AutoChunker: Structured Text Chunking and its Evaluation

Arihant Jain, Purav Aggarwal, Anoop Saladi
Amazon
{arihanta, aggap, saladias}@amazon.com

Abstract

Text chunking is fundamental to modern retrieval-augmented systems, yet existing methods often struggle with maintaining semantic coherence, both within and across chunks, while dealing with document structure and noise. We present AutoChunker, a bottom-up approach for text chunking that combines document structure awareness with noise elimination. AutoChunker leverages language models to identify and segregate logical units of information (a chunk) while preserving document hierarchy through a tree-based representation. To evaluate the chunking operator, we introduce a comprehensive evaluation framework based on five core tenets: noise reduction, completeness, context coherence, task relevance, and retrieval performance. Experimental results on Support and Wikipedia articles demonstrate that AutoChunker significantly outperforms existing methods, reducing noise while improving chunk completeness compared to state-of-the-art baselines. When integrated with an online product support system, our approach led to improvements in retrieval performance and customer return rates. Our work not only advances the state of text chunking but also provides a standardized framework for evaluating chunking strategies, addressing a critical gap in the field.

1 Introduction

The growing adoption of retrieval-augmented systems has made effective text chunking increasingly critical for information access and utilization. However, current chunking approaches face significant challenges in maintaining semantic coherence while handling real-world document complexity. Traditional methods often produce chunks that either fragment logical units of information or include irrelevant content, leading to degraded retrieval performance and poor user experiences in production systems.

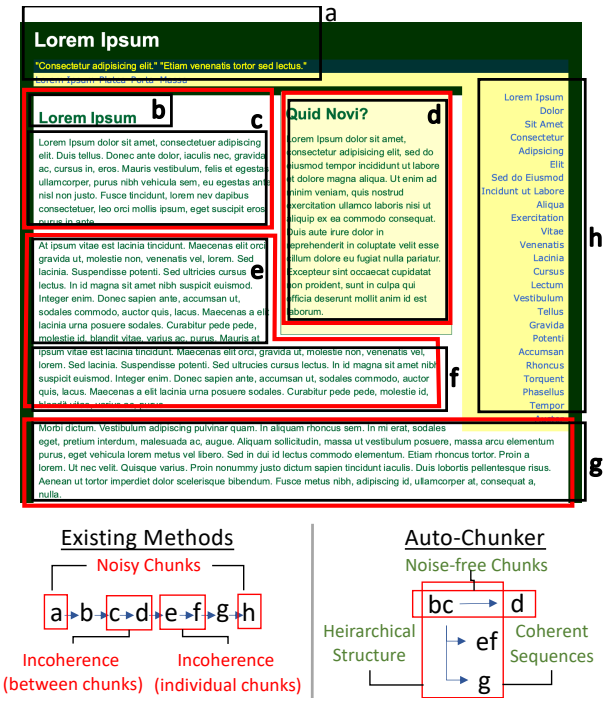

Figure 1: An illustration showcasing the limitations of the existing methods in general and how AutoChunker solves them by generating chunks that are noise-free, coherent and not well entailed.

These limitations are particularly evident in industrial applications, such as online product support systems, where documents often contain rich structure (headers, sections, lists) alongside noise (navigation elements, advertisements, boilerplate text) as shown in Figure 1. While recent approaches have attempted to address these challenges through embedding-based or language model-driven solutions, they typically operate in a top-down manner that struggles to preserve document hierarchy and eliminate noise effectively.

In this paper, we present AutoChunker, a bottom-up approach to text chunking that combines document structure awareness with intelligent noise elimination. Our method first converts documents

Feature	Recursive	Semantic	LGMGC	LLMSemantic	LumberChunker	AutoChunker
Structure Utilization	✓	✗	✗	✗	✗	✓
Noise Elimination	✗	✗	✗	✗	✗	✓
Context Aware Retrieval	✗	✗	✗	✗	✗	✓
Context Switching	✗	✗	✓	✓	✓	✓
Logit Free	-	-	✗	✓	✓	✓
Parameters Insensitivity	✗	✗	✓	✓	✗	✓

Table 1: Comparison of different methods across various features. Features are marked as not available (✗), partially available (✓), fully available (✓), or not applicable (-).

to a standardized markdown format, then employs language models to identify and aggregate logical units of information while preserving the document's hierarchical structure through a tree-based representation. This approach not only maintains semantic coherence within and across chunks but also enables context-aware retrieval through the hierarchical structure.

To systematically evaluate chunking effectiveness, we also introduce an evaluation framework based on five core tenets: noise reduction, completeness, context coherence, task relevance, and retrieval performance. Through extensive experiments on Support and Wikipedia articles, we demonstrate that AutoChunker significantly outperforms existing methods across all evaluation dimensions. In a real-world deployment for an online product support system, our approach led to improvements in both retrieval performance and customer return rates.

2 Related Work

2.1 Chunking Methods

Traditional **static chunking** methods often struggle to maintain logical coherence within and across data units. These methods typically employ fixed granularity levels such as sentences or paragraphs (Gao et al., 2024). More advanced static methods such as Langchain's Recursive chunker (Chase, 2022) employ priority-based separators, including paragraph breaks and new lines. While these methods are simple to implement, they lack the contextual understanding necessary to maintain semantic coherence across chunks.

To overcome the limitations of static chunking, researchers have explored intelligent **dynamic chunking** strategies. These methods aim to identify context switches within the data and create chunks based on semantic coherence rather than arbitrary divisions. Embedding/Semantic-based splitting (Chase, 2022; Smith and Troynikov, 2024) utilizes text embeddings to cluster semantically similar text segments. This method can effectively group related concepts, even when they span multiple paragraphs or sections. However, the quality of the chunks heavily depends on the underlying embedding model's performance. Some works, such as Bayomi and Lawless (2018); Eisenstein (2009); Kazantseva and Szpakowicz (2014), have explored the use of classical ML techniques for text segmentation, which typically rely on lexical and syntactic features to identify coherent segments of text.

Recently, researchers have explored leveraging the capabilities of **LLMs** to perform more intelligent chunking. LLMSemantic (Smith and Troynikov, 2024) provides text as input to an LLM and prompts it to identify splits that result in thematically consistent sections. Another work LumberChunker (Duarte et al., 2024) leverages LLMs to find paragraph splits where the content switches context. Unlike the previous two methods, LGMGC (Liu et al., 2025) utilizes the LLM's internal logits, specifically the probability of the end-of-sentence token `[EOS]`, to determine optimal split points. These LLM-based methods represent a top-down approach to chunking, starting with the full text and recursively identifying appropriate split points. While they offer improved semantic coherence, they may still struggle with noisy data and complex document layout.

2.2 Limitations of Existing Methods

Table 1 provides a comprehensive comparison of existing chunking methods, highlighting their major limitations across the following dimensions:

1. **Structure Utilization**: leveraging document structure (e.g., titles, subtitles) to guide the chunking process.

2. **Noise Elimination**: identifying and eliminating irrelevant content during chunking.

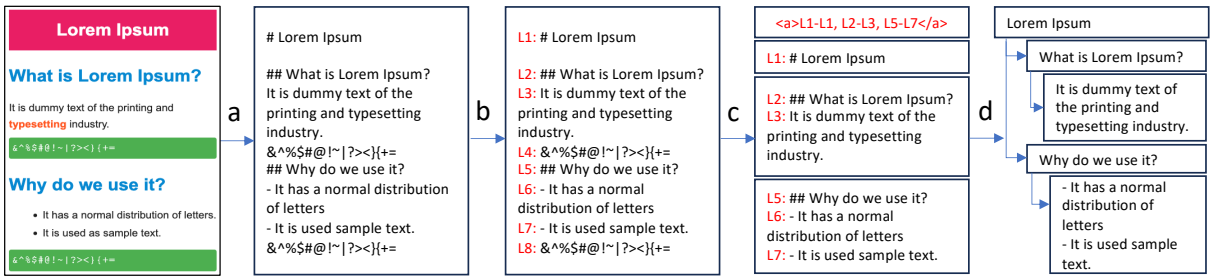

Figure 2: A block diagram of the proposed technique for Intelligent Document Chunking (a, b and c) and Hierarchical Tree Creation (d). The document is first converted to a common markdown format (a) and is then split into logical units (b). Intelligent aggregation and noise filtering (c) is then performed using an LLM.

3. **Context-Aware Retrieval**: effective information retrieval of the chunks using semantic matching.

Table 1 also highlights that additional differences based on their context switching, utilization of LLM logits, and reliance on hyperparameters. Notably, approaches that are *Logit Free* offer greater flexibility in LLM selection. This design choice enables the use of any LLM with API access, not limiting the method to open-source models only.

2.3 Lack of Evaluation

Evaluating the effectiveness of chunking techniques remains underexplored in literature. Traditional evaluation methods rely on downstream task performance, which solely may not directly reflect the quality of the chunking itself (Duarte et al., 2024; Liu et al., 2025). Also, reliance on retrieval is often impractical to assess due to the absence of comprehensive ground truth chunks. Moreover, retrieval performance may be influenced by factors beyond the chunking process itself, such as the embedding module and underlying retrieval algorithm, making it an indirect and potentially unreliable measure of chunking quality.

In light of these limitations, our work not only proposes AutoChunker to address the shortcomings of existing methods but also introduces an unsupervised evaluation framework. This framework utilizes LLMs as impartial judges to assess the quality of text chunks based on five core tenets of effective chunking, which we describe in Section 4. By addressing both the chunking process and its evaluation, we aim to advance the field of text chunking and improve its applicability.

3 Proposed Methodology

3.1 Intelligent Document Chunking

We propose a bottom-up approach to document chunking that preserves the logical structure of the text while enabling efficient retrieval. Unlike top-down methods that start with the full document and recursively split it, our bottom-up strategy begins at the most granular level - individual sentences. The process consists of three key steps:

1. **Document Preprocessing:** We convert the document into Markdown format, preserving heading, subtitles, content, and other structural elements (Figure 2a).

2. **Granular Splitting:** We split the document into its smallest logical units - individual sentences - each assigned a unique identifier (ID) (Figure 2b).

3. **Intelligent Aggregation:** These atomic sentences, along with their IDs, are then fed into an LLM with Prompt 1, present in Appendix B.1. The LLM analyzes the semantic relationships between sentences and identifies logical units of text by generating the start and end IDs of sentences that should be merged. During this aggregation process, the LLM simultaneously identifies and filters out noisy or irrelevant sentences that don't contribute meaningfully to the document's content, as illustrated in Figure 2c.

This approach offers several advantages:

- Ensures a non-lossy chunking by having the LLM generate only identifier tokens instead of summarizing text, thereby preserving fidelity while reducing computational overhead.

- Maintains logical coherence within chunks by dynamically adjusting boundaries based on semantic structure rather than imposing arbitrary length constraints, leading to more meaningful segmentation.
- Enhances retrieval by systematically eliminating irrelevant or noisy content, ensuring that retrieved chunks contain only high-value information relevant to downstream tasks.

3.2 Hierarchical Tree Creation

The noise-free chunks from the chunking process are organized into a hierarchical tree structure (shown in Figure 2d) based on the semantic structure present in the Markdown format. This representation leverages the inherent document hierarchy, where headings, subheadings, and content placement guide the tree's formation. The tree structure captures the document's organizational flow, enabling efficient navigation, retrieval, and preservation of contextual information.

To address the challenge of irregular chunk sizes and potential information loss in vector databases while embedding large chunks, we establish a maximum chunk size threshold. If a chunk exceeds this threshold, it is split into equal parts. While doing so, we maintain the relationships between these split chunks within the tree structure, preserving the original context and sequence. This approach ensures that embedding models can effectively process the chunks while retaining the document's logical structure.

The tree creation process offers several benefits:

- Maintains the document's original structure and hierarchy.
- Facilitates efficient navigation and retrieval of relevant content.
- Preserves the context of each chunk within the broader document layout.
- Optimizes chunk sizes for effective embedding and vector representation.

3.3 Context-Aware Retrieval

Our retrieval method leverages the hierarchical tree structure to provide context-rich results. When a query is processed, we compare it against each chunk in the vectorDB. For chunks that match the query criteria, we output a subtree with that chunk as the root node.

To address user requests for top-K chunks, we first perform a de-duplication process to eliminate overlapping subtrees. This is crucial as both parent and child nodes of a subtree may be retrieved, potentially leading to redundant information. We then rank the remaining subtrees and finally flatten them into a sequence of chunks and return the top-K.

This approach offers several advantages over traditional retrieval methods:

- Provides not just the relevant chunk but also its surrounding context within the document.
- Allows for more nuanced and accurate responses to queries by considering the hierarchical relationships between chunks.
- Enables the retrieval system to provide more comprehensive and contextually appropriate information to users.

4 Proposed Evaluation

We propose an unsupervised evaluation framework that utilizes LLMs as impartial judges (Gu et al., 2025; Jain et al., 2025) to assess the quality of chunks based on five core tenets of effective chunking. These tenets are:

- **Noise Reduction:** Does the chunking reduce noise in the data?
- **Completeness:** Are the chunks self-contained and meaningful?
- **Context Coherence:** Do the chunks minimize context switching?
- **Task Relevance:** Are the chunks relevant to the downstream task?
- **Retrieval Performance:** Does chunking improve the retrieval of relevant information?

4.1 Noise Reduction

To measure the percentage of noise present in the chunks, we provide each chunk to an LLM with the prompt 2, present in Appendix B.2, and ask it to identify if the chunk contains any noise. We define noisy elements as headers, footers, duplicate content, social media buttons, etc., which do not add value in answering the user query.

4.2 Completeness

We use LLMs to assess whether chunks are self-contained and meaningful as shown in Prompt 3 present in Appendix B.2. The completeness score is calculated as the percentage of chunks that are deemed complete.

4.3 Context Switch

We measure the percentage of chunks where there is no effective context switch. An LLM is prompted with 4, present in Appendix B.2, to check if there is any context switching present in the chunk.

4.4 Task Relevance

We calculate the percentage of chunks that are relevant to the downstream task. An LLM is prompted with 5, present in Appendix B.2, to assess if the chunk is relevant to the downstream task (e.g., question answering, support).

4.5 Retrieval Performance

To assess the retrieval performance of our unsupervised approach, we implemented the following methodology.

4.5.1 Query Generation

Since we lack actual queries, we utilized an LLM to generate synthetic queries. We randomly sampled chunks from our dataset and prompted the LLM to create relevant queries with Prompt 7 present in Appendix B.3.

4.5.2 Relevance Scoring

We used the generated queries to search through chunks using an embedding-based retrieval module. We analyzed the top-K retrieved chunks for relevance to the query using an LLM-based relevance scoring system. The prompt used for this scoring is provided in Prompt 6 present in Appendix B.2. The relevance scale is as follows:

- 0 - Irrelevant (no connection to query)
- 1 - Relevant (identifies the query)
- 2 - Somewhat Relevant (contains potential answer)
- 3 - Completely Relevant (contains both query and answer)
- 4 - Perfectly Relevant (exact match for query and answer)

We use weighted precision@K to measure the performance as:

$$\texttt{WP@K} = \frac{\sum_{i=1}^{K} \texttt{rel(i)}}{\texttt{max(rel)} \times \texttt{K}} \times 100$$

where `rel(i)` is the relevance score of the `i`-th retrieved chunk from top-K retrieved chunks, and `max(rel)` is the maximum relevance score (4 in this case).

5 Experimental Setup

5.1 Datasets

We evaluate our approach on two distinct domains: Support and Wikipedia. To obtain structured data for these domains, we used Common Crawl dataset (Crawl, 2025) containing raw HTML web pages.

For the Support domain, we filtered pages related to product support from top brands such as Apple and Samsung. The raw HTML text was extracted from the dataset, focusing on support pages addressing product issues. Here the content is usually structured with sections such as problem description, symptoms, and step-by-step solutions.

For the Wikipedia domain, we randomly sampled Wikipedia HTML pages. These pages cover a diverse range of topics, including products, countries, and notable individuals. The Wikipedia content is inherently structured, featuring sections like introduction, history, and references.

5.2 Baselines and Implementation Details

We compared our approach with static and dynamic chunking baselines using unstructured (raw text) and structured (HTML, Markdown) input formats. Static baselines include:

- **Recursive + Text**: We extracted text from raw HTML using BeautifulSoup (Richardson, 2007) and chunked it using Langchain's RecursiveCharacterTextSplitter (Chase, 2022).
- **Recursive + HTML**: We utilized the implementation released by Liu (2024), which is considered to be the most practical chunking method for HTML input.
- **Recursive + Markdown**: We converted HTML content to markdown and used Langchain's MarkdownHeaderTextSplitter (Chase, 2022) for chunking.

Dynamic baselines include:

Domain	Method	Input	Noise (↓)	Complete (↑)	Context Switch (↓)	Task Relevance (↑)
Support	Recursive	Text	27.56	15.36	23.60	84.38
	Recursive	HTML	25.59	55.75	2.96	45.16
	Recursive	Markdown	26.46	27.34	24.34	82.32
	Embedding	Markdown	35.86	9.41	59.41	57.92
	LLMSemantic	Markdown	24.00	71.21	6.81	76.89
	LumberChunker	Markdown	36.05	1.25	54.64	63.16
	AutoChunker	Markdown	**1.12**	**93.03**	**1.66**	**94.76**
Wikipedia	Recursive	Text	29.83	18.45	25.12	82.54
	Recursive	HTML	26.91	53.62	3.15	47.23
	Recursive	Markdown	28.13	25.67	26.45	80.91
	Embedding	Markdown	37.42	8.92	61.23	55.84
	LLMSemantic	Markdown	25.34	69.87	7.12	75.32
	LumberChunker	Markdown	38.21	2.14	56.78	61.45
	AutoChunker	Markdown	**2.31**	**91.24**	**2.05**	**92.87**

Table 2: Comparison of Different Chunking Techniques Across Domains. ↑ indicates higher is better, ↓ indicates lower is better. Best results are in **bold**.

Domain	Method	WP@1	WP@3	WP@5
Support	Recursive	60.75	51.25	39.15
	Embedding	16.75	14.25	13.65
	LLMSemantic	69.12	56.23	49.41
	AutoChunker	**75.42**	**63.42**	**56.84**
	AutoChunker + CAR	**75.42**	**68.74**	**63.22**
Wikipedia	Recursive	58.45	48.92	37.84
	Embedding	15.92	13.85	12.95
	LLMSemantic	66.78	54.32	47.65
	AutoChunker	**72.95**	**61.45**	**54.92**
	AutoChunker + CAR	**72.95**	**66.84**	**61.35**

Table 3: Comparison of Weighted Precision Scores Across Different Methods and Domains. CAR: Context Aware Retrieval. Best results are in **bold**.

- **Embedding**: We converted HTML content to markdown and utilized Langchain's SemanticChunker (Chase, 2022) with *cohere.embed-multilingual-v3* (Cohere, 2023).

- **LLMSemantic**: We used the code provided by the authors, employing the *claude-3.5-sonnet* (Anthropic, 2024) model as the LLM backbone.

- **LumberChunker**: We implemented this method using the code provided by the authors, also using the *claude-3.5-sonnet* model as the LLM backbone.

We used *claude-3.5-sonnet* for AutoChunker and all LLM-based evaluations, and *cohere.embed-multilingual-v3* as the embedding model for the retriever.

6 Results and Analysis

6.1 Chunking Quality Analysis

Table 2 presents the results comparing different chunking techniques across various metrics. Our approach significantly outperforms all baselines across all metrics. It achieves the lowest noise, highest completeness, minimal context switching, and highest task relevancy. The substantial reduction in noise can be attributed to our elimination mechanism, which addresses a critical gap in existing techniques.

6.2 Retrieval Performance

We evaluated the retrieval performance using weighted precision scores at different ranks. Table 3 shows these results. Our method consistently outperforms baselines in retrieval performance, with the highest WP@1. The addition of information via Context Aware Retrieval (CAR) further improves

WP@3 and WP@5 scores, demonstrating the effectiveness of our approach in maintaining context and relevance.

7 Industry Application and Impact

We implemented our chunking strategy to optimize the organization of support guides and troubleshooting content for an online product support store. The implementation of this strategy enhanced the retrieval performance of the online product store's customer support system. We observed a 7% increase in relevant content retrieval precision compared to the internal baseline that implements static chunking.

To leverage this improved content retrieval, we integrated our chunking strategy to chatbot system that utilizes a Retrieval-Augmented Generation (RAG) (Lewis et al., 2020). This chatbot serves as the primary interface for customers who have purchased products and are experiencing issues. The pipeline efficiently retrieves the most relevant chunked content from the vector database and uses it to generate contextually appropriate responses. The impact of this integration led to a 6.5 bps reduction in product return rates over a 4 week period following the system's deployment as we are able to provide more meaningful responses.

8 Conclusion

We introduce AutoChunker, an approach to text chunking that addresses critical limitations in existing works. Through its bottom-up strategy and structure-awareness, AutoChunker demonstrates improvements in chunk quality across multiple dimensions. Our evaluation framework, based on five core tenets, provides a systematic way to assess chunking effectiveness beyond traditional retrieval metrics. The integration of AutoChunker's processed chunks in an online product support system validates its practical utility, with measurable improvements in customer support and reduced product return rates. This real-world validation demonstrates that empirical improvements in chunking quality translate directly to industry impact.

References

Anthropic. 2024. Claude 3.5 sonnet model card addendum. https://www-cdn.anthropic.com/fed9cc193a14b84131812372d8d5857f8f304c52/Model_Card_Claude_3_Addendum.pdf.

Mostafa Bayomi and Séamus Lawless. 2018. C-HTS: A concept-based hierarchical text segmentation approach. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Harrison Chase. 2022. Langchain.

Cohere. 2023. cohere-embed-multi.

Common Crawl. 2025. Common crawl january 2025 crawl archive (cc-main-2025-05). https://commoncrawl.org.

André V. Duarte, João DS Marques, Miguel Graça, Miguel Freire, Lei Li, and Arlindo L. Oliveira. 2024. LumberChunker: Long-form narrative document segmentation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6473–6486, Miami, Florida, USA. Association for Computational Linguistics.

Jacob Eisenstein. 2009. Hierarchical text segmentation from multi-scale lexical cohesion. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, page 353–361, USA. Association for Computational Linguistics.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey. *Preprint*, arXiv:2312.10997.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2025. A survey on llm-as-a-judge. *Preprint*, arXiv:2411.15594.

Arihant Jain, Purav Aggarwal, Rishav Sahay, Chaosheng Dong, and Anoop Saladi. 2025. AutoEval-ToD: Automated evaluation of task-oriented dialog systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 10133–10148, Albuquerque, New Mexico. Association for Computational Linguistics.

Anna Kazantseva and Stan Szpakowicz. 2014. Hierarchical topical segmentation with affinity propagation. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 37–47, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020.

Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Jiarun Liu. 2024. Html chunking algorithm. https://github.com/KLGR123/html_chunking.

Zuhong Liu, Charles-Elie Simon, and Fabien Caspani. 2025. Passage segmentation of documents for extractive question answering. *Preprint*, arXiv:2501.09940.

Leonard Richardson. 2007. Beautiful soup documentation. *April*.

Brandon Smith and Anton Troynikov. 2024. Evaluating chunking strategies for retrieval. Technical report, Chroma.

A Additonal Results

Table 4 compared various document chunking techniques, including our proposed AutoChunker method, across multiple performance metrics. These metrics include average chunking time, token statistics (mean, 50th percentile, 90th percentile), and mean number of chunks per document. AutoChunker demonstrates competitive performance and is more efficient in terms of processing time compared to other LLM-based approaches. Moreover, it produces chunks with a balanced token distribution, suitable for both retrieval tasks and LLM context window limitations.

Method	Input	Time (in sec)	Mean tokens	p50 tokens	p90 tokens	Mean #chunks/doc
Recursive	Text	0.11	359.5	391	407	534
Recursive	HTML	5.61	14.2	10	34	13924
Recursive	Markdown	0.24	357.4	388	406	534
Embedding	Markdown	0.38	259.6	163	419	772
LLMSemantic	Markdown	49.39	95.3	76	195.1	2570
LumberChunker	Markdown	10.52	1314.5	633	4037	191
AutoChunker	Markdown	6.04	94.3	72	202	2223

Table 4: Comparative analysis of document chunking techniques across different parameters.

B Prompts

B.1 Intelligent Chunking Prompt

Prompt 1: AutoChunker

```
<task>
Your task is to analyze and merge paragraphs from a Markdown web page into coherent semantic
units. Each merged unit should be self-contained and logically complete. While doing so, also
identify and exclude any noise content (like navigation elements, empty paragraphs, redundant
headers, related articles) in the merged units.
</task>

<Input Format>
- Content is provided as numbered paragraphs within tags: <pXXX>content</pXXX>
- XXX represents the unique paragraph ID number
</Input Format>

<Output Requirements>
1. List the paragraph IDs that should be merged together
2. Present the merged IDs in the format: <merged>ID1-ID2,ID3-ID4,...</merged>
3. Just output the start ID and the end ID of the merged paragraphs in the merged tag.
</Output Requirements>

<Merging Guidelines>
1. Combine paragraphs that form complete thoughts or topics
2. Keep related content together (e.g., questions with their answers)
3. Maintain the natural flow of information
4. Preserve hierarchical relationships (headings with their content)
5. Group related FAQs or technical specifications together
6. All the steps present in a sequence should be present together.
7. Create a new paragraph unit only when a new topic is discussed or the context is changed.
8. Retain the product name in the merged units if there is any.
9. If an image is is associated with a logical unit, try to retain it.

Consider these elements as noise (typically exclude):
- Navigation menus
- Empty paragraphs
- Redundant headers
- Social media buttons
- Generic page elements (e.g., "Skip to main content")
- Footer content
```

```
- Duplicate content
- Related articles
- Support, Contact or chat with us related elements
- Callback request options
</Merging Guidelines>

Here is the input:
{input}
```

B.2 Evaluation Prompts

Prompt 2: Noise Scoring

```
You are given various paragraphs provided as numbered paragraphs within tags:
<pXXX>content</pXXX> where XXX represents the unique paragraph ID number. Your task is
to identify for each paragraph whether it contains any noisy content or not.

Consider these elements as noise:
- Navigation menus
- Empty paragraphs
- Redundant headers
- Social media buttons
- Generic page elements (e.g., "Skip to main content")
- Footer content
- Duplicate content
- Related articles
- Support, Contact or chat with us related elements
- Callback request options

Consider these elements as not noisy:
- Titles
- Question and Answers
- FAQs

<Output Requirements>
<p1>[Yes/No based on if it contains noise]</p1>
<p2>[Yes/No based on if it contains noise]</p2>
...
<pN>[Yes/No based on if it contains noise]</pN>
</Output Requirements>

Just output Yes or No within each tag in your response.
Now here is the input to you:
{paragraphs}
```

Prompt 3: Completeness Scoring

```
Analyze the following paragraphs for logical completeness. Each paragraph is enclosed in tags:
<pXXX>content</pXXX> where XXX is a unique paragraph ID.

A paragraph is considered COMPLETE if it:
1. Forms a self-contained logical unit
2. Conveys a complete thought or idea
3. Has proper context within itself
4. Doesn't leave readers with obvious unanswered questions
5. Doesn't end abruptly or start with connecting words referring to missing content

Examples:
- Complete: "What is photosynthesis? It is the process by which plants convert sunlight into
energy."
- Incomplete: "This led to several complications." (lacks context and previous reference)

Please evaluate each paragraph and respond ONLY with Yes/No in the following format: <p1>Yes</p1>
or <p1>No</p1>
```

```
<Output Requirements>
<p1>[Yes/No based on if it is complete]</p1>
<p2>[Yes/No based on if it is complete]</p2>
...
<pN>[Yes/No based on if it is complete]</pN>
</Output Requirements>

Paragraphs to analyze:
{paragraphs}
```

Prompt 4: Context Switch Scoring

```
Analyze each paragraph for internal context switching. Each paragraph is provided within tags:
<pXXX>content</pXXX> where XXX is the unique paragraph ID number.

DEFINITION OF CONTEXT SWITCHING:
A paragraph exhibits context switching if it:
1. Discusses more than 2 distinct topics/subjects
2. Shifts between unrelated ideas without clear transitions
3. Introduces multiple separate questions or problems
4. Changes perspective or narrative focus abruptly

EXAMPLES:
Context Switching (Yes):
- "The cat slept on the windowsill. Global warming is affecting polar bears. Students should
study more for exams."
- "AI technology is advancing rapidly. Speaking of which, my garden needs watering. The stock
market crashed yesterday."

No Context Switching (No):
- "The computer processes data through its CPU and RAM, which work together to execute programs."
- "Climate change affects both temperature and precipitation patterns, leading to various
environmental impacts."

OUTPUT FORMAT:
<p1>[Yes/No]</p1>
<p2>[Yes/No]</p2>
...
<pN>[Yes/No]</pN>

Respond ONLY with Yes/No within the paragraph tags.

PARAGRAPHS TO ANALYZE:
{paragraphs}
```

Prompt 5: Task Scoring (Support Specific)

```
You are a product support analysis system. Analyze the following paragraphs to identify
potential customer questions or troubleshooting scenarios about products.

For each paragraph provided within tags <pXXX>content</pXXX> (where XXX is the unique paragraph
ID), determine if it contains:
- A customer's potential question about a product
- A problem or issue that needs troubleshooting
- A request for help or clarification about product usage

Guidelines for identification:
- "Yes" if the paragraph contains:
* Questions about product features or functionality
* Problems or issues requiring resolution
* Requests for help or clarification
* Troubleshooting scenarios
* Customer concerns or confusion
* Product descriptions
- "No" if the paragraph contains:
* General statements or facts
```

```
* Marketing content
* Non-question related information

<Output Format Required>
<p1>[Yes/No]</p1>
<p2>[Yes/No]</p2>
...
<pN>[Yes/No]</pN>

Provide only Yes or No within each tag. No additional explanation needed.

Analyzing the following paragraphs:
{paragraphs}
```

Prompt 6: Relevance Scoring

```
Task: Analyze paragraphs for relevance to a customer query

Input Format:
- Customer query will be provided
- Multiple paragraphs marked with tags: <pXXX>content</pXXX> (XXX = unique paragraph ID)

Relevance Scoring Scale:
0 - Irrelevant (no connection to query)
1 - Relevant (identifies the issue)
2 - Somewhat Relevant (contains potential solution)
3 - Completely Relevant (contains both issue and solution)
4 - Perfectly Relevant (exact match for issue and solution)

Rules:
1. Each paragraph must be evaluated independently
2. Consider both semantic and contextual relevance
3. Score based on how directly the paragraph addresses the query
4. Multiple paragraphs can receive the same score
5. Assess both explicit and implicit relevance

Required Output Format:
<p1>[score]</p1>
<p2>[score]</p2>
...
<pN>[score]</pN>

Example:
<query>"How do I reset my password?"</query>
<p1>To reset your password, click on 'Forgot Password' and follow the instructions.</p1>
Output: <p1>2</p1>

Note: Scores should be integers between 0-4 only

Now here is the input to you:
<query>{query}</query>
{paragraphs}
```

B.3 Query Generation Prompt

Prompt 7: Query Generation

```
Given a set of text chunks, your task is to:

1. Analyze the content of the chunks carefully
2. Generate 5 diverse questions that:
- Can be directly answered using information from the provided chunks
- Range from simple fact-based to more complex analytical questions
- Are clearly worded and unambiguous
- Are non-repetitive and cover different aspects of the content
```

```
Format:
<Q1>[Question]</Q1>
<Q2>[Question]</Q2>

Text chunks:
{chunks}
```