The primary objective was to build a robust system capable of extracting, embedding, and retrieving content (text, images, and tables) from PDFs while leveraging state-of-the-art LLMs like Llama 3.1 for retrieval-augmented generation (RAG).

One of the challenges was extracting structured data (like images and tables) from PDF documents. PDF files are notoriously difficult to parse, as their content isn't stored in a simple text format. I used the fitz library (from PyMuPDF) to extract images and tables from PDFs, which is efficient but not always foolproof. Extracting images as base64 strings and then summarizing them with an AI model posed its own challenges, as it required integrating NVIDIA's vision model to interpret and generate metadata from these images. Initially, I used CLIP for image embeddings, but switched to base64-encoded images processed with llama-3.1-vision, which required adapting the metadata and creating a proper format for the Langchain Document schema.

The RAG approach itself was another technical hurdle. Setting up a robust retrieval chain that combines vector store searches with LLM-based responses, like ChatNVIDIA (a model for AI-based conversations), was crucial for allowing users to ask complex queries. One of the key challenges here was ensuring that the retrieval chain would efficiently filter through the embeddings and return concise and relevant information. Therefore, I designed an efficient prompt for the LLM that could handle context and offer concise answers to any query without being overly verbose.

The engine's interface was built using the Flask-based web app that provided a user interface for document uploads, search, and query handling. While Flask is lightweight and easy to use for small applications, handling multiple file uploads (PDFs in this case) and integrating the backend processing with the frontend search features required careful coordination. One of the challenges here was making sure the document uploads were handled smoothly, files were processed in the right order, and users could easily query the document database without confusion. Thus, I allowed the engine to be able to efficiently handle the input/output (I/O) operations and ensured that the process didn't break or slow down with large PDFs were challenges faced during testing. I wanted to ensure that the extraction, embedding, and storage process was as streamlined as possible to handle increasing document sizes.