Task Description

The task is to build a system that can extract names and last names from a scanned PDF document, identify their bounding box locations, and provide an API endpoint to perform fuzzy matching between the extracted names and the names provided in the request.

Create a Branch and make a pull request to this repository.

Requirements

1. Use an open-source OCR library to extract text from the scanned PDF document.
2. Implement a named entity recognition (NER) or a large language model (LLM) approach to identify and extract names and last names from the extracted text.
3. Parse the extracted text and OCR results to find the bounding box coordinates of the identified names and last names.
4. Create a RESTful API using FastAPI or Flask that accepts a scanned PDF file and a set of name-last name pairs as input.
5. Perform fuzzy matching (with a similarity threshold of 90%) between the extracted names and the provided names in the API request.
6. Return the extracted names, their bounding box coordinates, and the fuzzy matching results as a JSON response.
7. Implement a vector database (e.g., Chroma, Qdrant) to store the extracted text from the PDF documents.
8. Use an open-source embedding model (e.g., Sentence-BERT, Universal Sentence Encoder) to convert the extracted text into vector representations and store them in the vector database.
9. Create another RESTful API endpoint that accepts a question as input and returns an answer based on the RAG strategy.
10. Create a Docker container image for the entire application, including the text extraction, named entity recognition, bounding box parsing, vector database, and RAG components. Ensure that the container can be built and run using Docker commands or a Docker Compose file. Include instructions for running the containerized application and accessing the API endpoints.

Test Cases

A Driver's License and a Contract sample will be given with this technical assessment, and both documents will be used mainly to test different things in your submission:

- The Driver's License will be used to test the fuzzy search and bounding box implementation.
- The contract sample will be used to test the RAG implementation.

However, we might use the same documents to see how different parts of your submission behaves when dealing with single and multi-page PDF documents.

Take this into account when developing your solution.

Test Steps

Set up the Development Environment

- Create a new Python virtual environment and install the required libraries (e.g., PyMuPDF, Tesseract OCR, spaCy, Hugging Face Transformers, FastAPI or Flask).
- Obtain a sample scanned PDF document containing names and last names for testing purposes.

Implement Text Extraction from PDF

- Use PyMuPDF or a similar library to extract text from the scanned PDF document.
- Optionally, you can preprocess the extracted text (e.g., remove newlines, clean up formatting) for better NER or LLM performance.

Implement Named Entity Recognition (NER) or LLM Approach

- Use spaCy's pre-trained NER model or a custom NER model trained on a relevant dataset to identify names and last names in the extracted text.
- Alternatively, use a pre-trained NER (e.g., BERT, RoBERTa) from the Hugging Face Transformers library for the named entity recognition task. It is easier to use a Generative LLM for zero-shot NER and return the response as an usable JSON.

Parse Bounding Box Coordinates

- Integrate the OCR library (e.g., Tesseract OCR) to obtain the bounding box coordinates of the recognized words in the PDF document.
- Match the extracted names and last names with their corresponding bounding box coordinates from the OCR results.

Implement the RESTful API

- Use FastAPI or Flask to create a RESTful API endpoint that accepts a scanned PDF file and a set of name-last name pairs as input.
- Perform the text extraction, named entity recognition, and bounding box parsing steps on the uploaded PDF file.
- Implement fuzzy matching (e.g., using the python-fuzzy library) between the extracted names and the provided names in the API request, with a similarity threshold of 90%.
- Return the extracted names, their bounding box coordinates, and the fuzzy matching results as a JSON response.

Set up a Vector Database

- Choose and install an open-source vector database solution (e.g., Chroma, Qdrant, Milvus) compatible with your development environment. If you are using a docker image of an open-source database, make sure to include it in the docker compose file.
- Create a new database or collection within the vector database to store the extracted text from the PDF documents.

Implement Text Embedding

- Use an open-source embedding model like Sentence-BERT or Universal Sentence Encoder to convert the extracted text into vector representations.
- Integrate the embedding model with the vector database to store the text embeddings along with their corresponding text snippets.

Implement Retrieval-Augmented Generation (RAG)

- Create a new RESTful API endpoint that accepts a question as input.
- Implement the RAG strategy to retrieve relevant text snippets from the vector database based on the similarity between the question embedding and the stored text embeddings.
- Use a pre-trained language model (e.g., GPT-3.5, Gemini) to generate an answer based on the retrieved text snippets and the input question.
- Return the generated answer as the response to the API request.

Containerization

- Create a new Dockerfile in the project directory with all the required dependencies to run the API server.
- If required, generate a docker-compose file.

Documentation and Submission

- Document the code, and any external libraries or resources used.
- Provide instructions for setting up and running the system, as well as any additional dependencies or requirements.