Amphora Document AI Assignment

Problem Statement:

This assignment involves two main tasks: content extraction and entity extraction from multi-page PDF documents. For content extraction, an OOPS-based data model needs to be developed to store the extracted content. This involves identifying bounding boxes and the associated text, and effectively managing document entities. The data model should include properties, extraction functions, and utility functions to handle these requirements. For entity extraction, the task focuses on accurately identifying and storing specific entities, namely the shipper and their complete address, and the ports of loading and discharge/destination.

Instructions to run the code:

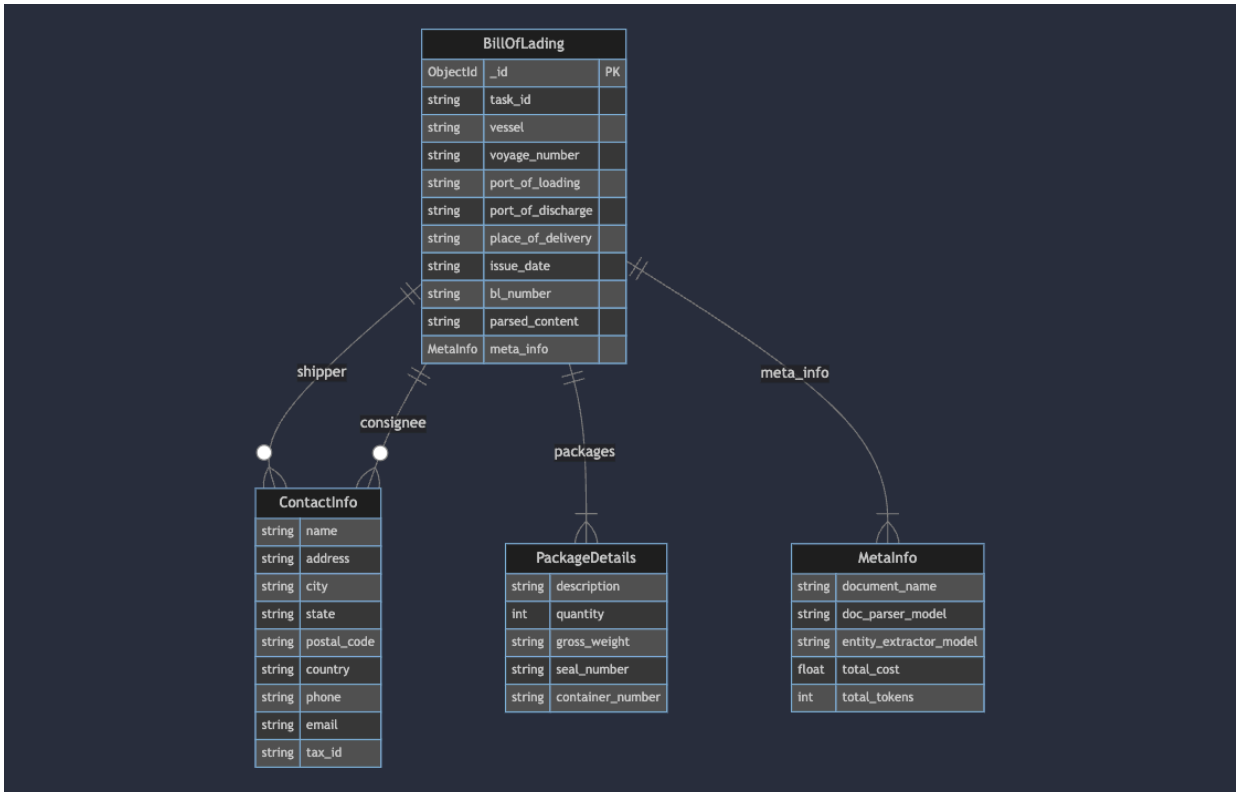
Please refer to the README.MD file which is present in the code for instructions on how to run the code and try out the tool yourself.

Database and database schema for storing the extracted entities:

A MongoDB database was chosen to store the extracted content and entities due to its flexibility and scalability. The key reasons include:

- **Schema Flexibility:** MongoDB's dynamic schema accommodates diverse and varying document structures, allowing easy adjustments.
- **Document-Oriented Storage:** MongoDB's structure suits the task of storing entities like shipper details, addresses, and ports as separate documents within collections.
- **Scalability:** MongoDB's horizontal scalability handles growing data volumes without performance issues.
- **Indexing and Querying:** MongoDB offers powerful indexing and querying capabilities for efficient retrieval of specific entities and their attributes.

The database schema used for this assignment is as follows,



Solutions:

LLM Based Solutions:

The solution was built as a fastapi API so that it can easily be scaled and also deployed. This fastapi endpoint allows the user to upload a document and the user can choose from any one of the below document parsers and LLMs to extract and store the entities in a mongodb database.

Document parsers,

- PyMuPDF
- Llama Parse

LLMs,

- GPT-4o
- GPT-4o-Mini
- Llama 3.1 (70B model from groq)

Various Document parsers advantages and disadvantages:

PyMuPDF:

- PyMuPDF is a python library which reads data from pdf and gives the output as a text.
- It is not able to parse images, and it parses the documents line by line which leads to it missing the relation of the content.
- The output of this is not very good as often times tables and the relation of the content in the pdf is very important to generate accurate results.
- It is free and very quick and for simple documents works extremely well.

Llama Parse:

- Llama Parse is an LLM based parser which uses an LLM and OCR to parse and generate the text outputs from documents.
- It is able to understand the relations between content much better than pymupdf, hence it is much more accurate.
- It has an OCR using which it can read images, tables and more complex data from pdfs.
- It is free for upto 1000 documents a day, and costs $3 / 1000 pages after that.

Summary of the results:

The json outputs for all the models and for various parsers is stored in the outputs folder in the zip file containing the code of the project. The below are some of the key highlights of this approach,

- When using pymupdf and using all the LLMs the output was accurate for some of the documents, but for others when more additional information is present in the document all the models struggled. Llama 3.1 and GPT-4o-mini suffered the most and the outputs were not correct in some instances but using GPT-4o the results were much better.
- When using LlamaParse since the documents extracted were much cleaner all the models performed pretty accurately. Even Llama 3.1 and GPT-4o-mini were enough to get good results.

Cost Considerations:

Model	Cost
GPT-4o	$15.00 / 1M output tokens, $7.50 / 1M output tokens
GPT-4o-mini	$0.600 / 1M output tokens, $0.300 / 1M output tokens
Llama 3.1 70B	Can be locally hosted, $0.59 / 1M output tokens, $0.79 / 1M output tokens
Llama Parse	free for upto 1000 documents a day, and costs $3 / 1000 pages after that

Other Experiments to try:

- Explore various other smaller models which can be locally hosted, and also finetune an open source model specifically for entity extraction task.
- Experimenting further to improve the prompt for extracting the entities more accurately.

Non-LLM Based Solutions:

I tried out transformer based pretrained models (Flan T5) from huggingface and Google Document AI pretrained models.

Summary of the results:

- The outputs from the pretrained Flan T5 model was decent but it was not able to extract the relevant entities accurately and tag them to the right entity name.
- Similarly the output from Google Document AI pretrained invoice parser was also struggling to identify all the entities accurately.
- The performance of both of these models can be largely improved by fine-tuning them on a large collection of documents, but due to unavailability of data was unable to try this experiment.

The results of the experiments and the code for these experiments can be seen in the experiments folder of the code which is shared as zip file.