GenAI Takeaway Assignment - Documentation

1. High-level architecture and Engineering Decisions

   Overview

   The system is a hybrid Retrieval Augmented Generation (RAG) pipeline that returns job listings for user queries. The RAG pipeline consists of various components such as preprocessing, embeddings, vector store, retriever, hybrid search and LLM integration coupled with thoughtful prompt design for accurate and concise LLM output. It also consists of an API endpoint, implemented using FastAPI. The main endpoint is a POST request to /api/query, designed to handle incoming queries in JSON format.

   The system first extracts key details like company, location, and job level from the user's query using a simple regex parser. Then it uses hybrid search to find the best job matches by combining two methods: dense search using sentence-transformer embeddings with a FAISS index, and sparse search using BM25 for keyword matching. The results are combined based on an adjustable weight (alpha). If a Cohere API key is available, the system provides an AI-generated summary of the top results, otherwise, it uses a basic fallback template. Finally, the system delivers a short, easy-to-read summary along with relevant job IDs and metadata.

   Components:

   i) Dataset

      The system uses a structured CSV file 'jobs.csv', which must contain at least the following columns: id, job category, job title, company name, publication date, job location, job level, tags, and job description.

ii) Preprocessing

After loading the CSV file, the text is cleaned and raw job data is prepared for search. Column names are normalized, and HTML tags present in the job description are cleaned. Then relevant fields (title, company, location) are concatenated into one searchable text. The long descriptions are then split into smaller chunks (1200 tokens with 200 overlap).

iii) Embeddings

The job descriptions are then embedded to create dense vector representations using the ‘all-MiniLM-L6-v2’ (Sentence-Transformers) model, enabling semantic search to provide contextually rich information to the large language model.

iv) Vector Store

FAISS (Facebook AI Similarity Search) is used to store all vector embeddings in a highly optimized index. FAISS works locally and is very fast for k-nearest neighbour searches.

v) Sparse Index

BM25 (Best Matching 25) is used to perform keyword based scoring to complement semantic search, because sometimes exact keyword matches can better help retrieve relevant chunks.

vi) Hybrid Retrieval

The hybrid retrieval system combines both dense and sparse retrieval methods. While dense retrieval excels at capturing semantic meaning, sparse retrieval is more effective at identifying keyword matches. Thus, combining the two we can get more accurate and relevant search results. Both the FAISS and BM25 scores are normalized and merged to rank results.

vii) Natural Language Filter Parser

The system includes a lightweight filter parser that extracts structured filters such as company, job level, category directly from the user's natural language query.

viii) Large Language Model summarization

The summarization component takes the top retrieved job listings and generates a concise, human-readable summary that highlights the most relevant results. Cohere is used, with carefully designed prompts to ensure clear and accurate summarization.

ix) FastAPI Endpoint

The system offers an API endpoint at POST /api/query that takes in a user's search query and returns a list of relevant job matches along with a brief summary.

2. Setup and installation instructions

Prerequisites:

- Python 3.9 +
- Cohere API Key
- Virtual Environment (venv recommended)

Installation:

Clone the repository

```
git clone <repo-url>
cd genai_assignment
```

Create a virtual environment

```
python3 -m venv venv
source venv/bin/activate     (on Mac/Linux)
```

venv\Scripts\activate (on Windows)

Install dependencies

pip install -r requirements.txt

Configure environment variables (.env)

EMBED_MODEL=all-MiniLM-L6-v2
DATASET_PATH=./data/jobs.csv
VECTOR_DB_PATH=./vectorstore/index.faiss
DOCSTORE_PATH=./vectorstore/docstore.jsonl
CHUNK_SIZE=1200
CHUNK_OVERLAP=200
TOP_K=5
HYBRID_ALPHA=0.6
COHERE_API_KEY=

Run the API

uvicorn main:app --reload

Open in Browser

http://127.0.0.1:8000/docs

3. Example usage:
    a. Requests

    {
        "query": "senior data engineer"
    }

    b. Expected responses

    {

"summary": "You inquired about senior data engineer job opportunities. Below is a list of jobs that match your query, along with their respective companies:\n\n- Senior Data Engineer at DataFinity Ltd.\n- Senior Data Engineer at Quantech Solutions Corp\n- Data Engineer at Mix Analytics\n\nThese postings seem to align with your search, do let me know if you require additional information on any of these positions.",
"results": [
{
"id": "LF0008",
"title": "Senior Data Engineer",
"company": "Fisher Investments",
"location": "",
"category": "Data and Analytics",
"level": "",
"tags": "",
"score": 1
},
{
"id": "LF0045",
"title": "Senior Data Engineer - Data Integration",
"company": "EPAM Systems",
"location": "",
"category": "Data and Analytics",
"level": "",
"tags": "",
"score": 0.9817
},
{
"id": "LF0155",
"title": "Senior Data Software Engineer",
"company": "EPAM Systems",

```
    "location": "",
    "category": "Data and Analytics",
    "level": "",
    "tags": "",
    "score": 0.9039
   },
   {
    "id": "LF0067",
    "title": "Senior Data Software Engineer",
    "company": "EPAM Systems",
    "location": "",
    "category": "Data and Analytics",
    "level": "",
    "tags": "",
    "score": 0.8923
   },
   {
    "id": "LF0245",
    "title": "Senior Software Engineer (Big Data)",
    "company": "EPAM Systems",
    "location": "",
    "category": "Software Engineering",
    "level": "",
    "tags": "",
    "score": 0.8802
   }
  ]
}
```

4. Assumptions made during development
    a. Job descriptions are split into chunks of 1200 tokens with 200 token overlap. These values were chosen as a balance between preserving enough context and staying within embedding model input limits.
    b. FAISS (dense embeddings) and BM25 (sparse keyword search) are combined for hybrid retrieval. The balance between the two is controlled by a weighting factor (alpha), assumed to be fixed for now.
    c. By default, the system retrieves the top 5 most relevant job postings (k=5). This is assumed to be an optimal tradeoff between providing enough options and avoiding information overload.
    d. The project is designed to run locally with .env files for configuration.
    e. The dataset (jobs.csv) is structured and consistently contains the following columns: d, job category, job title, company name, publication date, job location, job level, tags, and job description.
    f. Manual testing via FastAPI's /docs interface is assumed sufficient for demonstration.

5. Limitations
    a. There's no second-stage reranking using a cross-encoder, which could improve precision by re-evaluating top results in context.
    b. The query parser uses lightweight regex rules to extract filters like company, location, and job level, it may miss complex phrasing, uncommon abbreviations, or require consistent language structure.
    c. There's no Named Entity Recognition (NER) model to improve understanding of organizations, places, or roles.
    d. The embedding model, 'all-MiniLM-L6-v2' is not state-of-the-art.
    e. All jobs are treated equally regardless of how recently they were posted.
    f. There is no user-facing interface (e.g., React app), which may limit usability for non-technical users.

6. Future enhancements
    a. Dynamic chunking can be used at semantic boundaries like paragraphs or complete sentences to improve context preservation.
    b. Add lemmatization, fuzzy matching, and predefined skill dictionaries to handle variations in job titles and skills.
    c. Replace ‘all-MiniLM-L6-v2’ with more powerful embedding models. API-based embeddings (e.g., Cohere, OpenAI) could improve quality but introduce costs and external dependencies.
    d. Integrate a cross-encoder to re-rank top candidates with deeper contextual understanding.
    e. Apply a decay function to prioritize more recent job postings.
    f. Add an evaluation framework to measure retrieval quality using metrics like nDCG, MRR, and checks for answer faithfulness and groundedness in the RAG setup.
    g. Build a user-friendly web interface (e.g., using React) to make the system more accessible and easier to use.