Reconstructing Context

Evaluating Advanced Chunking Strategies for Retrieval-Augmented Generation

Anonymous authors

No Institute Given

Abstract. Retrieval-augmented generation (RAG) has become a transformative approach for enhancing large language models (LLMs) by grounding their outputs in external knowledge sources. Yet, a critical question persists: how can vast volumes of external knowledge be managed effectively within the input constraints of LLMs? Traditional methods address this by chunking external documents into smaller, fixed-size segments. While this approach alleviates input limitations, it often fragments context, resulting in incomplete retrieval and diminished coherence in generation. To overcome these shortcomings, two advanced techniques—late chunking and contextual retrieval—have been introduced, both aiming to preserve global context. Despite their potential, their comparative strengths and limitations remain unclear. This study presents a rigorous analysis of late chunking and contextual retrieval, evaluating their effectiveness and efficiency in optimizing RAG systems. Our results indicate that contextual retrieval preserves semantic coherence more effectively but requires greater computational resources. In contrast, late chunking offers higher efficiency but tends to sacrifice relevance and completeness.

Keywords: Contextual Retrieval · Late Chunking · Dynamic Chunking · Rank Fusion.

1 Introduction

Retrieval Augmented Generation (RAG) is a transformative approach that enhances the capabilities of large language models (LLMs) by integrating external information retrieval directly into the text generation process. This method allows LLMs to dynamically access and utilize relevant external knowledge, significantly improving their ability to generate accurate, contextually grounded, and informative responses. Unlike static LLMs that rely solely on pre-trained data, RAG-enabled models can access up-to-date and domain-specific information. This dynamic integration ensures that the generated content remains both relevant and accurate, even in rapidly evolving or specialized fields.

RAG models combine two key components: a retrieval mechanism and a generative model. The retrieval mechanism fetches relevant documents or data from a large corpus, while the generative model synthesizes this information into

coherent, contextually enriched answers. This synergy enhances performance in knowledge-intensive natural language processing (NLP) tasks, enabling models to produce well-informed responses grounded in the retrieved data.

The Context Dilemma in Classic RAG: Managing extensive external documents poses significant issues in RAG systems. Despite advancements, many LLMs are limited to processing a few thousand tokens. Although some models have achieved context windows up to millions of tokens [5], these are exceptions rather than the norm. Moreover, research indicates that LLMs may exhibit positional bias, performing better with information at the beginning of a document and struggling with content located in the middle or toward the end [11, 16]. This issue is exacerbated when retrieval fails to prioritize relevant information properly. Thus, documents are often divided into smaller segments or "chunks" before embedding and retrieval. However, this chunking process can disrupt semantic coherence, leading to:

- *Loss of Context:* dividing documents without considering semantic boundaries can result in chunks that lack sufficient context, impairing the model's ability to generate accurate and coherent responses.
- *Incomplete Information Retrieval:* important information split across chunks may not be effectively retrieved or integrated.

To address these issues, we analyse and compare two recent techniques—contextual retrieval[1] and late chunking [9]—within a unified setup, evaluating their strengths and limitations in tackling challenges like context loss and incomplete information retrieval. Contextual retrieval preserves coherence by prepending LLM-generated context to chunks, while late chunking embeds entire documents to retain global context before segmenting.

Our study rigorously assesses their impact on generation performance in question-answering tasks, finding that neither technique offers a definitive solution. This work highlights the trade-offs between these methods and provides practical guidance for optimizing RAG systems.

To further support the community, we release all code, prompts, and data under the permissive MIT license, enabling full reproducibility and empowering practitioners to adapt and extend our work.[2]

2 Related Work

Classic RAG. A standard RAG workflow involves four main stages: document segmentation, chunk embedding, indexing, and retrieval. During segmentation, documents are divided into manageable chunks. These chunks are then transformed into vector representations using encoder models, often normalized to ensure unit magnitudes. The resulting embeddings are stored in indexed vector databases, enabling efficient approximate similarity searches. Retrieval involves

[1] https://www.anthropic.com/news/contextual-retrieval
[2] https://anonymous.4open.science/r/rag-when-how-chunk-6093

comparing query embeddings with the stored embeddings using metrics such as cosine similarity or Euclidean distance, which identify the most relevant chunks. Seminal works like [15] and [13] have demonstrated the effectiveness of RAG in tasks such as open-domain question answering. More recent studies, including [7], have introduced advancements in scalability and embedding techniques, further establishing RAG as a foundational framework for knowledge-intensive applications.

Document Segmentation. Document segmentation is essential for processing long texts in RAG workflows, with methods ranging from *fixed-size segmentation* [7] to more adaptive techniques like *semantic segmentation*,[3] which detect semantic breakpoints based on shifts in meaning. Recent advancements include *supervised segmentation models* [14, 12] and *segment-then-predict models*, trained end-to-end without explicit labels to optimize chunking for downstream task performance [17]. In 2024, *late chunking* and *contextual retrieval* introduced novel paradigms. Both techniques have proven effective in retrieval benchmarks but remain largely untested in integrated RAG workflows. Despite several RAG surveys [7, 6, 8], no prior work has compared these methods within a comprehensive evaluation framework. This study addresses this gap by holistically analyzing late chunking and contextual retrieval, offering actionable insights into their relative strengths and trade-offs.

3 Methodology

To guide our study, we define the following research questions (RQs), aimed at evaluating different strategies for chunking and retrieval in RAG systems:

- **RQ#1**: Compares the effectiveness of **early versus late chunking** strategies, utilizing **different text segmenters** and embedding models to evaluate their impact on retrieval accuracy and downstream performance in RAG systems.
- **RQ#2**: Compares the effectiveness of **contextual retrieval versus traditional early chunking** strategies, utilizing **different text segmenters** and embedding models to evaluate their impact on retrieval accuracy and downstream performance in RAG systems.

3.1 RQ#1: Early or Late Chunking?

In this workflow, the main architectural modification compared to the standard RAG lies in the document embedding process Figure 3.1. Specifically, we experiment with various embedding models to encode document chunks, tailoring them to align with the early and late chunking strategies under evaluation. This adjustment allows us to explore how different embedding techniques influence the retrieval quality and, subsequently, the overall performance of the RAG system.

[3] https://docs.llamaindex.ai/en/stable/examples/node_parsers/semantic_chunking/

Additionally, we test dynamic segmenting models to further refine the chunking process, providing an adaptive mechanism that adjusts chunk sizes based on content characteristics. By evaluating the impact of these dynamic segmenting models, we aim to improve the overall retrieval efficiency and response generation within the RAG framework.

Early Chunking. Documents are segmented into text chunks, and each chunk is processed by the embedding model. The model generates token-level embeddings for each chunk, which are subsequently aggregated using mean pooling to produce a single embedding per chunk.

Late Chunking. Late chunking [9] defers the chunking process. As shown in Figure 3.1, instead of segmenting the document initially, the entire document is first embedded at the token level. The resulting token embeddings are then segmented into chunks, and mean pooling is applied to each chunk to generate the final embeddings. This approach preserves the full contextual information within the document, potentially leading to superior results across various retrieval tasks. It is adaptable to a wide range of long-context embedding models and can be implemented without additional training. The two approaches are tested with different embedding models.

3.2 RQ#2: Early or Contextual Chunking?

In this workflow, traditional retrieval is compared to Contextual Retrieval with Rank Fusion technique. This has been introduced by Anthropic in September 2024.[4] Three steps are added to the Traditional RAG process: Contextualization, Rank Fusion, Reraking.

Contextualization. After document segmentation, each chunk is enriched with additional context from the entire document, ensuring that even when segmented, each piece retains a broader understanding of the content (Fig. 3.2). In fact, when documents are split into smaller chunks, it might arise the problem where individual chunks lack sufficient context. For example, a chunk might contain the text: "The company's revenue grew by 3% over the previous quarter." However, this chunk on its own does not specify which company it is referring to or the relevant time period, making it difficult to retrieve the right information or use the information effectively. Contextualization improves the relevance and accuracy of retrieved information by maintaining contextual integrity.

Rank Fusion. In our methodology, we employ a rank fusion strategy that integrates dense embeddings with sparse embeddings of BM25 [19] to improve retrieval performance. Although embedding models adeptly capture semantic relationships, they may overlook exact matches, which is particularly useful for unique identifiers or technical terms. BM25 uses a ranking function that builds

[4] https://www.anthropic.com/news/contextual-retrieval

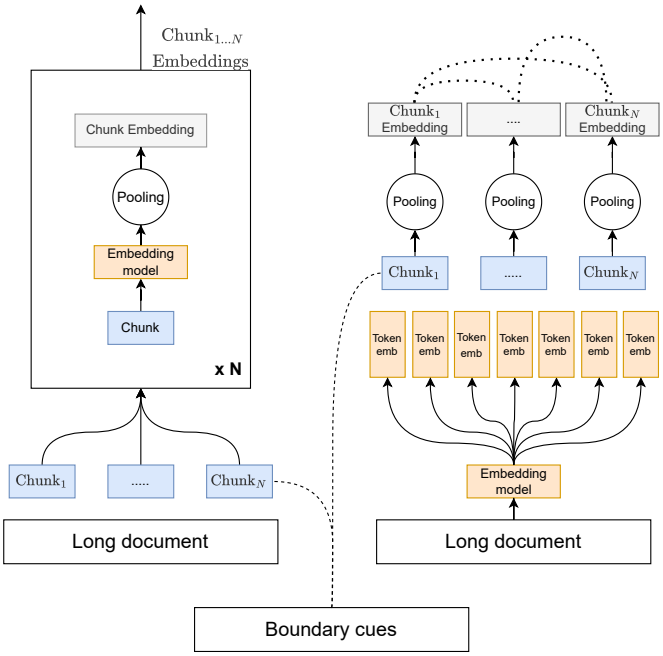

Fig. 1. Comparison of early chunking (left) and late chunking (right) approaches for processing long documents. In early chunking, the document is divided into chunks before embedding, with each chunk processed independently by the embedding model and then pooled. In contrast, late chunking processes the entire document to generate token embeddings first, using boundary cues to create chunk embeddings, which are subsequently pooled.

upon Term Frequency-Inverse Document Frequency (TF-IDF), addressing this limitation by emphasizing precise lexical matches. To combine the strengths of both approaches, we conduct searches across both dense embedding vectors and BM25 sparse embedding vectors generated from both the chunk and its generated context. Initially, the assigned relative importance in the search for the two vector fields has been set to be of equal intensity, resulting in lowering the scoring results in the retrieval evaluation. For this reason we use a weighting strategy assigning higher weights to dense vector fields, emphasizing them more in the final ranking. While different weight parameters have been tested, the final decision has been to define a ratio of importance 4:1 assigning weight 1 for the dense embedding vectors and 0.25 for the BM25 sparse embedding vectors. This ratio reflects Anthropic weight assignment.[5] By applying these weights, dense embeddings are emphasized more heavily, while still incorporating the contributions of

[5] https://github.com/anthropics/anthropic-cookbook/blob/main/skills/contextual-embeddings/guide.ipynb

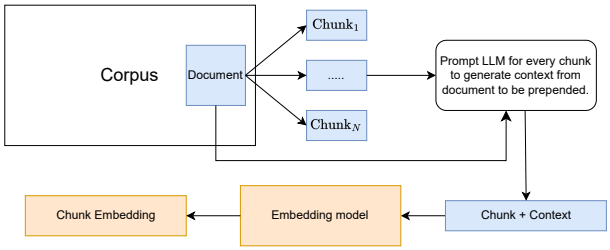

Fig. 2. Contextualization of each chunk is performed prior to embedding. The document is divided into chunks, and a prompt is used to query an LLM to generate contextual information from the document for each chunk. The context is prepended to the chunk, which is then processed by the embedding model to produce the final chunk embedding.

the BM25 sparse embeddings. This weighted rank fusion approach leverages the complementary strengths of semantic embeddings and lexical matching, aiming to improve the accuracy and relevance of the retrieved results.

Reranking Step. To boost and obtain consistent improved performance, the retrieval and ranking stages have been separated. This two-stage approach improves efficiency and effectiveness by leveraging the strengths of different models at each stage. After retrieving the initial set of document chunks, we implement an additional reranking step to enhance the relevance of the results to the user's query. This process involves reassessing and reordering the retrieved chunks based on their pertinence, prioritising the most relevant information. The reranker operates by evaluating the semantic similarity and contextual relevance between the query and each retrieved chunk. It assigns a relevance score to each chunk, and the chunks are then reordered based on these scores, with higher-scoring chunks placed at the top of the list presented to the user. This method ensures that the most pertinent information is readily accessible, improving the overall effectiveness of the retrieval system. Implementing this reranking step addresses potential limitations of the initial retrieval process, such as the inclusion of less relevant chunks or the misordering of pertinent information.

4 Experimental Setup

This study has focused on testing these techniques with open-source models. A particular focus was also given to resource usage, for real-world scenarios that lead towards the choice of the LLM for the question answering task to Microsoft's `Phi-3.5-mini-instruct`,[6] quantized to 4 bits. [1] This language

[6] https://huggingface.co/microsoft/Phi-3.5-mini-instruct

model is designed to operate efficiently in memory- and compute-constrained environments which is a crucial aspect of our work. The same language model has been used also to generate additional context to prepend to each chunk in the Contextual Retrieval Setup.

For what regards the embedding models these ones have been tested: `Jina V3` [20], `Jina Colbert V2` [10], `Stella V5` and `BGE-M3` [4], all present in the `MTEB` [18] leaderboard (see Table 4.1 for more details).

Dataset and Hardware. There are severe limitations in current datasets available for RAG systems evaluation. Many don't include together the labels for retrieval quality evaluation and answers labels for the quality of the generation in a question answering system. In our system initial Retrieval performance has been tested on the `NFCorpus` [3] dataset, while the subsequent Generation performance in question answering has been conducted over `MSMarco`[2].

Another important note for Contextual Retrieval (`RQ#2`). The `NFCorpus` dataset is characterised by a long average document length. Here appear the first limitations of the Contextual Retrieval approach to RAG. For the intrinsic nature of this approach, the segmented chunks are enhanced with a generated context taken from the document, prompting an LLM for the task, leveraging the new advents of Instruction Learning. Chunks and documents are passed together in a formatted prompt to the model. When a document reaches long lengths, the VRAM of the GPU gets filled up quickly. For chunk contextualization, around 20GB of VRAM use can be reached, limiting batch dimensions for generation and slowing down the times needed for effective chunk contextualization.

In our experimental setup, we utilized an Nvidia RTX 4090 with 24GB of VRAM. Due to GPU memory constraints, we employed a subset of the dataset, corresponding to 20% of the entire `NFCorpus` for `RQ#2`, while the full dataset was used for `RQ#1` workflow.

For datasets such as `MsMarco`, which include only passage texts rather than full documents, the system operates within a more constrained context for generating responses. This limitation arises because passages are typically shorter segments of text, providing less information for contextual understanding. As a result in `RQ#2`, the system's ability to generate contextually relevant and comprehensive responses can be affected by the brevity of the input text, potentially impacting the quality and depth of the generated content.

In `RQ#1`, the evaluation was conducted on the first 1,000 queries and approximately 5,000 documents/passages. For `RQ#2`, due to the significant computational requirements and hardware limitations, the experiments were restricted to 50 queries and around 300 documents.

4.1 Embedding generation

Common to both RQs. To generate embeddings for our experiments, we utilized different embedding models, as detailed previously (see section 4). Each segmentation model approach outlined was paired with an appropriate embedding model to evaluate its influence on downstream tasks.

Model	MTEB Rank	Model Size(M)	Memory(GB)	Embedding Dim	Max Token
Stella-V5	5	1,543	5.75	1,024	131,072
Jina-V3[20]	53	572	2.13	1,024	8,194
Jina-V2 [10]	123	137	0.51	1,024	8,194
BGE-M3 [4]	211	567	2.11	1,024	8,192

Table 1. Embedding models.

For fixed-size segmentation, we divided the text into equal-sized chunks with a predefined length of 512 characters. This approach ensures uniform chunk sizes, simplifying processing and offering a baseline for comparison with more adaptive methods.

For semantic segmentation, we used the `Jina-Segmenter API`,[7] which dynamically adjusts chunk boundaries based on the semantic structure of the text. This ensures that the segments capture meaningful content, improving the quality of embeddings generated.

All the generated embeddings were normalized to unit vectors, facilitating cosine similarity computations during the retrieval phase and ensuring uniformity across experiments.

RQ#1: In addition to the mentioned segmentation approaches, for this workflow, dynamic segmentation was tested, testing two models to assess their performance. The first model, `simple-qwen-0.5`, is a straightforward solution designed to identify boundaries based on the structural elements of the document. Its simplicity makes it efficient for basic segmentation needs, offering a computationally lightweight approach.

The second model, `topic-qwen-0.5`, inspired by Chain-of-Thought reasoning, enhances segmentation by identifying topics within the text. By segmenting text based on topic boundaries, this approach captures richer semantic relationships, making it suitable for tasks requiring a deeper understanding of document content.

RQ#2: In this workflow, for the ContextualRankFusion evaluation, contextualization of the chunks before the embedding is necessary. To contextualize each chunk, we prompt Microsoft's LLM model `Phi3.5-mini-instruct` to generate a brief summary that situates the chunk within the overall document, formatting the prompt with the chunk and its relative original document.

4.2 Retrieval Evaluation

RQ#2: In this workflow, specifically for the ContextualRankFusion retrieval, the document retrieval has been enhanced with two additional steps (see Section 3.2). In the reranking step, Rank Fusion was allowed through `Milvus` Vector Database, which integrates with `BM25` through the `BM25EmbeddingFunction`

[7] https://jina.ai/embeddings/

class, enabling hybrid search across dense and sparse vector fields.
After retrieving the top documents, these are reordered through the reranker model `Jina Reranker V2 Base` model,[8] that employes a cross-encoder architecture that processes each query-document pair individually, outputting a relevance score. This design enables the model to capture intricate semantic relationships between the query and the document before being given to the LLM.

Scorings. For both approaches in `RQ#1` and `RQ#2`, when querying the embedding database (generated in 4.1), the output will be a ranked list of chunks, ordered from the most similar to the query to the least similar. We employ a straightforward aggregation strategy to transition from chunk-level rankings to document-level rankings. Specifically, for each document, we consider the score of its most significant chunk as the representative value for the entire document. This approach ensures that a document's relevance is determined by its most relevant chunk.

Once the document scores are determined, we generate a ranked list of documents based on these scores. From this ranking, we extract the top-k documents, focusing on the Top 5 or Top 10 documents, depending on the specific evaluation scenario. This final document ranking is then used to assess the effectiveness of the retrieval process.

This methodology highlights the importance of individual chunks in influencing the overall document ranking and ensures that highly relevant chunks directly impact the document's position in the final ranking.

Metrics. To evaluate the performance of our model, we utilize three key metrics: NDCG , MAP, and F1-score. Each metric serves a specific purpose in assessing different aspects of the results. *Normalized Discounted Cumulative Gain (NDCG):* It measures the usefulness of an item based on its position in the ranking, assigning higher weights to items appearing at the top of the list. By using NDCG, we aim to assess the relevance of predictions in a way that prioritizes higher-ranked items. *Mean Average Precision (MAP)*: It calculates the mean of the Average Precision (AP) scores for all queries, where AP considers the precision at each relevant item in the ranked list. With MAP, we aim to quantify how effectively the model retrieves relevant results across different scenarios. *F1-score:* The F1-score, a harmonic mean of precision and recall, is employed to balance the trade-off between false positives and false negatives.

4.3 Generation evaluation

Generation evaluation was assessed through the `MSMarco` dataset using a question answering task. While for the Late Chunking technique, the scoring on the generation respects the retrieval performance, for the Contextual Retrieval Setup, chunks are enriched with additional generated context from the document that influences the output generation of an LLM. Although some differences were

[8] https://huggingface.co/jinaai/jina-reranker-v2-base-multilingual

measured in the scorings, they were not notable enough to assess a significant difference in generation performance.

5 Results and Analysis

5.1 Traditional Retrieval Versus ContextualRankFusion Retrieval

From the results in Table 2, and especially focusing the attention on the best performing embedding model `Jina-V3`, we show that Fixed-Window Chunking versus Semantic Chunking techniques do not differ much in terms of performance or not at all, while the first one being far easier implementable and faster than the second one. A more important finding underlines in the Rank Fusion technique. This technique shows improved results especially when chunks are enriched with additional context from the document. In this way, `BM25` matches search terms in both segments and contexts, leading to very good results. It is important to note that adding the final reranking step in the workflow is crucial to leverage this potential and see consistent improvements in the results.

5.2 Traditional Retrieval Versus Latechunking Retrieval

Upon analyzing the results in Table 3, we observe that the novel Late Chunking approach performs well in most cases when compared to the Early version. This indicates its potential as an effective retrieval strategy for many scenarios. However, it is important to note that Late Chunking does not consistently outperform the Early approach across all models and datasets.

For instance, with the `BGE-M3` model applied to the `NFCorpus`, the Early version demonstrates superior performance, highlighting a case where the Late Chunking approach falls short. This observation is further confirmed through testing on the `MsMarco` dataset using the `Stella-V5` model (Table 4), where once again the Early version outperforms the Late Chunking approach.

These findings suggest that while Late Chunking introduces promising improvements in certain contexts, its efficacy may vary depending on the dataset and model used, emphasizing the need for careful selection of retrieval strategies based on specific use cases.

5.3 Latechunking Versus ContextualRankFusion Retrieval

In table 5 we compare the best results obtained for ContextualRankFusion with Latechunking on the same subset of `NFCorpus` in order to compare the two techniques. The embedding model used is `Jina-V3`, for Fixed-Window Chunks. ContextualRankFusion obtains better results overall.

Model	CM	RM	NDCG@5	MAP@5	F1@5	NDCG@10	MAP@10	F1@10
Jina-V3	FUC	TR	0.303	0.137	0.193	0.291	0.154	0.191
		RFR	0.289	0.130	0.185	0.288	0.150	0.193
	SUC	TR	0.307	0.143	0.197	0.292	0.159	0.187
		RFR	0.295	0.135	0.194	0.287	0.152	0.189
	FCC	TR	0.312	0.144	0.204	0.295	0.159	0.190
		RFR	**0.317**	**0.146**	0.206	0.308	0.166	0.202
	SCC	TR	0.305	0.136	0.197	0.296	0.155	0.198
		RFR	0.317	0.146	**0.209**	**0.309**	**0.166**	**0.204**
Jina-V2	FUC	TR	0.206	0.084	0.138	0.202	0.096	0.137
		RFR	0.256	0.119	0.166	0.251	0.133	0.161
	SUC	TR	0.231	0.100	0.152	0.223	0.112	0.149
		RFR	0.274	0.127	0.179	0.262	0.140	0.168
	FCC	TR	0.232	0.098	0.155	0.219	0.109	0.143
		RFR	0.288	0.130	0.182	0.274	0.144	0.173
	SCC	TR	0.231	0.099	0.156	0.220	0.110	0.148
		RFR	0.297	0.134	0.191	0.283	0.148	0.180
BGE-M3	FUC	TR	0.017	0.006	0.015	0.018	0.007	0.014
		RFR	0.032	0.012	0.018	0.033	0.012	0.020
	SUC	TR	0.012	0.003	0.001	0.012	0.003	0.011
		RFR	0.029	0.008	0.017	0.026	0.009	0.018
	FCC	TR	0.007	0.001	0.003	0.012	0.003	0.012
		RFR	0.040	0.015	0.026	0.040	0.016	0.027
	SCC	TR	0.002	0.001	0.001	0.006	0.002	0.007
		RFR	0.034	0.014	0.021	0.030	0.015	0.019

Table 2. Comparative results on a subset of the `NFCorpus` dataset. 20% of the whole shuffled dataset was taken, deleting labels of documents not present in the subset dataset for retrieval evaluation. Scorings will be higher on the whole dataset.

CM: Chunking Methods (FUC: Fixed-Window Uncontextualized Chunks, SUC: Semantic Uncontextualized Chunks, FCC: Fixed-Window Contextualized Chunks, SCC: Semantic Contextualized Chunks).
RM: Retrieval Methods (TR: Traditional Retrieval, RFR: Rank Fusion with weighted strategy (1, 0.25) respectively for dense embedder models and BM25 embeddings – additional Reranking step for RFR).

5.4 Dynamic segmenting models

As shown in Table 3, the performance of pipelines utilizing dynamic segmentation, such as with `Jina-V3`, is superior to other approaches. However, this improvement comes at the cost of increased computational requirements and longer processing times. Specifically, embedding the `NFCorpus` dataset entirely with our experimental setup with fixed-size or semantic segmenter takes approximately 30 minutes. In comparison, the `Simple-Qwen` model requires twice the time, while the `Topic-Qwen` model requires four times as long.

Another drawback of these models is their generative nature, which can lead to inconsistencies. They do not always produce the exact same wording for chunks, rendering them less reliable in certain scenarios.

6 Conclusion

While both approaches are effective solutions at mitigating the challenge of context-dilemma, maintaining context in document retrieval in certain scenar-

Model	Chunk	Segm	Length	NDCG@5	MAP@5	F1@5	NDCG@10	MAP@10	F1@10
Stella-V5	Early	Fix-size	512	0.443	**0.137**	**0.226**	**0.414**	**0.161**	**0.247**
	Late	Fix-size	512	**0.445**	0.133	0.225	0.410	0.158	0.242
Jina-V3	Early	Fix-size	512	0.374	0.107	0.186	0.346	0.127	0.204
	Late	Fix-size	512	0.380	0.103	0.185	0.354	0.125	0.210
	Early	Jina-Sem	-	0.377	0.111	0.192	0.353	0.130	0.210
	Late	sim-Qwen	-	0.384	0.105	0.185	0.356	0.126	0.206
	Late	top-Qwen	-	0.383	0.102	0.179	0.351	0.122	0.203
Jina-V2	Early	Fix-size	512	0.261	0.064	0.124	0.237	0.075	0.137
	Late	Fix-size	512	0.280	0.069	0.125	0.255	0.081	0.146
	Early	Jina-Sem	-	0.294	0.079	0.144	0.269	0.092	0.158
	Late	sim-Qwen	-	0.278	0.071	0.130	0.253	0.083	0.146
	Late	top-Qwen	-	0.279	0.070	0.135	0.254	0.081	0.147
BGE-M3	Early	Fix-size	512	0.246	0.059	0.120	0.225	0.069	0.130
	Late	Fix-size	512	0.070	0.010	0.029	0.067	0.013	0.038
	Early	Jina-Sem	-	0.260	0.066	0.122	0.240	0.079	0.144
	Late	sim-Qwen	-	0.091	0.015	0.038	0.081	0.018	0.045
	Late	top-Qwen	-	0.110	0.019	0.044	0.097	0.022	0.048

Table 3. EarlyVsLate Retriever comparison on `NFCorpus`. Bold values indicate the best performance for each metric

Model	Chunk	Segm	Length	NDCG@5	MAP@5	F1@5	NDCG@10	MAP@10	F1@10
Stella-V5	Early	Fix-size	512	**0.630**	**0.501**	**0.019**	**0.632**	**0.502**	**0.011**
	Late	Fix-size	512	0.503	0.340	0.018	0.505	0.341	0.010

Table 4. EarlyVsLate Retriever comparison `MsMarco`. Bold values indicate the best performance for each metric.

Method	NDCG@5	MAP@5	F1@5	NDCG@10	MAP@10	F1@10
Late	0.309	0.143	0.202	0.294	0.160	0.192
Contextual	**0.317**	**0.146**	**0.206**	**0.308**	**0.166**	**0.202**

Table 5. Latechunking (Late) comparison versus ContextualRankFusion (Contextual) best performances, on same `NFCorpus` dataset subset (20% of the whole). Embedding Model: `Jina-V3`. Chunking Method: Fixed-Window Chunking.

ios, both cannot be considered definitive solutions to tackle the problem. Late chunking offers a more computationally efficient solution by leveraging the natural capabilities of embedding models. In contrast, contextual retrieval, with its reliance on LLMs for context augmentation and re-ranking, incurs higher computational expenses. It also notable that the type of document and it's length can affect the performances, together with the LLM chosen for the task, smaller and more efficient models performing worse.

This distinction is crucial for applications where computational resources are a significant consideration like in real-world scenarios.

References

1. Abdin, M., Aneja, J., Awadalla, H., Awadallah, A., Awan, A.A., Bach, N., Bahree, A., Bakhtiari, A., Bao, J., Behl, H., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, M., Cai, Q., Chaudhary, V., Chen, D., Chen, D., Chen, W., Chen, Y.C., Chen, Y.L., Cheng, H., Chopra, P., Dai, X., Dixon, M., Eldan, R., Fragoso, V., Gao, J., Gao, M., Gao, M., Garg, A., Giorno, A.D., Goswami, A., Gunasekar, S., Haider, E., Hao, J., Hewett, R.J., Hu, W., Huynh, J., Iter, D., Jacobs, S.A., Javaheripi, M., Jin, X., Karampatziakis, N., Kauffmann, P., Khademi, M., Kim, D., Kim, Y.J., Kurilenko, L., Lee, J.R., Lee, Y.T., Li, Y., Li, Y., Liang, C., Liden, L., Lin, X., Lin, Z., Liu, C., Liu, L., Liu, M., Liu, W., Liu, X., Luo, C., Madan, P., Mahmoudzadeh, A., Majercak, D., Mazzola, M., Mendes, C.C.T., Mitra, A., Modi, H., Nguyen, A., Norick, B., Patra, B., Perez-Becker, D., Portet, T., Pryzant, R., Qin, H., Radmilac, M., Ren, L., de Rosa, G., Rosset, C., Roy, S., Ruwase, O., Saarikivi, O., Saied, A., Salim, A., Santacroce, M., Shah, S., Shang, N., Sharma, H., Shen, Y., Shukla, S., Song, X., Tanaka, M., Tupini, A., Vaddamanu, P., Wang, C., Wang, G., Wang, L., Wang, S., Wang, X., Wang, Y., Ward, R., Wen, W., Witte, P., Wu, H., Wu, X., Wyatt, M., Xiao, B., Xu, C., Xu, J., Xu, W., Xue, J., Yadav, S., Yang, F., Yang, J., Yang, Y., Yang, Z., Yu, D., Yuan, L., Zhang, C., Zhang, C., Zhang, J., Zhang, L.L., Zhang, Y., Zhang, Y., Zhang, Y., Zhou, X.: Phi-3 technical report: A highly capable language model locally on your phone (2024), https://arxiv.org/abs/2404.14219
2. Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., Rosenberg, M., Song, X., Stoica, A., Tiwary, S., Wang, T.: Ms marco: A human generated machine reading comprehension dataset (2018), https://arxiv.org/abs/1611.09268
3. Boteva, V., Gholipour Ghalandari, D., Sokolov, A., Riezler, S.: A full-text learning to rank dataset for medical information retrieval. vol. 9626, pp. 716–722 (03 2016). https://doi.org/10.1007/978-3-319-30671-1_58
4. Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., Liu, Z.: Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation (2024), https://arxiv.org/abs/2402.03216
5. Ding, Y., Zhang, L.L., Zhang, C., Xu, Y., Shang, N., Xu, J., Yang, F., Yang, M.: Longrope: Extending LLM context window beyond 2 million tokens. In: Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net (2024), https://openreview.net/forum?id=ONOtpXLqqw
6. Fan, W., Ding, Y., Ning, L., Wang, S., Li, H., Yin, D., Chua, T.S., Li, Q.: A survey on rag meeting llms: Towards retrieval-augmented large language models (2024), https://arxiv.org/abs/2405.06211
7. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., Wang, H.: Retrieval-augmented generation for large language models: A survey (2024), https://arxiv.org/abs/2312.10997
8. Gupta, S., Ranjan, R., Singh, S.: A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions (10 2024). https://doi.org/10.48550/arXiv.2410.12837
9. Günther, M., Mohr, I., Williams, D.J., Wang, B., Xiao, H.: Late chunking: Contextual chunk embeddings using long-context embedding models (2024), https://arxiv.org/abs/2409.04701

10. Günther, M., Ong, J., Mohr, I., Abdessalem, A., Abel, T., Akram, M.K., Guzman, S., Mastrapas, G., Sturua, S., Wang, B., Werk, M., Wang, N., Xiao, H.: Jina embeddings 2: 8192-token general-purpose text embeddings for long documents (2024), https://arxiv.org/abs/2310.19923
11. Hsieh, C.Y., Chuang, Y.S., Li, C.L., Wang, Z., Le, L.T., Kumar, A., Glass, J., Ratner, A., Lee, C.Y., Krishna, R., Pfister, T.: Found in the middle: Calibrating positional attention bias improves long context utilization (2024), https://arxiv.org/abs/2406.16008
12. Jina.ai: Finding optimal breakpoints in long documents using small language models (Oct 2024), https://jina.ai/news/finding-optimal-breakpoints-in-long-documents-using-small-language-models/
13. Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., tau Yih, W.: Dense passage retrieval for open-domain question answering (2020), https://arxiv.org/abs/2004.04906
14. Koshorek, O., Cohen, A., Mor, N., Rotman, M., Berant, J.: Text segmentation as a supervised learning task (2018), https://arxiv.org/abs/1803.09337
15. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-augmented generation for knowledge-intensive nlp tasks (2021), https://arxiv.org/abs/2005.11401
16. Lu, T., Gao, M., Yu, K., Byerly, A., Khashabi, D.: Insights into llm long-context failures: When transformers know but don't tell (2024), https://arxiv.org/abs/2406.14673
17. Moro, G., Ragazzi, L.: Align-then-abstract representation learning for low-resource summarization. Neurocomputing **548**, 126356 (2023). https://doi.org/https://doi.org/10.1016/j.neucom.2023.126356, https://www.sciencedirect.com/science/article/pii/S0925231223004794
18. Muennighoff, N., Tazi, N., Magne, L., Reimers, N.: Mteb: Massive text embedding benchmark (2023), https://arxiv.org/abs/2210.07316
19. Robertson, S., Zaragoza, H.: The probabilistic relevance framework: Bm25 and beyond. Foundations and Trends in Information Retrieval **3**, 333–389 (01 2009). https://doi.org/10.1561/1500000019
20. Sturua, S., Mohr, I., Akram, M.K., Günther, M., Wang, B., Krimmel, M., Wang, F., Mastrapas, G., Koukounas, A., Wang, N., Xiao, H.: jina-embeddings-v3: Multilingual embeddings with task lora (2024), https://arxiv.org/abs/2409.10173