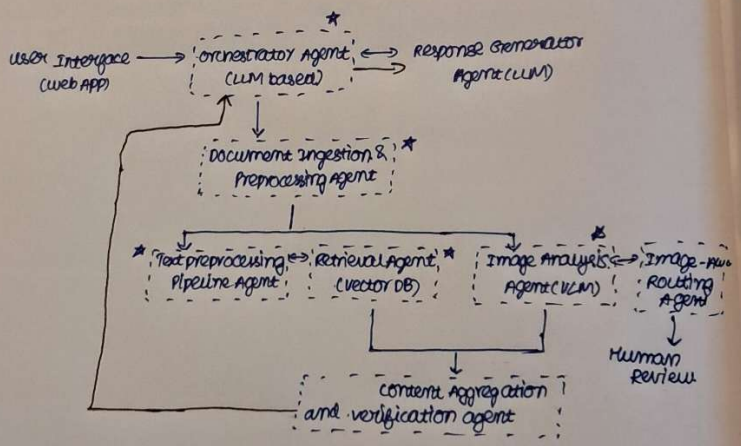
The provided Jupyter Notebooks outline a multi-phase project for building a Retrieval Augmented Generation (RAG) system that can process PDF documents containing both text and images.


User Interface (Web App)
Orchestrator Agent (LLM based)
Response Generator Agent (LLM)
Document Ingestion & Preprocessing Agent
Text preprocessing Pipeline Agent
Retrieval Agent (Vector DB)
Image Analysis Agent (VLM)
Image-Human Routing Agent
Human Review
Content Aggregation and verification agent

The below image states the intended flow of work while creating the multi agent system

Phase 1 (Imago_AI_Assignment_Phase_1_with_Image_Handling.ipynb): This phase focuses on extracting text from PDFs and identifying the presence and location of images.

It defines an EnhancedPyMuPDFLoader to load PDF documents and embed image placeholders (e.g., [IMAGE: img_1_1]) directly into the text content. Metadata about these images (ID, page number, position, format, dimensions) is stored.

An ImageAwareTextSplitter is used to chunk the documents while ensuring that image metadata is correctly associated with the chunks containing the corresponding image placeholders.

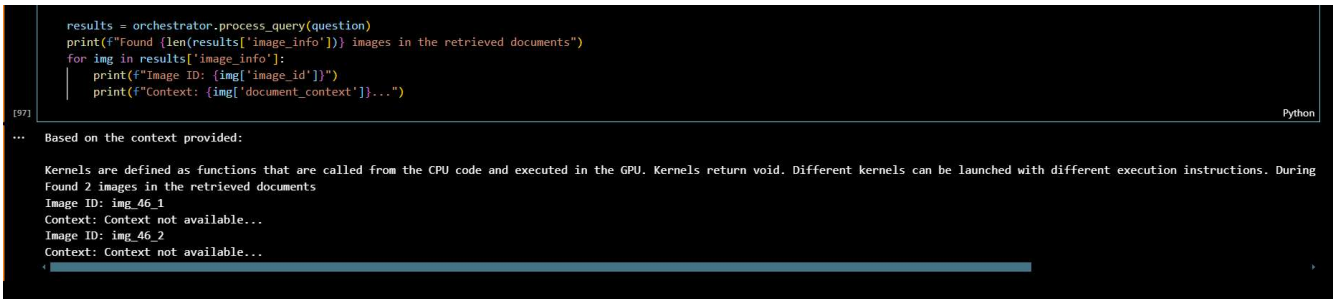
Text chunks are then embedded using a HuggingFace sentence transformer (intfloat/multilingual-e5-large-instruct) and stored in a Pinecone vector store.

A RetrievalAgent fetches relevant text chunks based on a user query.

An ImageAgent processes these retrieved chunks to extract information about any images mentioned.

A SimpleOrchestrator coordinates the retrieval and image identification steps.

Finally, an EnhancedResponseGenerator uses a Google Generative AI model (Gemini) to generate an answer. The prompt to the LLM includes the retrieved text and a summary of identified images and their context.



Phase 2 (Imago_AI_Assignment_Phase_2_with_Advanced_Image_Handling.ipynb): This phase enhances the system by performing OCR on the images to extract textual content from them.

Images are first extracted from the PDF and saved.

OCR (using Tesseract) is performed on these images. The extracted OCR text is then associated with the corresponding document page in its metadata (e.g., metadata['image_data'] = {'image_p1_1': 'ocr text from image 1 on page 1'}).

The document pages (now with OCR data in their metadata) are embedded and stored in Pinecone.

The RetrievalAgent retrieves relevant document pages.

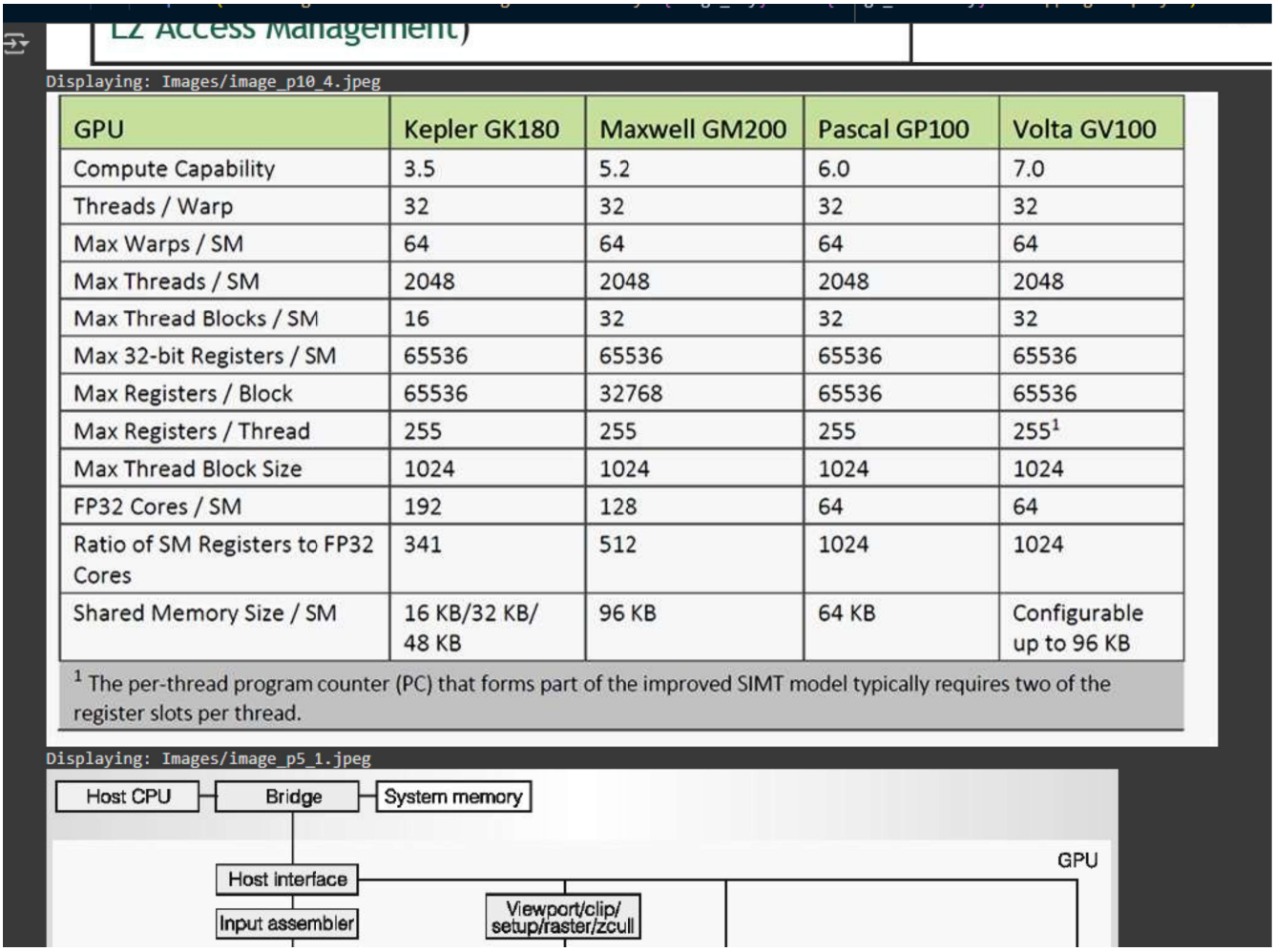
A new ResponseGenerator is introduced. This generator:

Retrieves documents based on the query.

For each retrieved document, it calculates the cosine similarity between the page's text content and the OCR text of images found on that page using the same embedding model.

If the similarity is above a certain threshold, the OCR text from those relevant images is appended to the page's main text content before being passed to the LLM (Gemini 1.5 Pro).

This allows the LLM to directly use the textual content of relevant images when formulating an answer.



The notebook also includes code to display the actual images that were part of the retrieved results.

In summary, the project evolves from identifying image presence (Phase 1) to extracting and utilizing the textual content within images via OCR and relevance filtering (Phase 2) to provide more comprehensive, context-aware answers from PDF documents. Both phases leverage Langchain, Pinecone, HuggingFace embeddings, and Google's Gemini models.

Important Research

During my work on this research project I have found a very useful tool called Llamaparse. It is a GenAI-native document parsing platform developed by LlamaIndex, specifically designed to transform complex, unstructured documents into formats optimized for large language model (LLM) applications such as Retrieval-Augmented Generation (RAG) and AI agents.

There are multiple features inherent in it such as using custom language OCR tools and setting document layout along with context so that the parsing can be very much clear.

This assignment taught me the possible background of such a tool and how it can be built.