Empowering RAG for complex documents with paragraph-based chunking

Francesco Cosimo Mazzitelli, Eugenio Zimeo
Department of Engineering
University of Sannio, Italy
f.mazzitelli@studenti.unisannio.it, zimeo@unisannio.it

Abstract

Retrieval Augmented Generation (RAG) combines chunks of documents with a prompt to be analyzed by an LLM to enhance the response's quality and reduce the risk of hallucinations. Existing chunking methods divides the text into chunks of predetermined length (fixed-size strategy), segments text at punctuation marks (recursive strategy), or aims at capturing the core meaning within each segment (semantic strategy). All these strategies limit context accessibility and encounter information loss, especially with larger documents and multi-layered information. They often require post-processing steps to avoid omitting crucial information, especially toward the documents' end. This paper introduces a paragraph-based chunking strategy that can benefit from larger context blocks. The proposed strategy leverages document structure metadata, such as font size and text formatting, to generate more semantically coherent chunks. The approach has been evaluated using both automatic and empirical approaches by exploiting standard metrics. The results demonstrate a promising enhancement for RAG applications by facilitating more effective document comprehension and improving context utilization in LLM-driven tasks when compared with existing strategies.

Index terms— Knowledge retrieval, Large Language Model, Retrieval Augmented Generation, Chunking

1 Introduction

Large Language Models (LLMs) have become fundamental to knowledge-intensive tasks involving the retrieval and synthesis of information from knowledge bases. In software engineering, they are used to query technical documentation, interpret source code, and assist in tasks such as bug triage and code generation. While effective, training LLMs for domain-specific tasks can be costly and often leads to reduced generalizability. To address these challenges, *Retrieval-Augmented Generation* (RAG) has emerged as a practical alternative. RAG [9] is a prominent framework that enhances the LLMs by incorporating external knowledge into the prompt context in the form of text segments, or *chunks*. This method often outperforms fine-tuned LLMs on the same data, with the advantage of avoiding retraining and enabling more interpretable and structured outputs. RAG systems are typically composed of two main components: the **Retriever**, which identifies the top-k most relevant chunks given a query, and the **Generator**, usually a pre-trained LLM that synthesizes a response using both the query and the retrieved context. RAG architectures are commonly classified into three categories: *Naive RAG*, *Advanced RAG*, and *Modular RAG* [6].

The **Naive RAG** pipeline, Figure 1, follows a straightforward three-step process. First, in the *Indexing* phase, raw documents are cleaned, segmented into fixed-size chunks, encoded into vector embeddings, and stored in a vector database. Then, during *Retrieval*, a query is converted into a vector representation, which is compared against stored embeddings to retrieve the top-k most similar chunks.

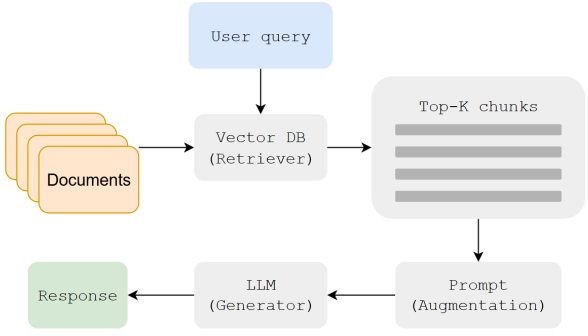

Figure 1: Naive RAG architecture

Finally, in the *Generation* step, the retrieved chunks are appended to the query and passed to an LLM to generate a final response.

DOI reference number: 10.18293/DMSVIVA2025-033

Advanced RAG introduces improvements in both the retrieval and generation stages. One such enhancement is *pre-retrieval optimization* [12], which improves indexing and query formulation by incorporating metadata or rewriting the query to achieve better semantic alignment. Another key improvement is *post-retrieval re-ranking* [13, 3], which reorders the retrieved chunks to prioritize the most relevant content in the prompt.

Modular RAG extends the architecture by introducing modular pipelines and deep learning components that adapt dynamically to the task. In the *Rewrite-Retrieve-Read* approach [12], an LLM rewrites the query to improve retrieval accuracy. The *Generate-Read* variant [23] replaces traditional document retrieval with LLM-generated contextual content. Finally, *Recite-Read* [8] renounces external retrieval entirely, relying instead on the internal knowledge stored within the model's weights.

Among the most commonly used retrieval models [4] are **BM25** [17], a probabilistic ranking function that improves upon traditional TF-IDF by incorporating term frequency saturation and length normalization; and **embedding-based retrieval**, which encodes textual inputs into dense vectors that capture semantic relationships, enabling retrieval based on the similarity between query and chunk embeddings. As LLMs continue to evolve, their capacity to process longer contexts is increasing. However, the challenge of effectively managing context remains. Limitations on maximum input length and risks of information loss persist, making the choice of the chunking strategy critical.

In this paper, we introduce a novel paragraph-based chunking approach designed to enhance both retrieval and comprehension in complex documents. By leveraging the natural structure of the content, this method aims to preserve semantic coherence and minimize information loss. It organizes content hierarchically through parent-child chunk relationships, allowing the identification of smaller, query-relevant segments within larger chunks. This approach mitigates the limitations of coarse-grained retrieval while still enabling access to broader, semantically rich content when needed, reducing fragmentation and improving the generator's ability to process and respond accurately.

The remainder of the paper is structured as follows: Section 2 reviews existing chunking strategies in RAG pipelines. Section 3 presents the proposed paragraph-based approach. Section 4 reports on the evaluation and comparison with baseline methods. Finally, Section 5 offers concluding remarks and future directions.

2 Related works

LLM, used as generator in the process, is not free from structural problems, e.g., the limitation of the context block provided in the input prompt strictly depends on how the generator was trained. To avoid context limitations, documents are split into several chunks in the pre-processing phase of the RAG pipeline. Several chunking strategies exist [22]:

Fixed size strategy [9]: This chunking strategy involves segmenting text into chunks of a predetermined number of characters. This approach organizes the text into manageable units for processing and analysis. An overlapping technique is commonly employed to minimize the loss of coherence. Rather than strictly respecting fixed boundaries, adjacent chunks partially overlap. This overlap ensures that essential contextual information and grammatical structures are preserved across chunk boundaries, thus maintaining the overall coherence and integrity of the text.

Recursive strategy [20]: This chunking strategy systematically breaks input text into smaller, more manageable segments. Its methodology revolves around using separators, allowing meticulous segmentation of the text's content. At its core, the strategy aims to align the chunking process with the text's structure, preserving the integrity of its meaning and coherence. Through the strategic use of separators, which may encompass punctuation marks, whitespace, or designated symbols, the text is partitioned into discrete units that encapsulate distinct sections or units of information. If the initial splitting does not result in chunks of the desired size or structure, the method calls itself, recursively, on the generated chunks with alternative separators or criteria. This continues until the desired chunk size or structure is obtained.

Document specific strategy [15] [18]: This chunking strategy involves breaking down a document into meaningful segments based on its unique structure and organization. Rather than relying solely on linguistic analysis, this strategy focuses on extracting specific metadata embedded within the document, such as tags, markers, or headings. These metadata indicate different sections, categories, or topics within the document. They provide cues for delineating the content into distinct chunks. Moreover, the strategy exploits meta-separators, specific markers, or separators within the document content that denote transitions between sections or subsections. These can be symbols, keywords, or formatting styles consistently used to demarcate different parts of the document. Once the metadata and meta-separators are identified, the chunking process follows the document's layout to segment the content accordingly. It parses the document from start to finish, identifying points where the structure transitions, such as between chapters, sections, or paragraphs. Each identified segment is treated as a separate chunk, preserving the document's logical organization.

Semantic strategy [16] [19] [20]: This chunking strategy aims to break down the text into meaningful chunks that capture the essence of the information. Instead of looking

at individual words or phrases, it considers the relationships and context within the text. This is obtained by taking each sentence in the text, converting it into embeddings, and then comparing the similarity of these embeddings. Sentences with similar meanings or contexts are grouped into chunks. By leveraging the semantic relationships encoded within the embeddings, this chunking strategy goes beyond surface-level analysis to capture the underlying meaning and connections present within the text.

Hybrid strategy [19]: This chunking strategy combines the previously reported strategies to provide more flexibility in handling different texts.

Although applying a chunking strategy may partially solve problems regarding the length of the context block it is not a silver bullet. Even with large contexts, it is possible to run into hallucinations and loss of information [11]. This study explains how changing the position of the relevant chunks can alter the generator's performance, indicating how the information contained in the center of the context block is not correctly accessible unlike those contained in the first or last part.

3 Paragraph-based chunking

The *paragraph-based chunking* strategy proposed in this paper tries to overcome the following problems:

- Information loss characterizing state-of-the-art chunking strategies;
- The need to use a natural language processing model for paragraph-based chunking.
- The need to use a Reranker for ordering chunks by relevance for a better selection.
- Accessibility of chunks positioned in the center of the context block.

This strategy leverages the natural structure of the input document to minimize information loss. Complex documents are often challenging for common chunking techniques due to the causal relationships between sentences, where a previous sentence may be referenced several sentences later. Using smaller chunks shifts the responsibility onto the retrieval component, increasing the risk of errors and potentially failing to retrieve chunks containing relevant information. While not all LLMs can deal with large context blocks, the proposed chunking strategy enables LLM to better understand complex documents by preserving these relationships.

3.1 Components

Figure 2 shows the architecture and the components chosen for the development of paragraph-based RAG the squared blocks represent artifacts and the rounded blocks functionalities.

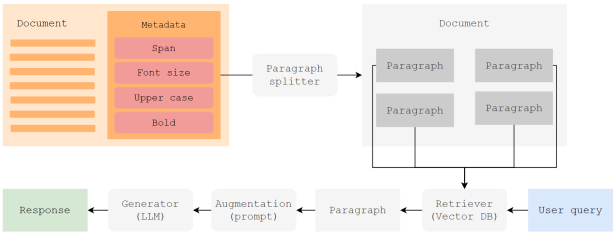

Figure 2: RAG architecture used in the proposed solution

The vector database is ChromaDB [7] within which IDs, text, and embedding of the document chunks are stored. Chroma was chosen since it makes easy providing custom embedding models during the database's creation, provides the possibility of choosing the similarity distance metric used for retrieving relevant chunks based on users' queries, and presents a great integration with Langhchain's framework [2].

The applied retrieval strategy is based on text embedding, which allows to capture relationships between each token. To correctly compute the embedding vector on text written in foreign languages "intfloat/multilingual-e5-large-instruct" [21] was involved, ensuring multilingual support.

The used LLM model is a fine-tuned version of Phi by Microsoft named "anakin87/Phi-3.5-mini-ITA" chosen for its ability to process large contexts and speak in Italian.

3.2 Chunking

The first step in the process involves reading the document and dividing it into blocks. Each block represents a section of content on a page delimited by a bounding box. If, for example, an image is placed between two sections on the same page, the document will be split into three blocks: one before the image, one for the image itself, which includes a single line of text with meta information; and one after the image. However, to prevent the loss of potentially critical information, data contained within images or tables is extracted using an integrated Optical Character Recognition (OCR) system. This splitting is necessary due to the structure of the data extracted from the document. Each block contains several fields, including the coordinates of its bounding box, the text within it, flags related to textual formatting, such as case (uppercase or lowercase), boldness, span, and font size.

The process of extracting content from a document and splitting it into distinct blocks based on visual elements and the spatial arrangement was performed using pymupdf. This is a Python module specifically designed for working with PDF documents. Each block is represented as a dictionary and then converted into Dataframes facilitating the extraction of further details using Pandas. The newly

created DataFrames are then read by a method that detects changes to the font size and whether the character of the analyzed span is bold. These metadata have been chosen based on the definition of paragraph opening and closing, assuming the presence of a title, typically with a larger font size, and a paragraph body with a smaller font size, regardless of punctuation or the presence of a double newline character as defined by the state-of-the-art recursive strategy. The checks on paragraph opening and closing are performed by defining a threshold value computed as the statistical mode of font size values in the DataFrames, assuming that the most frequent value relates to the font size of the body of the paragraph. The span of text characterized by a font size larger than the mode is marked as the title of the paragraph and everything else, smaller than the mode, is concatenated to it creating a single large string and therefore the chunk. However, the just-created chunks are characterized by dimensions, in terms of tokens, too large to perform an effective embedding-based search. To solve this problem and increase the precision of the retriever, these chunks marked as parents are divided into child chunks through the ParentDocumentRetriever of the Langchain framework [2]. When the search operation hits a child chunk, it only returns the parent chunk, avoiding the retrieval of other information to insert in the context block of the LLM. The source code is consultable at https://github.com/FrancescoMazzitelli/Paragraph-based-chunking. The documents and datasets extracted are not included in this repository due to restrictions imposed by the company that conducted the evaluation.

4 Evaluation

To assess the performance of the proposed solution, a two-step analysis was carried out: one automatic, using a set of largely adopted metrics, and one empirical, based on an interview.

It is important to note that widely recognized benchmarks, such as RAGBench [5], could not be employed in the evaluation. This limitation stems from the fact that the proposed approach relies on metadata extracted from documents, whereas benchmark datasets typically provide only raw text. As a result, a direct comparison with other strategies, designed to operate solely on raw text, would not have been meaningful or fair.

The evaluation process was firstly conducted by asking the experts to feed documents to the RAG and a set of questions for each of them. These were directly read from the file and submitted to:

Paragraph-based RAG: based on detecting the end of paragraph resulting in a larger chunk but with more information.

Fixed size RAG: based on creating chunks of 800 tokens with an overlap of a maximum of 50 tokens. These parameters were selected to match the average token length observed in the other strategies, ensuring a fair basis for comparison.

New Line RAG: based on detecting text separators like the \n\n character.

The other approaches cited in the related works section were not included in the evaluation. The chosen methods were selected to prioritize general-purpose functionalities, ensuring compatibility with most document types while maintaining the integrity of context and information.

The **recursive strategy** was excluded because it produced chunks identical to those generated by the fixed-size approach. This was due to the use of Langchain's RecursiveCharacterTextSplitter class [2] with a fixed chunk size, which ultimately resulted in the same output, leading to its integration into the fixed-size category.

The **document-specific strategy** was only partially considered through the inclusion of the newline-based method. Other variants were excluded due to their reliance on manual tagging to define document structure, which conflicts with the goal of developing an automated and general-purpose solution.

The **semantic strategy** produced poor-quality chunks. This method performed a further filtering of chunks based on their embeddings, reducing the dimension of the context block. This operation led to significant information loss, especially when handling complex input documents. This strategy was considered unsuitable because of the reduced context and lack of critical information.

4.1 Experiment plan

This section outlines the validation plan for assessing the proposed chunking strategies and their effectiveness within the Retrieval-Augmented Generation framework. To assess the quality of document retrieval, the generated responses and context utilization evaluations steps were conducted. The evaluation was conducted on a set of four territorial planning documents, differing in both content and structure, focused on the organization and delivery of waste collection and urban sanitation services. These plans serve as the primary tool for the technical, economic, and financial planning within the jurisdiction of the Optimal Territorial Ambit, which oversees the organization of waste collection and disposal activities. In addition, experts provided a set of 30 correct answers along with the ideal information chunks that contained the relevant data necessary to address the posed questions. The automatic evaluation focused on the quantitative assessment of both the generated responses and the retrieved chunks. To assess the effectiveness of the chunking strategy and its influence on the document retrieval process, we conducted a token-level comparison be-

tween the ideal chunk provided by the experts and the chunk retrieved by each chunking strategy. Furthermore, the analysis of response quality involved a comparison between the output of each chunking strategy (Paragraph-based, Fixed-size, and New Line) using standard evaluation metrics commonly applied in RAG tasks. These metrics enabled the assessment of accuracy, fluency, and relevance in the generated answers, taking into account both lexical and semantic similarity between the model's responses and the reference answers provided by the experts. In addition to the quantitative automatic analysis, an **empirical evaluation** was performed to gather user feedback on the quality of the generated responses. This step was necessary because the ambiguity of natural language may affect the evaluation of the model's output, and human judgment was crucial to understand the effectiveness of each chunking strategy in real-world applications. A questionnaire was submitted to domain experts who assessed the answers generated by the different RAG strategies, based on criteria such as relevance, clarity, and contextual accuracy. The results from this qualitative analysis provided feedback on how each approach handles complex, domain-specific content and whether the chunking methods preserved the crucial information needed to answer questions accurately. The combination of automatic and empirical evaluations allowed the assessment of the performances of each chunking strategy, ensuring the highlight of the most effective approach to optimize document comprehension and context utilization in RAG tasks.

4.2 Automatic evaluation

The first step involved initializing and loading the pre-trained model onto an Nvidia A100 GPU to enhance computation speed. Subsequently, the questions were imported from a CSV file using Pandas and stored in an iterable list for further processing. A text-generation pipeline was created with the initialized model and its corresponding tokenizer. The pipeline generated answers based on the context retrieved from the chunked documents stored in the vector database. The generated answers were then saved in a CSV file structured as: Question - Answer - Context.

The newly created output files were submitted for evaluation to the commonly used metrics in natural language processing: **BERTScore** [24]: BERTScore is a metric designed to evaluate the quality of generated text using contextual embeddings from the BERT model. Unlike traditional methods based on string matching, BERTScore measures semantic similarity by aligning tokens between candidate and reference texts based on their embeddings in a high-dimensional space. It aggregates precision, recall, and F1 scores to provide a comprehensive evaluation of contextual accuracy, capturing semantic equivalences such as synonyms or paraphrases.

BLEU [14]: BLEU, or Bilingual Evaluation Understudy, evaluates machine-generated translations by comparing them to reference translations using n-gram matching. It calculates the precision of overlapping n-grams and applies a brevity penalty to account for shorter translations. BLEU outputs a score between 0 and 1, where higher values indicate greater similarity to the reference translations.

METEOR [1]: METEOR, or Metric for Evaluation of Translation with Explicit ORdering, improves upon BLEU by incorporating additional features such as stemming and synonym matching to address word choice and linguistic variations. It evaluates both precision and recall, considers grammatical structure, and produces a weighted harmonic mean score that reflects both lexical and semantic accuracy, offering a more detailed comparison to human judgments of translation quality.

ROUGE [10]: ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, is used to assess the quality of machine-generated summaries by comparing them to reference summaries. It focuses on content overlap, using variants such as ROUGE-N (n-gram overlap), ROUGE-L (longest common subsequence), and ROUGE-Lsum (a weighted version of ROUGE-L). ROUGE emphasizes recall to evaluate how comprehensively the generated summary captures the key ideas from the reference summaries.

All metrics reported in Tables 1 and 2 represent averages of responses calculated across each strategy-document pair. The paragraph-based approach achieved the highest performance among the evaluated strategies. However, the differences in metric values across strategies are generally small, and the overall scores remain low in some cases. This can be attributed to the variation in length of retrieved chunks among different strategies, as well as to semantic equivalence between candidate and reference sentences that differ in their phrasing, making n-gram-based metrics less suitable for evaluation. Some errors may stem from the high similarity among extracted text segments, due to the frequent repetition of core concepts through different sections of technical documents. Moreover, this analysis does not account for the factual correctness of the responses. To address these limitations, a qualitative evaluation was conducted.

4.3 Empirical evaluation

Evaluating a RAG is not a simple task because there are several dimensions to be considered, such as the ability of the retrieval system to identify relevant chunks, the ability of the LLM to exploit and interpret the chunks, and, in general, the quality of text generation. To evaluate the proposed solution and overcome the limitations of the previous evaluation, 21 experts in urban sanitation were asked to assess the quality of responses provided by the various strategies. The goal was to identify which strategy offers the most effective

Doc	Strategy	Prec.	Rec.	F1	NDCG	MRR	Token Match	Unique Tok.
1	**Paragraph**	**0.662**	**0.370**	**0.420**	**0.457**	**0.671**	**106.031**	160.100
	Fixed size	0.493	0.285	0.342	0.318	0.361	74.750	151.406
	New line	0.328	0.326	0.312	0.451	0.322	86.218	215.656
2	**Paragraph**	**0.734**	**0.878**	**0.780**	**0.954**	**0.808**	**284.793**	403.103
	Fixed size	0.437	0.250	0.288	0.253	0.384	75.034	163.655
	New line	0.386	0.316	0.309	0.374	0.316	96.793	245.793
3	**Paragraph**	**0.588**	**0.549**	**0.543**	**0.647**	**0.729**	**155.250**	264.312
	Fixed size	0.507	0.332	0.374	0.339	0.411	89.437	173.250
	New line	0.536	0.479	0.474	0.578	0.447	152.187	286.625
4	**Paragraph**	**0.444**	**0.720**	**0.514**	**0.457**	**0.651**	**208.461**	443.807
	Fixed size	0.425	0.379	0.367	0.392	0.290	92.692	260.730
	New line	0.377	0.387	0.341	0.434	0.410	260.730	234.461

Table 1: Chunk quality metrics

Doc	Strategy	BERT Score			METEOR	BLEU			ROUGE			
		Prec.	Rec.	F1	Score	BLEU	B. Pen.	Len R.	R1	R2	RL	RLS
1	**Paragraph**	**0.862**	**0.847**	**0.853**	**0.264**	**0.169**	0.671	**0.714**	**0.365**	**0.245**	**0.348**	**0.347**
	Fixed size	0.842	0.820	0.830	0.126	0.135	0.654	0.702	0.201	0.125	0.191	0.191
	New line	0.842	0.821	0.830	0.127	0.136	**0.646**	0.696	0.207	0.125	0.191	0.190
2	**Paragraph**	**0.868**	**0.896**	**0.881**	**0.312**	0.304	0.926	**0.939**	**0.408**	**0.234**	**0.409**	**0.404**
	Fixed size	0.837	0.866	0.851	0.226	0.066	**0.826**	0.839	0.322	0.112	0.315	0.318
	New line	0.848	0.868	0.857	0.222	**0.606**	0.903	0.907	0.318	0.097	0.297	0.295
3	**Paragraph**	**0.875**	**0.888**	**0.882**	**0.493**	**0.267**	0.971	**0.984**	**0.532**	**0.437**	**0.501**	**0.491**
	Fixed size	0.840	0.849	0.850	0.307	0.134	**0.821**	0.835	0.359	0.272	0.320	0.330
	New line	0.860	0.872	0.865	0.335	0.120	0.964	0.969	0.424	0.312	0.368	0.363
4	**Paragraph**	**0.850**	**0.867**	**0.857**	**0.254**	**0.503**	**0.697**	0.734	**0.349**	**0.144**	**0.321**	**0.323**
	Fixed size	0.840	0.852	0.845	0.208	0.106	0.897	**0.902**	0.277	0.113	0.265	0.264
	New line	0.842	0.853	0.847	0.214	0.115	0.835	0.847	0.279	0.132	0.271	0.272

Table 2: Response quality metrics

answer. The interview was conducted by providing experts with a Google Form that displayed the original question, the correct answer, and checkboxes listing the answers generated by the RAG model using the chunking strategies. Experts were asked to select the machine-generated answers they considered comparable and similar to the correct answer. Many answers were marked as "not comparable"; therefore, all instances of these responses with values exceeding 60% of the total observations are identified and removed. Further analysis was conducted to identify outliers, highlighting user 8 and user 12 for the "new line" category due to their over-selection of this strategy, and user 15 for the "not comparable" category, as this user did not select that category even once. However, users 8 and 12 were not removed, as their responses to other questions showed a behavior consistent with other participants. User 15 was flagged as a "lazy voter" and subsequently removed due to the pronounced anomaly observed.

After removing the noisy components, it was possible to analyze the following results. According to Figure 3, the

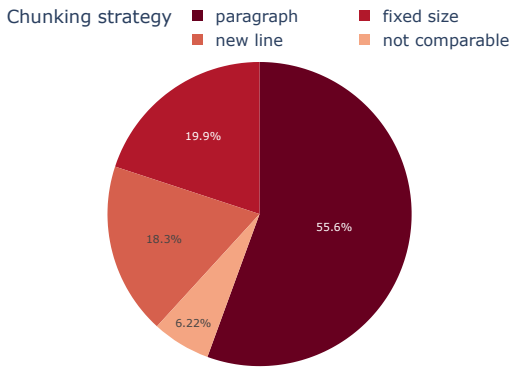

Figure 3: Distribution of feedback without incompatible responses

optimal strategy is the paragraph-based approach, which received an appreciation percentage of 55.6%. In comparison, the fixed-size strategy garnered 19.9%, while the new line method achieved 18.3%. These results indicate that the

paragraph-based strategy provides better answers regarding precision and coherence to the question.

5 Conclusion

This study demonstrates the effectiveness of a paragraph-based chunking strategy for enhancing Retrieval-Augmented Generation, especially in handling complex, document-length inputs. While advances in large language models have increased token capacity and improved semantic understanding, challenges remain in balancing context size constraints with the need to preserve relevant information. The proposed method addresses this trade-off by leveraging the document's inherent structure to generate hierarchical chunks that maintain logical and semantic coherence. Unlike fixed-size or newline-based strategies, paragraph-based chunking uses text metadata to group content into parent–child relationships, enabling the retrieval of larger, contextually consistent segments. This reduces ambiguity and improves both precision and recall in downstream tasks. Empirical results show enhanced contextual relevance and coherence, positioning this approach as a practical solution for RAG systems requiring deeper document comprehension. Future work may use this strategy to evaluate integration into different RAG architectures and explore scalability across broader applications.

References

[1] S. Banerjee and A. Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[2] H. Chase. Langchain development framework. https://www.langchain.com/, 2024. Accessed: 2024-04-14.

[3] G. V. Cormack, C. L. Clarke, and S. Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 758–759, 2009.

[4] P. Finardi, L. Avila, R. Castaldoni, P. Gengo, C. Larcher, M. Piau, P. Costa, and V. Caridá. The chronicles of rag: The retriever, the chunk and the generator. *arXiv preprint arXiv:2401.07883*, 2024.

[5] R. Friel, M. Belyi, and A. Sanyal. Ragbench: Explainable benchmark for retrieval-augmented generation systems, 2025.

[6] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[7] A. T. Jeff Huber. Chromadb. https://www.trychroma.com/, 2024. Accessed: 2024-04-14.

[8] O. Khattab, K. Santhanam, X. L. Li, D. Hall, P. Liang, C. Potts, and M. Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*, 2022.

[9] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[10] C.-Y. Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[11] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 02 2024.

[12] X. Ma, Y. Gong, P. He, H. Zhao, and N. Duan. Query rewriting for retrieval-augmented large language models. *arXiv preprint arXiv:2305.14283*, 2023.

[13] R. Nogueira, W. Yang, K. Cho, and J. Lin. Multi-stage document ranking with bert. *arXiv preprint arXiv:1910.14424*, 2019.

[14] K. Papineni, S. Roukos, T. Ward, and W. jing Zhu. Bleu: a method for automatic evaluation of machine translation. pages 311–318, 2002.

[15] M. Poliakov and N. Shvai. Multi-meta-rag: Improving rag for multi-hop queries using database filtering with llm-extracted metadata, 2024.

[16] R. Qu, R. Tu, and F. Bao. Is semantic chunking worth the computational cost?, 2024.

[17] S. Robertson, H. Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[18] R. Siegler. Optimizing vector search with metadata filtering and fuzzy filtering, 2024.

[19] I. S. Singh, R. Aggarwal, I. Allahverdiyev, M. Taha, A. Akalin, K. Zhu, and S. O'Brien. Chunkrag: Novel llm-chunk filtering method for rag systems, 2024.

[20] A. team. Optimizing retrieval-augmented generation with advanced chunking techniques: A comparative study, 2024.

[21] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei. Multilingual e5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*, 2024.

[22] A. J. Yepes, Y. You, J. Milczek, S. Laverde, and R. Li. Financial report chunking for effective retrieval augmented generation, 2024.

[23] W. Yu, D. Iter, S. Wang, Y. Xu, M. Ju, S. Sanyal, C. Zhu, M. Zeng, and M. Jiang. Generate rather than retrieve: Large language models are strong context generators. *arXiv preprint arXiv:2209.10063*, 2022.

[24] T. Zhang*, V. Kishore*, F. Wu*, K. Q. Weinberger, and Y. Artzi. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*, 2020.