The learnings regarding RAG(Retrieval-Augmented Generation)

RAG (Retrieval-Augmented Generation) is a technique in AI and NLP (Natural Language Processing) that combines **information retrieval** and **text generation** to improve the quality and relevance of the generated output.

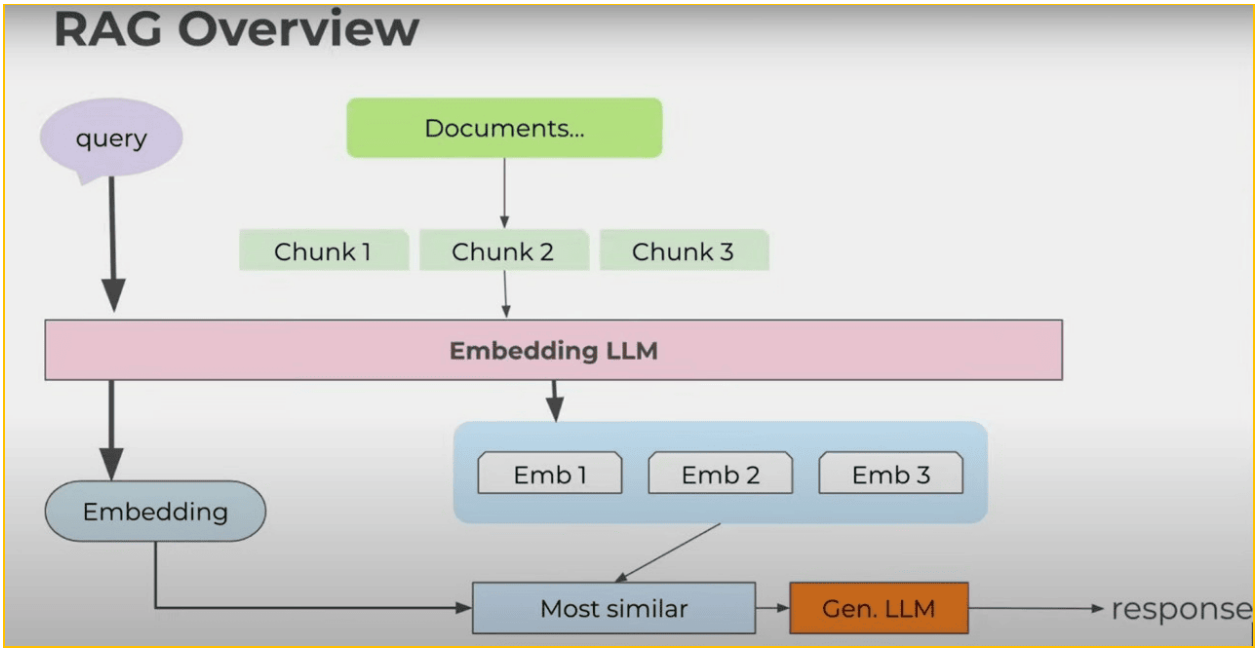


fig1: RAG overview

Steps in RAG (Retrieval-Augmented Generation):

- Data collection
- Data chunking
- Document embeddings
- Handling user queries
- Generating responses with an LLMs

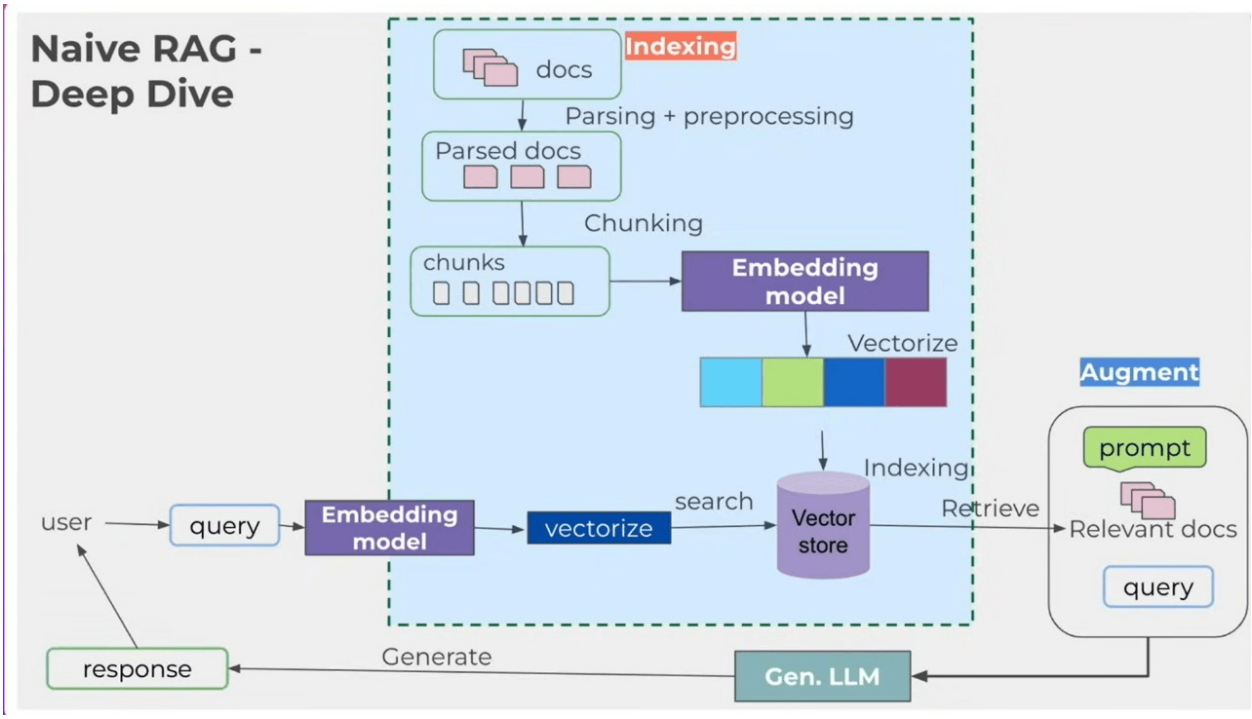


fig2: visualizing the steps

As a group project to complete the entire model of RAG, me and Vishal Pk were given the task of completing the first 2 steps i.e. data collection and data chunking.

We thought of making the RAG model as a medical chatbot which generates a response using gpt 2.0.

Data collection was done by using ChatGPT as we were not able to get the correct type of text for our project. we wanted the dataset to have the name of a specific disease and the discription in a ordered manner.

As we couldn't get that in huggingface , we tried adding the data manually by collecting from different sources which I have attached in references.

Data chunking was the next step. I have created a separate file that explains all the chunking methods and what it actually means in this workplace called chunking docs(1)

the codes which were considered what not used are:

```
from datasets import load_dataset

ds = load_dataset("Somraj77/formatted_ai_medical_chatbot_llama2")

print("Available fields:", ds['train'].column_names)

prompts = []
responses = []

if 'text' in ds['train'].column_names:
    texts = ds['train']['text']

    for i, text in enumerate(texts, 1):

        if "[INST]" in text and "[/INST]" in text:

            prompt = text.split("[/INST]")[0].replace("[INST]", "").strip()
            response = text.split("[/INST]")[1].strip()

            prompt = prompt.replace("<s>", "").replace("</s>", "").strip()
            response = response.replace("<s>", "").replace("</s>", "").strip()

            prompts.append(prompt)
            responses.append(response)
```

```
        else:

            print(f"Record {i} does not have proper markers and was skipped.")
else:
    print("The dataset does not have a 'text' field. Check the available fields and use the correct name.")

print("\nPrompts List:")
print(prompts[:5])
print("\nResponses List:")
print(responses[:5])
```

this was first considered for a specific type of document, but we ended up using another one.

```
from PyPDF2 import PdfReader
import re


pdf_path = 'nihms-1718244.pdf'

reader = PdfReader(pdf_path)

extracted_text = ""
for page in reader.pages:
    extracted_text += page.extract_text()


sentences = re.split(r'(?<=[.!?])\s+', extracted_text)

```

```
chunk_size = 1000
chunks = []
current_chunk = ""

for sentence in sentences:
    if len(current_chunk) + len(sentence) + 1 <= chunk_size:
        current_chunk += sentence + " "
    else:
        chunks.append(current_chunk.strip())
        current_chunk = sentence + " "

if current_chunk:
    chunks.append(current_chunk.strip())

output_path = 'chunked_text_for_RAG.txt'
with open(output_path, 'w', encoding='utf-8') as file:
    for idx, chunk in enumerate(chunks, 1):
        file.write(f"=== Chunk {idx} ===\n{chunk}\n\n")

print(f"Chunks saved to {output_path}")

```

this was not used for the same reasons.

```
chunks = []
with open('disease_dataset.txt', 'r') as file:
    current_chunk = ""
    for line in file:
        if line.strip().startswith("Disease:") and current_chunk:

```

```
            chunks.append(current_chunk.strip())
            current_chunk = line
        else:

            current_chunk += line

    if current_chunk:
        chunks.append(current_chunk.strip())


for i, chunk in enumerate(chunks):
    print(f"Chunk {i+1}:\n{chunk}\n")

```

the code which was used was:

```
def chunk_text_file(file_path, chunk_size):

    chunks = []
    with open(file_path, 'r', encoding='utf-8') as file:
        current_chunk = []
        for line in file:
            if line.strip():
                current_chunk.append(line.strip())
                if len(current_chunk) == chunk_size:
                    chunks.append("\n".join(current_chunk))
                    current_chunk = []

        if current_chunk:
            chunks.append("\n".join(current_chunk))
    return chunks


file_path = "input.txt"
chunk_size = 1
```

```
chunks = chunk_text_file("dataset.txt", chunk_size)



print(chunks)

```

This was used in the main code.

REFERENCES:

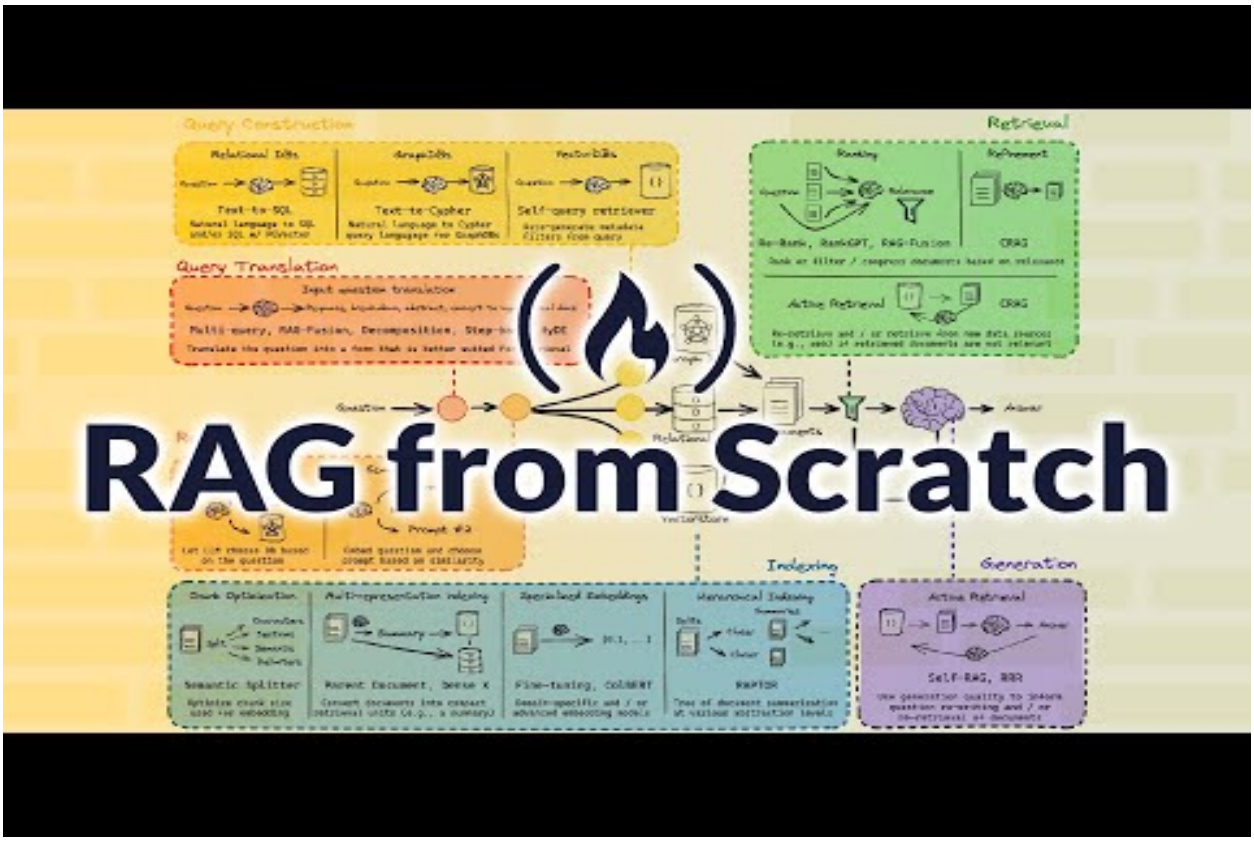
https://www.datacamp.com/blog/what-is-retrieval-augmented-generation-rag

https://writer.zoho.in/writer/open/g0keb0e096a2d800a4201b3c9732aa0d4e785

https://docs.cohere.com/v2/docs/retrieval-augmented-generation-rag

https://rahuld3eora.medium.com/from-basic-to-advanced-rag-every-step-of-the-way-dee3a3a1aae9

https://docs.databricks.com/en/generative-ai/tutorials/ai-cookbook/fundamentals-data-pipeline-steps.html


Query Construction
Retrieval
Text-to-SQL
Text-to-Cypher
Self-query retriever
Query Translation
Ranking
RePrompt
Re-Rank, RankGPT, RAG-Fusion
CRAG
Active Retrieval
CRAG
Question
Answer
RAG from Scratch
Indexing
Generation
Semantic Splitter
Raptor
Self-RAG, RRR


RAG
Fundamentals &
Advanced Techniques
docs
Indexing
Parsing + preprocessing
Parsed docs
Vectorize
model
vectorize
store
Indexing
TRUST-ME I'M A PROGRAMMER