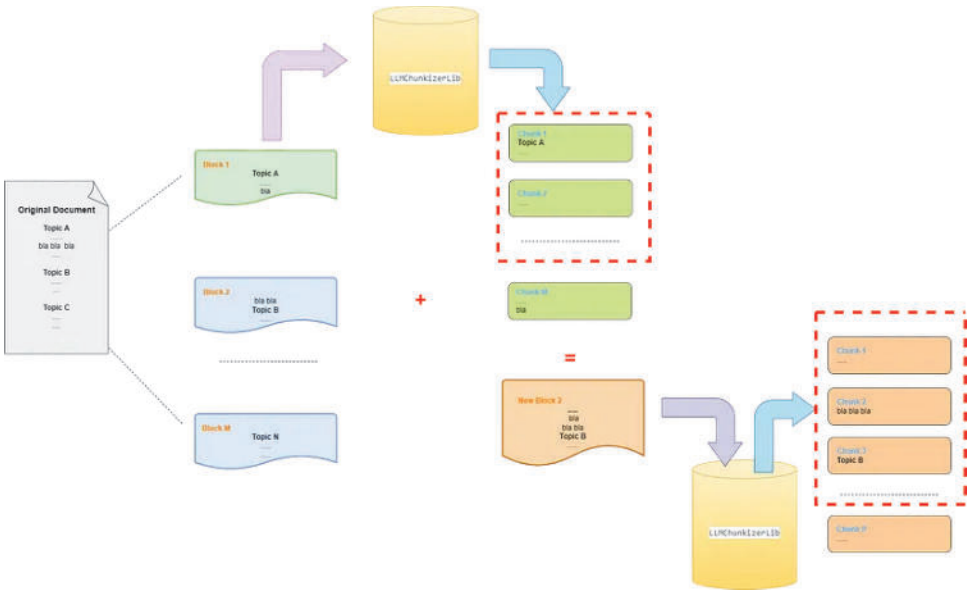
Efficient Document Chunking Using LLMs: Unlocking Knowledge One Block at a Time

Carlo Peron

Published in Towards Data Science · 8 min read · 6 days ago



The process of splitting two blocks — Image by the author

This article explains how to use an LLM (Large Language Model) to perform the chunking of a document based on concept of “idea”. *

I use OpenAI’s gpt-4o model for this example, but the same approach can be applied with any other LLM, such as those from Hugging Face, Mistral, and others.

Everyone can access this article for free.

Considerations on Document Chunking

In cognitive psychology, a chunk represents a “unit of information.”

This concept can be applied to computing as well: using an LLM, we can analyze a document and produce a set of chunks, typically of variable length, with each chunk expressing a complete “idea.”

This means that the system divides a document into “pieces of text” such that each expresses a unified concept, without mixing different ideas in the same chunk.

The goal is to create a knowledge base composed of independent elements that can be related to one another without overlapping different concepts within the same chunk.

Of course, during analysis and division, there may be multiple chunks expressing the same idea if that idea is repeated in different sections or expressed differently within the same document.

Getting Started

The first step is identifying a document that will be part of our knowledge base.

This is typically a PDF or Word document, read either page by page or by paragraphs and converted into text.

For simplicity, let’s assume we already have a list of text paragraphs like the following, extracted from ***Around the World in Eighty Days***:

```
documents = [
    """On October 2, 1872, Phileas Fogg, an English gentleman, left London for a
    He had wagered that he could circumnavigate the globe in just eighty days.
    Fogg was a man of strict habits and a very methodical life; everything was p
    He departed London on a train to Dover, then crossed the Channel by ship. Hi
    including France, India, Japan, and America. At each stop, he encountered va

    """However, time was his enemy, and any delay risked losing the bet. With th
    unexpected obstacles and dangerous situations.""",

    """Yet, each time, his cunning and indomitable spirit guided him to victory,

    """With one final effort, Fogg and Passepartout reached London just in time
    This extraordinary adventurer not only won the bet but also discovered that
]
```

Let’s also assume we are using an LLM that accepts a limited number of tokens for input and output, which we’ll call *input_token_nr* and *output_token_nr*.

For this example, we’ll set *input_token_nr* = 300 and *output_token_nr* = 250.

This means that for successful splitting, the number of tokens for both the prompt and the document to be analyzed must be less than 300, while the result produced by the LLM must consume no more than 250 tokens.

Using the tokenizer provided by OpenAI we see that our knowledge base documents is composed of 254 tokens.

Therefore, analyzing the entire document at once isn't possible, as even though the input can be processed in a single call, it can't fit in the output.

So, as a preparatory step, we need to divide the original document into blocks no larger than 250 tokens.

These blocks will then be passed to the LLM, which will further split them into chunks.

To be cautious, let's set the maximum block size to 200 tokens.

Generating Blocks

The process of generating blocks is as follows:

1. Consider the first paragraph in the knowledge base (KB), determine the number of tokens it requires, and if it's less than 200, it becomes the first element of the block.
2. Analyze the size of the next paragraph, and if the combined size with the current block is less than 200 tokens, add it to the block and continue with the remaining paragraphs.
3. A block reaches its maximum size when attempting to add another paragraph causes the block size to exceed the limit.
4. Repeat from step one until all paragraphs have been processed.

The blocks generation process assumes, for simplicity, that each paragraph is smaller than the maximum allowed size (otherwise, the paragraph itself must be split into smaller elements).

To perform this task, we use the *llm_chunkizer.split_document_into_blocks* function from the LLMChunkizerLib/chunkizer.py library, which can be found in the following repository — LLMChunkizer.

Visually, the result looks like Figure 1.

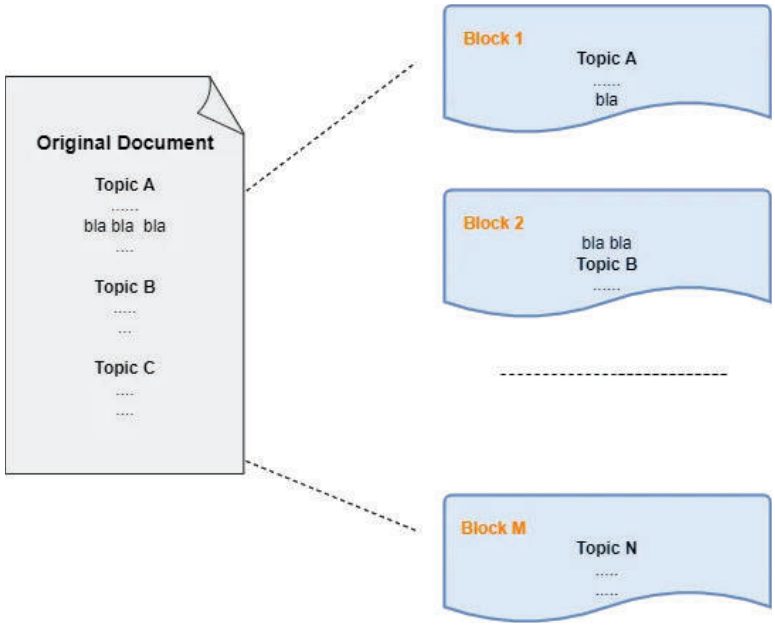


Figure 1 — Split document into blocks of maximum size of 200 tokens — Image by the author

When generating blocks, the only rule to follow is not to exceed the maximum allowed size.

No analysis or assumptions are made about the meaning of the text.

Generating Chunks

The next step is to split the block into chunks that each express the same idea.

For this task, we use the *llm_chunkizer.chunk_text_with_llm* function from the *LLMChunkizerLib/chunkizer.py* library, also found in the same repository.

The result can be seen in Figure 2.

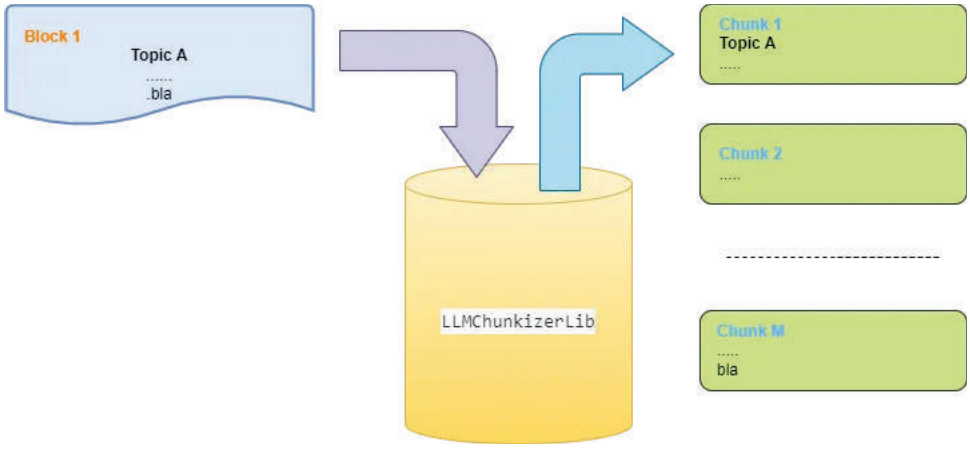


Figure 2 — Split block into chunks — Image by the author

This process works linearly, allowing the LLM to freely decide how to form the chunks.

Handling the Overlap Between Two Blocks

As previously mentioned, during block splitting, only the length limit is considered, with no regard for whether adjacent paragraphs expressing the same idea are split across different blocks.

This is evident in Figure 1, where the concept “bla bla bla” (representing a unified idea) is split between two adjacent blocks.

As you can see In Figure 2, the chunkizer processes only one block at a time, meaning the LLM cannot correlate this information with the following block (it doesn’t even know a next block exists), and thus, places it in the last split chunk.

This problem occurs frequently during ingestion, particularly when importing a long document whose text cannot all fit within a single LLM prompt.

To address it, *llm_chunkizer.chunk_text_with_llm* works as shown in Figure 3:

1. The last chunk (or the last N chunks) produced from the previous block is removed from the “valid” chunks list, and its content is added to the next block to be split.
2. The *New Block2* is passed to the chunking function again.

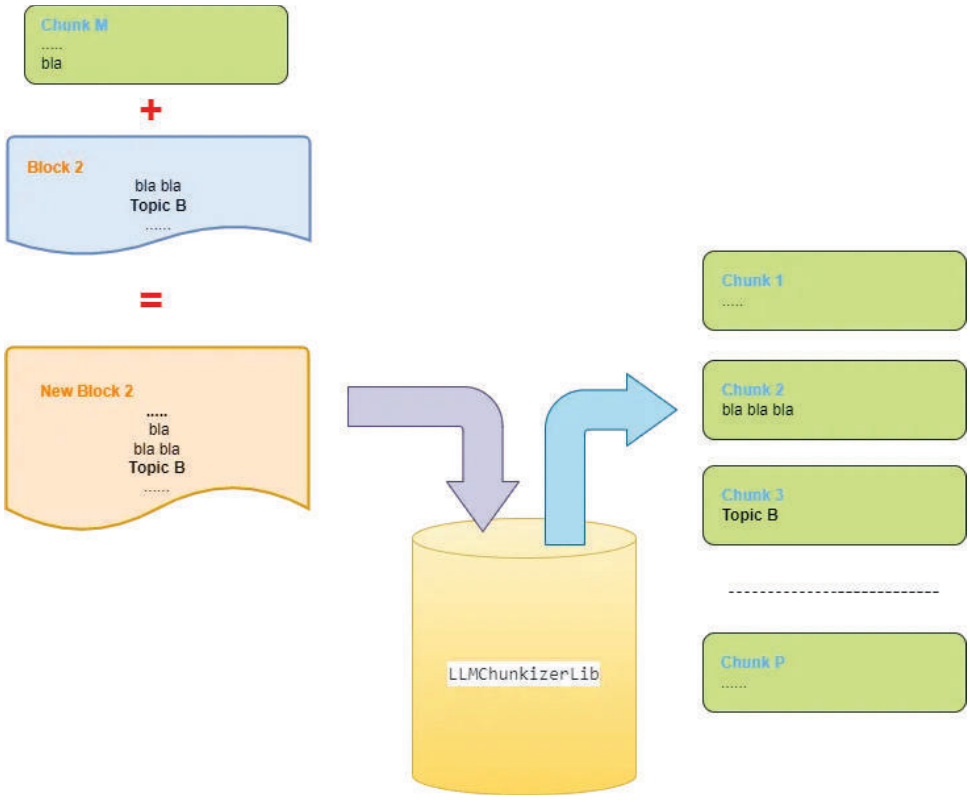


Figure 3 — Handling the overlap — Image by the author

As shown in Figure 3, the content of chunk M is split more effectively into two chunks, keeping the concept “bla bla bla” together

The idea behind this solution is that the last N chunks of the previous block represent independent ideas, not just unrelated paragraphs.

Therefore, adding them to the new block allows the LLM to generate similar chunks while also creating a new chunk that unites paragraphs that were previously split without regard for their meaning.

Result of Chunking

At the end, the system produces the following 6 chunks:

```
0: On October 2, 1872, Phileas Fogg, an English gentleman, left London for an ex
1: He departed London on a train to Dover, then crossed the Channel by ship. His
2: However, time was his enemy, and any delay risked losing the bet. With the he
3: Yet, each time, his cunning and indomitable spirit guided him to victory, whi
4: With one final effort, Fogg and Passepartout reached London just in time to p
5: This extraordinary adventurer not only won the bet but also discovered that t
```

Considerations on Block Size

Let’s see what happens when the original document is split into larger blocks with a maximum size of 1000 tokens.

With larger block sizes, the system generates 4 chunks instead of 6.

This behavior is expected because the LLM could analyzed a larger portion of content at once and was able to use more text to represent a single concept.

Here are the chunks in this case:

```
0: On October 2, 1872, Phileas Fogg, an English gentleman, left London for an ex
1: He departed London on a train to Dover, then crossed the Channel by ship. His
2: However, time was his enemy, and any delay risked losing the bet. With the he
3: With one final effort, Fogg and Passepartout reached London just in time to p
```

Conclusions

It’s important to attempt multiple chunking runs, varying the block size passed to the chunkizer each time.

After each attempt, the results should be reviewed to determine which approach best fits the desired outcome.

Coming Up

In the next article, I will show how to use an LLM to retrieve chunks — LLMRetriever .

You could find all the code and more example in my repository — LLMChunkizer.

If you'd like to discuss this further, feel free to connect with me on LinkedIn

Chunking | Retriever | Artificial Intelligence | Large Language Models | OpenAI

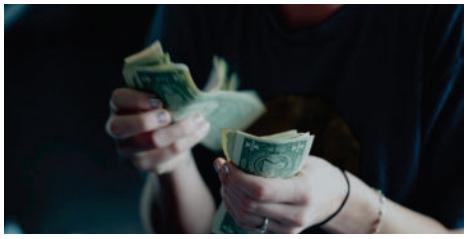
Written by Carlo Peron

Edit profile

41 Followers · Writer for Towards Data Science

You can get more information about me on https://www.linkedin.com/in/carlo-peron

More from Carlo Peron and Towards Data Science



Egor Howell in Towards Data Science

My 7 Sources of Income as a Data Scientist

Sara Nóbrega in Towards Data Science

5 Must-Know Techniques for Mastering Time-Series Analysis