Medium Search Write Sign up Sign in

Member-only story

Chunking : Strategies, Considerations and Optimization (Part 2 of RAG Series)

Bits, pieces and coherency

Chandan Durgia · Follow

6 min read · Jan 25, 2024

--

Date : 12/05/2024

Status : Completed

Photo by Markus Spiske on Unsplash

This is part 2 of the “Retrieval-Augmented Generation (RAG) — Basics to Advanced Series”. Links to other blogs in the series are at the bottom of this blog. Taking forward from part 1, in this blog we will focus on the “Chunking” component which is relevant for chunking of source content and chunking of query. (highlighted in Blue). Since, fundamentally the concept is similar, we will cover this together.

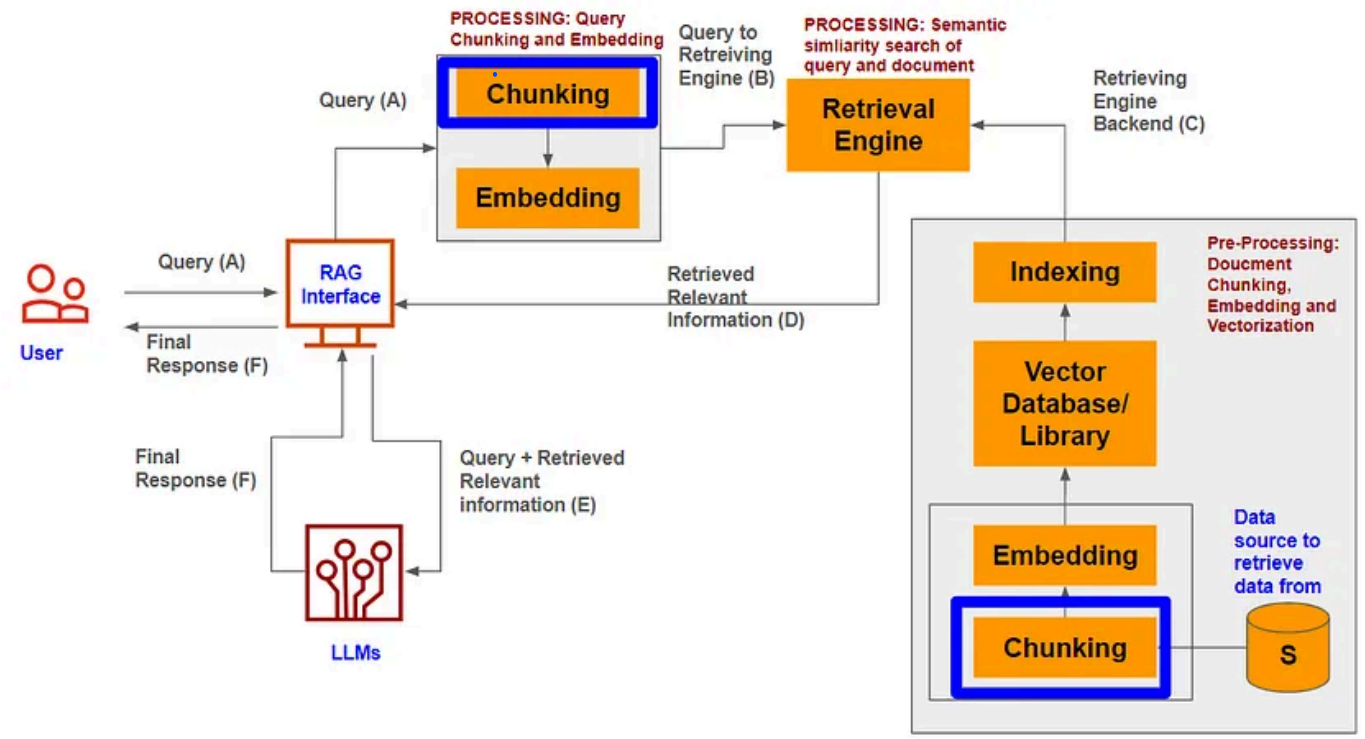


Image by Author: RAG Architecture

Chunking Basics

As discussed earlier, the data from the source document (for retrieving information) or query has to be converted into a mathematical form. The first step in this process is breaking down the text into smaller chunks of text. Hereafter called “Chunking”. The idea is once the text is chunked appropriately, every chunked part would be converted into a mathematical form (Embedding — next blog in the series).

Sounds straightforward right, but here comes some challenges — how to go about chunking — should we chunk it by every word, every few words, every sentence, few sentences together etc.? What would give us the best results?

Before we think about the right answer, let's think about what we are going to use these chunks for? We know in the "Retrieval Engine" these chunks will be compared with the query chunks and the chunks from the document which are contextually closest to the query chunks will be retrieved and sent to LLM.

Given this, below are some considerations we should keep in mind while defining a chunking strategy.

Considerations for chunking

The chunking strategy has a significant impact in terms of overall efficacy on the RAG output. The key element to keep in mind while defining the strategy is that every chunk to the extent possible should be able to encapsulate a "context" or a theme. i.e. from one chunk to another, the context could be different. The following are the key levers one could consider:

1. **Size of the document/texts:** If you are working with long documents — the context could be encapsulated in smaller chunks and vice versa for the shorter content.
2. **Query design:** If the query is long enough it means that the user has put in a lot of details in the query and is looking for a specific answer, for these cases a smaller chunk length would be a good option (improve the accuracy of the search results). On the other hand, shorter and simpler queries might require bigger chunks or no chunking, as the small queries can be processed as a single unit.
3. **Underlying data for embedding models:** It should be noted that the embedding models which convert the chunks into mathematical forms are trained in a manner that they perform well for specific chunk sizes. For example text-embedding-ada-002 embedding models perform well on 256 or 512 tokens chunks. So, one should review the underlying model and assess what chunking strategy would work well for the given model.

Given this background, we will now delve into the various chunking strategies. I intend to keep this short, concise and principle based.

Chunking Strategies

1. **Fixed size based chunking:** Fixed size chunks, for example, n number of words or n number of characters. It is quick and easy but could lead to context fragmentation (the text chosen doesn't capture the context). Low computational resources are required.

2. **Sentence based chunking**, using models like Spacy and NLTK to chunks with complete sentences. The sentences could be of different sizes and might be incoherent as separate sentences. Low computational resources are required.

3. **Coherence based**: Chunking whilst ensuring that the chunks are coherent. This methodology could include tracking topic changes — such that each chunk should have a semantic understanding. Requires high computational resources.

4. **Recursive chunking**: This is an intelligent and most commonly used chunking methodology which produces chunks in an iterative manner such that the desired chunk size and structure is obtained. This further includes a chunking structure which is flexible and hierarchical. High computational resources.

5. **Structural chunking:** This is applicable for files which have tags. i.e. HTML, Latex or markdown files. The chunks could be based on headers and sections and chunks are tagged with appropriate metadata.

These strategies form the foundational elements of chunking and many variants of these foundations chunking strategies have evolved over a period. But before we look into those, it is imperative to analyse one of the most important parameters related to chunking i.e. chunk size. i.e. how long or short a chunk should be.

Chunk Size: Size does matter

An intuitive and a simple way to think about chunking size is to find a balance between encapsulating context whilst ensuring accuracy. One should explore various chunk sizes like 128, 256 (smaller chunk category) or 512 or 1024 tokens (larger chunk category). Once can keep the following principles in mind:

- The larger chunks have more context, but would take longer for processing and could have a lot of noise (extra text or even text which could skew the output).
- On the other hand, smaller chunks would have less context, but would also have less noise and would require less processing.

Note that the common view in the industry currently is to use a **chunk size of 1024 tokens with overlapping chunks.** The chunk size of 1024 tokens is decent from a processing power perspective and overlapping chunks lessens

context fragmentation. There are a lot of advancements made in these areas — refer to Dynamic Chunking below.

Improving chunking performance

Though with the foundational chunking strategies one can get a decent accuracy and performance, however other avenues could be tried for fine tuning/improving chunking performance.

1. **Clean up your data before chunking:** This is also called Token Pruning. You should remove any noise, unnecessary filler words, non-value adding phrases, punctuations, HTML tags etc. from the data so that the context encapsulation can be done with the smaller chunks itself.
2. **Chunk indexing:** Indexing of chunks is a way to tag chunks to its indexes (attributes, topics, keywords etc.) so that the chunks can be easily searched by analysing these indexes. A good analogy is when one reads a paragraph one can write a few keywords, summary etc. for that paragraph which makes it easier to reference later.

a. Index category 1: Indexing of chunks **based on set of attributes and relationship to other chunks.**

b. Index category 2: Indexing of chunks **based on order of chunks within sub-parts** of the overall text and mapping it to its embedding vector**.**

c. Index category 3: Indexing of chunks **based on topics assignment** (knowledge domains based) and relevance.

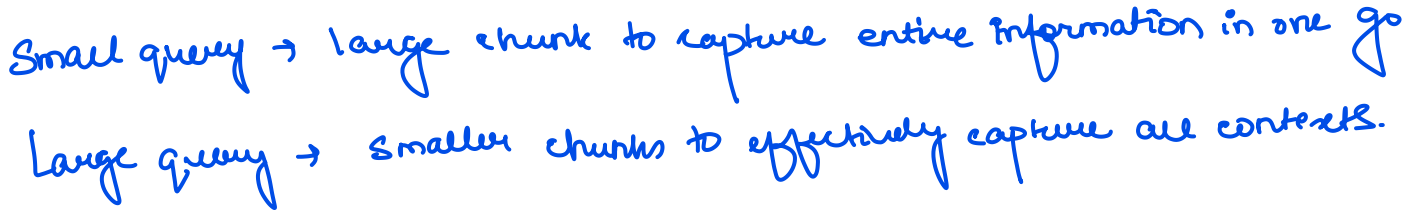
d. Index category 4: Assigning identifier **based on summary of each chunk.** This constitutes a chunking strategy called **small to big chunking.** Since summary chunks are tagged to actual text, the comparison can be done on the summary text (small text) and if required the complete text (big chunk) can be retrieved as well.

3. Multiprocessing Chunking: Applying distributed computing/parallel processing on small independent subqueries and then aggregate the results.

4. Dynamic Chunking: Depending on the query size and complexity, the chunk size is dynamically adjusted.

In conclusion, the chunking strategy plays a crucial role in enhancing the overall effectiveness of the RAG output. Therefore, it is essential to carefully assess the various factors and considerations associated with chunking. By understanding the significance of chunking and its impact on the final

results, individuals can make informed decisions and optimise their approach to achieve better outcomes. Evaluating the considerations for chunking allows for a more comprehensive understanding of how to effectively utilise this strategy and maximise its benefits.



Here is a view of what's in the series, feel free to let me know if you would like me to cover any specific aspect.

1. Retrieval-Augmented Generation (RAG) — Basics to Advanced Series (Part 1)

2. Pre-processing block — Chunking (part 2): strategies, considerations and optimization — this blog

3. Pre-processing block — Embedding (part 3): Types, Use cases and Evaluation.

4. Pre-processing block — Vector Databases and Vector Libraries (part 4)

5. Evaluation of RAG performance(part 5)

6. Improving RAG performance — A comprehensive structured approach (part 6A)

7. Improving RAG performance — A comprehensive structured approach (part 6B)

Do like and share and tag as much to help each other succeed.

Disclaimer: The views expressed in this article are opinions of the author in his personal capacity and not of his employer.

Llm Chunking Chunk Rag Optimization