Overview

The system is designed to process a folder of identity document images, extract key information (such as name, date of birth, address, and document-specific numbers), and return structured data.

Approach 1:

This approach involves a modular pipeline that processes identity document images by first reading and encoding them into base64, then using file name heuristics to determine whether each document is a passport or license. Based on this classification, the system dynamically constructs tailored prompts and Pydantic models to extract the relevant fields via a vision model.(deepseek v3)

Design Choices:

Functions for better readability:

The code is split into clearly defined functions:

encode_image*:* Encodes an image file to a base64 string. APIs and JSON payloads typically handle text, not raw binary. It also allows you to embed the image directly in the request (e.g., in a "data:" URL), simplifying the data flow.

extract_document_type_from_filename*:* Determines the document type (passport or license) based on file name heuristics.

extract_identity_info: Dynamically builds a Pydantic model and constructs a prompt based on the document type, then calls the vision AI model.

process_identity_documents**:** Iterates over files in a given folder, processes each image, and aggregates the results.

Constraints/ Trade-offs:

1. Misnamed files or documents that do not include expected keywords can lead to incorrect processing or fallback to a generic model.
2. A rigid pydantic prompt structure may fail to capture variations in document layout or content.While Pydantic models enforce data structure, the system assumes that the AI response will always conform to these models.
3. The quality of the output would depend on the underlying vision model.
4. The end-to-end process is designed for relatively small sets of images which may impede performance for larger datasets.
5. If the orientation of the document or the lighting is malformed then the vision model struggles to extract the necessary information. This may require some additional preprocessing steps to ensure better extraction.
6. Currently, we don’t have a system to introduce new document types. Future modifications with new document types would be tedious.

Future ideas:

1. We can use asynchronous processing (e.g., via Python's asyncio) to handle multiple tasks concurrently. For example, you could initiate several image reads and API calls simultaneously.
2. We can also enforce uniform orientation on documents such that content appears consistent across all images.

3. We can also use LangChain graph that can provide a structured, modular, and visual representation of the processing pipeline. In this system, we already branch based on the document type (passport vs. license). With a graph, these branches become explicit nodes, making it easier to introduce new document types or additional validation steps. LangChain Graph makes it straightforward to insert, remove, or modify nodes in your pipeline.

Approach 2:

This system is designed to extract structured data from identity document images by using LangChain Graph to model the process as a series of nodes (e.g., classification, extraction), providing clear modularity and routing.There are 3 Agents working.

1. Classification Agent : Classifies the document into 'license' or 'passport' based on the image.
2. License Agent : Specialized in extraction of license information with pydantic type validation
3. Passport Agent : Specialized in extraction of passport information with pydantic validation

Design Choices:

encode_image: converts the file into binary data, and then encodes it as a base64 string.

Process_classification: sends the base64 image to the language model with a prompt asking whether the document is a "license" or "passport."

Process_license_extraction: Extracts structured data from license images by sending a tailored prompt (including a JSON schema) to the language model.

Process_passport_extraction: Similar to license extraction, but with a prompt and schema tailored for passport documents.

Route_extraction: Acts as a conditional decision function, routing the state to the appropriate extraction node based on the classified document type.

Using LangGraph's StateGraph, nodes are added, and conditional edges are defined, ensuring that the data flows correctly through the classification and extraction stages.

Constraints/Trade-offs:

1. Rigidity in pydantic structure. Too many information extraction classification fields may yield incorrect results.
2. If the model's response cannot be parsed as JSON, the system returns an error message with raw response content.
3. The code does not process the requests asynchronously, making processing of large batches of documents inefficient.
4. This method also doesn't solve the issues related to misalignment of document pictures.
5. Duplicated logic between extraction functions might lead to maintenance challenges.

Both Approach 1 and Approach 2 provide very similar outputs substantiating the constraint that output is as good as the underlying AI Vision model used. For programmers, not well-versed in LangChain, understanding the code for future modifications could be challenging.