HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and Communication Technology

RETRIEVAL-AUGMENTED GENERATION AND THE IMPACT OF CHUNKING TECHNIQUES

Project I

744280 - IT3910E

Semester 2024.1

Bùi Nguyên Khải 20225501

Supervisor

Ph.D. Dang Tuan Linh

Hanoi, January 2025

Abstract

Retrieval-Augmented Generation (RAG) has emerged as a powerful framework that combines retrieval-based and generative models to address the limitations of purely generative systems, particularly in tasks requiring factual accuracy and contextual relevance. A critical factor influencing the performance of RAG systems is the choice of chunking technique used to preprocess source documents for retrieval. This study evaluates the impact of three prominent chunking strategies—Naive Chunking, Small2Big Chunking, and Proposition Chunking—on the performance of a RAG system applied to a question-answering task. Extensive experimentation highlights the trade-offs between these methods in terms of retrieval precision, response quality, and computational efficiency.

The results reveal that Naive Chunking offers a practical balance between simplicity and performance, making it ideal for real-time applications. Small2Big Chunking provides broader context at the cost of increased computational complexity, while Proposition Chunking excels in groundedness and precision but requires significant resources, making it more suitable for offline or domain-specific tasks. This study highlights the importance of aligning chunking strategies with task-specific requirements and underscores the critical role of preprocessing in optimizing RAG systems. The findings offer insights for improving the efficiency and effectiveness of retrieval-augmented frameworks across diverse applications.

Contents

1 Introduction

In the age of information overload, the demand for systems capable of generating accurate and contextually relevant responses has grown exponentially. Retrieval-Augmented Generation (RAG) [1] represents a transformative advancement in natural language processing, combining the strengths of retrieval-based and generative models to address the limitations of standalone generative approaches. By integrating external knowledge sources into the generation pipeline, RAG systems achieve enhanced factual accuracy, relevance, and contextual coherence, making them indispensable for applications such as question answering, summarization, and conversational agents.

A cornerstone of RAG systems is the preprocessing step known as chunking [2]. Chunking involves segmenting source documents into manageable units of text, which are then indexed and retrieved based on relevance to a user's query. The choice of chunking technique plays a pivotal role in determining the efficiency and effectiveness of the retrieval process. Overly large chunks can reduce relevance and overwhelm generative models with unrelated information [3], while overly small chunks may miss critical context, leading to fragmented or inaccurate responses. Despite its significance, the impact of chunking methods on RAG performance remains an area of active exploration.

This paper investigates three prominent chunking strategies—Naive Chunking [4], Small2Big Chunking [5, 6], and Proposition Chunking [7]-and evaluates their effects on the performance of a RAG system. Through comprehensive experiments on a question-answering task, this study assess the trade-offs these methods offer in terms of retrieval accuracy, response quality, and computational efficiency. The findings not only show the strengths and limitations of each technique but also provide practical insights for selecting chunking strategies tailored to specific application requirements.

By focusing on the relationship between chunking techniques and RAG performance, this study contributes to advancing the design and deployment of retrieval-augmented systems. It highlights the critical considerations for optimizing chunking strategies, ensuring RAG systems deliver accurate, grounded, and contextually rich outputs across diverse domains.

2 Retrieval-Augmented Generation Overview

Retrieval-Augmented Generation (RAG) is a hybrid framework that combines the strengths of retrieval-based and generative models to improve the quality and relevance of text generation tasks. Introduced in [1], RAG addresses the limitations of purely generative models, such as hallucination or lack of factual accuracy [8], by integrating external knowledge sources into the generation process. This approach is particularly effective for tasks requiring factual consistency, such as question answering, summarization, and dialogue systems [9] [10] [11].

RAG operates by dynamically retrieving relevant information from a large external corpus (e.g., Wikipedia or a domain-specific database) and using this information to condition the generation process. The framework consists of two main components: *the retriever* and *the generator*. These components work together to produce coherent and contextually accurate outputs.

2.1 Retriever

In RAG, the retriever plays a critical role in identifying and extracting relevant information from a corpus to support the generative model. The retriever ensures that the RAG system can access knowledge efficiently, enabling accurate and context-aware responses [12]. This section provides an overview of the key components of the retriever: source documents, chunks, embeddings, and the vector database.

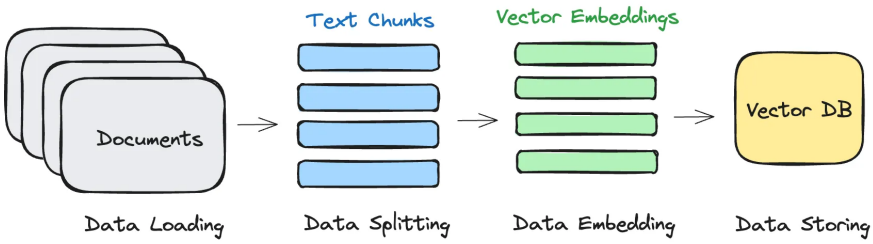

Figure 1: An illustration of the retriever framework. Source: [13].

2.1.1 Source documents

Source documents are the foundation of any retrieval system, serving as the primary knowledge base from which information is extracted. These documents can originate from diverse sources, including Wikipedia, search engines, or domain-specific databases [12]. The content must be preprocessed to ensure consistency, such as normalizing text formats, removing irrelevant metadata, and addressing language-specific nuances like tokenization or stemming [14].

2.1.2 Chunks

Chunks are smaller, manageable units of text derived from source documents. Chunking is essential for optimizing retrieval efficiency and accuracy, as smaller units allow the retriever to locate more granular and relevant information. The choice of chunking method can significantly influence retrieval performance, as overly large chunks may dilute relevance, while overly small chunks might miss necessary context [15]. Various chunking techniques will be explored in detail in Section 3.

2.1.3 Embeddings

Embeddings are vectorized representations of texts that encode semantic meaning, enabling the retriever to compare and rank the relevance of chunks [14]. Typically, pre-trained transformer-based models like BERT [16], models using NLP techniques like Word2Vec [17], or unsupervised models like GloVe [18] are used to generate these embedded vectors. Fine-tuning embedding models on domain-specific data can further improve retrieval quality [19]. Embedding vectors are often high-dimensional, capturing nuanced relationships between words, phrases, and concepts, and are optimized for similarity comparisons using metrics like cosine similarity or Euclidean distance [20]. However, [21] has shown that cosine similarity may not always be the most effective metric for certain tasks and should be used with caution.

2.1.4 Vector Database

A vector database serves as the infrastructure for storing and querying embeddings. It is optimized for similarity search, allowing efficient retrieval of the most relevant chunks based on their embeddings. Popular vector databases include FAISS [22], Pinecone [23], Weaviate [24], and Milvus [25, 26], each offering features like GPU support, scalability, approximate nearest neighbor (ANN) search, and integration with machine learning pipelines. The database indexes embedding vectors and enables fast lookups even in large-scale corpora, making it a cornerstone of effective RAG systems.

2.2 Generator

The generator in RAG is responsible for producing coherent and contextually accurate responses by leveraging both the user query and retrieved knowledge. It serves as the final step in the RAG pipeline, combining retrieved information with the user input to create enhanced outputs.

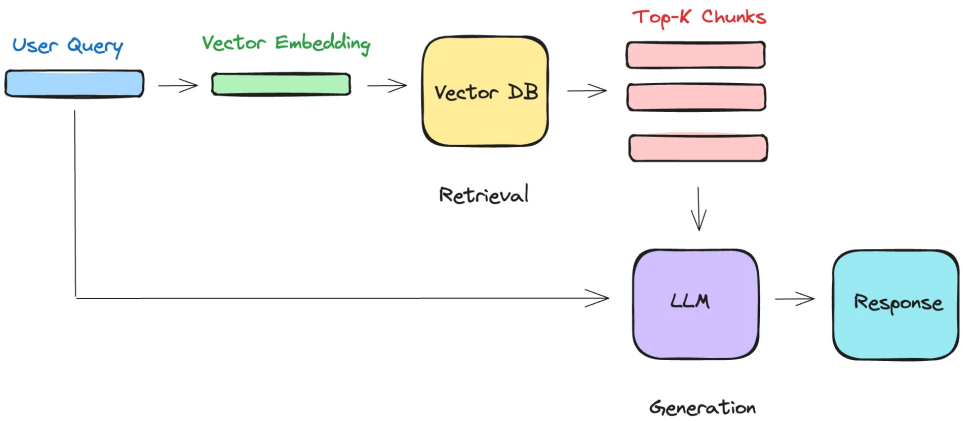

Figure 2: An illustration of the generator framework. Source: [13].

2.2.1 User Query

The generation process begins with the user's query, which is first preprocessed to ensure compatibility with the embedding model. Preprocessing may include tokenization, lowercasing, and removing wasteful characters [14]. The preprocessed query is then transformed into an embedding using the same model and techniques applied to the source document chunks. This ensures semantic alignment between the query and the knowledge base.

2.2.2 Similarity Search

The query embedding is used to perform a similarity search within the vector database, identifying the top-k most relevant chunks. These chunks, ranked based on their similarity scores to the query, represent the knowledge most relevant to the user's input [27]. The selection of k (e.g., top-5 or top-10) is a critical parameter that balances retrieval precision and the comprehensiveness of context.

2.3 Generation

The retrieved chunks are concatenated with the original user query to form a composited input for the large language model (LLM). This input construction ensures that the generative model has access to both the specific context provided by the user and the broader knowledge retrieved from the corpus. Prompt engineering techniques [28] are employed to effectively structure this input, ensuring that the query and the retrieved chunks are presented in a way that maximizes the model's ability to generate accurate and contextually relevant responses.

The composite prompt is then passed to the LLM, which processes it to generate a response. The LLM leverages its pretrained capabilities and the augmented input to produce outputs that are both contextually enriched and aligned with the user's intent. Fine-tuning the LLM on task-specific data can further enhance its performance, ensuring the generated responses meet the desired accuracy and relevance standards [29].

3 Chunking Methods

Chunking is a critical preprocessing step in RAG systems, as it directly impacts the efficiency and accuracy of information retrieval. The choice of chunking method is crucial, as overly large chunks may reduce relevance, while overly small chunks might miss necessary context. In this section, three main chunking methods will be explored: *Naive Chunking*, *Propositions Chunking*, and *Small2Big Chunking*. Each method has its unique approach to segmenting text, and their effectiveness varies depending on the task and the nature of the data.

3.1 Naive Chunking

Naive Chunking is one of the most commonly used methods for segmenting text in RAG systems. In Naive Chunking, the text is split into chunks of a fixed size, typically measured in tokens, such as 256, 512, or 1024 tokens [4]. This approach ensures that each chunk is of uniform size, which simplifies the retrieval process. However, the fixed size can sometimes lead to chunks that cut off sentences or ideas mid-way, potentially losing important context.

To mitigate the issue of losing context at chunk boundaries, chunking overlap [30] is often employed. Overlap involves allowing adjacent chunks to share a portion of their content. For instance, if the chunk size is 256 tokens, an overlap of 50 tokens means that the last 50 tokens of one chunk will also appear as the first 50 tokens of the next chunk. This overlap ensures that important context is preserved across chunk boundaries, reducing the risk of losing critical information.

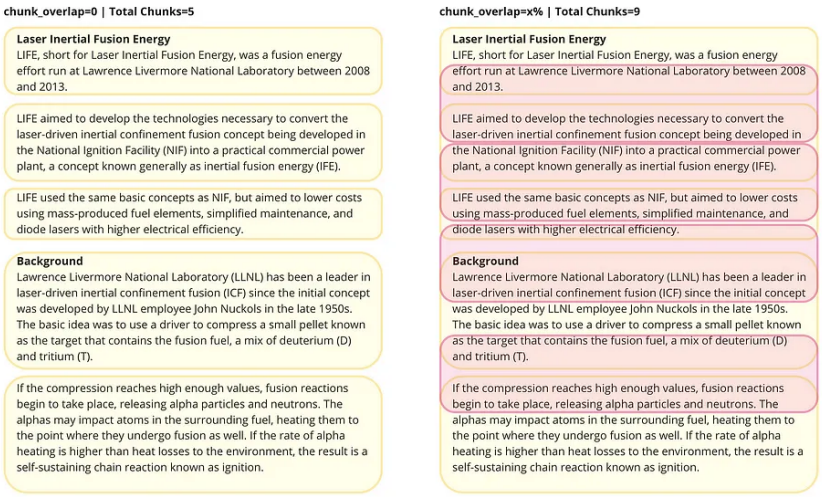

Figure 3: An illustration of chunk overlap. Source: [30].

The simplicity and ease of implementation make Naive Chunking a popular choice, especially in scenarios where computational efficiency is a priority.

3.2 Small2Big Chunking

The Small2Big Chunking method [5], also known as Parent Document Retriever [6], addresses a critical challenge in RAG systems: the trade-off between precision and context.

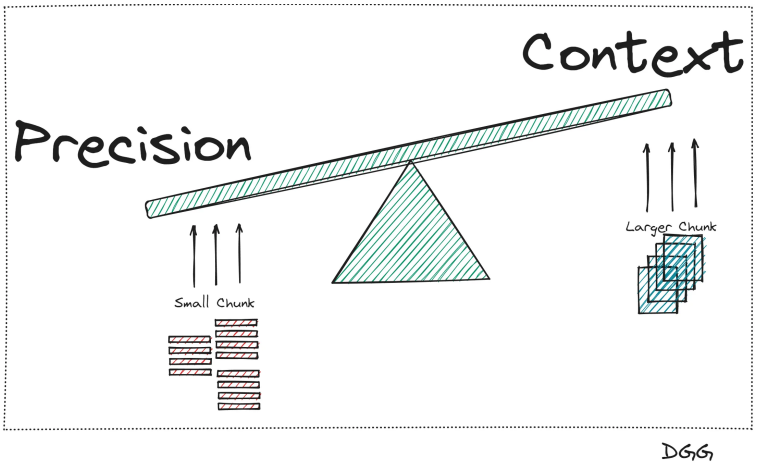

Figure 4: Representation of the balance between these two concepts. Source: [6].

While smaller chunks allow for more precise retrieval of relevant information, they often lack the broader context necessary for the language model (LLM) to generate coherent and accurate responses. Conversely, larger chunks provide more context but may reduce the relevance of the retrieved information [15]. Small2Big Chunking elegantly resolves this issue by combining the benefits of both fine-grained and coarse-grained chunking.

Small2Big Chunking solves this problem by introducing a hierarchical approach to chunking. The method involves two levels of chunking: parent chunks (larger, coarse-grained chunks) and child chunks (smaller, fine-grained chunks).

The key idea is to first retrieve the most relevant child chunks, which contain precise information, and then return the corresponding parent chunks to provide the necessary context. This dual-level approach ensures that the retriever can locate specific details while still maintaining the broader context required for accurate generation.

Step-by-Step Process:

1. **Create Parent Chunks:**
    - The first step is to segment the source documents into larger, *parent chunks*. These chunks are designed to capture broader context and are typically larger than the child chunks. For example, a parent chunk might consist of several paragraphs or an entire section of a document.
2. **Split Parent Chunks into Child Chunks:**
    - Once the parent chunks are created, they are further divided into smaller child chunks. These child chunks are designed to capture specific pieces of information. For instance, a child chunk might consist of a single sentence or a small group of sentences that convey a distinct fact or idea.
3. **Store Child Chunks in the Vector Store:**
    - The child chunks are then embedded and stored in a vector store. This allows the retriever to efficiently search for and retrieve the most relevant child chunks based on the input query.

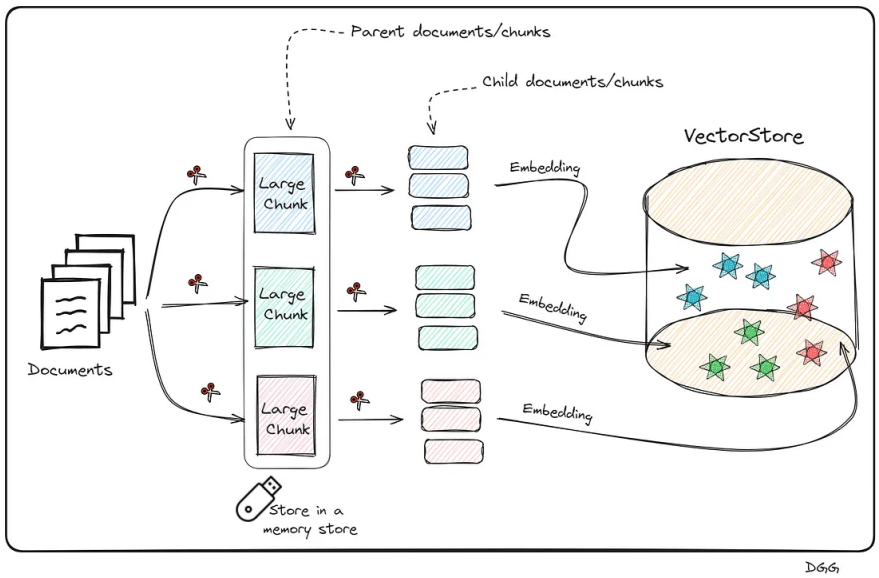

Figure 5: Visual representation of how child chunks are created from parent chunks, and their storage. Source: [6].

4. **Store Parent Chunks in Memory:**
    - The parent chunks are stored in memory, but they are not encoded into vector representations. Instead, they are kept as-is to provide context when needed. This approach reduces the computational overhead associated with encoding large chunks while still ensuring that the necessary context is available.

5. **Retrieve and Combine:**

   - During retrieval, the system first identifies the most relevant child chunks from the vector store. Once the top K child chunks are retrieved, the system maps them back to their corresponding parent chunks. The parent chunks are then used to provide the broader context needed for the LLM to generate accurate and coherent responses.

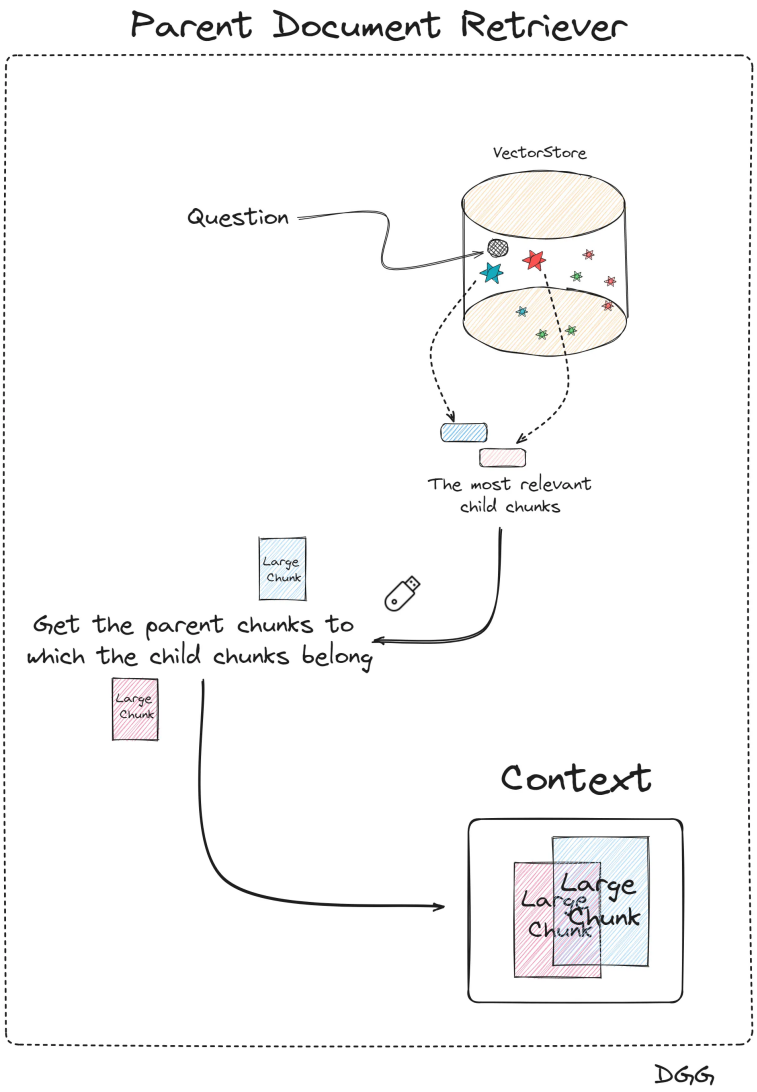

Figure 6: Visual representation of how a Parent Document Retriever works. Source: [6].

By combining fine-grained child chunks with coarse-grained parent chunks, Small2Big Chunking achieves a balance between precision and context. The child chunks allow for precise retrieval of relevant information, while the parent chunks ensure that the broader context is preserved. Moreover, storing only the child chunks in the vector store reduces the computational cost of encoding and searching large chunks. The parent chunks, which are stored in memory, provide context without the need for additional encoding.

3.3 Propositions Chunking

Propositions Chunking [7] addresses the shortcomings of typical retrieval units such as passages or sentences. While passages and sentences are commonly used in retrieval tasks, they often fail to capture the precise, self-contained units of meaning required for accurate and efficient retrieval. Sentences, for example, can still be complex and compounded, and they are often not self-contained, lacking necessary contextual information. This can lead to retrieval results that are either too broad or too fragmented, reducing the overall effectiveness of the retrieval process. This makes Propositions Chunking particularly effective in tasks such as domain-specific question answering, fact verification, and other information retrieval tasks where precision and context are critical.

[7] defined a proposition as an atomic expression within text that encapsulates a distinct fact or idea. Each proposition is minimal, meaning it cannot be further split into separate propositions, and it is self-contained, meaning it includes all the necessary context to interpret its meaning independently of the surrounding text. Propositions are inspired by cognitive psychology, where they are considered the building blocks of thought and memory.

By breaking down text into atomic propositions, Propositions Chunking allows for highly precise retrieval of relevant information. Each proposition represents a single, distinct fact or idea, making it easier for the retriever to locate specific pieces of information. Unlike sentences, which may rely on external context to convey their full meaning, propositions are designed to be self-contained. This ensures that each proposition includes all the necessary context, such as coreferences or background information, to interpret its meaning independently. Propositions Chunking reduces the noise often associated with larger retrieval units like passages or sentences. By focusing on atomic units of meaning, the retriever can avoid irrelevant information, leading to more accurate and relevant retrieval results.

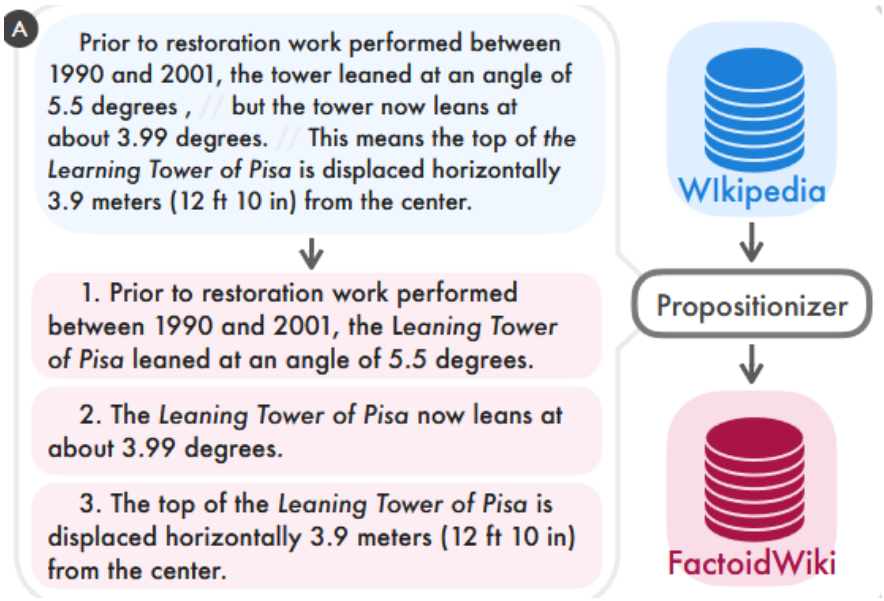

Figure 7: Visual representation of how a Propositionizer works. Source: [7].

To create propositions from text, [7] fine-tuned the flan-t5-large model [31] specifically for propositional tokenization. This fine-tuned model, referred to as the Propositionizer, is designed to parse text into atomic propositions. The pre-trained Propositionizer takes a passage of text as input and outputs a list of propositions.

4 Experiments and Results

This section will present the experimental setup and methodology used to evaluate the impact of different chunking techniques (Naive Chunking, Propositions Chunking, and Small2Big Chunking) on the performance of a RAG question answering system.

4.1 Models and Tools

LLM *Qwen2.5-7B-Instruct* [32], a state-of-the-art model among those with fewer than 7 billion parameters [33], was employed for generating responses based on the retrieved context [34]. The model was quantized using *Q5_K_S* configuration [35] to optimize memory usage and computational efficiency while maintaining high performance.

Embedding Model For encoding text into vector representations, *mxbai-embed-large-v1* [36, 37, 34] was employed, known for its high performance in semantic retrieval tasks. The embedding model was quantized using *Q5_K_M* configuration [38] to balance precision and computational efficiency.

Vector Store *FAISS (Facebook AI Similarity Search)* [22, 34] was used as the vector store for efficient indexing and retrieval of vectorized chunks. Its efficiency in managing and searching through high-dimensional vectors makes it an ideal choice for RAG applications, enabling rapid retrieval of relevant information to augment generative models. FAISS vector store is naively configured to retrieve top-5 chunks for all methods.

Source Documents The source documents for retrieval consisted of the scraped page contents [39] from top 2 Wikipedia articles [40, 34] and top 3 Google search results [41] relevant to the input queries. These documents ensure a diverse and comprehensive knowledge base for the RAG system.

4.2 Chunking Configurations

Naive Chunking For Naive Chunking [34], documents were segmented into fixed-size chunks of *2000 tokens*, with an overlap of *200 tokens* between adjacent chunks.

Small2Big Chunking For this method [34], documents were first divided into *parent chunks* of *10000 tokens*, with an overlap of *1000 tokens*. Each parent chunk was then further segmented into smaller *child chunks* of *400 tokens*, with an overlap of *40 tokens*. The child chunks allow for precise retrieval of specific information, while the parent chunks provide the broader context needed for accurate and coherent generation.

Propositions Chunking For Propositions Chunking, the pre-trained *propositionizer-wiki-flan-t5-large* model from [7] were utilized. In cases where the propositionizer failed to generate propositions [42], the method fell back to sentence-level chunking as a backup [2].

4.3 Evaluation methods

To evaluate the performance of the RAG system under different chunking techniques, an extensive evaluation framework was employed. This framework combines both automated metrics and human-like judgment using *LLM-*

as-a-judge [43] to assess the quality of the generated responses. This section will dive into the dataset, metrics, and evaluation criteria used in the experiments.

4.3.1 Dataset

100 question-answer pairs were randomly sampled from *Stanford/web_questions* [44] for evaluation. This dataset consists of question-short answer pairs derived from web queries, making it well-suited for testing the system's ability to retrieve and generate accurate responses from diverse and real-world knowledge sources.

4.3.2 Metrics

To assess the performance of the RAG system, a combination of automated metrics and rubric-based scoring were used. The metrics are designed to measure both the quality of the generated responses and the relevance of the retrieved information.

ROUGE

ROUGE-L (Recall-Oriented Understudy for Gisting Evaluation - Longest Common Subsequence) [45] measures the overlap between the generated response and the ground truth answer. It focuses on the longest common subsequence of words, capturing the semantic similarity between the two texts. A higher ROUGE-L score indicates that the response closely matches the ground truth answer.

LLM-as-a-Judge metrics

To ensure consistent and objective evaluation, an LLM fine-tuned for evaluation (quantized *prometheus-7b-v2.0*) [46, 47] was used as a judge to score the generated responses based on the rubric criteria. The LLM was provided with detailed instructions to ensure accurate and fair scoring.

Four key metrics were defined: *Correctness*, *Relevance*, *Groundedness*, and *Retrieval Relevance*. Each metric serves a distinct purpose in assessing the quality of the generated responses and the effectiveness of the retrieval process.

Correctness Measures how similar or correct the model's response is relative to the reference answer. This metric evaluates the factual accuracy of the generated response, ensuring that the system provides correct and reliable information.

Relevance Assesses how well the generated response addresses the initial user input or query. This ensures that the system's responses are directly relevant to the user's query, avoiding off-topic or tangential information.

Groundedness Evaluates the extent to which the generated response agrees with or is supported by the retrieved context. This metric ensures that the system's responses are well-grounded in the retrieved information, minimizing unsupported or speculative claims.

Retrieval Relevance Measures how relevant the retrieved documents or chunks are to the input query. This metric evaluates the quality of the retrieval process, ensuring that the system retrieves information that is both relevant and sufficient for generating accurate responses.

4.4 Results

All experiments were conducted on the Kaggle platform using 2 x T4 GPUs. Four methods will be compared: No RAG, Naive RAG, Small2Big RAG, and Proposition RAG. The results include metrics such as ROUGE-L, Correctness, Groundedness, Relevance, Retrieval Relevance, Average Documents Length, and Time taken to answer 100 questions. All rubric-based metrics (Correctness, Groundedness, Relevance, and Retrieval Relevance) were scored on a 1 to 5 scale, where 1 indicates poor performance and 5 indicates excellent performance.

Performance Comparison of Different RAG Chunking Methods

Method	Average ROUGE-L	Average Correctness	Average Groundedness	Average Relevance	Average Retrieval Relevance	Average Context Length	Total time taken
No RAG	0.29	1.56	1.08	3.62	2.83	8881.15	9 minutes
Naive RAG	**0.36**	**2.14**	1.23	**4.08**	**2.98**	8848.48	14 minutes
Small2Big RAG	0.31	**2.08**	1.12	3.65	2.64	13398.68	21 minutes
Proposition RAG	**0.34**	1.88	**1.81**	3.9	**2.99**	346.45	367 minutes

The analysis of results reveals distinct performance characteristics across the different RAG methods. The No RAG baseline, which relies solely on the language model without retrieval augmentation, achieved the lowest scores across all metrics, with a ROUGE-L score of 0.29, Correctness of 1.56, and Groundedness of 1.08. This highlights the limitations of the model in generating responses without external context, as the Relevance score of 3.62 indicates that while responses were somewhat relevant, they lacked depth and support. However, the absence of a retrieval process allowed it to answer 100 questions in just 9 minutes, making it the fastest but least effective method.

Naive RAG demonstrated significant improvements over the baseline, with a ROUGE-L score of 0.36, Correctness of 2.14, and Relevance of 4.08. The inclusion of retrieved context slightly improved Groundedness to 1.23, and the Retrieval Relevance score of 2.98 indicates that the retrieved documents were moderately relevant. The time taken increased to 14 minutes, reflecting the additional overhead of the retrieval process. This method strikes a balance between performance and efficiency, making it suitable for tasks requiring quick and relevant responses.

Small2Big RAG, which employs hierarchical chunking, achieved a ROUGE-L score of 0.31, Correctness of 2.08, and Relevance of 3.65. While these scores are slightly lower than Naive RAG, the system retrieved significantly more extensive context, as indicated by the high Average Context Length of 13398.68. However, this broader context came at the cost of precision, with a Retrieval Relevance score of 2.64. The time taken increased further to 21 minutes, reflecting the added complexity of hierarchical retrieval. This method provides a balance between context and precision but requires more resources compared to Naive RAG.

Proposition RAG, which uses fine-grained propositions, achieved a ROUGE-L score of 0.34, Correctness of 1.88, and the highest Groundedness score of 1.81. The Relevance score of 3.90 and Retrieval Relevance score of 2.99 demonstrate that the fine-grained

approach improved the relevance and support of responses. However, the computational complexity of generating propositions resulted in a significantly higher time of 367 minutes for 100 questions. Since each chunk is only a proposition, it's context length is much less than other chunking methods such as fixed-sized chunks or parent document chunks. As a result, the hyperparameter for the top-k chunks needs to be further tuned to ensure optimal performance. While this method excels in generating well-supported responses, its high computational cost makes it less suitable for real-time applications.

In summary, Naive RAG emerges as the most practical choice for tasks requiring quick and relevant responses, while Proposition RAG offers superior Groundedness at the cost of computational efficiency. Small2Big RAG provides a middle ground with broader context but requires more time and resources. The No RAG baseline underscores the importance of retrieval augmentation, as it consistently underperformed across all metrics, highlighting the critical role of external knowledge in enhancing system performance.

5 Discussion

This experiment provided valuable insights into the trade-offs and practical considerations of different RAG chunking methods. However, several limitations and challenges must be acknowledged to contextualize the findings and guide future work.

Dataset and Evaluation Limitations

The use of the Stanford Web Answers dataset introduces potential biases, as the online user's answers may not always reflect a true ground truth. This limitation is particularly relevant for evaluating systems like RAG, where the quality of retrieved information directly impacts response accuracy. Additionally, the reliance on LLM-as-a-judge (prometheus-7b-v2.0) for evaluation raises concerns about the reliability of the metrics [48]. While LLM-based evaluation is scalable, it may not consistently align with human judgment, especially for nuanced aspects like correctness and groundedness [49]. Future work should explore hybrid evaluation methods that combine automated metrics with human assessment to ensure more robust and trustworthy results.

Challenges with Small2Big Chunking

The Small2Big approach, which employs hierarchical chunking, presents significant challenges in terms of hyper-parameter tuning. Parameters such as parent chunk size, child chunk size, and chunk overlap require careful optimization to balance context breadth and precision. Moreover, the large document lengths generated by this method can strain some LLMs, making it difficult to process and utilize the retrieved information effectively [3]. While Small2Big is well-suited for tasks requiring extensive context, such as content creation and summarization, its complexity and resource demands limit its practicality for real-time applications.

Computational Complexity of Proposition Chunking

Proposition Chunking excels in generating fine-grained, well-supported responses, as evidenced by its high groundedness scores. However, its computational complexity makes it unsuitable for online retrieval scenarios. The process of generating propositions in real-time is resource-intensive, and the resulting short document lengths (due to the fine-grained nature of propositions) may limit the depth of retrieved information. Additionally, hyperparameter tuning for retrieving the top-k chunks adds another layer of complexity, requiring careful optimization to achieve optimal performance. This approach is better suited for offline use cases, such as domain-specific question answering, where propositions can be precomputed and stored in a database, and the top-k chunk hyperparameter can be optimized in advance [7].

Practicality of Naive Chunking

In contrast, Naive Chunking stands out for its simplicity and ease of use. It requires minimal hyperparameter tuning and performs well out-of-the-box, making it a practical choice for applications requiring online retrieval, such as conversational agents and chatbots. While it may not achieve the same level of groundedness or context breadth as Proposition Chunking or Small2Big, its efficiency and reliability make it a strong candidate for real-time applications such as Open-domain question answering [50, 51].

Future Directions

Future research should focus on addressing the limitations identified in this study. This includes developing more robust evaluation frameworks, optimizing hyperparameter tuning for complex chunking techniques, and exploring hybrid approaches that combine the strengths of different methods. Additionally, advancements in computational efficiency, particularly for Proposition Chunking, could expand its applicability to real-time use cases. By addressing these challenges, RAG systems can be further refined to meet the diverse needs of practical applications.

In conclusion, while each chunking technique has its strengths and weaknesses, the choice of method should be tailored to the specific requirements of the task at hand. By understanding these trade-offs, practitioners can make informed decisions to optimize the performance and efficiency of RAG systems in various domains.

6 Conclusion

This study highlights the critical role that chunking techniques play in the effectiveness and efficiency of Retrieval-Augmented Generation (RAG) systems. The comparative analysis of Naive Chunking, Small2Big Chunking, and Proposition Chunking provides valuable insights into their respective strengths, limitations, and practical applications.

Naive Chunking, with its simplicity and minimal computational overhead, is well-suited for real-time applications, offering a balanced trade-off between performance and efficiency. In contrast, Proposition Chunking excels in precision and groundedness, making it ideal for offline scenarios where fine-grained, contextually accurate responses are prioritized, albeit at the cost of significant computational complexity. Small2Big Chunking bridges the gap by combining precision and context, though its hyperparameter sensitivity and resource requirements limit its practicality for real-time use.

The choice of RAG chunking methods should be guided by the specific requirements of the application. For domain-specific question answering, where offline databases are available, Proposition Chunking can be highly effective due to its ability to generate precise, well-supported responses. For content creation and summarization, which require large amounts of context, Small2Big may be more appropriate despite its complexity. Finally, for conversational agents and chatbots, where real-time retrieval is essential, Naive Chunking offers a balance of performance and practicality.

Future work should aim to refine these chunking methods, addressing computational challenges and optimizing hyperparameters for broader applicability. Furthermore, advancements in evaluation methodologies, incorporating both automated and human assessments, are necessary to better capture nuanced performance differences across chunking techniques. By addressing these directions, RAG systems can be enhanced to deliver robust, efficient, and contextually rich responses across diverse use cases.

References

[1] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[3] Nelson F. Liu et al. *Lost in the Middle: How Language Models Use Long Contexts*. 2023. arXiv: 2307.03172 [cs.CL]. URL: https://arxiv.org/abs/2307.03172.

[4] Ravi Theja. *Evaluating the Ideal Chunk Size for a RAG System Using LlamaIndex*. 2023. URL: https://www.llamaindex.ai/blog/evaluating-the-ideal-chunk-size-for-a-rag-system-using-llamaindex-6207e5d3fec5.

[5] Sophia Yang. *Advanced RAG 01: Small to Big Retrieval*. 2023. URL: https://towardsdatascience.com/advanced-rag-01-small-to-big-retrieval-172181b396d4.

[6] Damian Gil. *Advanced Retriever Techniques to Improve Your RAGs*. 2023. URL: https://towardsdatascience.com/advanced-retriever-techniques-to-improve-your-rags-1fac2b86dd61.

[7] Tong Chen et al. "Dense X Retrieval: What Retrieval Granularity Should We Use?" In: *arXiv preprint arXiv:2312.06648* (2023). URL: https://arxiv.org/pdf/2312.06648.pdf.

[8] Gary Marcus. "The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence". In: *arXiv preprint arXiv:2002.06177* (2020). URL: https://arxiv.org/abs/2002.06177.

[9] Salman Rakin et al. "Leveraging the Domain Adaptation of Retrieval Augmented Generation Models for Question Answering and Reducing Hallucination". In: *arXiv preprint arXiv:2410.17783* (2024). URL: https://arxiv.org/abs/2410.17783.

[10] Darren Edge et al. "From Local to Global: A Graph RAG Approach to Query-Focused Summarization". In: *arXiv preprint arXiv:2404.16130* (2024). URL: https://arxiv.org/abs/2404.16130.

[11] Xin Cheng et al. "Lift Yourself Up: Retrieval-augmented Text Generation with Self Memory". In: *arXiv preprint arXiv:2305.02437* (2023). URL: https://arxiv.org/abs/2305.02437.

[12] Shangyu Wu et al. "Retrieval-Augmented Generation for Natural Language Processing: A Survey". In: *arXiv preprint arXiv:2407.13193* (2024). URL: https://arxiv.org/pdf/2407.13193.

[13] Julija Lazareva. *What is Retrieval-Augmented Generation (RAG)?* 2023. URL: https://medium.com/@drjulija/what-is-retrieval-augmented-generation-rag-938e4f6e03d1.

[14] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. 3rd. Upper Saddle River, NJ: Pearson, 2023.

[15] Akash. *The Impact of Chunk Sizes on Semantic Retrieval Results*. 2023. URL: https://medium.com/@csakash03/the-impact-of-chunk-sizes-on-semantic-retrieval-results-7fc721e2ec8a.

[16] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint* arXiv:1810.04805 (2019). URL: https://arxiv.org/pdf/1810.04805v2.

[17] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv preprint* arXiv:1301.3781 (2013). URL: https://arxiv.org/pdf/1301.3781.

[18] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162/.

[19] Phil Schmid. *Fine-tune Embedding Model for RAG*. 2023. URL: https://www.philschmid.de/fine-tune-embedding-model-for-rag.

[20] Frederik vom Lehn. *Understanding Vector Similarity*. 2023. URL: https://medium.com/advanced-deep-learning/understanding-vector-similarity-b9c10f7506de.

[21] Harald Steck, Chaitanya Ekanadham, and Nathan Kallus. "Is Cosine-Similarity of Embeddings Really About Similarity?" In: *arXiv preprint* arXiv:2403.05440 (2024). URL: https://arxiv.org/pdf/2403.05440.

[22] Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-scale similarity search with GPUs". In: *IEEE Transactions on Big Data* 7.3 (2019), pp. 535–547.

[23] Inc. Pinecone Systems. *Pinecone: Vector Database for Machine Learning*. 2023. URL: https://www.pinecone.io.

[24] Etienne Dilocker et al. *Weaviate*. URL: https://github.com/weaviate/weaviate.

[25] Jianguo Wang et al. "Milvus: A Purpose-Built Vector Data Management System". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2614–2627.

[26] Rentong Guo et al. "Manu: a cloud native vector database management system". In: *Proceedings of the VLDB Endowment* 15.12 (2022), pp. 3548–3561.

[27] Facebook AI Research. *FAISS: A Library for Efficient Similarity Search and Clustering of Dense Vectors*. 2023. URL: https://faiss.ai.

[28] Albert Ziegler and John Berryman. *A developer's guide to prompt engineering and LLMs*. 2023. URL: https://github.blog/ai-and-ml/generative-ai/prompt-engineering-guide-generative-ai-llms/.

[29] Chen Zhu et al. "Modifying Memories in Transformer Models". In: *arXiv preprint arXiv:2012.00363* (2020). URL: https://arxiv.org/pdf/2012.00363.

[30] Gustavo Espindola. *Chunk Division and Overlap: Understanding the Process*. 2023. URL: https://gustavo-espindola.medium.com/chunk-division-and-overlap-understanding-the-process-ade7eae1b2bd.

[31] Hyung Won Chung et al. *Scaling Instruction-Finetuned Language Models*. 2022. DOI: 10.48550/ARXIV.2210.11416. URL: https://arxiv.org/abs/2210.11416.

[32] An Yang et al. "Qwen2.5 Technical Report". In: *arXiv preprint arXiv:2412.15115* (2024).

[33] Stanford Center for Research on Foundation Models. *HELM Lite Leaderboard*. 2024. URL: https://crfm.stanford.edu/helm/lite/latest/#/leaderboard.

[34] Harrison Chase. *LangChain*. Oct. 2022. URL: https://github.com/langchain-ai/langchain.

[35] Maziyar Panahi. *Qwen2.5-7B-Instruct-GGUF*. 2024. URL: https://huggingface.co/MaziyarPanahi/Qwen2.5-7B-Instruct-GGUF.

[36] Sean Lee et al. *Open Source Strikes Bread - New Fluffy Embeddings Model*. 2024. URL: https://www.mixedbread.ai/blog/mxbai-embed-large-v1.

[37] Xianming Li and Jing Li. "AnglE-optimized Text Embeddings". In: *arXiv preprint arXiv:2309.12871* (2023).

[38] ChristianAzinn. *mxbai-embed-large-v1-gguf*. 2024. URL: https://huggingface.co/ChristianAzinn/mxbai-embed-large-v1-gguf.

[39] Chip Huyen. *lazynlp: Library to scrape and clean web pages to create massive datasets*. 2019. URL: https://github.com/chiphuyen/lazynlp.

[40] Jonathan Goldsmith. *Wikipedia Python Library*. 2013. URL: https://github.com/goldsmith/Wikipedia.

[41] Nv7-GitHub. *googlesearch: A Python library for scraping the Google search engine*. 2024. URL: https://github.com/Nv7-GitHub/googlesearch.

[42] Stefano Baccianella. *JSON Repair - A python module to repair invalid JSON, commonly used to parse the output of LLMs*. Version 0.28.3. Aug. 2024. URL: https://github.com/mangiucugna/json_repair.

[43] Lianmin Zheng et al. "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena". In: *arXiv preprint arXiv:2306.05685* (2023).

[44] Jonathan Berant et al. "Semantic Parsing on Freebase from Question-Answer Pairs". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1533–1544. URL: https://www.aclweb.org/anthology/D13-1160.

[45] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013.

[46] Seungone Kim et al. *Prometheus 2: An Open Source Language Model Specialized in Evaluating Other Language Models*. 2024. arXiv: 2405.01535 [cs.CL].

[47] PrunaAI. *prometheus-eval-prometheus-7b-v2.0-AWQ-4bit-smashed*. 2024. URL: https://huggingface.co/PrunaAI/prometheus-eval-prometheus-7b-v2.0-AWQ-4bit-smashed.

[48] Kayla Schroeder and Zach Wood-Doughty. *Can You Trust LLM Judgments? Reliability of LLM-as-a-Judge*. 2024. arXiv: 2412.12509 [cs.CL]. URL: https://arxiv.org/abs/2412.12509.

[49] Guiming Hardy Chen et al. "Humans or LLMs as the Judge? A Study on Judgement Bias". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 8301–8327. DOI: 10.18653/v1/2024.emnlp-main.474. URL: https://aclanthology.org/2024.emnlp-main.474/.

[50] Qin Zhang et al. “A Survey for Efficient Open Domain Question Answering”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 14447–14465. DOI: 10.18653/v1/2023.acl-long.808. URL: https://aclanthology.org/2023.acl-long.808/.

[51] Fengbin Zhu et al. *Retrieving and Reading: A Comprehensive Survey on Open-domain Question Answering*. 2021. arXiv: 2101.00774 [cs.AI]. URL: https://arxiv.org/abs/2101.00774.