2024 3rd International Conference on Computer Applications Technology (CCAT) | 979-8-3315-3979-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/CCAT64370.2024.00021

Reordering of Double Pass Merging Chunking to Improve Retrieval Augmented Generation

Xiaohang Gong*
Department of Computer Science
University of Science and Technology of China
Hefei, 230026, China
gxh2018@mail.ustc.edu.cn

Runpeng He
Department of Computer Science
McGill University
Montreal, H3A 0G4, Canada
runpeng.he@mail.mcgill.ca

Qihan Peng
Department of EEE
Hong Kong Polytechnic University
Hong Kong, 999077, China
22101057d@connect.polyu.hk

***Abstract*—Large Language Models(LLMs) do not perform well in certain tasks or queries that are out of the training database. One prevailing notion is Retrieval-Augmented Generation (RAG), which means using the external database to provide external knowledge for LLMs. Chunking methods play a crucial role in the performance of RAG. In this work, we experiment with a semantic chunking method called double pass merging, explore the effects of merge order on the quality of chunks, and propose an improved merge algorithm. Our experimental results show that changing the merge order can significantly influence both chunking outputs and RAG performance and our method achieved a performance improvement of at least 15% on the test set. However, more research is required in order to find the ideal merge ordering.**

***Keywords*—RAG, chunking methods, double pass merging chunking, merging order**

I. Introduction

Nowadays, even though current large language models (LLM) have been developed very smart and intelligent, there are still some problems, such as inaccurate and untraceable responses. There are currently some issues with LLMs: 1) LLMs may generate hallucinations [1] or inaccurate and untraceable responses due to outdated databases or when they deal with domain-specific or knowledge-intensive tasks [2]. 2)The LLMs are always with a fixed knowledge base once the training is completed. And the cost (always retraining) of updating the knowledge base is unaffordable. 3)The performance of LLMs do not achieve the expectation when handling long or complicated text, and there is a risk of overlooking or misinterpreting relevant context. 4)Online LLMs, such as ChatGPT, are not well personalized, and users cannot feel comfortable providing private information. They may cause privacy and security concerns.

Therefore, the Retrieval-Augmented Generation (RAG) [3] is a modern concept that means using the external database to provide external knowledge for LLMs and helps them to generate accurate and appropriate responses based on the external context. RAG combines retrieval and generation methods in NLP to mitigate the drawbacks of a pure retrieval model or generative one. The basic idea of RAG is to collect the most relevant document chunks or pieces of information from an external database based on user Queries, followed by putting this information along with users' queries to LLMs in order for them to generate a response rooted in this broader knowledge base.

Therefore, enhancing the performance of RAG is an important issue. The existing RAG methods can be divided into three categories: naive RAG, Advanced RAG, and modular RAG [4]. However, all kinds of RAG methods include these steps: chunking, embedding, and retrieval. Focusing on the chunking aspect of RAG, we optimize the of double pass merging chunking (a semantic chunking method) by reorder the merging order and evaluating chunking performance, and indicating how we can enhance the overall performance of RAG by improving chunking.

II. Background

A. *RAG's Problem: Long Context*

The fundamental purpose of RAG is to use the external database to augment the large language models' knowledge to generate more precise and specific responses to users' prompts. For every user's query, after the RAG retrieves the external database and finds files that contain the most relevant information, it combines these files with user input and provides the LLM. However, in the practical environment, this approach may not be appropriate and face some problems.

The following are some reasons why external documents should be divided into chunks.

1. The **input limitation** (context length) of the large language model prompts this approach.

2. Long contexts **negatively impact the** performance of the large language model in terms of time and accuracy. This is due to long contexts contain a lot of irrelevant information which causes the LLM is difficult to find relevant information fast and accurately. Also, longer files lead to longer response time in the embedding, retrieval, and generate steps, because models need longer time to scan and find. Additionally, Chunking enhanced retrieval by fitting within the context window limits of LLMs.

3. **Resource constraints**. One of the designed aspects of Rag is to help individual users to build their own local AI agent with their personal database in a security environment. However, personal users have different devices and resources to implement their own RAG. Thus, the third problem is the resource constraints such as GPU, token usage, and some other

personal factors because a long context prompt requires too many resources for implementation and running for personal work.

Thus, the common approach to solve these problems is to first divide long context files into smaller chunks and then input the query and relevant chunks into the LLM. This chunking approach significantly impacts the performance of RAG in situations with long context. For example, if the size of chunks is too small, the continues semantic contexts will be split up; on the other hand, if the size is too big, previous problems will occur again. Therefore, how to chunk long contexts is a very important and necessary feature of RAG that needs to be improved.

B. RAG's Background

The common RAG process is demonstrated by Figure 1, illustrating the stages in a standard RAG process.

a. External Database

The external database in a naive RAG process serves as an extra source of information or knowledge, and the model can retrieve relevant context by the queries to generate more accurate responses. This database could be a text corpus, knowledge base or any naturally structured set of information the model can search through.

b. Chunking ❶

Chunking refers to the process of dividing external documents into smaller, manageable segments or called chunks. The logic behind this is LLMs' input length limits and faster retrieval. Each chunk holds a fragment of the information that may be searched and retrieved according to the query. Chunking helps RAG to be practical and speeds up the retrieval process as it allows the RAG to put a part of external documents into the LLMs instead of putting all of them.

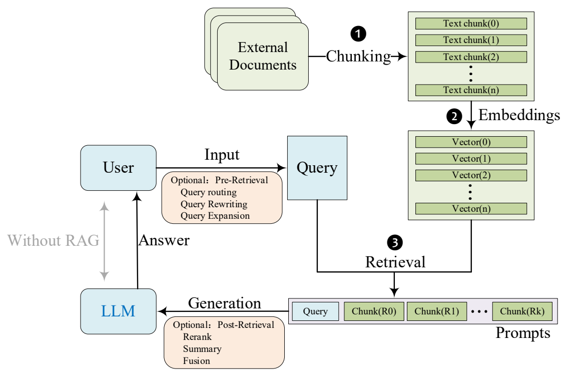


Fig. 1. Typical naive RAG process

(options highlighted in the pale yellow show Advanced RAG)

c. Embedding ❷

Embedding means converting text or data into a vector. In the process of RAG, all chunks of the external data and the queries of user are embedding into vectors. These vectors reflect semantic meanings of text in a high-dimensional space. The vectors are used to find similarities between the queries and the chunks produced in Chunking step.

d. Retrieval ❸

Retrieval is the process of searching the relevant chunks based on the query. The query is embedded into vector, and compared with embedding vectors of the chunks to find the most relevant one. Similarity is usually measured by similarity metrics like cosine similarity.

e. LLM

The Large Language Model (LLM) is the text generation part. After retrieving relevant chunks based on the query, the LLM accepts both the query and the retrieved context to generate the response.

Naive RAG is a text generation augmentation method that retrieves information from an external database, and concatenates the retrieved content with queries. The external database plays a role of information bank or knowledge source. Hence, this database could be in the form of a text corpus, knowledge base, or any information (structured data) that can help the model search for more meaningful context/fact beyond its bounded internal parameters. Also, chunking is necessary in RAG, where the external database has to be split into chunks so that it can cater to the input limit of LLMs and be easier for retrieval. Information in each of these chunks is a part of the whole external document and can be searched quickly, which reduces the time to retrieve data by decreasing the likelihood that the conversation will need to search through all records for potential matching information. The second step is embedding, and text data is turned into a vector through embedding models. Next, the chunks from an external database and the query are both converted into embedding vectors that signify semantic meaning in high-intent space. We use these embedding vectors to compare/query similarity with the chunks. Finally, the retrieval step is to find matching chunks from an external database given query (usually by searching similarity metrics like cosine_similarity), also, the chunks get returned and used as further context when building a response. Finally, a large language model (LLM) is the key to text generation in the naive RAG framework. LLM generates a text that will be based on the query along with retrieved chunks, which means it utilizes its internal knowledge plus external contextual information to provide better or more accurate responses for context-specific queries. Despite the fact that naive RAG is powerful, it has some disadvantages: because chunks with low relevance to the query are retrieved (which can lower precision); while they fail to return enough relevant chunks during retrieval time (thus lowering recall), leading into hints when merging these returned chunks from retrieving phase and then including them with our original input text. These issues might cause redundancy/repetition in generated content as well. Thus, chunking methods affect the performance of RAG significantly, and we will show the order in the "semantic double pass merge chunking" method and how to improve it.

Advanced RAG adds a number of improvements to make naive-RAG more efficient, accurate, and powerful. Prevailed methods include query rewriting, query transformation, query expansion, and other techniques [5-8]. This encompasses pre-retrieval index optimization with data granularity improvement and optimized indexes fortified by acceleration techniques using metadata-aware-and-mixes so as to retrieval kinds. A final attempt to refine the retrieval process [9]: further optimize queries. Post-retrieval, all those things you can do to better paste the question into retained relevant content by reordering chunks and compressing context or ensuring more coherent, clearer, most pertinent answers.

The Modular RAG is one level beyond the well-known pipeline architecture of RAG, it extends this to introduce independent modules and more customizability for tasks. This paradigm facilitates the replacement or reconfiguration of modules, thus making dynamic organization of the RAG process in a context-specific way. Its modularity also enables to evolve and adapt the RAG architecture continuously according to a particular scenario, being more efficient in practice.

C. Chunking Methods

The chunking method plays a crucial role in determining the performance of RAG. It can be broadly categorized into two types: conventional chunking and semantic chunking. In the conventional approach, the text is divided based on specific identifiers like punctuation marks, which fail to recognize semantic dependencies between adjacent statements and are susceptible to formatting variations in the source file. On the other hand, an advanced approach involves semantically chunking external documents by merging tokens with high similarity into the same chunk. Various chunking methods are elaborated below, and Table 1 lists them in detail [10].

1) Conventional chunking methods:

a. Split the text directly based on characters or source text content without considering tokens or sentence boundaries.

Advantages: simple and easy to implement, flexible.

Disadvantages: may not consider word boundaries, resulting in incoherent semantics.

b. Divide the text into smaller paragraphs based on a predetermined quantity.

Advantages: easy to implement, suitable for fixed-length processing.

Disadvantages: may ignore semantic structure and information repetition.

c. Sentence-based segmentation, where each chunk contains one or more complete sentences.

Advantages: utilizes natural language structure, easy to understand and analyse.

Disadvantages: may be imprecise when dealing with complex sentences and inconsistent lengths.

d. Recursively split larger chunks until the desired granularity is achieved.

Advantages: fine control over segmentation granularity.

Disadvantage: computationally complex and inefficient

e. Split the text based on its hierarchical structure, such as chapters and paragraphs.

Advantage: suitable for complex document structures, enabling hierarchical analysis. Disadvantage: depends on document format, poor performance when handling poorly formatted documents.

2) Chunking-free：

The process of information retrieval and generation in the absence of explicit document segmentation relies on analysing the entire document or substantial sections of text, rather than dividing it into smaller fragments.

3) Semantic chunking methods (state of the art)

a. semantic double pass merge chunking

Advantages: Improve segmentation accuracy and handle interrupted content.

Disadvantages: Complex implementation rely on high-quality similarity calculation.

b. k-means clustering: Generate sentence embeddings through embedding models and then cluster similar sentences using k-means.

Advantages: Cluster similar content and capture semantics.

Disadvantages: May lose sentence order, not suitable for time series data.

c. Using large language models (LLMs) to determine the optimal segmentation of content and size

Advantages: Accurate segmentation based on semantic analysis.

Disadvantages: High computational cost, require high-performance LLM.

d. Split based on differences between embeddings when exceeding a predefined threshold value. Advantages: Precise control of splitting points, good semantic preservation.

Disadvantages: Complex computation requires embedding models.

Table 1. Chunking methods' features

Chunking Category	Chunking Methods	Advantage	Disadvantage
Conventional Chunking	fixed-length characters	Implement: Simple suitable: fixed-length processing.	Ignore semantic structure &information repetition.
	paragraphs	Implement: Simple & Flexible Understand and analyse: easy	Complex query: imprecise
	sentences		
	recursively splitting	Chunking granularity: Fine control	Complex and inefficient
Semantic Chunking	k-means clustering	Cluster similar content, capture semantics.	May lose sentence order, not suitable for time series data.
	large language models (LLMs) determination	Accurate segmentation based on semantic analysis.	High computational cost, require high-performance LLM.
	differences between embeddings	Precise control of splitting points, good semantic preservation.	Complex computation, require embedding models.
	semantic double pass merge chunking	Improve segmentation accuracy, handle interrupted content.	Complex implementation, rely on high-quality similarity calculation.

D. *Evaluation Methods*

The completion of chunking necessitates an evaluation method to ascertain its efficacy. Generally, the results can be evaluated at three stages: post-chunking, post-retrieval, and after the LLM generates a response based on RAG outcomes.

1) chunking evaluation (after chunking)

In this evaluation category, we evaluate the results of chunking. That means we get several chunks after chunking, and have to design evaluate method to assess the results. LlamaIndex and Chroma show how to do that.

a. LlamaIndex [11]

LlamaIndex made use of the chunking evaluation method as described in one of its blog posts to identify what is an optimal size for a RAG, i.e. The process involves creating the evaluation environment, generating questions from a data set and time measurement metrics like average response time, how faithful are the responses on an answer level or asker query level-to what extent they span required content. The idea is that you are trying to find the right chunk size between not having enough information (you will have false alarms) and too much info, which may make the system more unresponsive. The implication is that there is an "ideal" chunk size that would maximize the quality of returned responses without bloating any single query, as well.

b. Chroma Research [12]

The Chroma Research chunking evaluation technique involves comparing the effectiveness of different document chunking strategies for retrieval performance in AI applications. It is a token-level specific retrieval methodology, and various precision-recall Intersection over the Union (IoU) methods are used to measure the efficiency of this approach. Question and text corpora (queries and examples) are passed to the model; it generates chunks of useful contexts from the sentences in example texts — they help us understand what parts of which tokens were given relevance for a query. This helps in routing the best chunking strategies, which, on one side, are capturing the right things and not a huge number of waste tokens.

2) RAG evaluation (after LLM) [13]

In this evaluation category, we evaluate the output of LLMs. RAG input both queries and their relevant context into LLMs, and the evaluation will compare the outputs or response text. The key points include:

a. Process: 3 main steps
  i. Retrieval: Get relevant information from external knowledge chunks, search the most relevant context based on the query.
  ii. Generation: Using the retrieved chunks to generate responses, involving prompting and inferencing phases.
  iii. Evaluation: Compare and evaluate the responses generated by the LLMs

b. Main evaluation metrics:
  i. precision, recall. etc: Evaluate the correctness of LLMs' response.
  ii. Latency: Measuring response time.
  iii. Fluctuation: Responses are affected by the LLMs, thus we must measure the fluctuation.
  iv. Noise robustness: Accuracy is affected when encounter irrelevant or misleading information.
  v. Negative rejection: LLMs may reject to response when the information is not sufficient.

c. Challenges: Evaluating RAG is complex due to:
  i. Dynamic knowledge bases: Retrieval will encounter vast, dynamic information sources.
  ii. Correctness: This evaluation must choose a powerful LLM, we can't get any meaningful data if the LLM fails to answer the query.
  iii. Fluctuation: LLMs seem to be a black box and will generate different responses even for the same prompt.

d. Exist evaluation framework: Auepora Framework: The authors introduce "A Unified Evaluation Process of RAG (Auepora)," focusing on the targets that defining what aspects of the system to evaluate (e.g., relevance, accuracy, faithfulness), the datasets and the metrics(e.g., precision, recall, BLEU, ROUGE).

3) retrieval evaluation (after retrieval)

In this evaluation category, we evaluate the results of retrieval. That's to say, we get a list of related chunks for each query and measure the similarity or relevancy between the query and chunks in the list.

The retrieval evaluation in LlamaIndex using the RetrieverEvaluator to evaluate the quality of retrieval [11,14]. This involves defining the metrics used to evaluate (e.g., hit-rate and MRR) that would measure how well the retrieval does based on ground truth.

III. Semantic Double-pass Merge Chunking Order and Evaluation

In this section, we evaluate the effect of merge order on the results of Semantic double-pass merging chunking.

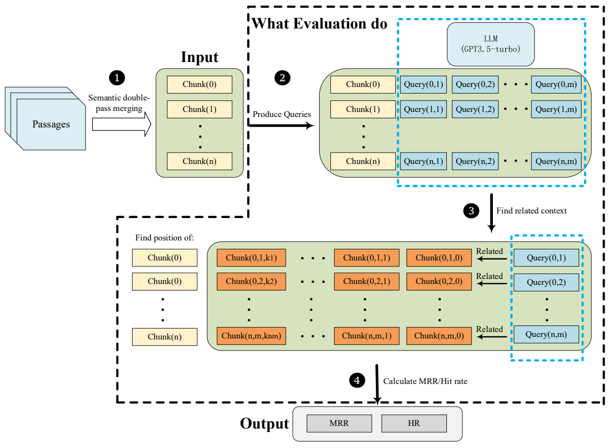


Fig. 2. Total workflow

Our workflow is shown in Figure 2. We implement the double pass merging chunking method to chunking the external document. Then, a set of chunks is produced, and we design an evaluation process for chunking results. The key idea of evaluating chunking lies in using LLM to generate questions for each chunk, then retrieving the most relevant *chunks* for each question. The basic principle is that an ideal chunk should be semantic independent, including all necessary and relevant information. Hence, we choose Mean Reciprocal Rank (MRR) and Hit Rate metrics for evaluating the quality of chunking results.

A. Documents Chunking ❶

The traditional chunking methods sometimes generate chunks that lose necessary context information. For example, they may split a sentence into two chunks or separate two paragraphs that are related closely. Consequently, this leads to uncompleted context and affects the accuracy of LLMs' responses.

So currently available traditional chunking methods encounter challenges. The primary obstacle lies in paragraphs that are highly relevant but lack semantic similarities with the surrounding text, such as those containing mathematical formulas, code/algorithm blocks, or citations. Due to significant differences between the embedded encodings of these fragments, documents are often erroneously segmented using traditional semantic chunking approaches. Instead of creating a larger chunk that better describes the given fragment, multiple smaller chunks (including individual mathematical formulas) are frequently generated because currently employed chunks are "terminated" upon encountering the first fragment with semantic differences from their content.

The aforementioned issues have resulted in the development of a chunking algorithm known as "**Semantic double-pass merging**". The initial phase of this algorithm bears resemblance to classical semantic chunking, which relies on mathematical metrics such as percentiles and standard deviations. What **distinguishes it is the incorporation of a second pass scan, enabling the merging of previously generated chunks into larger and more informative ones**. During the second pass scan, the algorithm examines two chunks ahead. If the current chunk exhibits adequate cosine similarity with the subsequent one, even if their similarity is low (potentially due to textual dissimilarity but still possessing semantic relevance), the algorithm will merge these three chunks—the current one and its two consecutive counterparts. This feature proves particularly advantageous for texts encompassing mathematical formulas, code/algorithms blocks or citations that may perplex traditional semantic chunking algorithms (which solely assess similarity between adjacent sentences).

Algorithms 1 and 2 are described as double-pass merging chunking in detail [15].

Algorithm 1: first pass merging Algorithm

```
Input:     Original documents
Parameter: initial_threshold, appending_threshold
           max_chunking_size
Output: chunks after first pass merging(sentence)
1: Split the text into sentences.
2: Calculate cosine similarity (c.s.) for the first two
   available sentences.
3: If (cosine similarity value > initial_threshold)
       merge those sentences into one chunk.
   Else
       The first sentence becomes a standalone chunk and
     return to step 2 with the second sentence and the
     subsequent one.
4: If(chunk size >= max_chunking_size)
       stop its growth and proceed to step 2 with the two
     following sentences.
5: Calculate cosine similarity between the last two
   sentences of the existing chunk and the next sentence.
6: If (cosine similarity value >= appending_threshold)
       add the next sentence to the existing chunk and return
   to step 4.
7: Finish the current chunk and return to step 2
```

Algorithm 2: second pass merging Algorithm

```
Input:     Original documents
Parameter: merging_threshold, max_chunking_size
Output: chunks after second pass merging(chunks)
1: Take the first two available chunks.
2: Calculate cosine similarity between those chunks.
3: If ( cosine similarity value > merging_threshold &&
       chunk_size_aftermerging < max_chunking_size)
        merge those two chunks, then take the next
        available chunk and return to step 2
   Else
        If the length exceed the limit then finish the current
     chunk and return to step 1 with second chunk used in
     that comparison and next available chunk. Elsewhere
     move to step 4
4: Take next available chunk and calculate cosine
   similarity between first examined chunk and the new
   (third in that examination) one.
5: If (cosine similarity value >= merging_threshold)
        Merge three chunks, ensuring that the length of
   these chunks does not exceed max_chunking_size. Then take
   the next available chunk and return to step 2. If the length
   does exceed the limit, then finish the current chunk and return
   to step 1 with the second and third chunks used in that
   comparison.
```

B. Evaluation Chunking

❷ In Part ❶, we have already segmented the external document into several chunks. As the first step in evaluating RAG chunking, we use LLM to generate several questions for each segmented chunk. The results of this step are shown in Figure 2, and we get n pairs of (chunk, queries), denoted as

$$P(i) = (chunk(i), \{Q(i,j) | j \in [1,m]\}), i \in [1,n]$$

Since Q(i,j) is generated by chunk(i), chunk(i) is the most relevant chunk to Q(i,j)

❸ We analyse $n * m$ queries, and for each query, we execute retrieval to find the top k most relevant chunks of total n chunks. For each query, we get a list of k most relevant chunks, where the position of an element represents its similarity with the query.

$$Q(i,j) = (Query(i,j), \{chunk(i,j,t) | t \in [1,k]\}), i \in [1,n], j \in [1,m]$$

$$Flag(Q(i,j)) = i$$

The flag represents that for Q(i,j), the most relevant chunk should be chunk(i). We use cosine similarity to find the top k most similar chunks for each query.

❹ We use Mean Reciprocal Rank (MRR) and Hit Rate to evaluate the results of chunking and retrieval. They can be calculated by $Q(i,j)$ and $Flag(Q(i,j))$.

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

$$Hit\ Rate@k = \frac{1}{|Q|}\sum_{i=1}^{|Q|}1\,(rank_i \leq k)$$

- $|Q|$: num of queries
- $rank_i$: rank position of the first relevant document for the ith query

Meanwhile, we show the Precision and recall rate in our experiments in chapter 4. Here $Flag(Q(i,j))$ is the TruePositive(TP) item for $Q(i,j)$.

C. Parameters

The parameters mentioned below are adjustable in our workflow.

Figure3 shows thresholds and max chunking size in chunking process

i)thresholds:

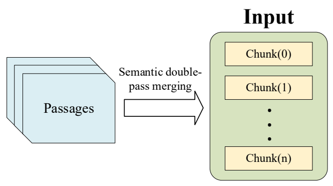
1)initial threshold: The similarity needed for initial sentences to form a new chunk. Higher values create more focused but smaller chunks.

2)appending threshold: The minimum similarity required to add sentences to an existing chunk. Higher values ensure cohesiveness but fewer additions.

3)merging threshold: The similarity level for merging chunks. Higher values consolidate related chunks but risk merging unrelated ones.

ii)max chunking size:

The maximum allowable length for a chunk. This parameter ensures chunks do not exceed a certain size, maintaining manageability. Also, for LLM prompt limitation.



```
splitter = SemanticDoubleMergingSplitterNodeParser(
        initial_threshold=0.4,
        appending_threshold=0.5,
        merging_threshold=0.5,
        max_chunk_size=5000,
)
```

Fig. 3. Parameters in double pass merging chunking(our default value in experiments)

iii) num_questions_per_chunk:

Figure 4 shows parameter num_questions_per_chunk in workflow. The number of questions generated per chunk, affecting the evaluation scope and detail of the chunk's content.

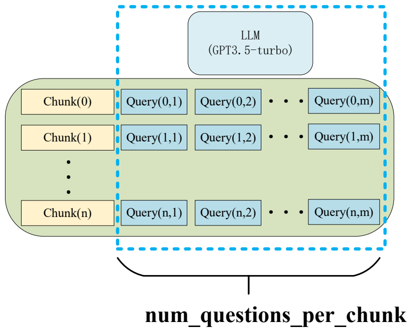


Fig. 4. Parameters: num_questions_per_chunk

iv)similarity_top_k:

Figure 5 shows parameter similarity_top_k in workflow. The number of top similar items considered during retrieval. This parameter impacts the precision and relevance of retrieved information.

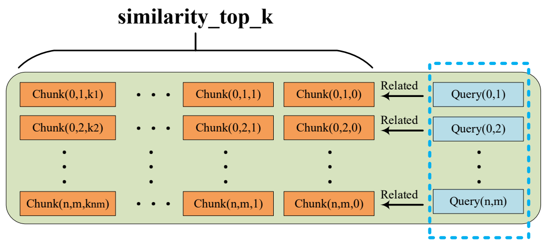


Fig. 5. Parameters: similarity_top_k

IV. Experiment and Results

Note that there are some random variations in the results when we execute this evaluation program, particularly because of an LLM used within it to generate queries for every chunk. The fluctuations caused by random factors in our evaluation process are shown in Figure 6. We use the same external documents and the same chunks, simply run the same evaluation program several times, and collect outputs. Consequently, we have noticed that running results do not fluctuate much with the same conditions, and these fluctuations are reasonable and affect around 8% of our results.

After analysing the fluctuation, we evaluate chunking results. Initially, we divided our input external document into sentences, resulting in over 1000 small chunks. In the next step, to investigate whether the order of these chunks affects the final outputs, we moved the first sentence to the end of the original text file and generated a new file. We repeated this process sequentially for each sentence in the original document. It is important to note that the relative order of each sentence remained unchanged after each movement.

Since there are over 1000 sentences, it is impractical to explore the patterns by rearranging the entire file's small chunks. Therefore, we chose to use random sampling in this situation. Specifically, we randomly selected 100 scenarios from the more than 1000 possible cases for analysis. We randomly picked 100 numbers within the total number of sentences to determine the positions of the sentences to be rearranged. The newly generated 100 external files were then input into our known large language model (GPT-3.5turbo), and we obtained the MMR and Hit-rate corresponding to these 100 samples.

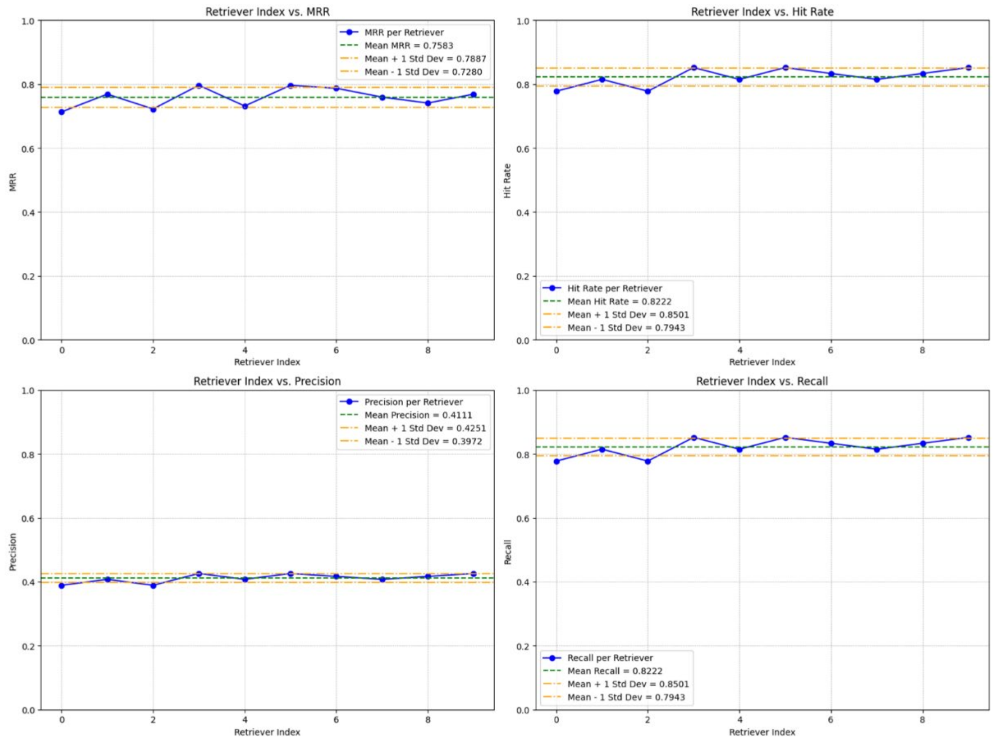


Fig. 6. Fluctuation of our evaluation(num_questions_per_chunk=2，similarity_top_k =2)

Figure 7 shows 100 sample reordering merge's evaluation results. Overall, no clear pattern emerged. However, when we focused on specific details, we noticed that many adjacent points had very similar statistical values. This suggests that during the merging process, the meanings of these sentences, or their overall context, are similar enough to be combined into a single chunk. Similarly, by attempting to rearrange all sentences, we can identify the method that maximizes accuracy, i.e., the method that yields the highest MRR and hit rate.

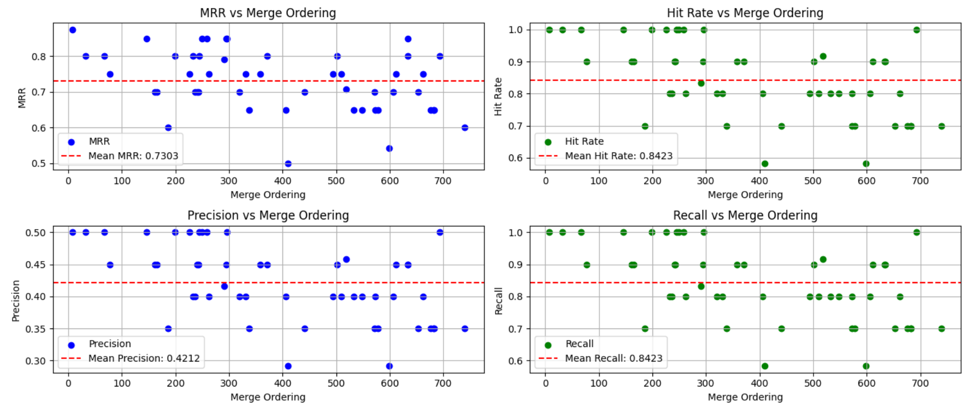


Fig. 7. Reorder merge Results(100 samples order, num_questions_per_chunk=2, similarity_top_k =2)

Meanwhile, we also created plots to represent the relationship between the parameters and the MRR/HitRate/Precision/Recall. We keep the merge thresholds and the input file the same, adjust the value of num_questions_per_chunk and similarity_top_k. Results showed as figure 8-figure 11.

From a mathematical perspective, both MRR and Hit Rate increase as the value of k increases. Our results support the theory.

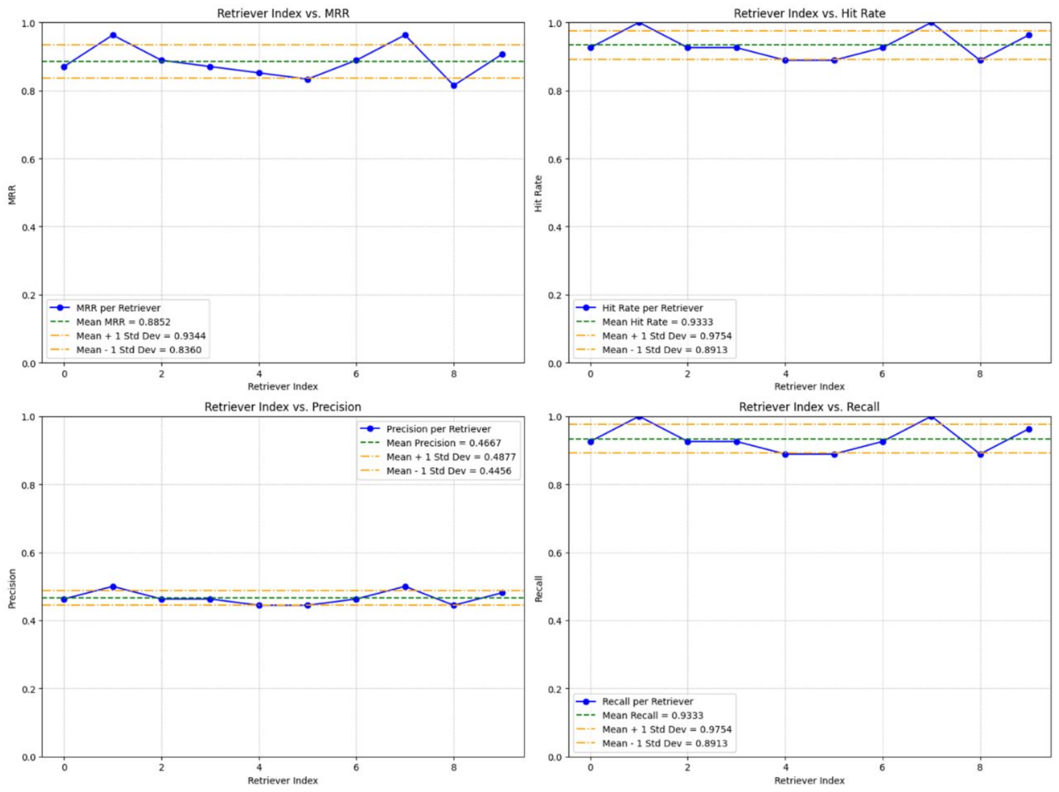


Fig. 8. Results: Affect of parameters(num_questions_per_chunk=1，similarity_top_k =2)

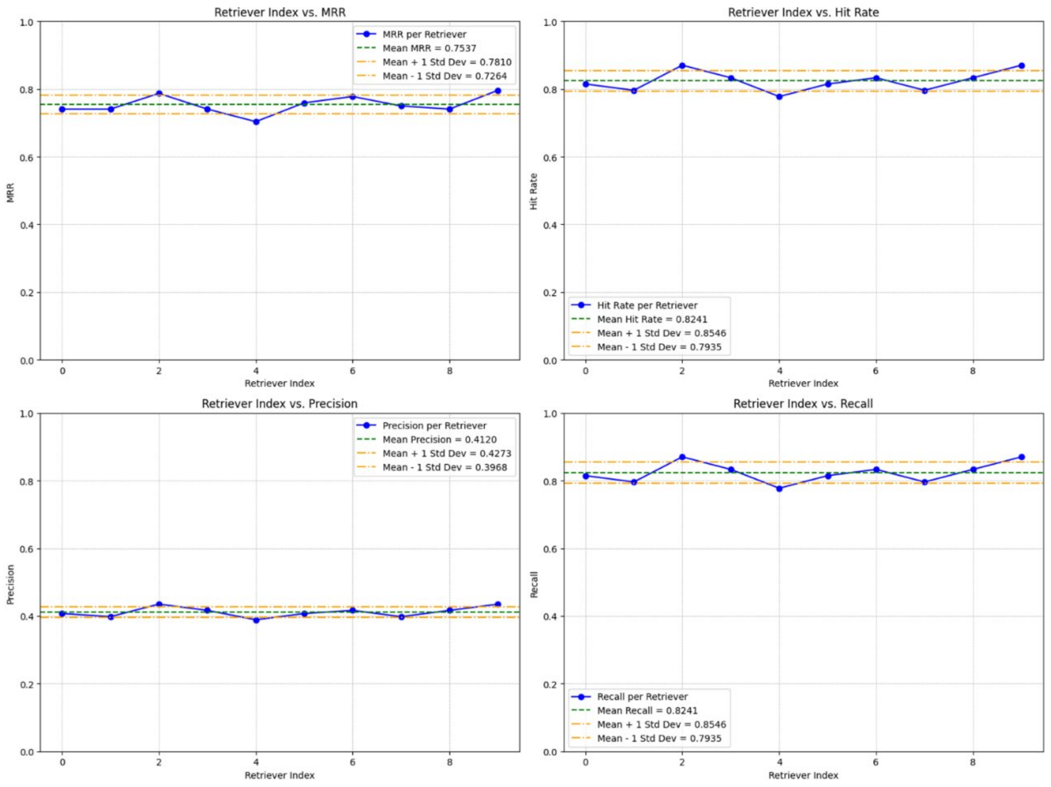


Fig. 9. Results: Affect of parameters(num_questions_per_chunk=2，similarity_top_k =1)

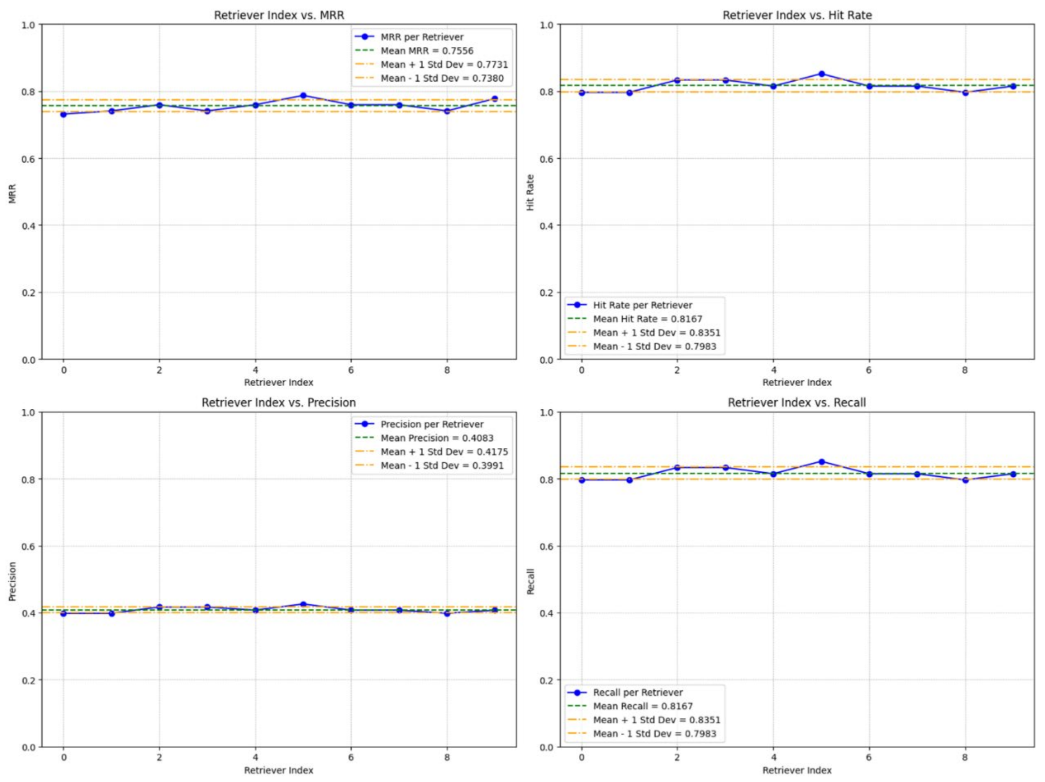


Fig. 10. Results: Affect of parameters(num_questions_per_chunk=2，similarity_top_k =3)

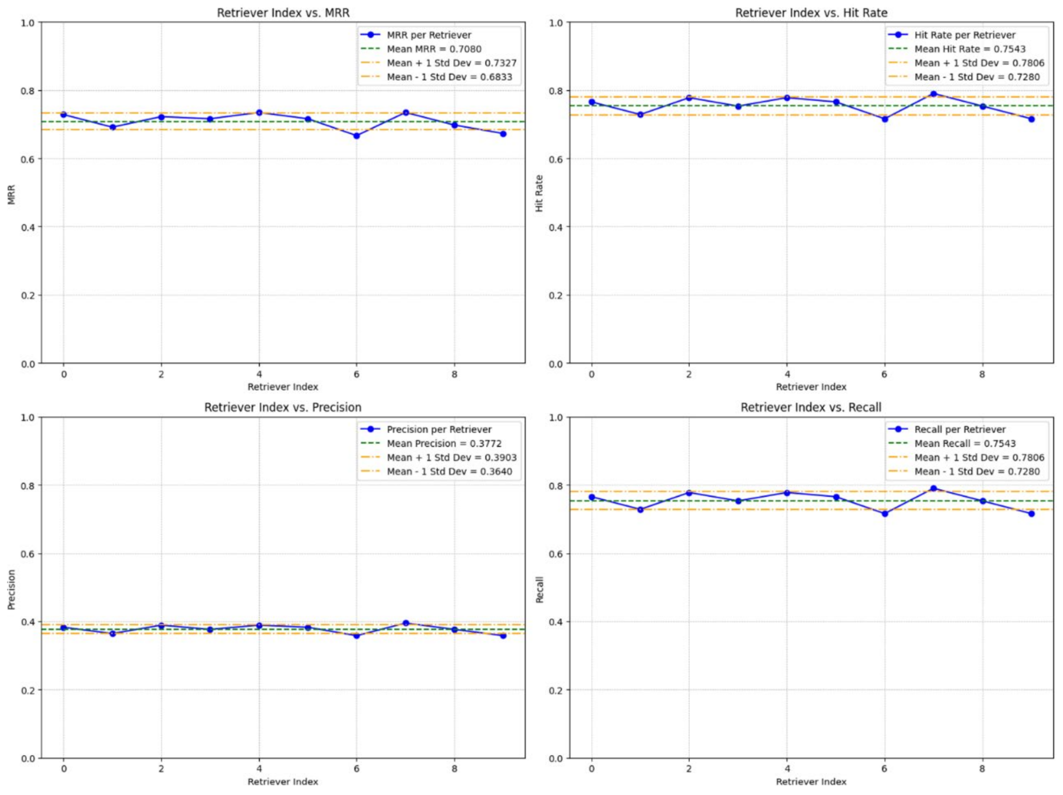


Fig.11. Results: Affect of parameters(num_questions_per_chunk=3，similarity_top_k =2)

The results(fig6-fig11) show that changing the merge order has an impact of about 100% (from 0.5 to close to 1) on the evaluation results. Meanwhile, its own fluctuation is only about 8%. From this, we are confident that the merge order has a significant impact on chunking.

We propose that preferentially merging the most semantically relevant sentences in external documents may yield better results. To verify this, we change the order of merge, modifying the Semantic double-pass merging algorithm of the first pass to merge the most semantically relevant sentences. Our modified algorithm is shown in algorithm 3.

We use different external documents to evaluate our improvement methods. We set the parameters to the same value, changed only the merge algorithm for the first pass, and tested it on a different external document (a series of different chunks). The results are shown in Figure 12.

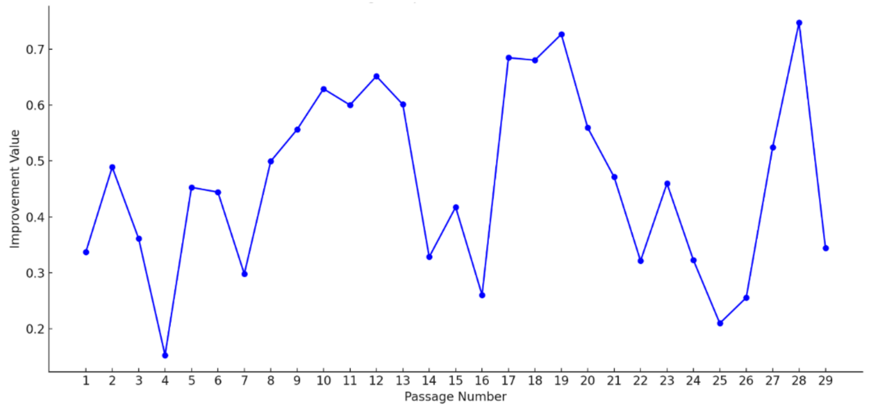


Fig.12. Results: Performance improvement from changing the merge order

It is clear that by modifying the algorithm in the first pass, RAG performance has improved by at least 15% on the test cases. This shows that the merge order has a significant impact on Chunking results, but we are not sure that this is the optimal merge order and chunking method.

```
Algorithm3: improved first pass merging Algorithm
Input:      Original documents
Parameter: initial_threshold,    appending_threshold
           max_chunking_size
Output: chunks after first pass merging
1:  Split the text into sentences.
2:  Calculate cosine similarity (c.s.) for all adjacent
    sentences
    Then find the most similar two sentences, start merging
    from the first one of them.
3:  Choose the two available two sentence, until go back to
    the start sentence.
4:  If (cosine similarity value > initial_threshold)
        merge those sentences into one chunk.
    Else
        The first sentence becomes a standalone chunk and
      return to step 3 with the second sentence and the
      subsequent one.
5:  If(chunk size >= max_chunking_size)
        stop its growth and proceed to step 3 with the two
      following sentences.
6:  Calculate cosine similarity between the last two
sentences of the existing chunk and the next sentence.
7:  If (cosine similarity value >= appending_threshold)
      add the next sentence to the existing chunk and return
    to step 4.
8:  Finish the current chunk and return to step 3
```

V. Future Work

The impact of different merge orders on chunking results needs to be further evaluated to determine the optimized merge order. Additionally, there are several intriguing avenues for future research.

1) Semantic Chunking Order (Especially in Semantic double merging chunking)

As mentioned in the paper, experiments are made with several chunking parameters, including thresholds, maximum chunking size, number of questions per chunk, and similarity top k, which is the number of similar chunks chosen during retrieval. Adjusting these parameters will also affect experiment results and overall performance. Thus, designing an algorithm based on those parameters to find the best merge ordering can be further discussed.

2) Graph RAG

Graph RAG is also a popular feature in the field of RAG; it focuses on transforming the external database into knowledge graphs. Therefore, the graph structure and algorithms can efficiently handle complicated relationships between data nodes and help understand the semantic meanings between contexts. However, transforming files to knowledge graphs is very expensive and requires a long time, and graphs need to be regenerated once the database is updated. Therefore, how efficiently create graphs needs to be further researched.

3) Vision Language Model and RAG

Currently, common RAG systems cannot capture the vision factors of files such as graphs, charts, and so on, so vision language models (VLM) are introduced to help RAG systems to be more accurate. Moreover, key research areas include improving the chunking and embedding of the visual factor and developing advanced indexing and retrieval techniques by VLM.

VI. Conclusion

In conclusion, chunking is the basic and important step in RAG, and chunking methods play a crucial role in this step. In this paper, we evaluate the semantic chunking method "Semantic double merging chunking", propose an improved merging order, and estimate the effect of merge order on chunking results. Moreover, the experiment results show that the merge order significantly influences both chunking results, LLM responses, and RAG performance, and our method can improve the performance of RAG by producing better chunks. However, how to find the optimal merge order remains an area requiring further research.

Acknowledgments

The authors made equal contributions to this work. I would like to express my great appreciation to Professor David P. Woodruff for his invaluable guidance and to all the esteemed authors for their significant contributions to this scholarly work

References

[1] Y. Zhang *et al.*, "Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models," Sep. 24, 2023, *arXiv*: arXiv:2309.01219. doi: 10.48550/arXiv.2309.01219.

[2] N. Kandpal, H. Deng, A. Roberts, E. Wallace, and C. Raffel, "Large Language Models Struggle to Learn Long-Tail Knowledge," Jul. 27, 2023, *arXiv*: arXiv:2211.08411. doi: 10.48550/arXiv.2211.08411.

[3] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Apr. 12, 2021, *arXiv*: arXiv:2005.11401. doi: 10.48550/arXiv.2005.11401.

[4] Y. Gao *et al.*, "Retrieval-Augmented Generation for Large Language Models: A Survey," Mar. 27, 2024, *arXiv*: arXiv:2312.10997. doi: 10.48550/arXiv.2312.10997.

[5] X. Ma, Y. Gong, P. He, H. Zhao, and N. Duan, "Query Rewriting for Retrieval-Augmented Large Language Models," Oct. 22, 2023, *arXiv*: arXiv:2305.14283. doi: 10.48550/arXiv.2305.14283.

[6] W. Peng *et al.*, "Large Language Model based Long-tail Query Rewriting in Taobao Search," Mar. 04, 2024, *arXiv*: arXiv:2311.03758. doi: 10.48550/arXiv.2311.03758.

[7] H. S. Zheng *et al.*, "Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models," Mar. 12, 2024, *arXiv*: arXiv:2310.06117. doi: 10.48550/arXiv.2310.06117.

[8] L. Gao, X. Ma, J. Lin, and J. Callan, "Precise Zero-Shot Dense Retrieval without Relevance Labels," Dec. 20, 2022, *arXiv*: arXiv:2212.10496. doi: 10.48550/arXiv.2212.10496.

[9] I. ILIN, "Advanced RAG Techniques: an Illustrated Overview," Medium. Accessed: Sep. 14, 2024. [Online]. Available: https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6

[10] BitPeak, "Chunking methods," Chunking methods. Accessed: Sep. 14, 2024. [Online]. Available: https://bitpeak.com/chunking-methods-in-rag-overview-of-available-solutions/

[11] Meta, "Retrieval Evaluation - LlamaIndex." Accessed: Sep. 14, 2024. [Online]. Available: https://docs.llamaindex.ai/en/stable/examples/evaluation/retrieval/retriever_eval/

[12] Trychroma, "Evaluating Chunking Strategies for Retrieval." Accessed: Sep. 14, 2024. [Online]. Available: https://research.trychroma.com/evaluating-chunking

[13] H. Yu, A. Gan, K. Zhang, S. Tong, Q. Liu, and Z. Liu, "Evaluation of Retrieval-Augmented Generation: A Survey," Jul. 03, 2024, *arXiv*: arXiv:2405.07437. doi: 10.48550/arXiv.2405.07437.

[14] Meta, "Evaluating the Ideal Chunk Size for a RAG System using LlamaIndex — LlamaIndex, Data Framework for LLM Applications." Accessed: Sep. 14, 2024. [Online]. Available: https://www.llamaindex.ai/blog/evaluating-the-ideal-chunk-size-for-a-rag-system-using-llamaindex-6207e5d3fec5

[15] BitPeak, "Semantic double-pass merging," Semantic double-pass merging. Accessed: Sep. 14, 2024. [Online]. Available: https://bitpeak.com/chunking-methods-in-rag-methods-comparison/