**INDUSTRIAL AI INTERVIEW**

OPTICAL CHARACTER RECOGNITION –

I initially attempted to directly extract text from the PDF. However, due to the presence of images, this approach proved ineffective. To overcome this, I used the PyMuPDF library to convert the PDF pages into images, followed by basic preprocessing techniques to enhance the image quality.

For text extraction from these images, I first experimented with Tesseract OCR. However, I realized that using regular expressions to extract relevant text from extracted text of OCR would be difficult since the invoice formats would vary across companies. Given the importance of accurately extracting items from the images, I opted to utilize a multimodal LLM (Gemini-1.5-Flash) model to perform the text extraction. The structured nature of customer request PDFs simplifies the extraction process for the model. To further improve accuracy, I provided an example within the prompt to guide the LLM in understanding the specific extraction requirements.

An alternative approach could involve combining OCR with the capabilities of an LLM. We could first use Tesseract OCR to extract text from the images and then leverage the LLM to clean up the extracted text and accurately identify and extract the items requested by the user. This would save the cost since we could use open source LLMs like Llama-3.

MAPPING ENGINE –

For mapping each line item to a product in the product database, I used embeddings generated by two types of models: a bi-encoder and a cross-encoder. Bi-encoders are good at finding similarity between two sentences. We encode two sentences into two vectors and then compute similarity between vectors. Bi encoders are faster and are suited for semantic search tasks. On the other hand, cross-encoder encode the two sentences simultaneously. Cross-encoder embeddings are dependent on each other, and hence cross encoders are better suited for ranking the results from bi-encoder. In my approach, I used the bi-encoder model to identify the top 20 most similar products, then applied the cross-encoder model to rank these products based on the similarity between the extracted products and the requested item. Finally, I selected the top 10 products according to this ranking.

I utilized open-source models from the METB leaderboard for embedding generation. The models I chose are small, fast, and offer a decent level of accuracy, allowing them to run efficiently without the need for dedicated hardware.

For storing the embeddings from the product database, I used the FAISS vector database. I selected flat indexing to retrieve relevant products based on the requested item embeddings, as this method offers higher accuracy compared to other indexing options.

STREAMLIT FRONT-END –

A screenshot of a computer

Description automatically generated

I designed a simple front-end using Streamlit, providing an intuitive interface for users to upload PDFs. The application processes the uploaded PDF by first converting it into images. These images are then sent to the Gemini model, which extracts text from them. Once the text is retrieved, it is passed to a bi-encoder model that generates the top 20 product recommendations based on similarity.

These 20 recommendations are subsequently refined by a cross-encoder model, which ranks them to identify the top 10 most relevant products. Users can then download a file containing these top 10 recommendations directly from the Streamlit application.