Advanced-RAG-Methodologies

By Farid Ghorbani

Summer 2024 – Northeastern University

This document aims to explore advanced RAG techniques, focusing on their practical applications. The goal is to provide an overview of these methodologies, offering a clear pathway to integrating these technologies into your own projects.

1. What is RAG?

Retrieval-Augmented Generation (RAG) is a sophisticated architecture that blends search capabilities with Large Language Models (LLMs) to enhance the relevance and accuracy of generated responses. RAG is particularly useful in scenarios where the LLM needs to provide detailed, factual, or context-specific responses, such as in chatbots, virtual assistants, and question-answering systems.

Here's a breakdown of how RAG works:

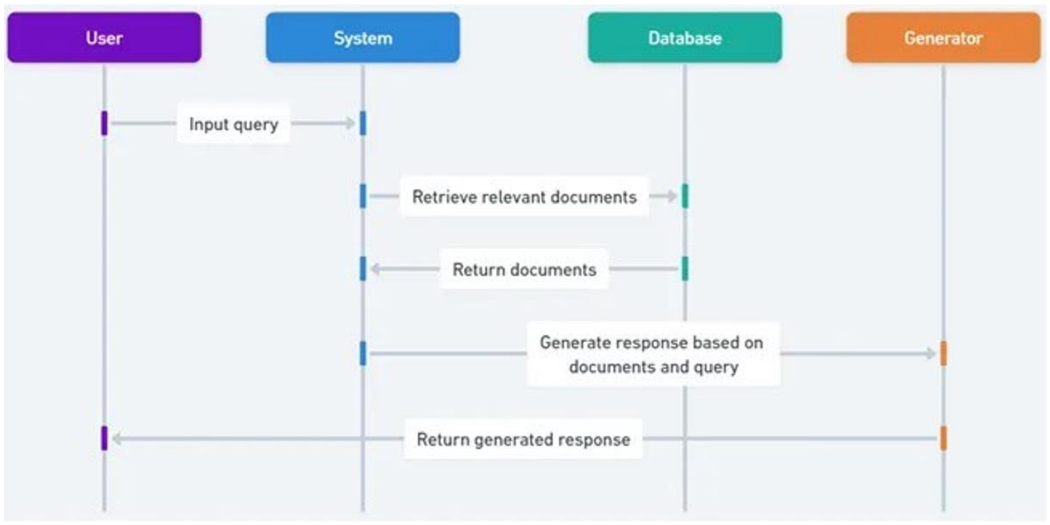
2. Simple RAG Workflow

Our journey begins with the basic RAG structure:

Retrieval: Given a query, the model first retrieves relevant documents or passages from a large corpus. This is typically a vector database where we store our context embeddings. The most relevant documents or passages (usually the top-k) are selected to be used in the next stage.

Augmentation: The retrieved documents or passages are used to augment the input query. This provides additional context and information that the language model can use during the generation phase.

Generation: Finally, the augmented input (query plus retrieved documents) is fed into a generation model, typically a Transformer-based LLM like GPT-3.


User
System
Database
Generator
Input query
Retrieve relevant documents
Return documents
Generate response based on documents and query
Return generated response

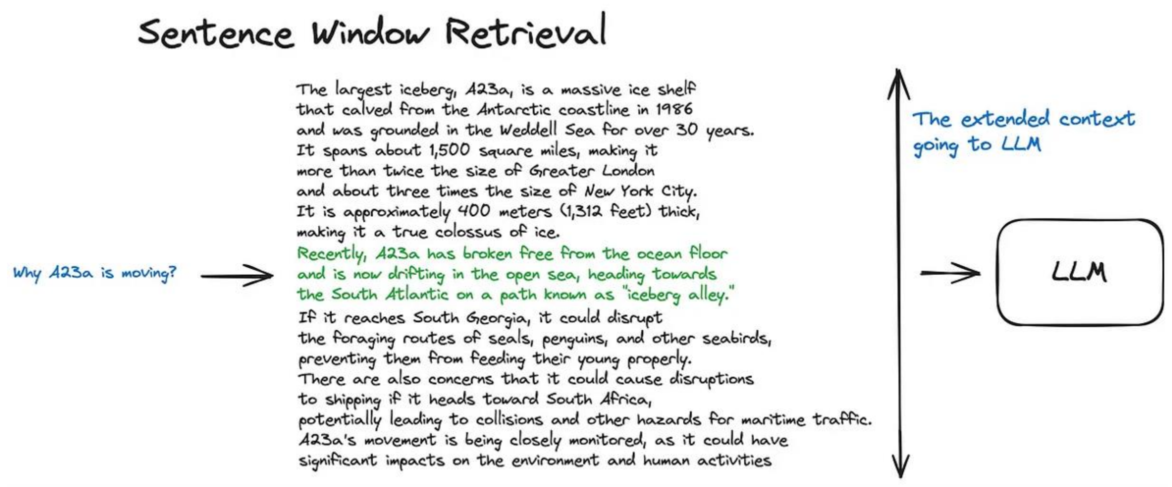
For further information about the simple RAG pipeline, you can find my Simple RAG chatbot repository.

3. Advanced RAG Techniques (Index Retrieval)

Diving into advanced RAG techniques, this section focuses on key index retrieval methods. The essence of the RAG system lies in its ability to store and retrieve vectorized content efficiently.

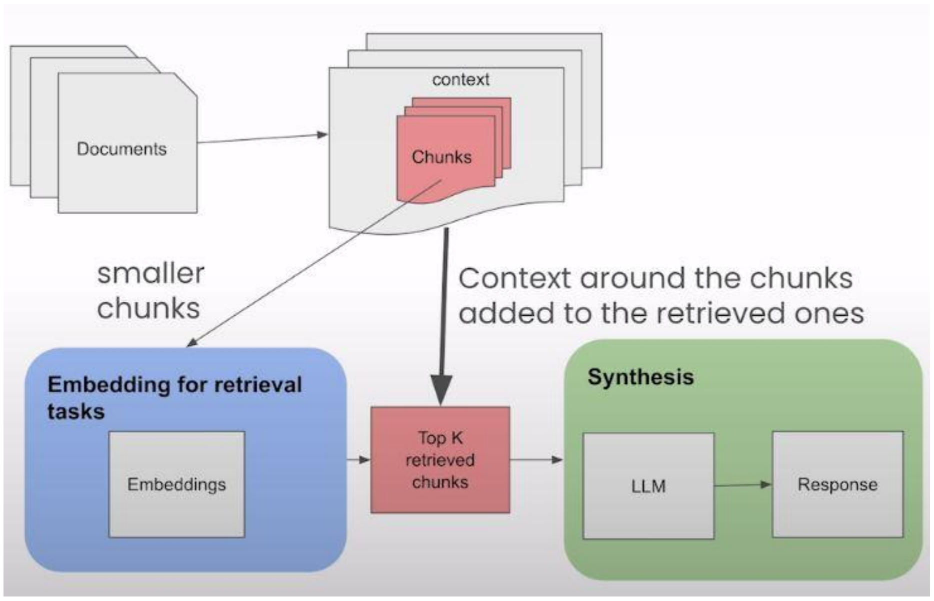
3.1. Sentence Window Retrieval

This technique enhances search precision by embedding individual sentences and extending the search context to include neighboring sentences. This not only improves the relevance of the retrieved data but also provides the LLM with a richer context for generating responses.



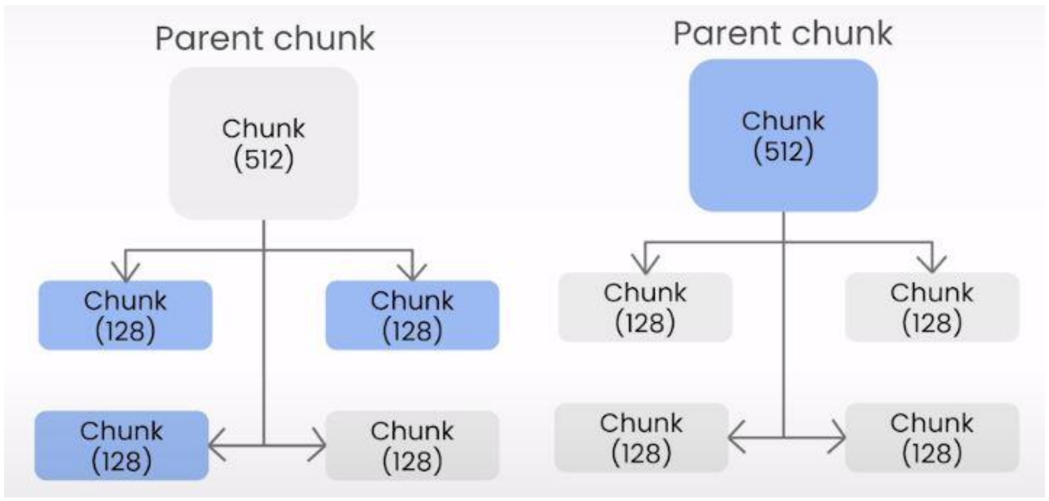
The largest iceberg, A23a, is a massive ice shelf
that calved from the Antarctic coastline in 1986
and was grounded in the Weddell Sea for over 30 years.
It spans about 1,500 square miles, making it
more than twice the size of Greater London
and about three times the size of New York City.
It is approximately 400 meters (1,312 feet) thick,
making it a true colossus of ice.
Recently, A23a has broken free from the ocean floor
and is now drifting in the open sea, heading towards
the South Atlantic on a path known as "iceberg alley."
If it reaches South Georgia, it could disrupt
the foraging routes of seals, penguins, and other seabirds,
preventing them from feeding their young properly.
There are also concerns that it could cause disruptions
to shipping if it heads toward South Africa,
potentially leading to collisions and other hazards for maritime traffic.
A23a's movement is being closely monitored, as it could have
significant impacts on the environment and human activities

To set it up, we first segment the text into smaller sentences and store them in a vector database. We also add the context of the sentences that occur before and after. During retrieval, we retrieve the sentences that are most relevant along with their surrounding context.



3.2. Auto-Merging Retriever

The issue with the naive approach is retrieving many fragmented chunks of context, leading to a disjointed context. Here, we use an auto-merging heuristic to merge smaller chunks into a larger parent chunk, ensuring a more coherent context.



To set it up, first, identify a hierarchy of smaller chunks linked to larger parent chunks. During retrieval, if the number of small chunks linked to a parent chunk exceeds a certain percentage threshold, we merge the smaller chunks into the larger parent chunk. This helps ensure a more coherent context.

In simple terms, documents are segmented into a hierarchy of chunks, and smaller, more relevant pieces are initially retrieved. If multiple small chunks relate to a larger segment, they are merged to form a comprehensive context, which is then presented to the LLM.

These two methods focus on granularity while extending the context more broadly. However, there are many more techniques that you can explore online. Here, we just introduce some popular methods.

4. Evaluation of Different RAG Techniques

Choosing the right methods and techniques for your RAG pipeline highly depends on your use case, and you need to try and evaluate your RAG pipeline accordingly.

Each RAG pipeline has three main components: query, context, and response. The query is the user input, the context refers to the retrieved information based on the user's query, and the response is the LLM output based on the user's query and context retrieval. Based on this structure, we can evaluate a RAG pipeline in two key areas:

Context Relevance: This refers to how relevant the retrieved context is to the query.

Answer Relevance: This refers to how relevant the responses are to the query.

5. Summary

The index retrieval techniques discussed earlier are methods to improve our context relevance, which in turn enhances answer relevance, our ultimate goal.

Visit my GitHub repository for further discussion about evaluating RAG pipelines.

6. References

1. Advanced RAG Techniques: Unlocking the Next Level [Tarun Singh]
2. Evaluating Advanced Retrieval Augmented Generation [Sachin Kumar]
3. Building and Evaluating Advanced RAG Applications [DeepLearning.ai]