Talent Acquisition Hub

A RAG-Based Chatbot for Assisting Talent Acquisition in Candidate Selection

Prepared by: Yara Mahfouz

Date: January 15, 2025

Contents

1 Executive Summary

The Talent Acquisition Hub is an innovative, cloud-based solution designed to revolutionize the recruitment process. It integrates advanced technologies such as Retrieval-Augmented Generation (RAG), large language models, and vector databases to streamline the analysis and retrieval of candidate information. This system provides talent acquisition professionals with a robust, interactive chatbot interface capable of understanding and responding to complex queries related to candidates' qualifications.

The project addresses key challenges in the recruitment process, such as efficiently parsing resumes, managing large datasets, and providing context-aware responses. By leveraging technologies like LangChain, Milvus, HuggingFace, and GPU acceleration, the system ensures fast, accurate, and intelligent processing of candidate resumes.
Key features include:

- Automated CV parsing and metadata extraction.
- Query-based retrieval of candidate information.
- Contextualized responses using advanced NLP models.
- Seamless integration with vector databases for efficient data storage and retrieval.

The Talent Acquisition Hub empowers recruiters to make data-driven hiring decisions, reducing manual effort and improving the overall efficiency of the recruitment process. With its scalable and flexible architecture, the system is well-positioned to address the evolving needs of modern talent acquisition.

2 Architecture

The Talent Acquisition Hub is designed with a modular and scalable architecture, enabling efficient handling of recruitment workflows. The system is divided into four main layers: **Frontend**, **Local Backend**, **Cloud Backend**, and **Vector Database**, each playing a distinct role in the processing and retrieval of data. (Figure 1)

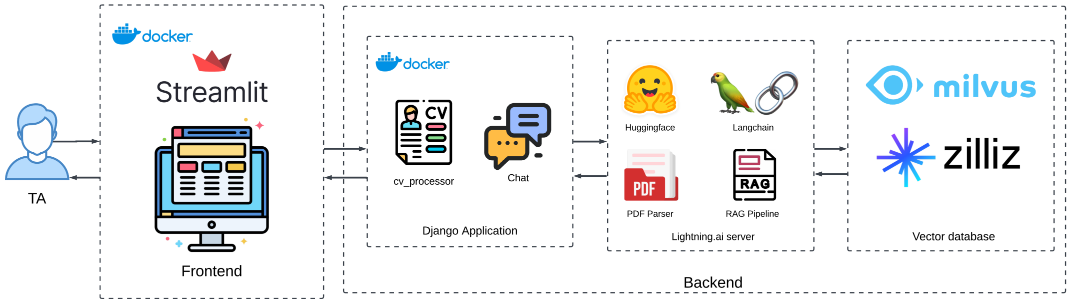


Figure 1: Overall System Architecture

2.1 Frontend

- **Technology:** Built with Streamlit, providing a clean and interactive user interface.
- **Purpose:**
  - Allows talent acquisition professionals to upload CVs and interact with the chatbot.
  - Displays retrieved insights and responses in real-time.
- **Deployment:** Dockerized for portability and scalability.

2.2 Local Backend

- **Technology:** Built with Django.
- **Purpose:** Acts as the intermediary between the Streamlit frontend and the cloud-hosted Lightning.AI server such that it Manages communication between them.
- **Handles lightweight tasks such as:**
  - Forwarding user messages to the chatbot pipeline.
  - Receiving parsed data or chatbot responses from the cloud backend.
  - Passing the processed data back to the frontend.
- **Benefits:**

    - Reduces latency by offloading computationally intensive tasks to the cloud backend.
    - Simplifies the integration of cloud services with the local deployment.
- **Deployment:** Dockerized for portability and scalability.

2.3 Cloud Backend

The Lightning.AI Flask backend is the core computational engine that powers the system's resource-intensive operations. It is designed with modularity in mind and is divided into two main modules:

1. **Offline Application:**
    - Handles preprocessing tasks such as parsing CVs, generating vector embeddings, and interacting with the vector database.
    - Ensures efficient data preparation for downstream retrieval tasks.
2. **Online Application:**
    - Focuses on real-time interactions, including processing user queries, retrieving relevant information from the vector database, and generating context-aware responses.
    - Uses advanced NLP pipelines and retrieval-augmented generation (RAG) workflows to deliver accurate and insightful answers.

3 API Documentation

This section provides details about the APIs used in the Talent Acquisition Hub, including endpoints in the **Django backend** and the **Lightning.AI server**. Each API is documented with its purpose, input, output, and functionality.

3.1 Local Backend APIs

1. **ParseAndStoreCVsView**
    - **Endpoint:** /retriever/parse-and-store-cvs/
    - **HTTP Method:** POST
    - **Purpose:** processes batch uploads of PDF CVs by forwarding them to a cloud-based parser for extraction. It then splits the processed data into manageable chunks and calls another cloud-based service to create a structured collection in a vector database.
    - **Input:**
        - **Files:** A batch of PDF files uploaded through a 'multipart/form-data' request.

- **Output:**
  - **On Success:** A JSON response with the parsing status and the newly created collection name. [language=Bash] ”results”: ”CVs parsed and stored successfully”, ”collection$_{n}ame$” : ”$collection_{<}uuid >$ ”
  - **On Failure:** An error message. [language=Bash] ”error”: ”Error description”
- **Error Codes:**
  - **400:** Missing files or invalid file format (e.g., unsupported file type or corrupted file).
  - **500:** Errors during file transfer, parsing, or storage, including issues with the cloud parser or the vector database.

2. **SendMessageView**

- **Endpoint:** /chatbot/send-message/
- **HTTP Method:** POST
- **Purpose:** Acts as a bridge between the frontend and the cloud-hosted chatbot API. It forwards user messages and returns responses.
- **Input:** [language=Bash] ”collection$_{n}ame$” : ”$string$”, ”$message$” : ”$string$”
- **Output:**
  - **On Success:** A JSON response containing the chatbot's reply. [language=Bash] ”response”: ”Chatbot response”
  - **On Failure:** An error message. [language=Bash] ”error”: ”Error description”
- **Error Codes:**
  - **400:** Missing ‘collection_name‘ or ‘message‘ field, or invalid field format (e.g., empty string).
  - **500:** Internal server error, such as connectivity issues with the chatbot API.

3.2 Cloud Backend APIs

1. **Parse PDFs**

- **Endpoint:** /parse-pdfs
- **HTTP Method:** POST
- **Purpose:** Handles batch uploads of PDF documents, parses them into text, and returns the parsed data as a ZIP file.
- **Input:**
  - **Files:** A batch of PDF files uploaded through a ‘multipart/form-data‘ request.

- **Output:**
  - **On Success:** A ZIP file containing the parsed documents.
  - **On Failure:** A JSON error message. [language=Bash] ”error”: ”Error description”
- **Error Codes:**
  - **400:** No files uploaded or invalid file types (non-PDF).
  - **500:** Server-side errors during file parsing or handling.
- **Workflow:**
  - Uploaded PDF files are saved temporarily.
  - The ‘parse_pdf‘ function processes the files and extracts their content.
  - The parsed content is zipped and sent as the response.
  - Temporary files are cleaned up after the operation.

2. **Add Collection**

- **Endpoint:** /add-collection
- **HTTP Method:** POST
- **Purpose:** Creates a new collection in the vector database and stores parsed document embeddings for later retrieval.
- **Input:** [language=Bash] ”collection$_{n}ame” : ”string”, ”documents” : [”id” : ”string”, ”con$
- **Output:**
  - **On Success:** A JSON response confirming the collection creation. [language=Bash] ”message”: ”Collection ’collection$_{n}ame'addedsuccessfully$”
  - **On Failure:** A JSON error message. [language=Bash] ”error”: ”Error description”
- **Error Codes:**
  - **400:** Missing ‘collection_name‘ or ‘documents‘.
  - **500:** Server-side errors during collection creation or embedding storage.
- **Workflow:**
  - The input documents are converted into LangChain ‘Document‘ objects.
  - The ‘RetrieverManager‘ adds the documents to the vector database under the specified collection name.
  - LangChain memory is cleared after adding a new collection as it signifies the start of a new conversation.

3. **Send Message**

- **Endpoint:** /send-message
- **HTTP Method:** POST

- **Purpose:** Processes user queries using the RAG (Retrieval-Augmented Generation) pipeline by retrieving relevant information from the vector database and generating chatbot responses.
- **Input:** [language=Bash] ”collection$_n$*ame*” : ”*string*”,”*message*” : ”*string*”
- **Output:**
  - **On Success:** A JSON response containing the chatbot’s reply. [language=Bash] ”response”: ”Chatbot response”
  - **On Failure:** A JSON error message. [language=Bash] ”error”: ”Error description”
- **Error Codes:**
  - **400:** Missing ‘collection_name‘ or ‘message‘.
  - **500:** Errors during retrieval or response generation.
- **Workflow:**
  - The ‘collection_name‘ is used to retrieve the corresponding vector database collection.
  - A ‘ContextualCompressionRetriever‘ is set up with the selected retriever and an LLM-based compressor.
  - The query is processed using the chatbot pipeline, which generates a response based on the retrieved context.

4 System Components

4.1 Frontend

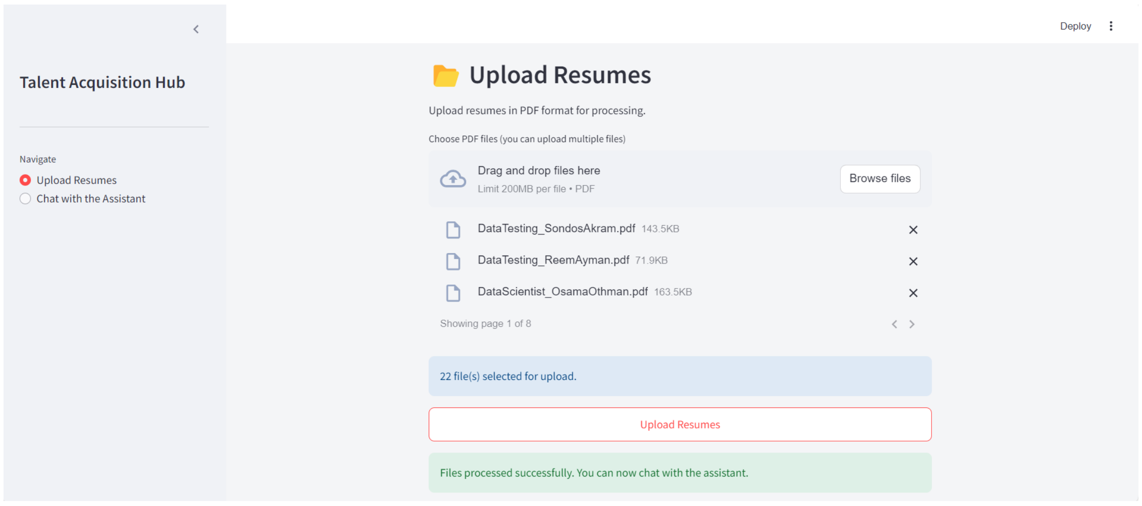
The frontend of the Talent Acquisition Hub offers a simple and user-friendly interface for recruiters. It has two main pages: Upload Resumes and Chat with the Assistant, accessible via a sidebar navigation menu.

The Upload Resumes page (Figure 2) allows users to upload multiple resumes in PDF format. These resumes are sent to the backend for parsing and storage in the vector database. Once the files are processed, a collection is created, and users are notified. This page serves as the entry point for preparing data for chatbot queries.

The Chat with the Assistant page (Figure 3), which is only accesible after cvs are processed, provides a conversational interface where users can ask questions about the uploaded resumes. Messages are dynamically displayed in a chat history, with responses from the backend-generated chatbot. Users can enter queries in the input box, and the chatbot retrieves relevant data from the vector database to provide context-aware answers.

4.2 Cv Parsing and Storage

The CV parsing and storage process begins when resumes in PDF format are uploaded through the frontend. These files are sent to the backend, where they are processed using a dedicated parser. The parser extracts text and metadata from the documents,


Talent Acquisition Hub

Navigate

● Upload Resumes

○ Chat with the Assistant

Deploy

Upload Resumes

Upload resumes in PDF format for processing.

Choose PDF files (you can upload multiple files)

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

Browse files

DataTesting_SondosAkram.pdf 143.5KB

DataTesting_ReemAyman.pdf 71.9KB

DataScientist_OsamaOthman.pdf 163.5KB

Showing page 1 of 8

22 file(s) selected for upload.

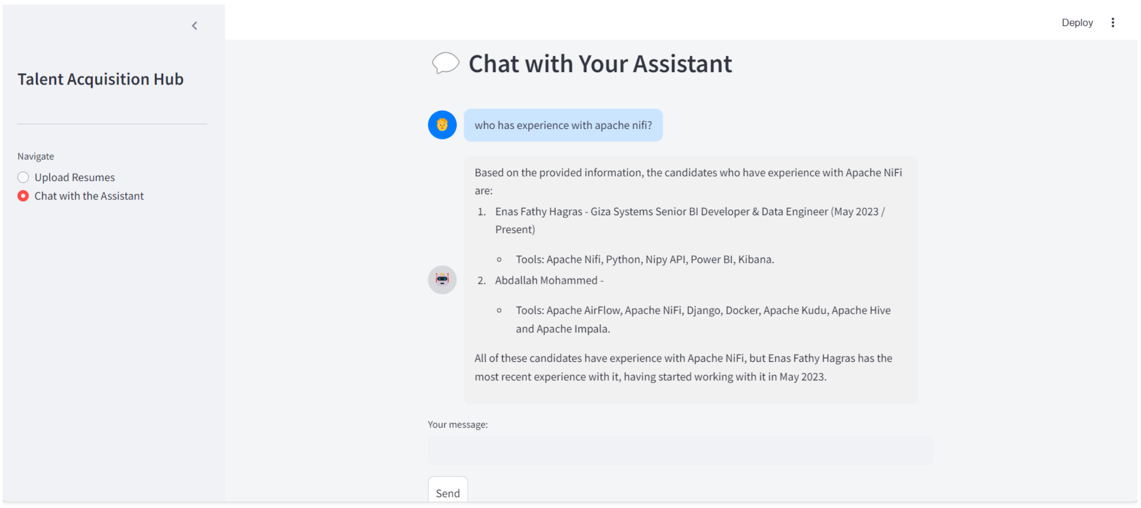
Upload Resumes

Files processed successfully. You can now chat with the assistant.

Figure 2: cv-upload page

and converts them into a structured format suitable for downstream processing. The extracted data is then split into manageable chunks using RecursiveCharacterTextSplitter with the candidate's name extracted using openAI and appended to the chunk's metadata to ensure efficient storage and retrieval.

Once parsed, the document chunks are transformed into dense vector embeddings using the dunzhang/stella_en_400M_v5 model. These embeddings, along with their associated metadata, are stored in the Milvus vector database under unique collections (Figure 4. Each collection represents a set of resumes, allowing for efficient similarity-based retrieval. This setup ensures that the system can quickly and accurately retrieve relevant information when queried, forming the foundation for the chatbot's context-aware responses.



who has experience with apache nifi?

Based on the provided information, the candidates who have experience with Apache NiFi are:

1. Enas Fathy Hagras - Giza Systems Senior BI Developer & Data Engineer (May 2023 / Present)
   - Tools: Apache Nifi, Python, Nipy API, Power BI, Kibana.
2. Abdallah Mohammed -
   - Tools: Apache AirFlow, Apache NiFi, Django, Docker, Apache Kudu, Apache Hive and Apache Impala.

All of these candidates have experience with Apache NiFi, but Enas Fathy Hagras has the most recent experience with it, having started working with it in May 2023.

Figure 3: Chat Page

4.3 RAG Pipeline

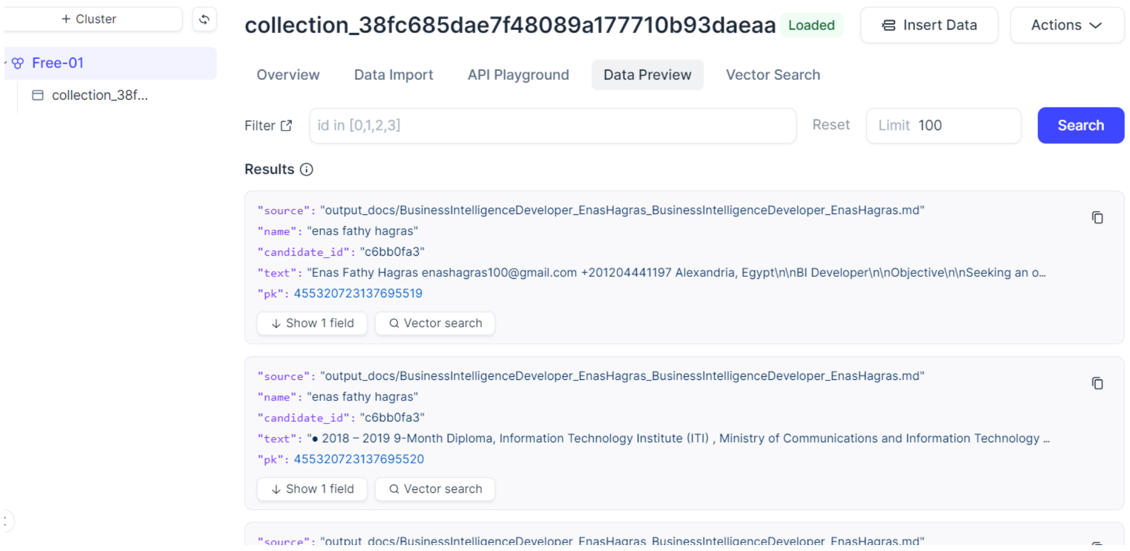
The Retrieval-Augmented Generation (RAG) Pipeline is the core of the Talent Acquisition Hub, enabling the chatbot to generate intelligent, context-aware responses by combining vector-based document retrieval with advanced language models. It consists of these core classes below:

1. **RetrieverManager:** responsible for managing and retrieving relevant information from the Milvus vector database. It connects to the database using specified URI and token credentials For each user query, the RetrieverManager fetches document chunks that match the query context, ensuring accurate and relevant information retrieval.

2. **ModelManager:** powers the language generation capabilities of the pipeline. It utilizes the meta-llama/Llama-3.2-3B-Instruct model. The ModelManager integrates with a Hugging Face text-generation pipeline, enabling dynamic language modeling. To enhance retrieval quality, the pipeline incorporates LLMListwiseRerank, a GPT-4-powered contextual compression retriever. This ranks and filters retrieved document chunks, extracting the top six most relevant results.

3. **Chatbot:** orchestrates the retrieval and generation processes. It uses a ConversationBufferWindowMemory to maintain conversational history, enabling continuity and context in multi-turn dialogues. A dynamic prompt shown below is constructed by combining chat history, retrieved context, and the user's query, which is then processed through the text-generation pipeline to produce detailed and professional responses tailored to the recruiter's needs.

   [language=] You are an intelligent talent acquisition assistant chatbot. Your primary role is to assist recruiters by analyzing candidates resumes, understanding their qual-



```
"source": "output_docs/BusinessIntelligenceDeveloper_EnasHagras_BusinessIntelligenceDeveloper_EnasHagras.md"
"name": "enas fathy hagras"
"candidate_id": "c6bb0fa3"
"text": "Enas Fathy Hagras enashagras100@gmail.com +201204441197 Alexandria, Egypt\n\nBI Developer\n\nObjective\n\nSeeking an o...
"pk": 455320723137695519

"source": "output_docs/BusinessIntelligenceDeveloper_EnasHagras_BusinessIntelligenceDeveloper_EnasHagras.md"
"name": "enas fathy hagras"
"candidate_id": "c6bb0fa3"
"text": "• 2018 – 2019 9-Month Diploma, Information Technology Institute (ITI) , Ministry of Communications and Information Technology ...
"pk": 455320723137695520

"source": "output_docs/BusinessIntelligenceDeveloper_EnasHagras_BusinessIntelligenceDeveloper_EnasHagras.md"
```

Figure 4: Documents stored in a milvus

ifications, and answering questions about their suitability for specific roles. Provide detailed, professional, and context-aware responses. this is the previous question and answer history if needed: history Relevant Candidates Information: context Recruiter's Question: question

5 Future Work

1. **Marker-Pdf Chunk Parsing:** accelerates large-scale PDF parsing by distributing documents across multiple GPUs.

2. **Context Aware Retrieval:** The current RAG pipeline struggles with maintaining context in conversational settings. For example, a follow-up query like "Tell me more about his technical experience" may fail to retrieve relevant documents if the candidate's name was only mentioned earlier. Query reconstruction solves this by using recent chat history and the user's query to create a more context-aware prompt, such as "Tell me more about the technical experience of Mohamed Maher, based on his CV." This ensures the retriever maintains continuity and delivers precise, context-relevant responses.

3. **Candidates Summaries:** The current RAG pipeline handles specific candidate queries well but struggles with comprehensive summaries because it operates on chunked document embeddings, making it difficult to aggregate information into a cohesive profile. Its complexity increases significantly when summarizing information for multiple candidates. The pipeline lacks the ability to consolidate diverse data points into unified overviews,. To address this, the Django backend's Candidate-CV model, storing full CVs linked to candidate names, enables efficient retrieval for

summarization tasks. A logical router directs queries to either the RAG pipeline for specific details (e.g., "What is Mohamed Maher's educational background?") or to a summarization pipeline for holistic profiles (e.g., "Summarize Mohamed Maher's CV for a managerial role.").