LumberChunker: Long-Form Narrative Document Segmentation

André V. Duarte[1], João Marques[1], Miguel Graça[1],
Miguel Freire[2], Lei Li[3], Arlindo L. Oliveira[1]
[1]INESC-ID / Instituto Superior Técnico, [2]NeuralShift, [3]Carnegie Mellon University
{andre.v.duarte, joao.p.d.s.marques, arlindo.oliveira}@tecnico.ulisboa.pt
miguel.graca@inesc-id.pt
miguel@neuralshift.ai
leili@cs.cmu.edu

Abstract

Modern NLP tasks increasingly rely on dense retrieval methods to access up-to-date and relevant contextual information. We are motivated by the premise that retrieval benefits from segments that can vary in size such that a content's semantic independence is better captured. We propose LumberChunker, a method leveraging an LLM to dynamically segment documents, which iteratively prompts the LLM to identify the point within a group of sequential passages where the content begins to shift. To evaluate our method, we introduce GutenQA, a benchmark with 3000 "needle in a haystack" type of question-answer pairs derived from 100 public domain narrative books available on Project Gutenberg[1]. Our experiments show that LumberChunker not only outperforms the most competitive baseline by 7.37% in retrieval performance (DCG@20) but also that, when integrated into a RAG pipeline, LumberChunker proves to be more effective than other chunking methods and competitive baselines, such as the Gemini 1.5M Pro.

1 Introduction

The rapid expansion of Large Language Models (LLMs) has paved the way for tackling a wide range of tasks, including translation (Alves et al., 2024), summarization (Kedia et al., 2021), and question answering (Huang et al., 2023), among others (Zhao et al., 2023). However, a significant issue arises when these models are tasked with generating responses based on information they lack, often resulting in "hallucinations" - responses that, while seemingly plausible, are factually incorrect (Zhang et al., 2023). Given the broad accessibility of these models, less informed users may accept all generated content as accurate, potentially causing severe misinterpretations and adverse consequences, like the recent incident where a lawyer cited fictitious cases produced by ChatGPT (OpenAI, 2022) in court, resulting in sanctions for the lawyer and highlighting the severe risks of unverified AI-generated information (Bohannon, 2023).

In the field of question answering, where precision and accuracy of information are paramount, Retrieval Augmented Generation (RAG) systems present a viable solution to hallucinations by grounding the model's generation on contextually relevant documents (Lewis et al., 2020).

One often overlooked part of the RAG pipeline is how textual content is segmented into 'chunks', which can significantly impact the dense retrieval quality (Shi et al., 2023). Real-world applications tend to simplify this step and consider sentences, paragraphs, or propositions as the usual granularity level (Tiedemann and Mur, 2008; Chen et al., 2023).

In this paper, we propose LumberChunker, a novel text segmentation method based on the principle that retrieval efficiency improves when content chunks are as independent as possible from one another. This independence is best achieved by allowing chunks to be of dynamic sizes. Given that language models excel at analyzing text, we leverage their capabilities to identify optimal segmentation points. Specifically, we repeatedly instruct a language model to receive a series of continuous paragraphs and determine the precise paragraph within the sequence where the content starts diverging. This approach ensures that each segment is contextually coherent yet distinct from adjacent segments, thereby enhancing the effectiveness of information retrieval.

We introduce a new benchmark: GutenQA, which consists of 100 public domain narrative books manually extracted from Project Gutenberg[2]. We create 3000 high-quality question-answer pairs

[1]Code and Data available at: https://github.com/joaodsmarques/LumberChunker

[2]https://www.gutenberg.org/

Findings of the Association for Computational Linguistics: EACL 2024, pages 6473–6486

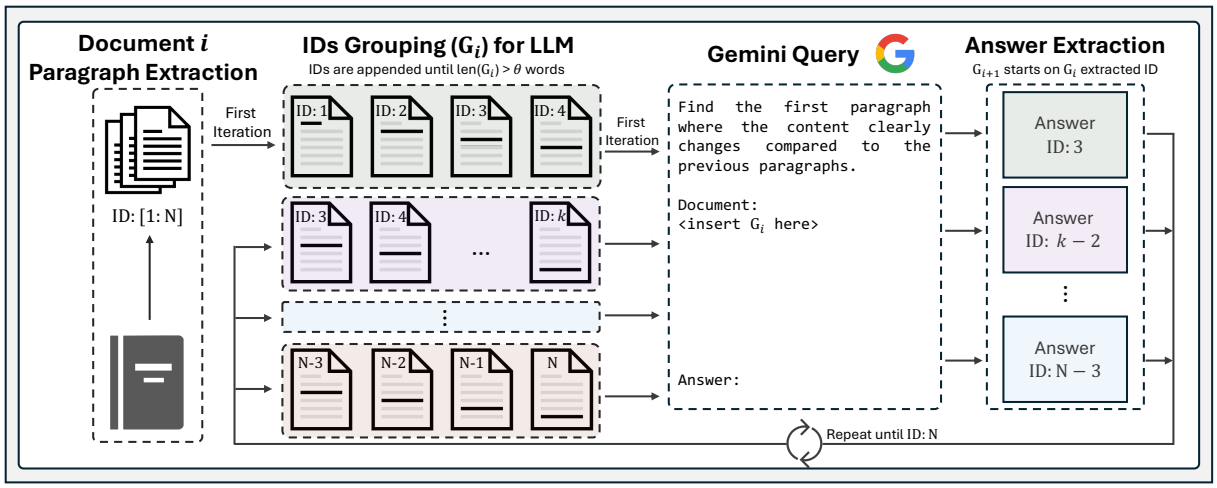

Figure 1: LumberChunker follows a three-step process. First, we segment a document paragraph-wise. Secondly, a group (G_i) is created by appending sequential chunks until exceeding a predefined token count θ. Finally, G_i is fed as context to Gemini, which determines the ID where a significant content shift starts to appear, thus defining the start of G_{i+1} and the end of the current chunk. This process is cyclically repeated for the entire document.

from these books to evaluate the impact of LumberChunker on retrieval. Finally, we integrate LumberChunker into a RAG pipeline for a downstream QA task to assess its effect on the accuracy of the generated outputs.

Our main contributions are as follows:

- We present LumberChunker, a novel text segmentation method that leverages an LLM to create semantically independent chunks, enhancing retrieval performance for dense passage retrieval tasks.
- We introduce GutenQA, a new benchmark composed of 100 narrative books from Project Gutenberg, featuring 3000 question-answer pairs designed to evaluate the effectiveness of text segmentation models in retrieval tasks.
- We present results showing that LumberChunker outperforms the best prior segmentation baseline by 7.37% in DCG@20. Our analysis also reveals that LumberChunker chunks align more closely with manually created chunks, further supporting its effectiveness in text segmentation.

2 Background

The retrieval granularity at which a document is segmented plays an essential role as ineffective chunking strategies can lead to chunks with incomplete context or excessive irrelevant information, which damage the performance of retriever models (Yu et al., 2023).

Beyond typical granularity levels like sentences or paragraphs (Gao et al., 2023b), other advanced methods can be employed. Recursive character splitting (Langchain, 2023) segments text based on a hierarchy of separators such as paragraph breaks, new lines, spaces, and individual characters. While this method better respects the document's structure, it may lack contextual understanding. To address this, semantic-based splitting (Kamradt, 2024) utilizes embeddings to cluster semantically similar text segments. This approach ensures that chunks maintain meaningful context and coherence by identifying breakpoints based on significant changes in embedding distances.

The recent work by Chen et al. (2023) introduces a novel retrieval granularity termed *propositions* - minimal textual units, each conveying an individual fact in a clear, self-sufficient natural language format. While this concept is valid for contexts with fact-based information, like Wikipedia, it may be less effective for narrative texts where the flow and contextual continuity play a critical role (as illustrated in Appendix A). Furthermore, the shift in the search space from the original text to these new representations also places considerable reliance on the generating LLM to ensure their quality.

Retrieval granularity is often viewed at the document level, but it can also involve adjusting the query itself. Gao et al. (2023a) suggests the Hypothetical Document Embeddings (HyDE) method, where an LLM transforms the query into a potential answer document.

3 Methodology

3.1 LumberChunker

Our main contribution is a novel method for document segmentation named LumberChunker, which employs an LLM to dynamically segment documents into semantically independent chunks. Our approach is grounded in the principle that retrieval benefits from segments that can vary in size to better capture the content's semantic independence. This dynamic granularity ensures that each chunk encapsulates a complete, standalone idea, enhancing the relevance and clarity of the retrieved documents. By feeding the LLM a set of sequential passages, LumberChunker autonomously determines the most appropriate points for segmentation. This decision process takes into account the structure and semantics of the text, thereby enabling the creation of chunks that are optimally sized and contextually coherent.

Figure 1 displays the overall pipeline of LumberChunker. We start by splitting the target document paragraph-wise, with each paragraph being uniquely identified by an incremental ID number. Each paragraph is sequentially concatenated into a group G_i until its collective token count surpasses a pre-determined threshold, θ, which is strategically set based on empirical insights, further discussed in 5.1. The goal is to set θ large enough to avoid bisecting relevant larger segments while ensuring it is small enough to prevent overwhelming the model with excessive context, which could hinder its reasoning accuracy. The group G_i is given as input to the LLM (we choose Gemini 1.0-Pro (Team et al., 2023)), which we instruct to pinpoint the specific paragraph within G_i where the content is perceived to diverge significantly from the preceding context. This detection marks the end of a chunk. The document keeps being sequentially partitioned into chunks in a cyclical manner, with the starting point of each new G_{i+1} group being the paragraph identified in the previous iteration. The prompt used is provided in Appendix B.

3.2 GutenQA

Our proposed benchmark comprises a collection of 100 English books sourced from Project Gutenberg. Due to the diverse and inconsistent HTML structures of these books, we extract the content manually. This avoids potential errors that often arise with automatic text extraction, as discussed in Appendix C.

We use ChatGPT (`gpt-3.5-turbo-0125`) to generate questions and answers for each book. Initially, more than 10000 questions are automatically generated, which are then filtered down manually such that each book has 30 high-quality questions. To better evaluate retrieval performance, we prioritize factual and specific questions that target unique information, typically answerable in 1-2 sentences. This selection strategy favors 'what,' 'when,' and 'where' questions over 'why' and 'how' questions.

The model is additionally instructed to output a concise verbatim span from the text that contains the answer to each question. This span serves as the ground-truth passage for evaluating retrieval accuracy. Given the size and specificity of the span, irrespective of the chunking method employed, the correct chunk will fully contain the specified string. Our evaluation method relies on performing an exact match search for this substring within the retrieved text chunks.

The prompt used to instruct the model to generate questions, along with statistics about the distribution of question types within the dataset, is provided in Appendix D.

4 Experiments

We evaluate LumberChunker using a series of diverse experiments. The key questions that guide our experimental evaluation are as follows:

- **What is the optimal threshold for target token count in each LumberChunker prompt?** The number of tokens in the prompt provided to the LLM directly correlates with the token count in the resulting chunk. Intuitively, one might hypothesize that increasing the prompt length reduces retrieval difficulty, as larger chunks reduce the overall search space, thereby increasing the likelihood of retrieving the correct chunk. To test this hypothesis, we analyze how LumberChunker's DCG@k scores, where k is the number of chunks being retrieved, are influenced by different prompt lengths $\theta \in [450, 1000]$ tokens.

- **Does LumberChunker enhance retrieval?** We evaluate LumberChunker's ability to locate highly specific information within the documents, as represented by our GutenQA questions. We compare its DCG@k and Recall@k scores against other baseline methods, such as Semantic or Proposition-Level chunking.

Table 1: Passage retrieval performance (DCG@k and Recall@k) on GutenQA with different granularities on the questions[†] and on the retrieval corpus passages. The best scores in each column are highlighted in **bold**.

	DCG @ k					**Recall @ k**				
	1	2	5	10	20	1	2	5	10	20
Semantic Chunking	29.50	35.31	40.67	43.14	44.74	29.50	38.70	50.60	58.21	64.51
Paragraph-Level	36.54	42.11	45.87	47.72	49.00	36.54	45.37	53.67	59.34	64.34
Recursive Chunking	39.04	45.37	50.66	53.25	54.72	39.04	49.07	60.64	68.62	74.35
HyDE[†]	33.47	39.74	45.06	48.14	49.92	33.47	43.41	55.11	64.61	71.61
Proposition-Level	36.91	42.42	44.88	45.65	46.19	36.91	45.64	51.04	53.41	55.54
LumberChunker	**48.28**	**54.86**	**59.37**	**60.99**	**62.09**	**48.28**	**58.71**	**68.58**	**73.58**	**77.92**

- **Do LumberChunker chunks enhance generation quality?** It is natural to question whether the increased computational cost of segmenting a document with our method is worthwhile. To address this, we evaluate if LumberChunker chunks improve generation quality in a QA task. For this purpose, we integrate our chunks into a RAG pipeline and create a smaller QA test set comprising 280 questions based on four narrative autobiographies. As these elements are outside the paper's main scope, further details are provided in Appendix E. We compare our approach with other RAG pipeline variants using different chunking techniques, including manually created chunks, which we consider the gold standard for optimal retrieval. We also employ non-RAG baselines like Gemini 1.5 Pro (Reid et al., 2024), capable of processing up to 1.5 million input tokens, and a closed-book setup where Gemini Pro relies solely on internal knowledge.

- **How similar are LumberChunker chunks to the manually created ones?** We employ the Rouge-L metric to quantify the similarity between the chunks produced by LumberChunker and the manually created ones for the smaller QA test set. This approach enabled us to assess the degree to which LumberChunker can replicate human judgment in chunk creation and whether this alignment correlates with differences in accuracy performance.

5 Results and Discussion

5.1 Context Size

Figure 2 reveals that among the various thresholds tested, $\theta = 550$ leads to the best performance, achieving the highest DCG@k scores across all values of k tested. This indicates that prompts with around 550 tokens optimize the quality of document retrieval by effectively balancing context capture and passage length. Following this, thresholds $\theta = 450$ and $\theta = 650$ show similar but slightly lower performances, suggesting that while they are effective, they do not capture the optimal balance as well as $\theta = 550$. The threshold $\theta = 1000$ performs the worst, with noticeably lower DCG@k scores. Given that this task requires advanced reasoning, an excessively long prompt may overwhelm the model's capacity to focus on the relevant parts of the input, thus compromising its performance.

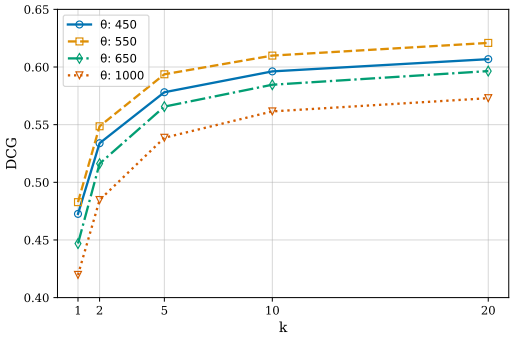

Figure 2: Optimizing Context Size θ ($\approx$ number of tokens in the LumberChunker prompt.)

5.2 Main Results

The results presented in Table 1 highlight that for all values of k, LumberChunker consistently outperforms every other baseline both on DCG@k and Recall@k metrics[3]. This is particularly evident at $k = 20$, where LumberChunker's DCG score reaches 62.09, while the closest competitor, Recursive chunking, only achieves a score of 54.72. Similarly, in terms of Recall@k, LumberChunker

[3]Chunks are encoded with `text-embedding-ada-002` embeddings from OpenAI.

attains a score of 77.92 at $k = 20$, compared to Recursive Chunking's 74.35.

A closer examination of the baselines reveals that methods like Paragraph-Level and Semantic chunking fail to scale effectively as k increases, indicating their limitations in maintaining relevance over a larger number of retrieved documents. HyDE, which uses Recursive chunking as its document granularity level, also fails to outperform its simpler counterpart for every value of k. This suggests that the additional augmentation layer the HyDE introduces may not be suited for this task.

The scores for Proposition-Level chunking are notably lower than those of LumberChunker. While Proposition-Level chunking excels in contexts with fine-grained, fact-based information, such as Wikipedia text, it is less effective for narrative texts where flow and contextual continuity play a critical role. For details on the segmentation of GutenQA, refer to Appendix F.

5.3 Impact on QA Systems

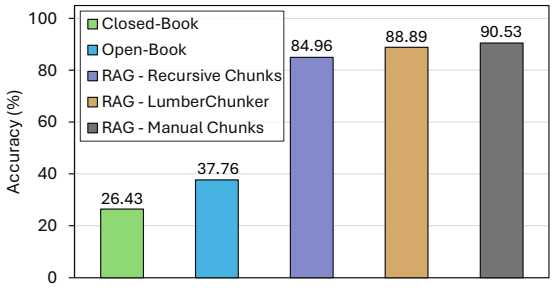

Figure 3: QA Accuracy on Autobiographies Test Set.

In Figure 3, we observe the performance of different QA methodologies applied to several autobiographies. Surprisingly, the Open-Book method achieves an accuracy of 37.76%. Despite being designed to handle large inputs, the model struggles to efficiently process and utilize all the information. In contrast, all RAG methods outperform Open-Book, highlighting the effectiveness of retrieval-based approaches.

Furthermore, our proposed LumberChunker, when integrated into the RAG pipeline, outperforms again Recursive chunking, which was the most competitive baseline in the retrieval evaluation (Table 1). This reinforces our view that better retrieval directly improves accuracy. LumberChunker only falls short of the RAG with manual chunks, which is the expected gold standard for the task.

5.4 Chunk Similarity

Table 2 presents the average Rouge-L scores, comparing the chunks generated by LumberChunker and Recursive Chunks (our top two automated methods from the QA experiment) against the manually created (gold standard) chunks.

Table 2: Average Rouge-L scores of methods compared to Manual Chunks.

Method	Average Rouge-L Score
LumberChunker	0.709
Recursive Chunks	0.689

The high Rouge-L scores indicate a significant overlap between the chunks produced by both LumberChunker and Recursive Chunks when compared to the manual chunks. LumberChunker achieves the highest Rouge-L score, supporting our hypothesis that it generates chunks more closely aligned with the manual segmentation. This similarity is likely a key factor contributing to the improved QA accuracy observed in Figure 3, where the RAG system using LumberChunker chunks outperforms the Recursive Chunks baseline.

6 Conclusions

In this study, we introduce LumberChunker, a novel text segmentation method leveraging LLM textual capabilities to dynamically segment documents into semantically coherent chunks.

We also present GutenQA, a collection of 100 carefully parsed public domain books, segmented with LumberChunker, for which we create 3000 curated question-answer pairs.

Our experiments demonstrate that LumberChunker significantly improves retrieval performance over competitive baselines, and when integrated into a RAG pipeline, it also enhances the accuracy of generated answers.

Acknowledgments

We acknowledge the financial support provided by the Recovery and Resilience Fund towards the Center for Responsible AI (Ref. C628696807-00454142), and the financing of the Foundation for Science and Technology (FCT) for INESC-ID (Ref. UIDB/50021/2020). Lei Li is partly supported by the CMU CyLab seed grant.

Limitations

Despite LumberChunker demonstrating superior performance compared to all baselines, it should be highlighted that it requires the use of an LLM, which automatically renders it more expensive and slower compared to traditional methods like Recursive chunking (further details in Appendix G).

LumberChunker is designed for narrative texts, which are somewhat unstructured and benefit from semantic textual interpretation. However, for scenarios with highly structured texts like those in the legal domain, LumberChunker may be an unnecessarily complex solution because it would likely achieve similar segmentations as those employing document-wise structure parsing (like Markdown segmentation), but at a more expensive cost.

Within the present methodology, LumberChunker also faces scalability issues with the length of individual documents and the volume of documents that need to be processed. While each document needs to be processed only once, the iterative nature of prompting the language model to identify segmentation points can become a drawback when dealing with large number of documents.

Ethical Considerations

This paper focuses on improving text segmentation methods by leveraging existing large language models. Our released dataset, GutenQA, uses only public domain texts, ensuring there are no privacy concerns or handling of sensitive data. Regarding LumberChunker, we do not foresee any direct impact on malicious activities, disinformation, surveillance, or any significant environmental impact beyond the typical computational requirements.

The only ethical consideration we would like to highlight is our extensive use of black box models. Unlike traditional chunking techniques like Recursive Chunking, which are fully transparent and easily reproducible, black box models introduce some uncertainty regarding their outputs. As a result, it is not impossible that our methodology might have some biases we are unaware of.

References

Duarte Miguel Alves, José Pombal, et al. 2024. Tower: An Open Multilingual Large Language Model for Translation-Related Tasks. In *First Conference on Language Modeling*.

FP Balsemão. 2021. *Memórias*. Porto Editora.

Molly Bohannon. 2023. Lawyer Used ChatGPT In Court And Cited Fake Cases—A Judge Is Considering Sanctions. Accessed: 2024-05-20.

Tong Chen, Hongwei Wang, Sihao Chen, et al. 2023. Dense X Retrieval: What Retrieval Granularity Should We Use? *Preprint*, arXiv:2312.06648.

B Franklin. 2006. *Autobiography of Benjamin Franklin*. Project Gutenberg. https://www.gutenberg.org/ebooks/20203.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023a. Precise Zero-Shot Dense Retrieval without Relevance Labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777, Toronto, Canada. Association for Computational Linguistics.

Yunfan Gao, Yun Xiong, Xinyu Gao, et al. 2023b. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

M Ghandi. 1929. *An Autobiography or The Story of My Experiments with Truth*. Navajivan Publishing House. https://www.mkgandhi.org/ebks/An-Autobiography.pdf.

Jiaxin Huang, Shixiang Gu, Le Hou, et al. 2023. Large Language Models Can Self-Improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1051–1068, Singapore. Association for Computational Linguistics.

Greg Kamradt. 2024. Semantic Chunking. https://github.com/FullStackRetrieval-com/RetrievalTutorials/tree/main/tutorials/LevelsOfTextSplitting.

Akhil Kedia, Sai Chetan Chinthakindi, and Wonho Ryu. 2021. Beyond Reptile: Meta-learned dot-product maximization between gradients for improved single-task regularization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 407–420.

H Keller. 2000. *The Story of My Life*. Project Gutenberg. https://www.gutenberg.org/ebooks/2397.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The NarrativeQA Reading Comprehension Challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328.

Langchain. 2023. RecursiveCharacterTextSplitter Documentation.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Nelson F Liu, Kevin Lin, John Hewitt, et al. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

OpenAI. 2022. Introducing Chat-GPT. https://openai.com/blog/chatgpt. Accessed: 2022-11-30.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *Preprint*, arXiv:2403.05530.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Jörg Tiedemann and Jori Mur. 2008. Simple is Best: Experiments with Different Document Segmentation Strategies for Passage Retrieval. In *Coling 2008: Proceedings of the 2nd workshop on Information Retrieval for Question Answering*, pages 17–25, Manchester, UK. Coling 2008 Organizing Committee.

Wenhao Yu, Hongming Zhang, Xiaoman Pan, Kaixin Ma, Hongwei Wang, and Dong Yu. 2023. Chain-of-Note: Enhancing Robustness in Retrieval-Augmented Language Models. *Preprint*, arXiv:2311.09210.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.

Xuandong Zhao, Siqi Ouyang, Zhiguo Yu, Ming Wu, and Lei Li. 2023. Pre-trained Language Models Can be Fully Zero-Shot Learners. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15590–15606, Toronto, Canada. Association for Computational Linguistics.

A Propositions Example on Narrative Texts

Table 3: Example of **good** passage transformation into propositions.

Passage 1: *Elon Musk's attraction to risk was a family trait. In that regard, he took after his maternal grandfather,* ***Joshua Haldeman****, a daredevil adventurer with strongly held opinions who was raised on a farm on the barren plains of central Canada.* ***He*** *studied chiropractic techniques in Iowa, then returned to his hometown near Moose Jaw, where he broke in horses and gave chiropractic adjustments in exchange for food and lodging.*
Proposition 1.1: Elon Musk's attraction to risk was a family trait.
Proposition 1.2: His maternal grandfather was Joshua Haldeman.
Proposition 1.3: Joshua Haldeman was a daredevil adventurer with strongly held opinions.
Proposition 1.4: Joshua Haldeman was raised on a farm on the barren plains of central Canada.
Proposition 1.5: Joshua Haldeman studied chiropractic techniques in Iowa.
Proposition 1.6: Joshua Haldeman returned to his hometown near Moose Jaw.
Proposition 1.7: In his hometown, Joshua Haldeman broke in horses and gave chiropractic adjustments.
Proposition 1.8: Joshua Haldeman received food and lodging in exchange for his chiropractic services.

Table 4: Example of **poor** passage transforming into propositions.

Passage 2: ***He*** *was eventually able to buy his own farm, but he lost it during the depression of the 1930s. For the next few years, he worked as a cowboy, rodeo performer, and construction hand. His one constant was a love for adventure.* ***He*** *married and divorced, traveled as a hobo on freight trains, and was a stowaway on an oceangoing ship.*
Proposition 2.1: He was eventually able to buy his own farm.
Proposition 2.2: He lost his own farm during the depression of the 1930s.
Proposition 2.3: For the next few years, he worked as a cowboy.
Proposition 2.4: For the next few years, he worked as a rodeo performer.
Proposition 2.5: For the next few years, he worked as a construction hand.
Proposition 2.6: His one constant was a love for adventure.
Proposition 2.7: He married and divorced.
Proposition 2.8: He traveled as a hobo on freight trains.
Proposition 2.9: He was a stowaway on an oceangoing ship.

Comment: Unlike the example in Table 3, the pronoun '**He**' in Table 4 passage cannot be accurately co-referenced, resulting in somewhat ambiguous propositions. Consequently, if a user asks a question like 'Who in Elon Musk's family worked as a rodeo performer?', a model that uses only propositions as retrieval units will not be able to provide an accurate response.

B LumberChunker Gemini Prompt

Table 5: LumberChunker Gemini Prompt example for the book: *Winnie the Pooh* by A. A. Milne.

Prompt: You will receive as input an English document with paragraphs identified by 'ID XXXX: <text>'.

Task: Find the first paragraph (not the first one) where the content clearly changes compared to the previous paragraphs.

Output: Return the ID of the paragraph with the content shift as in the exemplified format: 'Answer: ID XXXX'.

Additional Considerations: Avoid very long groups of paragraphs. Aim for a good balance between identifying content shifts and keeping groups manageable.

Document:
ID 0001: *Here is Edward Bear, coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it. And then he feels that perhaps there isn't. Anyhow, here he is at the bottom, and ready to be introduced to you. Winnie-the-Pooh.*

ID 0002: *When I first heard his name, I said, just as you are going to say, "But I thought he was a boy?"*

ID 0003: *"So did I," said Christopher Robin.*

ID 0004: *"Then you can't call him Winnie?"*

ID 0005: *"I don't."*

...

ID 0018: *So I tried.*

ID 0019: *Once upon a time, a very long time ago now, about last Friday, Winnie-the-Pooh lived in a forest all by himself under the name of Sanders.*

ID 0020: *"What does 'under the name' mean?" asked Christopher Robin.*

Answer: ID 0019

C Project Gutenberg - Manual Extraction

Table 6: Passage manually extracted from Project Gutenberg regarding the book: *Anna Karenina*, by Leo Tolstoy.

GutenQA Verbatim: They were carrying something, and dropped it. "I told you not to sit passengers on the roof," said the little girl in English; "there, pick them up!" "Everything's in confusion," thought Stepan Arkadyevitch; "there are the children running about by themselves." And going to the door, he called them. They threw down the box, that represented a train, and came in to their father. The little girl, her father's favorite, ran up boldly, embraced him, and hung laughingly on his neck, enjoying as she always did the smell of scent that came from his whiskers. At last the little girl kissed his face, which was flushed from his stooping posture and beaming with tenderness, loosed her hands, and was about to run away again; but her father held her back. "How is mamma?" he asked, passing his hand over his daughter's smooth, soft little neck. "Good morning," he said, smiling to the boy, who had come up to greet him. He was conscious that he loved the boy less, and always tried to be fair; but the boy felt it, and did not respond with a smile to his father's chilly smile.

Table 7: Passage from NarrativeQA (Kočiský et al., 2018) regarding the book: *Anna Karenina*, by Leo Tolstoy.

NarrativeQA Verbatim: They were carrying something, and dropped it.\n\nâ\x80\x9cI told you not to sit passengers on the roof,â\x80\x9d said the little girl in\nEnglish; â\x80\x9cthere, pick them up!â\x80\x9d\n\nâ\x80\x9cEverythingâ\x80\x99s in confusion,â\x80\x9d thought Stepan Arkadyevitch; â\x80\x9cthere are\nthe children running about by themselves.â\x80\x9d And going to the door, he\ncalled them. They threw down the box, that represented a train, and\ncame in to their father.\n\nThe little girl, her fatherâ\x80\x99s favorite, ran up boldly, embraced him,\nand hung laughingly on his neck, enjoying as she always did the smell\nof scent that came from his whiskers. At last the little girl kissed\nhis face, which was flushed from his stooping posture and beaming with\ntenderness, loosed her hands, and was about to run away again; but her\nfather held her back.\n\nâ\x80\x9cHow is mamma?â\x80\x9d he asked, passing his hand over his daughterâ\x80\x99s smooth,\nsoft little neck. â\x80\x9cGood morning,â\x80\x9d he said, smiling to the boy, who had\ncome up to greet him. He was conscious that he loved the boy less, and\nalways tried to be fair; but the boy felt it, and did not respond with\na smile to his fatherâ\x80\x99s chilly smile.

Comment: The differences between the passages on Table 6 and Table 7 are significant. The latter passage is marked by misplaced '\n' characters and the presence of non-standard characters such as 'â' and '\x80\x9c', which likely result from encoding issues during the data extraction process. In contrast, the first passage, manually extracted from Project Gutenberg, is more readable and free from such errors.

D Artificial Test Data Generation - GutenQA

Table 8: Test set questions prompt template for the book: *Anna Karenina*, by Leo Tolstoy.

System Prompt: Your task is to generate a question-answer pair that is specific to the provided text excerpts from the book "Anna Karenina" by Leo Tolstoy. The question should be unique to the passage, meaning it cannot be easily answered by other parts of the book.
Instructions:

Read the Passage: Carefully read the provided text excerpt from the book. Understand the context, key events, and specific details mentioned.

Formulate a Question: Create a question that is:

Directly related to the passage: The question should be based on the specific information or events described in the text.

Unique to the passage: The question should not be answerable with information from other parts of the book.

Type: Focus on creating a "When/What/Where" question to encourage specificity and conciseness.

Provide a Concise Answer: Write an answer that is:

Direct and informative: Limit the answer to a maximum of two sentences. Ensure it directly addresses the question and is supported by the passage.

Self-contained: The answer should make sense on its own and should not require additional context from outside the passage.

Cite the Supporting Passage: Include the passage that contains the information needed to answer the question. This will be used to verify the accuracy of the answer and the relevance of the question. Do not use '...'. The passage should be quoted without breaks.

User Prompt: *Example:*
Passage: *"Vous comprenez l'anglais?" asked Lidia Ivanovna, and receiving a reply in the affirmative, she got up and began looking through a shelf of books. "I want to read him 'Safe and Happy,' or 'Under the Wing,'" she said, looking inquiringly at Karenin."*

Question: *What book does Countess Lidia Ivanovna want to read to Karenin?*

Answer: *She wants to read him 'Safe and Happy,' or 'Under the Wing.'*

Supporting Passage: *"I want to read him 'Safe and Happy,' or 'Under the Wing,'" she said, looking inquiringly at Karenin.*

Table 9: Frequency of the first token in GutenQA questions.

	What	**How**	**Why**	**Who**	**Where**	**When**	**Other**
Frequency	49.3%	20.1%	15.2%	7.7%	2.8%	0.3%	4.5%

E LumberChunker impact on Generation - Details

E.1 RAG Pipeline

We build a RAG-based QA pipeline specifically tailored to biographical books. We employ a hybrid retrieval format, combining OpenAI `text-ada-embedding-002` dense embeddings with BM25. Based on Figure 4, we outline the step-by-step process employed in our system.

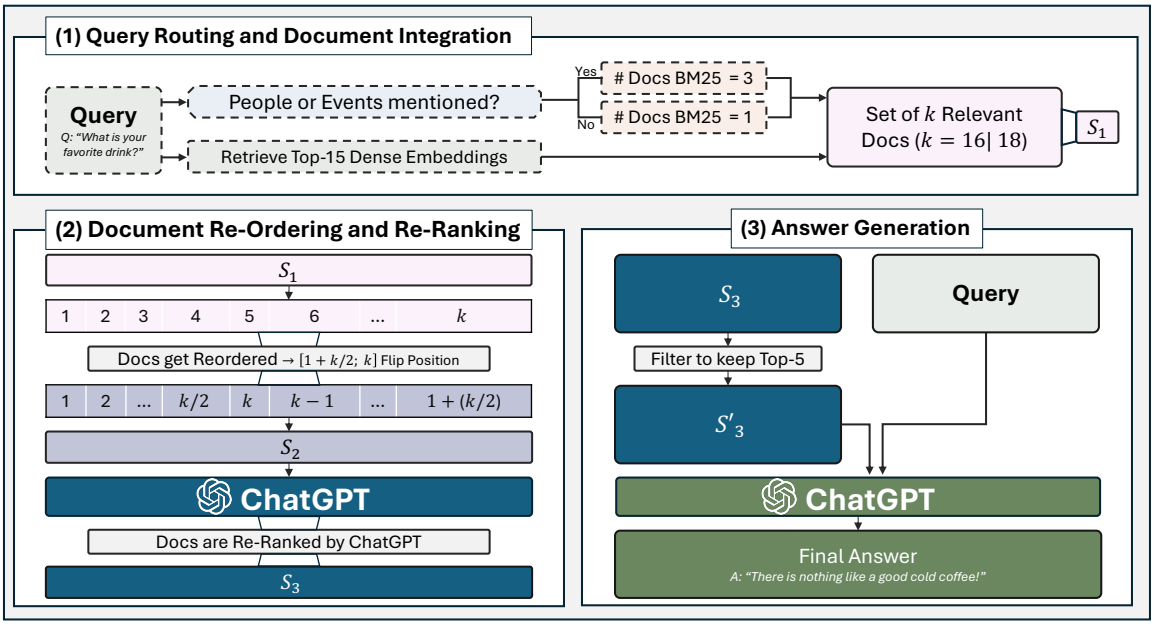

Figure 4: RAG Pipeline for QA on Autobiographies

Query Routing and Document Integration: Each query undergoes an evaluation by a detector that identifies mentions of people or events. If the detector identifies relevant mentions within the query, the top-3 most relevant chunks are retrieved using the BM25 algorithm. A single document is retrieved as a precautionary measure if no mentions are detected (acknowledging occasional detector failures). Concurrently, the top-15 chunks are retrieved through a dense retrieval mechanism. This step is designed to enhance the retrieval quality by accessing deeper semantic relations that BM25 might miss. We implement an intersection check between the documents retrieved by BM25 and those from dense retrieval, and overlapping documents from BM25 are removed to avoid redundancy. The highest-ranked document from BM25 is then prioritized at the top of the retrieval list, with the second and third (if available) placed at the end, ensuring a blend of retrieval strategies.

Document Re-Ordering and Re-Ranking: The retrieved chunks are integrated into the context window of ChatGPT (gpt-3.5-turbo). If the context includes six or more chunks, we employ a strategy to reverse the order of the chunks from the midpoint onwards. This re-ordering aims to potentially minimize the model's 'Lost in the Middle' problem, where model performance degrades for information located in the middle of long contexts but starts recovering towards the end, creating a U-shaped performance curve (Liu et al., 2024). ChatGPT is then prompted to identify and re-order the documents based on their decreasing relevance to the query.

Final Answer Generation: The response is generated in this final step. The top-5 documents, as determined by the model, are retained for the final answer generation. The model synthesizes the information from the top documents into a coherent and contextually accurate answer, aiming to address the query comprehensively.

E.2 Autobiography Questions

We created a dataset based on four autobiographies: (1) Mahatma Gandhi, (2) Helen Keller, (3) Benjamin Franklin, and (4) Francisco Pinto Balsemão (Ghandi, 1929; Keller, 2000; Franklin, 2006; Balsemão, 2021). Each book includes 70 questions, a combination of manually crafted and ChatGPT-generated ones. Specifically, 40 questions are auto-generated following the methodology described for the GutenQA, while the remaining 30 questions are manually created by a human while reading the book.

Each book contains manually created questions and is also manually segmented, as one of the baselines for the task is a RAG Pipeline with manual chunks. Both tasks are time-consuming but crucial for ensuring the true impact of LumberChunker on the generation quality.

F LumberChunker and Baseline methods - Chunks Statistics

Table 10: The average number of tokens per chunk and the total number of chunks after segmenting each book in the GutenQA.

	Avg. #Tokens / Chunk	**Total #Chunks**
Semantic Chunking	185 tokens	191059
Paragraph Level	79 tokens	248307
Recursive Chunking	399 tokens	31787
Proposition-Level	12 tokens	914493
LumberChunker	334 tokens	36917

From Table 10, we observe that the average number of tokens per chunk for LumberChunker is approximately 40% below the intended input size of 550 tokens. This suggests that, on average, the LLM does not frequently select IDs near the end of the input context. This is a positive sign, as consistently choosing IDs near the end could imply the model is not reasoning over the input effectively, leading to the 'lost in the middle' problem (Liu et al., 2024).

Both Paragraph, Recursive, and Proposition-Level chunking exhibit values that align with expectations. The high prevalence of dialogue in the dataset explains the relatively small average number of tokens per chunk at the paragraph level. Recursive chunking achieves an expected value since the `RecursiveCharacterTextSplitter` from `langchain`[4] was configured with an optimal value of 450 tokens in mind. Additionally, the average chunk size for propositions is consistent with findings by Chen et al. (2023), which reported an average proposition length of 11.2 tokens for the processed Wikipedia corpus.

On the other hand, Semantic chunking appears to have a relatively small average token size. We hypothesize that this is primarily due to the nature of the documents (narrative books), which often contain significant amounts of dialogue, resulting in short paragraphs. Consequently, Semantic chunking may not fully capture the broader semantic context intended for each paragraph, thus fragmenting the text more than necessary.

[4] https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/recursive_text_splitter/

G Computational Efficiency and Cost Analysis - LumberChunker and Baselines

G.1 Efficiency

Table 11: The time required to apply LumberChunker or one of the baselines on each book.

	Avg. Seconds to Complete a Book	
	A Christmas Carol (710 Paragraphs)	The Count of Monte Cristo (14339 Paragraphs)
Semantic Chunking	212 seconds	4978 seconds
Recursive Chunking	0.1 seconds	0.6 seconds
HyDE	75 seconds	79 seconds
Proposition-Level	633 seconds	10302 seconds
LumberChunker	95 seconds	1628 seconds

From Table 11, we observe the impact of document size on the completion times of the tested approaches. Recursive chunking, although demonstrating a sixfold time increase between both books, is still the faster method. We attribute this efficiency mainly to the fact that Recursive chunking does not involve any LLM API requests. HyDE, despite employing an LLM, maintains a constant number of LLM queries per book (always 30 API requests), resulting in its completion time being invariant to the document size. On the other hand, LumberChunker, Semantic, and Proposition-Level chunking exhibit significant increases in completion time with larger documents. It is important to note that both Semantic and Proposition-Level chunking can be optimized by making asynchronous OpenAI API requests, significantly reducing completion times. However, LumberChunker does not allow for such optimization because its methodology requires dynamic queries to the LLM that cannot be pre-established. While LumberChunker does enhance retrieval performance over every other baseline, we acknowledge the potential for further optimization.

G.2 Costs

We conducted tests to evaluate the cost of segmenting books. The costs are based on Gemini-1.0-Pro with an input window size of ±550 tokens which, at the time of writing, charges $ 0.50 per million tokens.

Table 12: Cost comparison of segmenting two books with significantly different lengths.

	Book Name	
	A Christmas Carol	**The Count of Monte Cristo**
Length	28k words	450k words
#LumberChunker Chunks	92	1476
Cost	$ 0.03	$ 0.4

These estimates illustrate that while smaller books incur minimal costs, larger documents may require more consideration due to the increased processing requirements. However, segmentation is a one-time process, and the additional cost can be justified by the improved performance it yields. Moreover, as LLM pricing continues to decrease and model capabilities advance, LumberChunker is likely to benefit from these trends in the future.