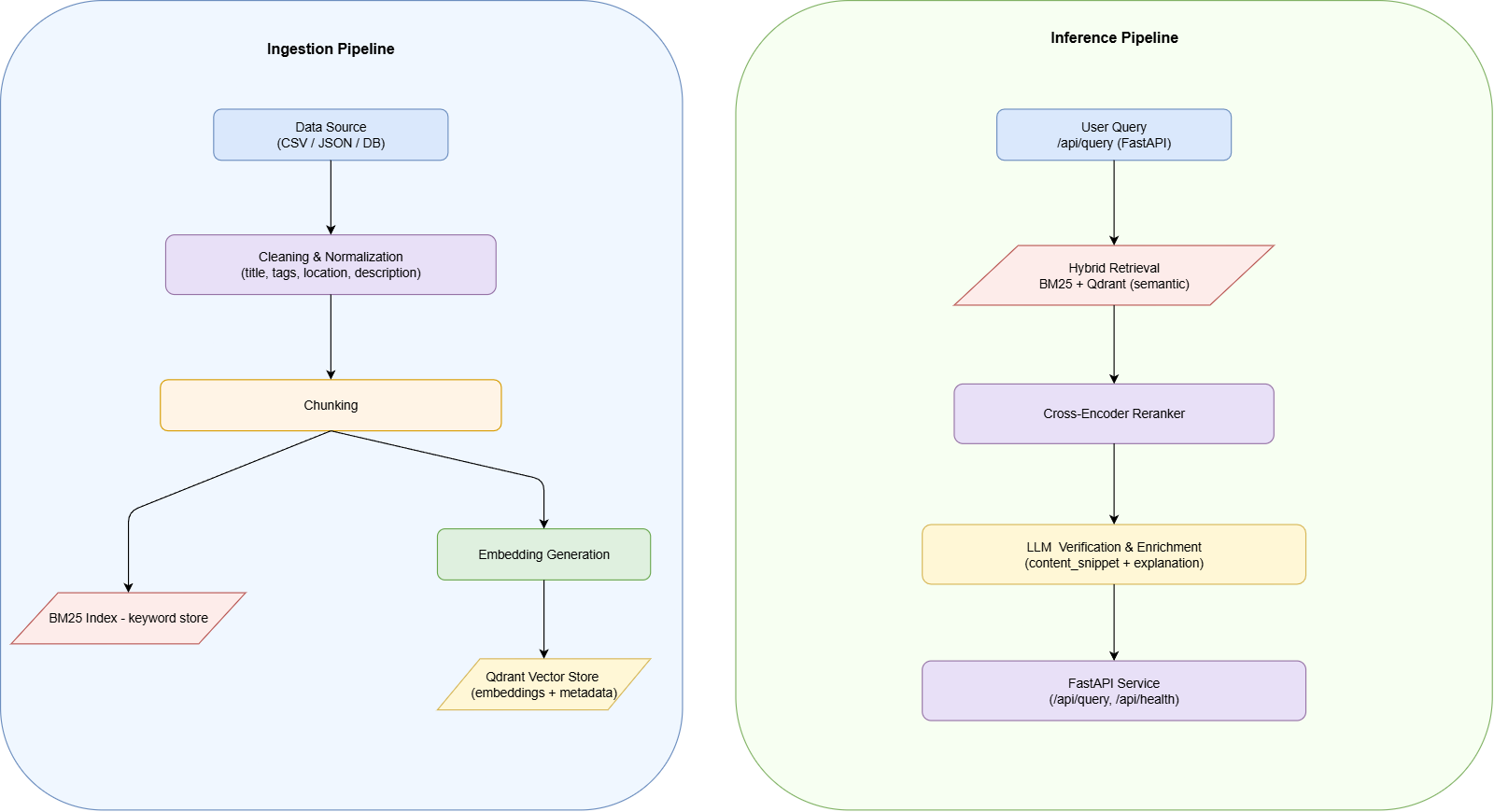


Fig: High Level architecture diagram of the system depicting ingestion and inference pipelines.

# 2. Pipelines

## 2.1 Ingestion Pipeline

Goal: Preprocess job postings and store them for efficient retrieval.  
  
Steps:  
1. Data ingestion – Job postings are ingested from the given Excel file.  
  
2. Cleaning & normalization – Titles, tags, locations, and descriptions are standardized.  
 Reason: Consistency improves both keyword retrieval and embedding quality.  
  
3. Chunking – Long job descriptions are split into section-wise chunks while keeping job metadata.  
 Reason: By chunking job descriptions into well-defined sections (e.g., *Responsibilities*, *Requirements*, *Benefits*), the most contextually relevant section may be searched.Also, Embeddingseparate sectionsseparately captures the semantic intent of each, improving similarity search in the vector store.  
  
4. BM25 Index (rank-bm25) – Chunks are stored in a BM25 inverted index.  
 Reason: BM25 is simple, fast, and strong for exact keyword matches.  
  
5. Embedding Generation (all-MiniLM-L6-v2) – Dense embeddings are created for each chunk.  
 Reason: This lightweight model balances semantic quality and speed, suitable for local/medium-scale setups.  
  
6. Vector Storage (Qdrant Cloud) – Embeddings + metadata are stored in Qdrant.  
 Reason: Cloud-hosted Qdrant scales well, and removes local infrastructural burden.

Manual Trigger: The ingestion pipeline must be run manually by an admin before starting the API server.

## 2.2 Query Pipeline

Goal: Retrieve, rerank, and explain jobs based on user queries.  
  
Steps:  
1. Query Input – User submits a query via /api/query.  
  
2. Hybrid Retrieval (BM25 + Qdrant) – Both keyword and semantic search results are retrieved. Combination is done via Reciprocal Rank Fusion (RRF).  
 Reason: RRF balances precision and recall by fairly merging rankings from both sources.  
  
3. Cross-Encoder Reranker (cross-encoder/ms-marco-MiniLM-L-6-v2) – Each (query, job\_chunk) pair is rescored.  
 Reason: Cross-encoders provide better semantic alignment than embeddings alone, especially for nuanced queries.  
  
4. LLM Verification & Explanation (LangChain + Groq) – The LLM takes top results and outputs:  
 - Job ID  
 - Title  
 - Location  
 - Explanation (why it matches)  
 Reason: Instead of returning raw retrieved text, the LLM provides an interpretable justification, improving trust and usability.  
  
5. Response Construction – Unlike strict JSON APIs, the response is returned as a string.

# 3. Usage

## Setup

1. Pull the repo from github

Clone:

git clone <https://github.com/MohitAryal/RAG-Job-Search.git>

Change current directory:

cd RAG-job-Search

2.Create a virtual environment:

python -m venv venv

# Activate on Windows

venv\Scripts\activate

# Activate on macOS/Linux

source venv/bin/activate

3. Install dependencies:  
 pip install -r requirements.txt

4. Add environment file (.env) with the following:

# Vector Database Configuration - Qdrant

VECTOR\_DB\_URL=link for qdrant node

VECTOR\_DB\_API\_KEY=your qdrant api key

VECTOR\_DB\_COLLECTION\_NAME=choose your collection name

VECTOR\_SIZE=based on embedding model used

# LLM Configuration - Groq

LLM\_MODEL=your chosen llm model

LLM\_TEMPERATURE=choose how deterministic you want the answers to be

LLM\_MAX\_TOKENS=limit the maximum number of output token

# Groq API Key

LLM\_API\_KEY=your groq api key

# Embedding Configuration

EMBEDDING\_MODEL=choose an embedding model

EMBEDDING\_DIMENSION=based on the model

EMBEDDING\_BATCH\_SIZE=how many chunks to embed in a single batch

# Search Configuration

DEFAULT\_TOP\_K=how many jobs to retrieve from hybrid search

# Reranking

RERANKER\_MODEL=choose a reranker model

RERANKER\_TOP\_K=how many jobs to return from reranker

# Keywords Search

BM25\_K1=controls how much term frequency (how often a term appears in a document) affects the relevance score.(1.2 to 2.0)

BM25\_B=adjusts how much document length normalization impacts the score.(0-1)

# Hybrid Search

K=choose your constant to use for combining search result using RRF

# Data Directories

DATA\_DIR=app/data

RAW\_DATA\_DIR=app/data/raw

PROCESSED\_DATA\_DIR=app/data/processed

CHUNKED\_DATA\_DIR=app/data/chunks

EMBEDDINGS\_DATA\_DIR=app/data/embeddings

KEYWORD\_RETRIEVER\_DIR=app/data/keyword\_retriever

# File Name

FILE\_NAME=LF Jobs.xlsx

PROCESSED\_FILE\_NAME=processed.json

CHUNKED\_FILE\_NAME=chunked.json

EMBEDDINGS\_FILE\_NAME=embeddings.npy

KEYWORD\_RETRIVER\_FILE=keyword\_retriver.pkl

Note : Use the same settings as in config file for similar result as mine.

4. Run ingestion pipeline (must be executed before queries can be served):  
 python -m app.ingestion\_pipeline

5. Start FastAPI:  
 uvicorn app.main:router --reload

## Example Request

"senior data scientist machine learning"

## Example Response

**LF0068 - 'Machine Learning and AI Engineering Lead'**

- This job was chosen because the job title exactly matches the user's query.

- The job description also mentions machine learning and AI, which aligns with the user's query.

- **Location:** Haguenau, London, Lunel, France, United Kingdom, USA

- **Seniority:** Senior Level

**LF0119 - 'Lead Machine Learning Engineer'**

- This job was chosen because the job title exactly matches the user's query.\n- The job description mentions machine learning and designing, building, and deploying production applications and data pipelines, which aligns with the user's query.

**Location:** Gavarr, Armenia

**Seniority:** Senior Level

**LF0053 - 'Lead Data Scientist - Healthcare Delivery'**

- This job was chosen because the job title mentions data scientist and machine learning, which aligns with the user's query.

- The job description mentions a care and risk modeling team and machine learning, which aligns with the user's query.

- **Location:** Trenton, NJ, USA

- **Seniority:** Senior Level"

# 4. Limitations

- No strict structured filters (e.g., remote=true) are enforced programmatically; filtering relies on chunks returned.  
- LLM adds latency and cost.  
- Ingestion must be run manually before the system is operational.

# 5. Future Improvements

- Add structured filters (remote, location, seniority) to reduce reliance on LLM.  
- Automate ingestion at app startup or schedule it periodically.  
- Implement query/result caching to save LLM calls.