THE CHUNK SPLITTING PROBLEM: A COMPARATIVE ANALYSIS OF INDEXING METHODS FOR RAG

Maximilian Huber
Khoury College of Computer Science
Northeastern University
huber.maxi@northeastern.edu

ABSTRACT

The primary focus of this project is to address and provide a clearer definition for an existing issue in RAG (Retrieval-Augmented Generation) systems, referred to in this paper as the "Chunk Splitting Problem." The aim is to explore and evaluate the effectiveness of different indexing methods in mitigating this problem. The methods under investigation are:

- Rule-Based Chunking
- Semantic Chunking
- Slide Retrieval
- Neural Network Estimation Retrieval (NNER)

These methods will be compared based on several criteria: the number of tokens they embed, the time required to generate an index, the retrieval time for context, and the quality and relevance of the retrieved information. By conducting this comparative analysis, the goal is to uncover the strengths and weaknesses of each approach, specifically in relation to the Chunk Splitting Problem, and provide actionable insights for optimizing indexing methods in practical RAG systems.

***K*eywords** RAG · Retrieval · Neural Networks

1 Introduction

Retrieval Augmented Generation (RAG) is a popular framework in natural language processing, which combines the strengths of information retrieval and large language models (LLM's). This framework is useful for generating informative and relevant responses to a user's queries. It accomplishes this by using the user's question as a query to retrieve valuable context from a prepared set of documents, and then prompting the LLM with the added context information. This has been shown to improve responses dramatically, especially on domain-specific topics.

RAG system's are reliant on indexes: structured representations of an informative text corpus which can be ranked and retrieved from based on semantic content. Indexing a set of documents usually consists of two main steps: chunking and embedding. Through chunking, the documents in the corpus are split into smaller text chunks, which are then individually converted into vector representations called embeddings. When it comes time to retrieve from the index, the query is converted into an embedding also, which allows the text chunks to be ranked in order of vector similarity to the query embedding. The chunks with the most similar vectors are expected to be the most semantically similar to the question, so they are also expected to be the most useful in providing the LLM with additional context for answering the question. However, there is a problem with this approach.

1.1 The Chunk Splitting Problem

When you chunk a set of documents, you run the risk of separating a continuous passage of contextually related text, because there is currently no practical way to choose where the boundaries of individual text chunks occur. This splitting can result in a partial loss of information, as a query that requires the context information which is caught

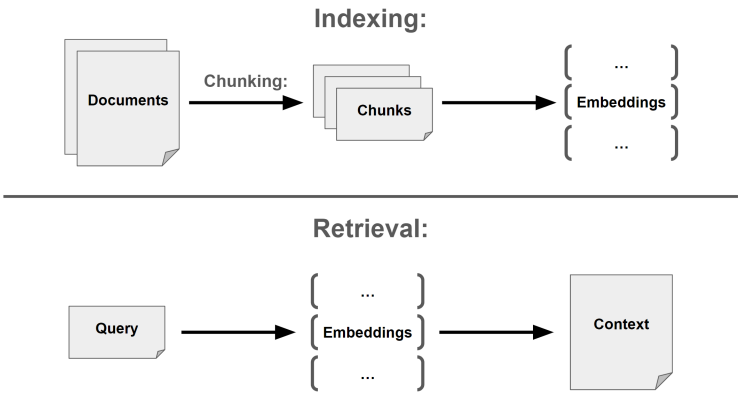

Figure 1: Indexing, Chunking, and Retrieval

between text chunks may only end up retrieving one of the two chunks. This can be referred to as the *"chunk splitting problem"* and is the primary motivation behind the experimental methods introduced later in this paper.

2 Indexing Methods

2.1 Rule-Based Chunking

This is the most common method for creating an index from a text corpus. The documents are broken up into chunks, usually by splitting text on a specified delimiter or by defining a maximum chunk size. The chunks are then embedded using a selected embedding model (for this paper, all chunks are embedded using the **text-embedding-ada-002** model from OpenAI, which outputs 1536 dimensional vectors). When the index is queried, the query is also embedded and used to perform a similarity search over all the chunks, and the most similar are returned as context. For this experiment, the industry-standard LlamaIndex library will be used with default parameters to implement this method. [1]

Parameters:

- **Maximum Chunk Size:** The maximum number of tokens included in each chunk
- **Delimiter:** A separator character (like "/n") used to identify splits in the text
- **Chunk Overlap:** How many tokens each chunk shares with its following neighbor
- **Return Count:** How many of the most similar chunks are retrieved

2.2 Semantic Chunking

This is an existing alternative to rule-based chunking which aims to improve semantic similarity within chunks and to avoid the splitting of semantically related text blocks, as often happens with rule-based chunking. This is done by first generating sentence embeddings for the entire corpus (NLTK sentence embeddings are used for this experiment[1]). Then a predefined similarity threshold is used to either add each sentence to the current chunk, or make it the first sentence of the next one. Once all chunks are created, the rest of the indexing process is the same as for the rule-based method. The retrieval process is also the same.

Parameters:

- **Threshold Variable:** determines how similar a sentence must be to a preceding one to be included in the same chunk
- **Maximum Chunk Size:** The maximum number of tokens included in each chunk
- **Return Count:** How many of the most similar chunks are retrieved

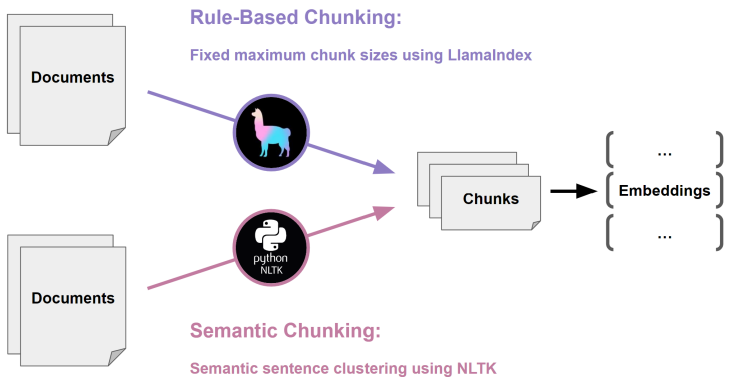

Figure 2: Rule-Based Chunking and Semantic Chunking

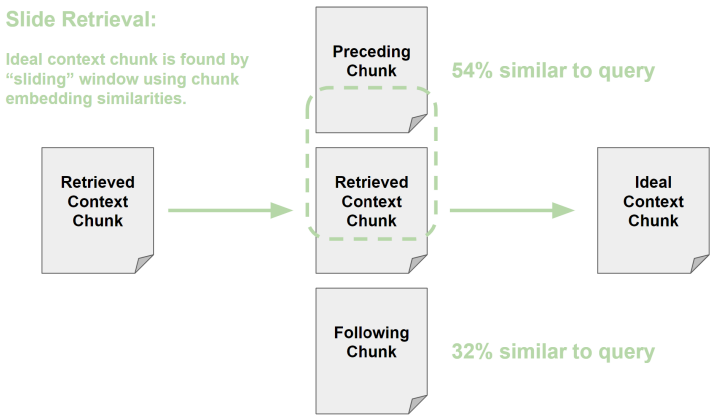

Figure 3: Slide Retrieval

2.3 Slide Retrieval

This is an augmentation of the standard indexing technique which aims to mitigate the chunk splitting problem. The proposed remedy is to retrieve the chunks whose embeddings are most similar to the query embedding, as we would normally, but to then combine these chunks with their neighboring chunks (the preceding chunk and the following chunk), creating very large chunks. Then, using the similarity between these neighbors and the query we can calculate a "sliding" move for a chunk-sized window which gives us our "ideal" chunk. The intuition is that if a chunk and one of its neighbors are both very similar to the query, the "ideal" chunk will lie somewhere between them, and so this method should help prevent losing information which is split between two chunks. This method notably only augments the retrieval process, and so could be used in conjunction with either of the chunking methods that were already described. For this experiment, it will be combined with a custom implementation of rule-based chunking.

Parameters:

- **Window Size:** The number of tokens included in each retrieved ideal chunk
- **Return Count:** How many of the most similar chunks are retrieved

2.4 Neural Network Estimation Retrieval (NNER)

This is an even more ambitious and experimental approach to indexing, which aims to prevent the chunk splitting problem entirely.

Because text chunks are converted into vector embeddings, an index represents a spatial representation of the corpus. Instead of conceiving of the corpus as a sequence of separate chunks, this method envisions the corpus as a continuous

[1]The semantic chunking approach is inspired by Solano Todeschini's "How to Chunk Text Data - A Comparative Analysis"[2]

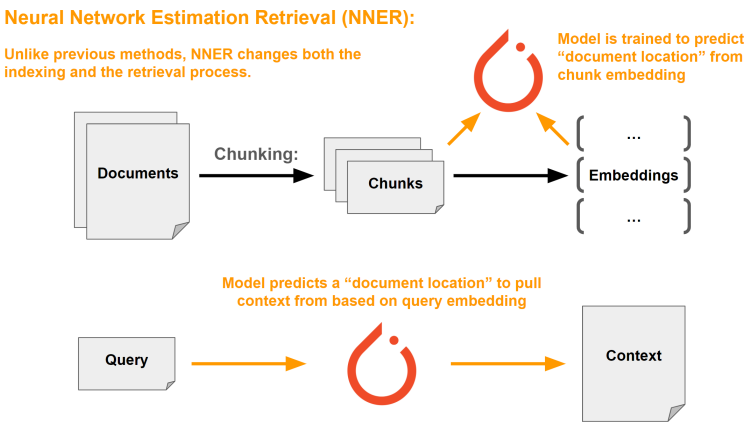

Figure 4: Neural Network Estimation Retrieval (NNER)

curving line through the vector space, in which every chunk embedding that the line passes through corresponds to a location in the corpus. If we use this representation of the corpus, we are not confined to retrieving individual chunks, and can instead pick a continuous value which represents any location in the corpus.

In order to achieve this representation of the corpus, we need to train a model which can estimate our curve through the vector space. We can do this by creating a model which takes in an embedding and returns its predicted location in the corpus. We gather evenly spaced chunk embeddings from the entire length of the corpus, and then train a model on these location-embedding pairs, with the hope that we can predict the most similar location of the corpus given any new embedding. This also means that this method does not need to embed every chunk, which saves on the embedding cost.

When it comes time to retrieve, we embed our query as before, and then use our trained model to predict a location in the document. Then we return a chunk of the document surrounding that location, defined by a window size parameter.

method may decrease the amount of embedding generation required to construct an index, and does not suffer from the chunk splitting problem like rule-based chunking does, but its feasibility is ultimately dependent on whether a corpus can really be modeled this way, and if the model can be properly trained given the limited data provided. Also, with the current implementation, this method differs from all previous methods in that it only returns a single location in the corpus, which may be a critical weakness that outweighs its potential benefits.

Parameters:

- **Model Architecture:** The combination of layers used in the estimation model will affect how well this method performs
- **Epochs:** The number of epochs to train the model for
- **Chunk Gap:** How much space is left between chunks which are embedded
- **Maximum Chunk Size:** The maximum number of tokens included in each chunk
- **Window Size:** The number of tokens included in each retrieved chunk

3 Data

3.1 Documents

In order to test these indexing methods, we need documents to index. For this experiment the documents consist of 30 Wikipedia articles of different lengths about various neural network, machine learning, and AI topics. They were scraped using the Wikipedia Python library. Using this data was preferable because Wikipedia articles are very informative, and the data was clean enough to use without preprocessing.

3.2 Queries

In order to compare how well our generated indexes actually work, we need a set of queries which pose questions that can be answered by the documents. For this experiment the 110 test queries (and associated answers) were generated using GPT-4. All the queries can be answered with information found in the documents.

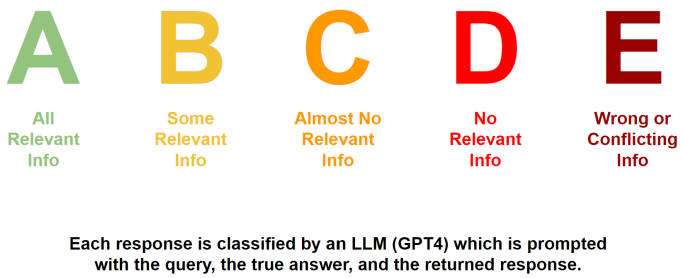

Figure 5: Response Quality Classes

All the documents and queries used in this experiment can be found in the following repository:

`https://github.com/MaxHuber888/comparative-indexing-analysis`

4 Experimental Setup

To analyze how the various methods compare to one another, we used each method to generate an index of the documents, and then ran all the test queries on each index. The metrics that were tracked throughout the experiment were:

- **Index Generation Time:** the total time it takes each method to generate an index
- **Average Query Time:** the average time it takes for each method to retrieve context
- **Tokens Embedded:** the number of tokens each method embeds while generating indexes
- **Response Quality:** the quality and relevance of the information retrieved

The number of tokens embedded were tracked using the Tiktoken library (this allows for counting the number of tokens in each chunk using the same rules as the one used by OpenAI's embedding model).

For scoring the quality of the responses, all responses were saved to file while querying. Then, for each query, associated answer, and returned context information, GPT-4 was prompted to classify the returned context using one of the five classes seen in Figure 5. The number of responses which received each classification for each indexing method were recorded as a measure of that method's general response quality[2].

5 Results

5.1 Index Generation Time

In Figure 6 we can see the number of seconds that passed while each method was generating its index. The red line in the graph represents the one hour mark. These results are to be expected, as semantic chunking involves generating sentence embeddings for the whole corpus, and NNER involves training the predictor model, so both of these methods are much more computationally expensive while generating the index. The difference between rule-based and slide retrieval is most likely due to the fact that the LlamaIndex library (which was used for the rule-based method) is heavily optimized. Since the slide retrieval method is only changing the retrieval process, and could easily use the same chunking method as the rule-based method (i.e. could be adapted to use LlamaIndex for chunking/indexing), this difference can be ignored.

5.2 Average Query Time

In Figure 7 we see the average number of seconds that passed while each method retrieved context for a query. The methods are much closer for this metric, and in fact none of the methods took more than a second on average to retrieve

[2]This approach to measuring the quality of RAG responses is adapted from Doug Safreno's "How to test rag systems" [3]

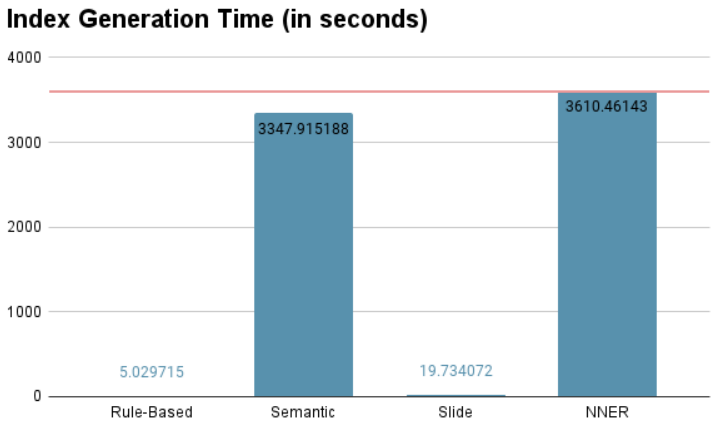

Figure 6: Index Generation Time

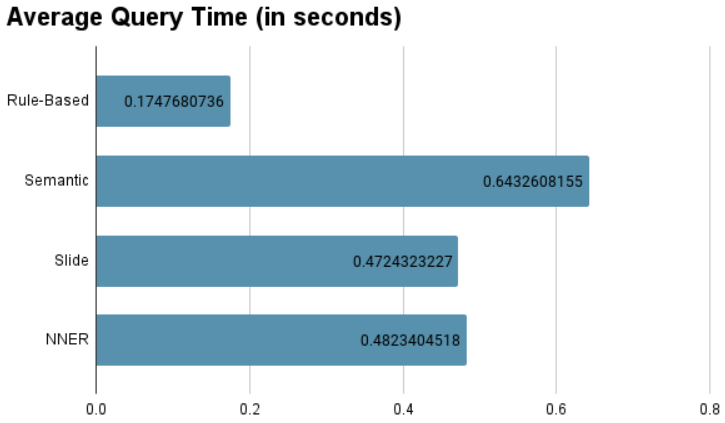

Figure 7: Average Query Time

context, which makes all of them viable for use in a real-time setting like an RAG chatbot. The optimized efficiency of LlamaIndex still shows, as rule-based is still by far the fastest method.

5.3 Tokens Embedded

In Figure 8 we see the number of tokens embedded by each method while generating an index. The main motivation for showing this metric was to show that NNER can generate an index with much fewer embedded tokens. Because every method used the OpenAI embedding model, there was a real per-token financial cost for embedding these indexes. That makes NNER more financially preferable, as it cost less than half of what the rule-based method cost.

This metric also revealed an oversight in standardization of the methods: the rule-based method embedded more tokens than slide retrieval because LlamaIndex uses a chunk overlap of 20 tokens and a maximum chunk size of 1024 tokens by default. Slide retrieval, in this experiment, used a maximum chunk size of 1000 tokens and no chunk overlap. In a more controlled experiment slide retrieval and rule based would share identical values here. We would expect the tokens embedded by semantic chunking to just be the total number of tokens in the corpus, because it is embedding every chunk (like slide retrieval and rule-based) but is not using chunk overlaps (not needed, as ideally semantically related information is already chunked together).

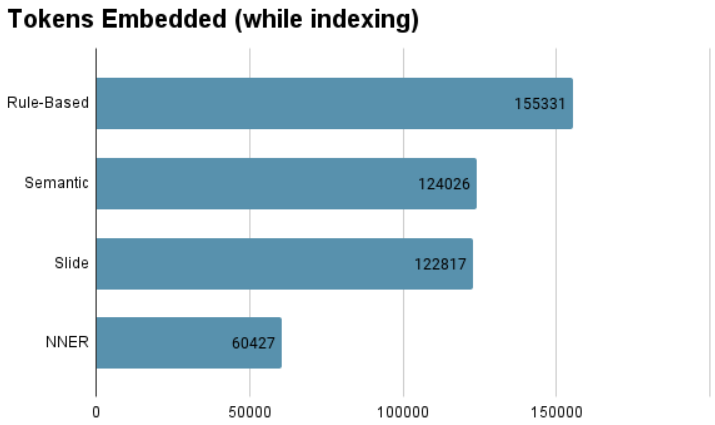

Figure 8: Tokens Embedded

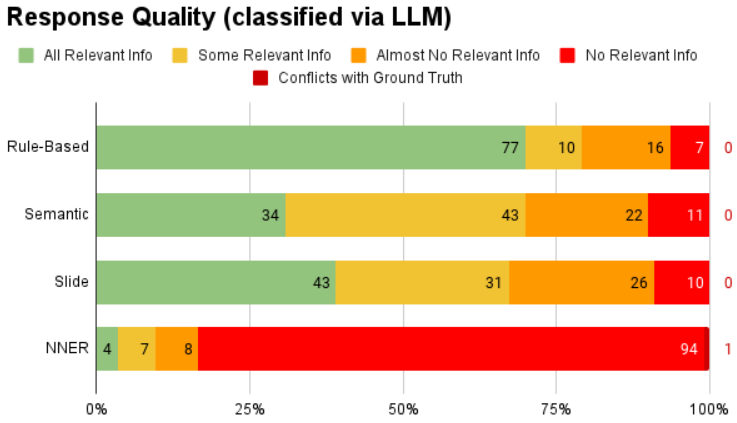

Figure 9: Response Quality

5.4 Response Quality

This metric is arguably the most important, as if a method is not able to return contextually relevant text, then it cannot perform the functions required of it for use in an RAG system. As can be seen in Figure 9, the response quality of NNER is far below what is acceptable if it is to be considered as an alternative to rule-based chunking. The other methods which we implemented also underperformed, indicating that there may be something wrong with our implementations. But based on this metric alone, rule based chunking via LlamaIndex is the clear best choice (and really the only practical choice, at least for now).

5.5 Results Analysis

Despite the disappointing results for NNER, semantic chunking, and slide retrieval, we don't believe that these results are definitive evidence that these approaches are not worth developing further. A big part of this conviction lies in the multitude of ways in which this experiment can be further fleshed out and expanded. There are so many factors to consider when comparing these methods, and so many ways in which the methods could potentially be improved, many of which could be the difference between terrible and amazing results. We believe that it is at least worth considering the efficacy of these experimental methods, and making further attempts at successfully implementing them before dismissing them. In the final section we will discuss all the potential improvements that could be made to this experiment in a future trial.

6 Future Work

6.1 Chunk Overlap and LlamaIndex

One big oversight of this project was neglecting the chunk overlap parameter in the implementation of slide-retrieval. If done again, it would probably be best to rely more heavily on the LlamaIndex library for the construction of all methods, as the difference in efficiency is clear from the results. If the slide retrieval method was implemented as an augmentation to the LlamaIndex rule-based chunking, then we might actually see the improved performance we are expecting.

6.2 Combined Methods

Another unexplored potential in this experiment was the inherent modularity of the indexing process. Both rule-based chunking and semantic chunking could be combined with slide retrieval, which could result in improved performance. The current implementations of these methods are all very isolated, which makes them near impossible to combine in the current code base, but in fact their strength comes from the fact that they can be combined, and so their implementation should reflect that.

6.3 Parameter Optimization

All of the methods have various parameters which are more or less randomly chosen. In an ideal scenario, the methods would be run multiple times with various parameter configurations to get a sense for what combination of parameters leads to optimal performance for each model. One could even imagine a genetic algorithm approach being used to learn the best parameter configuration for each method.

6.4 NNER Architecture

In a very similar sense, the structure of the NNER predictor model could be optimized. For this experiment it was just a very simple set of fully connected layers which go from the 1536 dimensional embedding vector to the single continuous value representing a document location. One potential improvement could be figuring out some way to allow the model to output multiple locations within the corpus, as right now NNER is the only method with this restriction, and that could make it considerably less performant.

6.5 Human Review of Responses

The scoring of response quality was all done by an LLM, but it would be good to verify these results by hand. It may be that the LLM is over-estimating or under-estimating the performance of all methods, but probably not a problem for comparing them to one another.

6.6 Alternative Embedding Algorithms

For this experiment, all models used the text-embedding-ada-002 embedding model from OpenAI. This is far from the only embedding model available, but it was chosen because it is the default for LlamaIndex, and LlamaIndex served as the baseline for this experiment. It would be interesting to try embedding methods which count tokens differently, and even methods which do not have an associated financial cost (like the NLTK sentence embeddings used for semantic chunking). It may also be the case that other embedding algorithms provide a better spatial representation of a corpus for the NNER predictor to train on.

6.7 Hybrid Search + Reranking

There are other techniques for improving RAG performance which we did not apply here. Using a hybrid search (combination of the semantic similarity search used here and a keyword search) or doing some kind of reranking of the retrieved chunks may yield better results.

6.8 Expanded Dataset

Finally, this experiment used a very small dataset of non-sequential documents (each article is independent of the others). It would be interesting to try the experiment with a much larger dataset, and also a sequential document set like a book series.

Acknowledgments

I would like to express my sincere gratitude to Dr. Amir Tahmasebi for his invaluable support and guidance throughout this research project. His expertise, insightful feedback, and encouragement were instrumental in shaping this work and helping me navigate the challenges I encountered. Dr. Tahmasebi's dedication to fostering academic growth and his willingness to share his knowledge have been truly inspiring.

Code Repository

`https://github.com/MaxHuber888/comparative-indexing-analysis`

References

[1] Llamaindex documentation. `https://docs.llamaindex.ai/en/stable/understanding/indexing/indexing.html` [Accessed 18 Apr. 2024].

[2] Solano Todeschini. How to chunk text data - a comparative analysis. `https://towardsdatascience.com/how-to-chunk-text-data-a-comparative-analysis-3858c4a0997a` [Accessed 18 Apr. 2024].

[3] Doug Safreno. How to test rag systems. `https://gentrace.ai/blog/how-to-test-rag-systems?t` [Accessed 18 Apr. 2024].