GenAI Resume Screening Chatbot - Documentation

1. High-Level Solution Architecture

Overview

The **GenAI Resume Screening Chatbot** is designed to streamline candidate selection by leveraging **Retrieval-Augmented Generation (RAG)**. The architecture consists of:

- **Data Extraction & Processing Layer**: Extracts text from CVs (PDF/DOCX) and chunks them into semantically meaningful segments.
- **Embedding & Storage Layer**: Uses **Sentence Transformers** to generate embeddings and stores them in **Milvus Vector Database**.
- **Search & Retrieval Layer**: Enables natural language queries to fetch the most relevant candidate profiles.
- **Response Generation Layer**: Utilizes **OpenAI's GPT model** to generate structured responses.
- **User Interface (Streamlit)**: Allows users to upload CVs, search for candidates, and interact with the chatbot.

Technology Stack:

- **Python 3**
- **Milvus (Zilliz Cloud)** for vector storage
- **Sentence Transformers (all-MiniLM-L6-v2)** for embedding generation
- **OpenAI GPT-4o-mini** for natural language response generation
- **Streamlit** for user interface
- **Docker** for containerization

2. Methodology and Used Models

Methodology:

1. **Data Extraction**:
    - Extracts raw text from **PDF and DOCX** using PyPDF2, pdfplumber, and python-docx.

2. **Semantic Chunking**:
    - Uses NLTK and Sentence Transformers to split CVs into semantically meaningful chunks.
3. **Vectorization & Storage**:
    - Embeds the chunked text using all-MiniLM-L6-v2.
    - Stores embeddings in **Milvus Vector DB**.
4. **Candidate Retrieval**:
    - Accepts job descriptions as queries.
    - Computes similarity scores and fetches relevant CVs from **Milvus**.
5. **AI Response Generation**:
    - Constructs a structured recommendation using **GPT-4o-mini**.
6. **User Interaction (Streamlit UI)**:
    - Uploads & processes CVs.
    - Accepts job descriptions as input.
    - Displays matching candidates and AI-generated recommendations.

3. The Implemented Features

Core Functionalities:

Resume Processing

- Upload and process **multiple CVs**.
- Extract text from **PDF and DOCX**.
- Convert extracted text into structured embeddings.

Vector Storage & Search

- Stores embeddings in **Milvus Vector Database**.
- Implements **fast candidate retrieval** using **L2 distance similarity**.

Chatbot Interaction

- Accepts **job descriptions** or **skill queries**.

- Returns **top-ranked candidates** based on embedding similarity.
- Uses **GPT-4o-mini** to provide structured recommendations.

Streamlit UI

- Upload **CV files**.
- Search for candidates using natural language.
- Display retrieved candidate details and recommendations.

Dockerized Deployment

- Fully containerized with Dockerfile.
- Runs seamlessly using docker run.

4. Technical Challenges

1. **Resume Extraction**
   - Issue: Handling PDFs, DOCX, tables.
   - Solution: Used PyPDF2, pdfplumber, python-docx.
2. **Chunking**
   - Issue: Random sentence cuts, loss of meaning.
   - Solution: Used semantic chunking with Sentence Transformers.
3. **Llama 3 Issues**

- Issue: Crashed due to size.
- Solution: Switched to GPT-4o-mini.

5. Enhancements and Future Work

Better User Interface:

- Drag-and-drop file uploads.

Advanced Search Filters:

- Add filters for years of experience, skills, and job roles.

Switch from L2 Distance to Cosine Similarity:

Add User Authentication:

- Ensure only authorized recruiters can access the system.

Allow follow-up questions about previously found candidates.

6. Demo

How to Run the Chatbot:

1. **Run Locally:**

pip install -r requirements.txt

streamlit run app/app.py

Access UI at http://localhost:8501

2. **Run via Docker:**

docker build -t my-streamlit-app .

docker run -p 8501:8501 my-streamlit-app

Key Features in Action:

- Uploading & processing resumes
- Searching for candidates based on job descriptions
- AI-generated recommendations