Technical Report

Intelligent CV Analysis and Candidate Database System

Tong Nguyen Minh Khang

Table of *contents*

01

Introduction

Challenges, objectives,

Challenge of processing CVs in modern recruitment

- **High Volume of Applications**: Recruiters often receive hundreds or thousands of CVs for a single job posting, making manual screening time-consuming and inefficient.
- **Inconsistent Formats**: CVs come in various layouts, file types, and structures, making automated parsing complicated.
- **Unstructured Data**: Candidate information is typically unstructured, making it hard to extract key details like skills, experience, and education.
- **Human Error & Bias**: Manual screening is prone to oversight and unconscious bias, potentially overlooking qualified candidates.
- **Speed & Efficiency Demands**: Businesses require fast hiring processes, and delays due to manual CV review can slow down recruitment.
-

What Will an CV Analysis and Candidate Database System Solve?

Manual Screening Overload
Automates CV parsing to save time and reduce recruiter workload.
Inconsistent CV Formats
Uses AI (LLM) to interpret and extract data regardless of layout, style, or file type.
Data Fragmentation
Centralizes all candidate information into a searchable, structured database.
Slow Hiring Process
Speeds up shortlisting by instantly extracting and matching candidate skills, experience, and qualifications.

What Will an CV Analysis and Candidate Database System Solve?

Poor Candidate Matching
Enhances job-candidate fit through intelligent keyword and semantic analysis.
Scalability Limitations
Supports processing thousands of CVs without manual effort.
Lack of Insightful Analytics
Enables reporting and analytics on talent pools, diversity, skill gaps, and hiring trends.

02

Solution Architecture

Key Components

CV Text Extraction Module

CV Information Extraction Module

CV Processing Controller

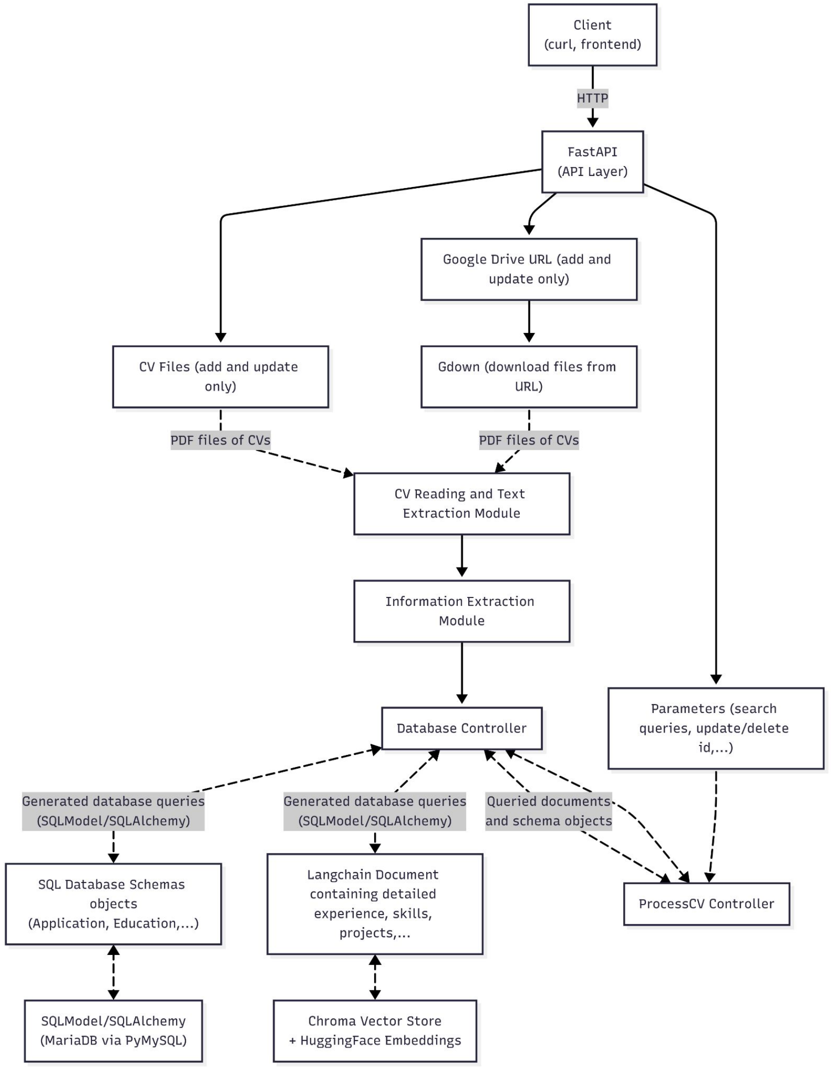
Database Controller

Technologies Used

- **SQL Database:** MariaDB
- **Programming Language:** Python
- **Backend Framework:** FastAPI
- **PDF Processing & NLP:** PyMuPDF
- **AI & LLM Interaction:**
  - Langchain
  - **Vector database:** ChromaDB
  - **LLM:** OpenAI API

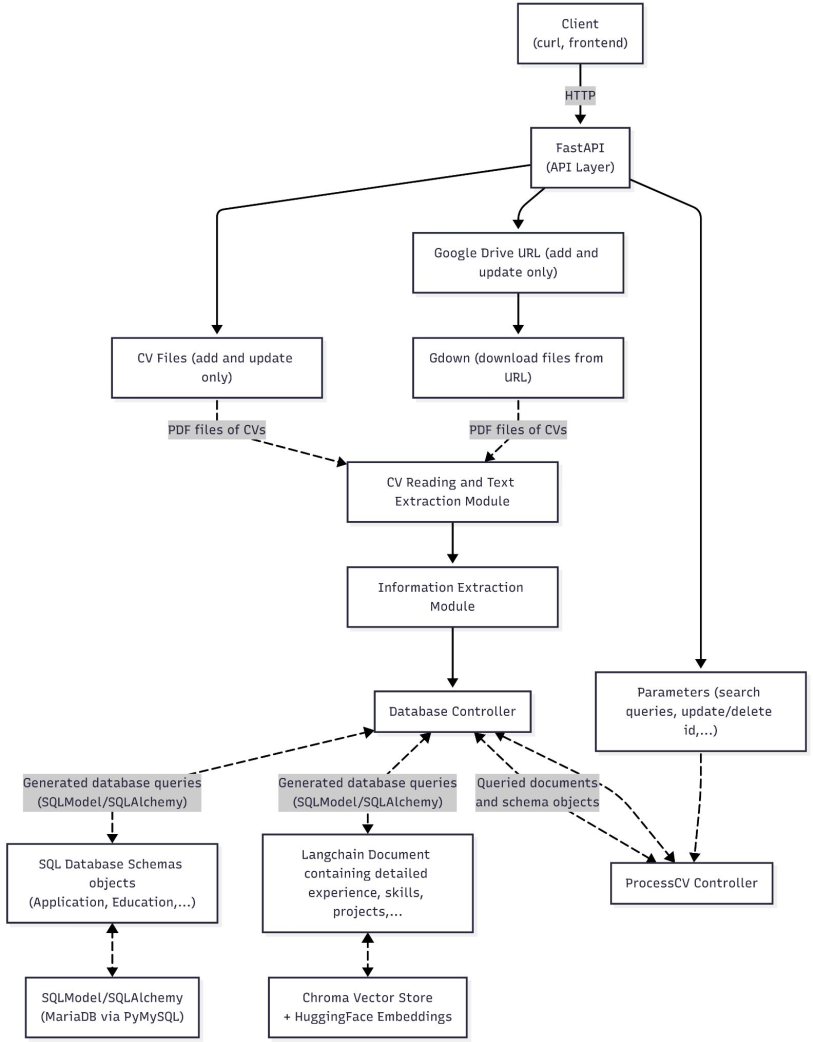
Architecture Diagram

- The solution is an API to process CV files, extract information and store candidate information in the databases.
- The solution uses REST APIs to provide way to upload CV files and integrate with other applications more easily.
- ProcessCV Controller handling requests, and retrieve documents from the database controller, and building the response.
- Database Controller: Building queries (for adding CVs, updating CV files, search CVs, get CVs, delete CV,...)and handling storing, retrieval, updating and deleting corresponding schema objects in the SQL database and documents in vector database.


Client (curl, frontend)
HTTP
FastAPI (API Layer)
Google Drive URL (add and update only)
CV Files (add and update only)
Gdown (download files from URL)
PDF files of CVs
PDF files of CVs
CV Reading and Text Extraction Module
Information Extraction Module
Database Controller
Parameters (search queries, update/delete id,...)
Generated database queries (SQLModel/SQLAlchemy)
Generated database queries (SQLModel/SQLAlchemy)
Queried documents and schema objects
SQL Database Schemas objects (Application, Education,...)
Langchain Document containing detailed experience, skills, projects,...
ProcessCV Controller
SQLModel/SQLAlchemy (MariaDB via PyMySQL)
Chroma Vector Store + HuggingFace Embeddings

Architecture Diagram

- CV Reading and Text Extraction module will fetch and download CV files from URLs and files from client and extract the texts to the Information Extraction Module.
- Information Extraction Module will extract the texts in the CV files and return structured data of each application's CV file to be stored in the databases.
-



Client (curl, frontend)
HTTP
FastAPI (API Layer)
Google Drive URL (add and update only)
CV Files (add and update only)
Gdown (download files from URL)
PDF files of CVs
PDF files of CVs
CV Reading and Text Extraction Module
Information Extraction Module
Database Controller
Parameters (search queries, update/delete id,...)
Generated database queries (SQLModel/SQLAlchemy)
Generated database queries (SQLModel/SQLAlchemy)
Queried documents and schema objects
SQL Database Schemas objects (Application, Education,...)
Langchain Document containing detailed experience, skills, projects,...
ProcessCV Controller
SQLModel/SQLAlchemy (MariaDB via PyMySQL)
Chroma Vector Store + HuggingFace Embeddings

03

Components

CV Text Extraction And Ingestion Module

GDriveDownload.py

- Fetches public Google Drive files/folders via gdown
- Filters to valid CV types (.pdf, .docx, .odt).
- Download the CV files in

File Processors (inherits from ICVFileProcessor.py)

- Includes file processor for .pdf, .docx and .odt files to parse and get text for each CV file type.
- Load pages into LangChain “Document” objects.

CVProcessor.py

- The CVProcessor class orchestrates download or local directory scan and extract text from the downloaded CV files.
- Routes each file to its processor (for each file type) and return
- Merges page contents of multiple documents corresponding to each CV file into one “Document” object per CV.

Information Extraction Module

ParseCVFiles Chain

Uses LangChain's OpenAI's custom integration with a custom prompt which instruct GPT-4.1-mini to parse each CV into a XML-like object.

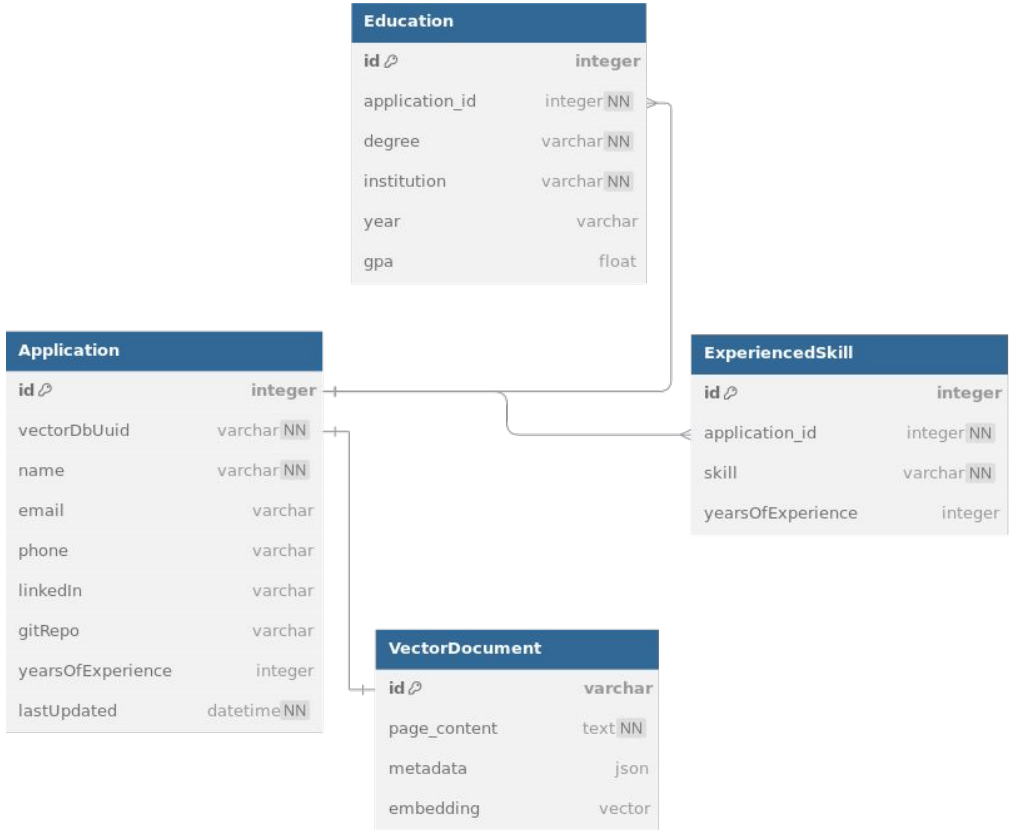
Each field wrapped in an XML tag, and each CV file wrapped between <CV> and </CV> object.

Parsing each CV

Regex-based parser that converts each CV information returned by the LLM into a "ParsedCV" object which holds name, contact info, education, work experiences, projects, skills, years of experience and other information.

Return the list of parsed CV each in an ParsedCV object conta

Database Schemas



Education	
id	integer
application_id	integer NN
degree	varchar NN
institution	varchar NN
year	varchar
gpa	float

Application	
id	integer
vectorDbUuid	varchar NN
name	varchar NN
email	varchar
phone	varchar
linkedIn	varchar
gitRepo	varchar
yearsOfExperience	integer
lastUpdated	datetime NN

ExperiencedSkill	
id	integer
application_id	integer NN
skill	varchar NN
yearsOfExperience	integer

VectorDocument	
id	varchar
page_content	text NN
metadata	json
embedding	vector

VectorDocument schema is indicating the document in the vector database, other schemas are in SQL database

Database Schemas

SQL Database Schema:

- Including Application schema (to store personal information in the CV, LinkedIn URL, git repository URL (Github, GitLab,...), total work years of experience (yearsOfExperience).
- Education schema indicates an educational institution in the application with application ID application_id, containings institution name, degree, duration (year column), and GPA.
- ExperiencedSkill table contains each skills with significant experience (work projects, other significant projects in the CV) with corresponding years of experience.
- Both ExperiencedSkill and Education have delete cascade relation to the Application schema.

Vector Database Schema:

- There will be one document associated to each application, containing address, skills (those without specified years of experience, work experience details, project details,...).

Database Controller

- The DatabaseController class will process and add information in the CV to the databases (SQL and vector databases).
- This controller currently supports the following methods
    - **processParsedCV:** Return the schema objects containing the application personal info, education and skills with experiences (stored in the SQL database) and the document containing the rest of the the CV to be stored in the vector database from the ParsedCV object from the information extraction module.
    - **addApplication:** First, use the processParsedCV to get an Application schema object and Document object from the parsed CV object; add the schema object (with Education and ExperiencedSkill objects associated) to the SQL database (MariaDB), and the Document object to the vector database.

Database Controller

- Other methods:
  - **updateApplication:** First, use the processParsedCV to get an Application schema object and Document object from the parsed CV object.
    - Find from the SQL database if there is an application with the updating id (if not return 404 error).
    - Update the schema object (with Education and ExperiencedSkill objects associated) to the SQL database (MariaDB), and the Document object to the vector database (by first deleting the old document object in the vector database using the old vector database UUID in the old document, then adding the updated document with the new UUID (of the new Application document's generated vector database document UUID).

Database Controller

- Other methods:
    - **getApplication:** Parameter is the id to get the details of the application (CV information), fetching the application with that id in the Application schema in the database, if a record exists, fetch the corresponding Document (using the vectorDbUuid column in Application schema which correspond to the document in the vector database), and return both of them to the client.
    - **searchApplications:** Parameters is defined in the SearchCVQuery object (details of possible values and structure is in the API documentation), for personal information (name, email, phone,...) and skills with experience, it will build a query based on the values present to search in the SQL database; for skills without specified experiences, address, work experience details, it will search in the vector database, fetch the ids of the documents and finally the controller will filter based on the ids of the vector database documents.

Database Controller

- Other methods:
  - **deleteApplication:** Parameters is the id to delete, during deletion first the controller will find if there is an application with that id. If there is none, return 404 error that no application with that id is in the database, else delete the document in vector database, then delete the Application schema object (which will also delete all Education and ExperiencedSkill objects associated with the candidate Application row as it has delete cascade relation in the database).
  - **getApplications** (get application by page): Fetching applications paginated and sorted. There are three parameters, page (start from 1), size (determine the page size and maximum return number of applications), and orderBy. After selecting the Application schema objects in the SQL database, each application will have the corresponding document in the vector database fetched and both will be returned to the client.

Process CV Controller

- This is the controller to handle each API point (searching, updating and uploading CV files,…).
- **Adding/Updating CV application:** First retrieve the files from client (sanitize file names for cases of getting CV files from API) or fetch Google Drive files if it's Google Drive URLs to a specified directory in _generateDownloadFolder private method, fetch these files (the above are done using the CVProcessor class in CV preprocessing module, and then parse them using parseCVs function in extracting CV information module.
    - For adding: use addApplication method in DatabaseController, get the returned ids of the added documents, return it the user or else return the error from the database controller.
    - For updating: Get the id of the document to update, use updateApplication method

Process CV Controller

- **Deleting Document, Search Document, Get Document by Id:** Calling the corresponding methods in the database controller and passing the correct parameters.
- **Get Documents By Page (getApplications method):** Fetching applications paginated and sorted. There are three parameters, page (start from 1), size (determine the page size and maximum return number of applications), and orderBy, which supports one of these following parameter values (default is sorting by last updated time descending):
    - "name": sort by names of the application ascending in the dictionary order.
    - "Id": sort by the id in the database.
    - "nameDesc": like "name", but sorting descending in the dictionary order.
    - "lastUpdated": sort applications by last updated time ascending.

API Overview

POST /cv/upload: Upload new CV(s) (files or Google Drive URL)

PUT /cv/update/{id}: Update existing CV by application ID

GET /cv: List all CVs (supports `page` & `size` query params)

GET /cv/{id}: Retrieve a single CV's details by ID

DELETE /cv/{id}: Delete a CV record by ID

POST /cv/search: Search CVs using structured filters + semantic keywords

Details of parameters, return results,... of these APIs is in the API documentation, or you can try in **/docs** after running the server.

04

Documentation, Reporting and Future Enhancements

Overview of Documentation & Reports

Comprehensive documentation are in “/documentation” in the project directory, containing these files:

- API Documentation: Documenting API routes and points, parameters, response, example responses and requests, error responses, and supported inputs.
- Schema Documentation: Documenting schemas and structure in the SQL and vector databases used, details of each field, data flow, database controllers, and file location for each schema and controller source files.
- Technical Report Presentation (this presentation): Containing the overall architecture, how each component works and interact, technologies used, and future planned enhancements.
- Daily Journal and Project Executions: Documenting the progress and what has been done, as well designs choice, challenges/issues (and solutions), and next steps,...

Future Enhancement:

- Add a frontend UI to the application for easier uses.
- Learn about Vertex AI for deploying pipelines, managing models,... (I couldn't because I couldn't register a Google Cloud account).
- Increasing API response speed (especially in the CV parsing, as getting response from the LLM takes a very long time).
- Integrating with Google Drive APIs for better integration with Google Drive.

Thanks for Watching!