Notes on Building Production RAG over complex documents

compiled by D. Gueorguiev, 7/14/2025

# Introduction

Large Language Models (LLMs) are revolutionizing how users search for, interact with, and generate new content. Some recent stacks and toolkits around Retrieval-Augmented Generation (RAG) have emerged, enabling users to build applications such as chatbots using LLMs on their private data. However, while setting up naive RAG is straightforward, building production RAG is very challenging, especially as users scale to larger and more complex data sources. A classic example is a large number of PDFs with embedded tables. RAG is only as good as your data, and developers must carefully consider how to parse, ingest, and retrieve their data to successfully build RAG over complex documents. This Notes record the in-depth exploration of this entire process; you will get an overview of the process around building a RAG pipeline that can handle messy, complicated PDF documents. This includes implementing a parsing strategy for parsing a complex document with embedded objects. This consists of an indexing strategy to process these documents beyond simple chunking techniques. We will then explore various advanced retrieval algorithms to handle questions about the tabular and unstructured data and discuss their use cases and tradeoffs.

Knowledge Assistant Use Case:

Human wants to query a Knowledge Base and wants to get back an answer.

The types of question can be asked are various – it could be a question for specific details on specific issue / aspect / concept or it could be a question which can be answered only generally / in very broad terms, or it could be a question which would require detailed analysis and research before providing an answer.

The output could be structured output , short answer to question of the first type or research report for question of the third type.

A computer screen shot of a computer

Description automatically generated

A diagram of a data flow

Description automatically generated

Retrieval-Augmented Generation (RAG) contains of two main components –

* Data Parsing and Ingestion
* Data Querying

Data Parsing and Ingestion can be viewed as ETL applied to your unstructured data.

Being able to parse, process, and index that data is *the first step* toward building a RAG. *The second step* is querying relevant data using appropriate storage system like vector DB or SQL DB and you can treat access to this storage as API interface for the querying purposes. Querying will be done by the LLM. If we have more tools which we can use in order to obtain the final answer we can combine those with the storage facility.

Naïve RAG:

A diagram of a diagram

Description automatically generated

We have a tool which allows us to load some document (e.g. PyPDF) and then we do some sort of splitting from the loaded document – e.g. sentence or paragraph splitting with appropriate chunk size. We can choose also more elaborate splitting scheme such based on NLTK tokens or semantic similarity based splitting.

# References

[1] [Building Production RAG Over Complex Documents, Jerry Liu, LlamaIndex by Databricks, (youtube video) July 7, 2024](https://youtu.be/dI_TmTW9S4c?si=qP6wuZieU2SCZdc3)

[2] [The Big Book on MLOps and LLMOps, Databricks, 10/2023](https://github.com/dimitarpg13/rag_architectures_and_concepts/blob/main/docs/2023-10-EB-Big-Book-of-MLOps-2nd-Edition.pdf)

[3] PyPDF User Guide and API Reference: <https://pypdf.readthedocs.io/en/stable/>

[4] [Chunking(Text Splitting) Strategies- LLMs, Rahul, March 30, 2024, Medium](https://medium.com/@rahulpant.me/chunking-text-splitting-strategies-llms-579ab4ede2eb)

[5] the NLTK suite of libraries: <https://github.com/nltk/nltk>

[6] [LangChain NLTKTextSplitter API Reference](https://python.langchain.com/api_reference/text_splitters/nltk/langchain_text_splitters.nltk.NLTKTextSplitter.html)

[7] [LangChain NLTKTextSplitter Examples](https://lagnchain.readthedocs.io/en/latest/modules/indexes/text_splitters/examples/nltk.html)