Adaptive Chunking: Optimizing Chunking-Method Selection for RAG

Paulo Roberto de Moura Júnior, Jean Lelong, Annabelle Blangero
Ekimetrics, France
jean.lelong@ekimetrics.com, annabelle.blangero@ekimetrics.com

Abstract

The effectiveness of Retrieval-Augmented Generation (RAG) is highly dependent on how documents are chunked, that is, segmented into smaller units for indexing and retrieval. Yet, commonly used "one-size-fits-all" approaches often fail to capture the nuanced structure and semantics of diverse texts. Despite its central role, chunking lacks a dedicated evaluation framework, making it difficult to assess and compare strategies independently of downstream performance. We challenge this paradigm by introducing *Adaptive Chunking*, a framework that selects the most suitable chunking strategy for each document based on a set of five novel intrinsic, document-based metrics : *References Completeness* (RC), *Intrachunk Cohesion* (ICC), *Document Contextual Coherence* (DCC), *Block Integrity* (BI), and *Size Compliance* (SC), which directly assess chunking quality accross key dimensions. To support this framework, we also introduce two new chunkers, an *LLM-regex splitter* and a *split-then-merge recursive splitter*, alongside targeted post-processing techniques. On a diverse corpus spanning legal, technical, and social science domains, our metric-guided adaptive method significantly improves downstream RAG performance. Without changing models or prompts, our framework increases RAG outcomes, raising answers correctness to 72% (from 62-64%) and increasing the number of successfully answered questions by over 30% (65 vs. 49). These results demonstrate that adaptive, document-aware chunking, guided by a complementary suite of intrinsic metrics, offers a practical and effective path to more robust RAG systems. Code available at https://github.com/ekimetrics/adaptive-chunking.

Keywords: Retrieval-Augmented Generation, Document Chunking, Intrinsic Evaluation Metrics, Adaptive Systems, Natural Language Processing, Hybrid Search

1. Introduction and Related Work

Retrieval-Augmented Generation (RAG) has become a key paradigm for enhancing Large Language Models (LLMs) by grounding them in external knowledge sources (Zhao et al., 2025; Merola and Singh, 2025; Wang et al., 2025). In essence, RAG combines two steps: retrieval, which fetches relevant information from a knowledge base, and generation, where an LLM uses this retrieved context to produce more accurate and informed responses. The effectiveness of RAG strongly depends on the retrieval pipeline's ability to fetch relevant context from a knowledge base for each user query (Wang et al., 2025). Despite the growing size of LLM context windows, document segmentation remains critical for efficient retrieval (Smith and Troynikov, 2024; Developer, 2025). Current chunking approaches often impose arbitrary boundaries that break contextual links, a problem referred to as the context-preservation dilemma (Qu et al., 2025; Duarte et al., 2024; Jain et al., 2025). This fragmentation scatters essential information across disconnected chunks, degrading retrieval quality (Duarte et al., 2024; Anthropic, 2024; Merola and Singh, 2025).

Some existing chuncking methods attempt to maintain a degree of semantic coherence, but often at the expense of other essential properties. *Sentence-based splitters* (Qi et al., 2020) divide text into a fixed numbers of semantically cohesive sentences, but often break logical blocks such as paragraphs. Recursive splitters, such as LangChain's `RecursiveCharacterTextSplitter` (LangChain, 2025a) apply a hierarchy of separators to enforce length constraints, but they can group unrelated topics together, reducing cohesion. *Semantic chunkers* (LangChain, 2025b; Ni et al., 2025; LlamaIndex, 2024) use embeddings to split text at semantic boundaries, improving cohesion but introducing high computational cost. Finally LLM-driven methods (Duarte et al., 2024; Jain et al., 2025), leverage language models to infer natural boundaries, which improves completeness but still assumes a single uniform strategy for an entire corpus and incurs significant overhead.

Each of these methods comes with inherent drawbacks, a trade-off between properties, and they all share a common limitation: relying on one fixed chunking strategy for all documents, regardless of structural or contextual differences.

Recent studies increasingly recognize that a "one-size-fits-all" chunking strategy is inadequate (Zhao et al., 2025) and that the optimal approach is inherently dependent on the document's structure and content (Developer, 2025). A critical gap in evaluation exacerbates this reliance on a monolithic strategy, as most studies measure chunking quality extrinsically via downstream retrieval met-

Proceedings of the Fifteenth Language Resources and Evaluation Conference (LREC 2026), pages 11535–11551
11–16 May 2026.

rics such as Hits@k, Recall@K, and Normalized Discounted Cumulative Gain (nDCG@K) (Günther et al., 2025; Anthropic, 2024; Merola and Singh, 2025), making it difficult to isolate the impact of the chunking strategy itself (Smith and Troynikov, 2024). This highlights a pressing need for robust, intrinsic metrics capable of directly assessing chunk quality.

To optimize retrieval, we consider the hypothesis that ideal chunks should simultaneously satisfy four key properties. They should be (1) **self-contained and logically complete**, expressing a full thought that can be understood in isolation; (2) **respectful of length constraints**, balancing the need for sufficient context with the token limits of embedding models; (3) **semantically cohesive**, focusing on a single topic to avoid informational noise; and (4) **context-preserving**, aligning with the document's natural structure, such as paragraphs or sections. Each property addresses a failure mode identified in prior work: breaking logical units scatters essential information across chunks (Duarte et al., 2024; Anthropic, 2024), oversized or undersized chunks degrade embedding quality and waste retrieval slots (Günther et al., 2025; Smith and Troynikov, 2024), and mixing unrelated topics dilutes semantic signals (Qu et al., 2025).

In this work, we propose **Adaptive Chunking**, a framework that moves beyond the single-strategy paradigm by selecting the most suitable chunking method for each document. Our contribution is threefold:

1. A suite of five intrinsic metrics: References Completeness, Intrachunk Cohesion, Document Contextual Coherence, Block Integrity, and Size Compliance—that directly assess chunk quality;
2. New chunking techniques, including an LLM-guided regex splitter and a split-then-merge recursive splitter, complemented by targeted post-processing to enforce size constraints;
3. An evaluation pipeline that measures both retriever quality and downstream RAG performance using a novel contextual completeness metric.

Together, these components enable document-aware chunking that improves retrieval and question answering without modifying models or prompts.

2. Method

2.1. Document Collection and Pre-processing

Our experiments use a real-world corpus of 33 PDF documents from the Ekimetrics CLAIR project, spanning the legal, technical, and social science domains. This collection is intentionally diverse in its formatting, vocabulary and length, making it an robust test case for evaluating optimized chunking strategies.

	Tech	Legal	Social
Docs (count)	9	16	8
Tokens / doc (mean)	5 257	30 895	79 862
Tokens / doc (max)	18 605	118 424	391 660
Tokens / doc (min)	1 475	523	8 013
Pages / doc (mean)	12	46	114
Total corpus tokens	47 313	494 320	638 896

Table 1: Corpus statistics per domain. Tokens were counted using OpenAI `o200k_base` tokenizer (Byte-pair encoding).

The documents were parsed into markdown text using Microsoft Azure AI Document Intelligence (ADI) (Microsoft, 2025), which provides high-quality structural outputs. We then implemented a custom Markdown generation pipeline based on ADI's section tree, extending its built-in capabilities with three enhancements:

- Improved table handling by splitting oversized tables (>1000 tokens) into sub-tables, while preserving headers and reducing unnecessary token overhead.
- grouping page headers and footers into single blocks to avoid redundant markers;
- preserving inseparable spans such as tables, figures, titles with body text, sequences of short items (<100 tokens), and footnotes with their preceding text.

After parsing and Markdown formatting, the total corpus comprises approximately 1.18 million tokens.

The overall statistics of the collection are presented in Table 1.

2.2. Chunking Methods

We introduce two new chunking methods designed to balance structural awareness with efficiency:

1. **LLM Regex splitter**: this method combines the structural analysis capabilities of a language model with the determinism of regex-based splitting. We prompt the LLM with chunking guidelines, an output schema, and an example, then provide a sample document segment (first 8,000 tokens in our experiments). The LLM generates a regex pattern that is applied to the full document using Python's regex library. This approach is particularly effective for structured texts such

as legal documents, where respecting natural boundaries (e.g., article delimiters) is critical—even when articles vary in length or span multiple pages.

2. **Split-then-Merge Recursive Splitter**: Inspired by LangChain's recursive splitter (LangChain, 2025a), our variant introduces a two-pass strategy to reduce tiny, context-poor chunks (Günther et al., 2025). In the first pass, the text is recursively split using a prioritized separator list (titles → sections → sentences → characters) until each segment is $\leq S$, where S is the chunk size. In the second pass, adjacent segments are greedily merged without exceeding S, backtracking when necessary to maintain overlap and re-splitting oversized parts. This design improves size compliance and preserves context compared to single-step recursive methods. In our experiments, we include two variants of our recursive splitter (target sizes: 1,100 and 600 tokens).

Both methods integrate seamlessly with our Markdown pipeline, leveraging structural cues for better alignment with document hierarchy.

As final regularization steps, we propose two post-processing passes.

1. **Oversized-chunk splitting**: Chunks exceeding the maximum size (1,100 tokens in our experiments) are re-split using the same separator cascade, leaving well-sized chunks untouched. This prevents embedding vectors from representing too many distinct ideas, which can dilute semantic signals (Günther et al., 2025) and obscure key information during retrieval (Wang et al., 2025).

2. **Tiny-chunk merging**: Chunks smaller than the minimum size (100 tokens) are merged with adjacent segments, provided the combined size remains below the upper limit (1,150 tokens). This reduces context-poor fragments that waste retrieval slots and degrade overall context quality. These steps ensure size compliance without compromising document structure.

For comparison, we also evaluate baseline strategies:

- Page-based chunking, i.e. chunking the documents by pages (with and without post-processing).
- LangChain's recursive splitter (with default parameters and with the same parameters as our custom recursive splitter).
- LangChain's experimental semantic splitter (LangChain, 2025b) with `gradient` thresholding.
- Sentence-based chunking, i.e. chunking by a fixed number of sentences (in our experiments, 5 sentences), using Stanza NLP model (Qi et al., 2020).

Baseline methods are evaluated without post-processing, to preserve how they were designed and how they are typically used in practice.

2.3. Chunking Metrics

We introduce five intrinsic metrics to evaluate chunk quality at the document level. All metrics are computed at the document level, enabling both per-document tuning and corpus-level optimization.

2.3.1. References Completeness (RC)

Measures the fraction of entity–pronoun pairs that remain intact within a chunk. Breaking these pairs across boundaries can lead to incomplete retrieval when queries reference pronouns without their antecedents. The metric computation requires, first, the extraction of mention clusters using Maverick coreference resolution model (Martinelli et al., 2024), and then we extract the entity-pronoun pairs from the clusters, mapping pronouns to entities. Let $P = \{(e_i, p_i)\}_{i=1}^{N}$ denote the set of entity-pronoun pairs, with $s_i = \text{start}(e_i)$ and $t_i = \text{end}(p_i)$ their respective span boundaries. Let B be the set of interior chunk boundaries, i.e., all chunk starts except the very first and all chunk ends except the very last. **The Reference Completeness (RC)** score is defined as

$$m_i = \mathbf{1}\left[\exists\, b \in B :\ s_i < b \leq t_i\right],$$

$$\text{RC} = 1 - \frac{1}{N}\sum_{i=1}^{N} m_i,$$

where m_i is an indicator that equals 1 if any boundary lies strictly between the entity and its pronoun, in which case the reference is considered missing. Note that **RC** can only be computed for English-language documents, as the Maverick coreference resolution model is limited to English text.

2.3.2. Block Integrity (BI)

Evaluates whether structural units, such as paragraphs, tables, figures, and title-body pairs, remain unbroken. Splitting these blocks reduces interpretability (e.g., a table split across chunks becomes unusable). **BI** uses parser-provided block spans and computes the proportion of intact blocks, applying a tolerance margin of 5 characters to avoid false positives. Let the list of gold block boundaries

Chunking method	mean	max	min	std. dev.	# chunks	time [s]
* LLM regex (GPT-5)	518	1146	69	332	2279	**146.85
* our recursive ($s = 1100$)	878	1141	104	217	1345	31.42
* our recursive ($s = 600$)	496	691	102	101	2381	28.3
* page (post-processed)	663	1146	72	235	1780	4.09
† LC recursive ($s = 1100$)	706	1101	2	324	1675	5.63
† LC recursive (default)	773	1364	57	129	1557	0.04
† page (raw)	669	1765	0	277	1765	0.01
† semantic	693	17146	1	1095	1706	389.25
† sentence	146	1025	4	68	8125	30.43
Adaptive Chunking	724	1146	86	247	1631	210.66

Table 2: Chunk sizes in tokens (OpenAI's `o200k_base`) and runtime statistics. Chunking methods marked with * are included in the Adaptive Chunking results. Those marked with † were not post-processed; ** was computed considering asynchronous API calls.

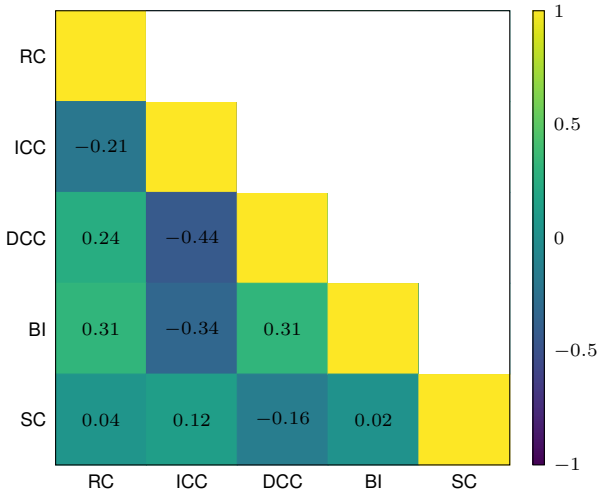

Figure 1: Spearman correlation (ρ) between chunking metrics, computed using the metric results for each document and chunking method.

be $G = \{0, d_1, \ldots, d_M, L\}$ with L the document length, and B be the set of interior chunk boundaries. A block $[d_j, d_{j+1}]$ is considered broken if there exists a predicted chunk boundary $b \in B$ such that

$$d_j + \tau < b < d_{j+1} - \tau,$$

where τ is a tolerance margin in characters (in our experiments, $\tau = 5$). Equivalently, the block is *intact* if no such b lies strictly inside it. The Block Integrity score is the fraction of gold blocks that remain intact,

$$I_j = \mathbf{1}[\nexists b \in B : \; d_j + \tau < b < d_{j+1} - \tau]\,,$$

$$\mathrm{BI} = \frac{1}{|G| - 1} \sum_{j=0}^{|G|-1} I_j\,.$$

2.3.3. Intrachunk Cohesion (ICC)

Quantifies semantic consistency within a chunk by comparing sentence embeddings to the chunk embedding. For each chunk, ICC is the mean cosine similarity between its sentences and the full chunk embedding, computed using Jina AI's jina-embeddings-v3 model (Sturua et al., 2024). In more details, let the document be split into chunks $C = \{c_1, \ldots, c_K\}$, and let each chunk c_k contain a sequence of text blocks $S_k = \{s_{k1}, \ldots, s_{kn_k}\}$ obtained using the starting indexes of text spans from the parsing outputs. Let $\mathbf{v}(c_k) \in \mathbb{R}^d$ denote the normalized embedding of chunk c_k, and $\mathbf{v}(s_{kj})$ the normalized embedding of sentence s_{kj}. The semantic cohesion of a single chunk is defined as the mean cosine similarity between its sentences and its full embedding,

$$\mathrm{Cohesion}(c_k) \;=\; \frac{1}{n_k} \sum_{j=1}^{n_k} \mathbf{v}(s_{kj})^\top \mathbf{v}(c_k), \qquad n_k \geq 2,$$

where chunks with fewer than two blocks are ignored, since cohesion cannot be meaningfully computed. The overall Intrachunk Cohesion (ICC) score is the mean across all valid chunks,

$$\mathrm{ICC} \;=\; \max\{0, \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \mathrm{Cohesion}(c_k)\},$$

where $\mathcal{K} \subseteq \{1, \ldots, K\}$ indexes the chunks with $n_k \geq 2$ and any negative score is clipped to zero.

2.3.4. Document Contextual Coherence (DCC)

Assesses how well chunks align with their broader local context (Günther et al., 2025). We construct sliding windows of up to 3,000 tokens and compute the mean cosine similarity between each chunk and its window embedding (computed using Jina AI's jina-embeddings-v3 model (Sturua et al., 2024)).

This ensures chunks are understandable in isolation while preserving document-level context. As previously, let the document be segmented into ordered chunks $C = \{c_1, \ldots, c_K\}$ with character spans $[a_k, b_k)$. Let $\mathrm{tok}(\cdot)$ denote a token-count function and let T be a token budget (in our experiments, $T = 3000$ tokens). We build a sequence of context windows $\{W_m\}$ as follows: starting at chunk index i, we concatenate the non-overlapping tails of consecutive chunks $c_i, c_{i+1}, \ldots$ (i.e., only the text not already included in the window) until the cumulative token count of the added tails would exceed T, always requiring each window to contain at least two chunks. The window starting index then advances by a stride of `window_step` chunks, and the process repeats. This yields windows W_m and their associated chunk index sets $\mathcal{C}_m \subset \{1, \ldots, K\}$. Let $\mathbf{v}(c_k) \in \mathbb{R}^d$ be the *normalized* embedding of chunk c_k, and let $\mathbf{w}(W_m) \in \mathbb{R}^d$ be the *normalized* embedding of window W_m. For each window, define the window coherence via mean cosine similarity,

$$\mathrm{Coherence}(W_m) \;=\; \frac{1}{|\mathcal{C}_m|} \sum_{k \in \mathcal{C}_m} \mathbf{w}(W_m)^\top \mathbf{v}(c_k),$$

and finally aggregate all window coherence scores to build the Document Contextual Coherence as

$$\mathrm{DCC} = \max\{0, \frac{1}{|M|} \sum_{m=0}^{M} \mathrm{Coherence}(W_m)\},$$

where M is the total number of windows.

2.3.5. Size Compliance (SC)

Measures the proportion of chunks whose token length falls within predefined bounds (100–1,100 tokens in our experiments). Oversized chunks dilute embeddings, while tiny chunks waste retrieval slots and degrade context quality. Let $C = \{c_1, \ldots, c_K\}$ be the set of chunks, $\mathrm{tok}(\cdot)$ a token-count function, and m and M the minimum and maximum allowed tokens (in our experiments $m = 100$, $M = 1100$). For each chunk, let $\tau_k = \mathrm{tok}(c_k)$, then the score is the mean compliance

$$\mathrm{SC} \;=\; \frac{1}{K} \sum_{k=1}^{K} \mathbf{1}\big[\, m \le \tau_k \le M \,\big].$$

2.4. Retrieval and Experiments Setup

2.4.1. Pipeline Steps

To evaluate the impact of chunking strategies on RAG performance, we implement a hybrid-search pipeline combining keyword-based and semantic retrieval, reranking, and generation (Figure 2). Here are the different steps:

1. **Query Embedding**: Encode user queries using Qwen3-Embedding-4B, selected for strong performance on retrieval tasks.
2. **Keyword Search**: Apply BM25 to retrieve the top-50 candidates based on lexical similarity.
3. **Semantic Search**: Perform dense retrieval using cosine similarity between query embeddings and chunk embeddings, selecting the top-50 candidates.
4. **Candidate Merging**: Combine BM25 and semantic results (100 candidates), deduplicate, and pass to reranking.
5. **Reranking**: Use snowflake-arcticembed-l-v2.0, a state-of-the-art reranker optimized for hybrid retrieval, to select the top-10 most relevant chunks.
6. **Answer Generation**: Provide these top-10 chunks as context to GPT-4.1 with temperature=0 and top-p=1 to minimize variability. The model is instructed to answer "I don't know based on the provided context" if the retrieved chunks are insufficient.

All models, hyperparameters, and prompts remain fixed; only the chunking method varies to isolate its effect.

2.4.2. Evaluation Setup

To rigorously assess the impact of chunking strategies on RAG performance, we designed an evaluation protocol that compares three chunking strategies using two complementary metrics: one measuring retrieval quality and the other assessing the correctness of generated answers.

Chunking Strategies Compared
We benchmark three approaches:

1. **Adaptive Chunking** which dynamically selects the best chunking method for each document based on the average of five intrinsic metrics.
2. **LangChain Recursive Splitter** (LangChain, 2025a) with default parameters and no post-processing, representing a widely used baseline in retrieval systems.
3. **Page-based Chunking** without post-processing, serving as the simplest and most common baseline in production pipelines.

Question–Answer Generation For each document in our corpus, we generate three question–answer pairs using GPT-4.1 with a 10,000-token context window. This ensures that

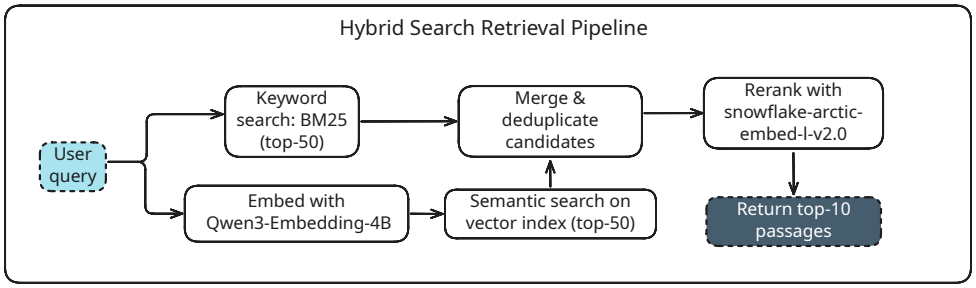

Figure 2: Hybrid search retrieval pipeline.

questions are grounded in the document content and that answers reflect realistic retrieval scenarios.

Evaluation Metrics

- **Retrieval Completeness:** Based on DeepEval (Confident AI, 2025b), we designed this metric to measures how well the retrieved context supports the ground truth answer. GPT-4.1 acts as a judge, labeling each context as incomplete (0), partially complete (1), or complete (2). Scores are normalized to [0,1]. In our qualitative experiments, we observed that DeepEval's Contextual Recall produced inconsistent results for cases where the LLM had skipped answering queries due to insufficient context, whereas Retrieval Completeness yielded more consistent results.
- **Answer Correctness:** Using G-Eval implemented in DeepEval (Liu et al., 2023; Confident AI, 2025a), we compare the generated answer to the ground truth for factual alignment on a $0\tilde{}1$ scale. This metric prioritizes correctness over stylistic differences and lightly penalizes omissions.

By combining these two metrics, we capture both retrieval quality and generation accuracy. This setup ensures that any observed performance differences can be attributed to chunking strategies rather than confounding factors such as model choice or prompt design.

3. Results

3.1. Chunking Quality Analysis

Overview. Table 3 reports our five intrinsic metrics: *References Completeness* (RC), *Intrachunk Cohesion* (ICC), *Document Contextual Coherence* (DCC), *Block Integrity* (BI), and *Size Compliance* (SC), together with their mean across documents. The split-then-merge recursive splitter with a target size of $s{=}1100$ tokens obtains the highest overall mean (90.68%), followed closely by post-processed page chunking (90.52%) and the LLM-guided regex method (89.80%). The Adaptive policy, which picks the best method per document, yields the highest corpus-level mean (91.07%), indicating that no single method dominates across all documents.

Per-metric bests. Method-level winners are consistent with the intended design trade-offs:

- *RC:* our recursive $(s{=}1100)$ reaches the top score (99.0% $\pm$ 1.8), suggesting fewer entity–pronoun splits.
- *ICC:* sentence-based chunking is best (78.4% $\pm$ 2.7), as smaller units reduce topical mixing.
- *DCC:* our recursive $(s{=}1100)$ achieves the highest coherence with local context (89.7% $\pm$ 2.5).
- *BI:* raw page chunking is perfect (100.0% $\pm$ 0.0) because structural blocks rarely cross page boundaries.
- *SC:* both recursive variants hit 100.0% SC, confirming the effectiveness of split-then-merge size control.

The corresponding chunk sizes and runtime statistics are shown in Table 2.

3.2. Effect of Post-processing on Size Regularization

Post-processing has a clear and consistent effect. Methods with the tiny-chunk merge and oversized-chunk re-split maintain substantially higher *minimum* chunk sizes (69–104 tokens) than their raw counterparts, which can produce near-empty fragments (0–4 tokens) (Table 2). These small fragments carry little semantic content yet may intrude into the top-k candidate set, displacing informative chunks. Empirically, post-processing improves SC and the mean intrinsic score by **+6 to +16** percentage points across methods, confirming the importance of enforcing size bounds without altering well-formed segments.

Chunking method	RC	ICC	DCC	BI	SC	mean
* LLM regex (GPT-5)	98.0 ± 2.9	70.9 ± 5.1	82.4 ± 5.5	98.1 ± 2.5	99.6 ± 1.3	89.80
* our recursive ($s = 1100$)	$\mathbf{99.0 \pm 1.8}$	66.6 ± 3.8	$\mathbf{89.7 \pm 2.5}$	98.1 ± 1.9	$\mathbf{100.0 \pm 0.1}$	**90.68**
* our recursive ($s = 600$)	97.2 ± 2.9	69.6 ± 4.1	84.7 ± 3.1	94.8 ± 3.6	$\mathbf{100.0 \pm 0.0}$	89.24
* page (post-processed)	97.2 ± 3.5	69.2 ± 3.6	86.4 ± 4.6	99.9 ± 0.3	99.9 ± 0.4	90.52
† LC recursive ($s = 1100$)	98.4 ± 3.1	64.7 ± 3.4	86.8 ± 2.8	98.6 ± 1.4	93.3 ± 7.2	88.37
† LC recursive (default)	96.1 ± 4.3	65.6 ± 3.6	88.8 ± 2.4	95.0 ± 2.8	97.7 ± 6.3	88.62
† page (raw)	97.1 ± 3.5	69.3 ± 3.4	86.1 ± 5.0	$\mathbf{100.0 \pm 0.0}$	92.7 ± 9.7	89.03
† semantic	97.5 ± 3.1	69.3 ± 4.1	76.3 ± 7.3	91.3 ± 4.0	48.1 ± 17.6	76.49
† sentence	86.3 ± 11.1	$\mathbf{78.4 \pm 2.7}$	72.5 ± 5.8	61.9 ± 10.3	67.2 ± 23.1	73.26
Adaptive Chunking	99.0 ± 1.5	68.2 ± 4.7	88.8 ± 3.5	99.4 ± 1.2	99.9 ± 0.3	**91.07**

Table 3: Chunking performance results (% mean ± % st. dev.) for *References Completeness* (RC), *Intrachunk Cohesion* (ICC), *Document Contextual Coherence* (DCC), *Block Integrity* (BI), and *Size Compliance* (SC). "LC" denotes LangChain, and s is the chunk size parameter in tokens. Rows marked with * are included in the Adaptive Chunking results; those with † were not post-processed and are here for reference and comparison. Differences between Adaptive Chunking and all individual methods are statistically significant (Wilcoxon signed-rank test, $p < 0.001$).

3.3. Metric Complementarity and Trade-offs

Spearman correlations in Figure 1 are weak to moderate ($-0.44 < \rho < 0.31$), implying that the five metrics capture complementary phenomena rather than a single latent factor. Two notable tensions emerge:

1. **ICC vs. DCC.** Smaller chunks are more internally cohesive (higher ICC) but lose surrounding signal (lower DCC). Conversely, larger chunks preserve local context windows (higher DCC) but risk topical mixing (lower ICC).

2. **ICC vs. BI.** Keeping structural units intact (high BI) may place heterogeneous elements (e.g., title + table + prose) in the same segment, which depresses ICC.

These trade-offs justify a multi-metric objective and motivate a per-document selection policy rather than a corpus-wide "best" chunker.

3.4. Adaptive Selection Behavior

Although the recursive splitter with $s{=}1100$ achieves the highest average intrinsic score, Table 4 shows that the Adaptive policy chooses *page (post-processed)* for **48%** of documents and *recursive* $s{=}1100$ for **42%**; the LLM-regex and recursive $s{=}600$ variants account for the remainder (6% and 3%). This dispersion reflects corpus heterogeneity: documents with strong pagination semantics (e.g., tables, footnotes, or article headers aligned to pages) benefit from page-level integrity, while long narrative or hierarchical content benefits from recursive segmentation tuned to size and overlap. The selection data substantiate the central claim that *no single method is universally optimal*.

Selected method	% selection
page (post-processed)	48
our recursive ($s = 1100$)	42
LLM regex (GPT-5)	6
our recursive ($s = 600$)	3

Table 4: Results for chunking method selection in the Adaptive Chunking method with respect to the entire document corpus. Other chunking methods were not selected and omitted here.

3.5. Impact on Retrieval and Generation

Retrieval quality. Table 5 shows that Adaptive Chunking improves Retrieval Completeness to **67.68%**, outpacing LangChain recursive default (58.08%, **+9.60 pp**) and raw page chunking (59.09%, **+8.59 pp**). In relative terms, these are ~16.5–18.0% gains. Since the retrieval configuration and models are held constant, these deltas are attributable to chunking alone.

Answer correctness. Adaptive Chunking also yields higher G-Eval correctness (**78.01%**) than both baselines (70.11%, **+7.90 pp** vs. LC recursive; 73.33%, **+4.68 pp** vs. page). This shows that better retrieval context translates into more accurate answers under the same generator (GPT-4.1, $T{=}0$, $p{=}1$).

Answer rate. The system with Adaptive Chunking answers **65/99** queries versus **49/99** for each baseline (**+16** questions, **+32.7%**). The increase is consistent with higher Retrieval Completeness: when the retrieved context is more often complete, the model is less likely to abstain ("I don't know based on the provided context.").

Amplification effect. The intrinsic metric differences between the top chunking methods in Table 3 are modest (0.4–2.4 percentage points), yet they

translate into substantially larger RAG performance gaps (8–10 pp in Retrieval Completeness, 5–8 pp in Answer Correctness). This suggests that intrinsic chunk quality improvements compound through the retrieval and generation pipeline: slightly better chunks lead to more relevant retrieved passages, which in turn yield more complete and accurate answers.

3.6. Runtime and Scalability

Chunking cost. Table 2 indicates the LLM-regex approach has the highest average runtime (146.85s), driven by LLM calls (asynchronous). Our recursive splitters are efficient (28–31s) and scale well. The Adaptive pipeline's total (~210.66s) reflects running multiple candidate chunkers and computing metrics per document, which is amortized at indexing time.

Evaluation cost. Table 6 shows Document Contextual Coherence (15:58) and entity–pronoun extraction (13:13) dominate evaluation time, whereas computing chunk embeddings (3:04) and token metrics (0:17) are comparatively inexpensive. These bottlenecks suggest optimizations: (i) caching token-level embeddings once per document and composing window/chunk vectors; (ii) batched coreference clustering.

3.7. Practical Implications

The results support three practical recommendations for building RAG over heterogeneous corpora:

1. **Prefer document-aware selection over a global default.** Even when a single chunker has the best average score, per-document selection delivers higher retrieval completeness and answer rates.

2. **Always regularize sizes.** Tiny-chunk merging and oversized re-splitting provide consistent gains at negligible additional cost, improving both SC and the aggregate mean.

3. **Balance cohesion and context.** Use multi-metric scoring (ICC+BI+DCC+RC+SC) rather than optimizing one dimension in isolation; this avoids overfitting to cohesion at the expense of context (or vice versa).

4. Conclusion

This work introduced *Adaptive Chunking*, a framework for document-specific chunking in Retrieval-Augmented Generation (RAG). Our approach combines a robust parser-to-Markdown pipeline, two novel chunkers (LLM-guided regex and split-then-merge recursive splitter), and targeted post-processing to enforce size constraints. We proposed five intrinsic metrics: *References Completeness* (RC), *Intrachunk Cohesion* (ICC), *Document Contextual Coherence* (DCC), *Block Integrity* (BI), and *Size Compliance* (SC), to enable document-level evaluation and guide chunker selection. Experiments on a heterogeneous corpus show that adaptive, metric-driven selection consistently improves retrieval completeness and answer correctness without modifying models or prompts. These findings demonstrate that principled, document-aware chunking offers a practical path to more robust RAG systems.

5. Ethics Statement

All experiments were conducted on publicly available documents from legal, technical, and social science domains. No personal or sensitive data were used. The proposed methods do not involve human subjects, demographic profiling, or any form of discriminatory processing. All evaluations were performed using automated systems under controlled conditions, ensuring compliance with ethical standards for reproducibility and transparency.

6. Limitations

Computational Efficiency Our evaluation pipeline introduces computational overhead (Table 6), primarily from Document Contextual Coherence and entity-pronoun extraction. DCC currently recomputes token embeddings for each sliding window; computing all token embeddings once at the document level and mapping them to windows would significantly reduce this cost. Entity-pronoun extraction is constrained by Maverick's lack of batch clustering support; a custom batched implementation could accelerate this step substantially.

Language and Domain Coverage The Maverick coreference model limits References Completeness to English text only, constraining multilingual applicability. Our corpus focuses on formal documents (legal, technical, social science); generalization to informal text, creative writing, or heavily multimodal content requires further validation. While we report detailed results on one corpus for reproducibility, we validated our approach on additional datasets with consistent performance gains.

Hyperparameters and Flexibility Hyperparameter choices (chunk size bounds of 100-1,100 tokens, sliding window size of 3,000 tokens, equal metric weighting) remain heuristic and user-dependent.

Metric	Adaptive Chunking	LC recursive (default)	page (raw)
Retrieval Completeness	**67.68**	58.08	59.09
Answer Correctness	**78.01**	70.11	73.33
Mean	**71.77**	62.07	63.80
Answered queries	**65**	49	49
Total queries	99	99	99

Table 5: RAG performance results (% mean) evaluated using OpenAI's GPT-4.1 (*temperature* $= 0$) as the LLM judge. Retrieval Completeness is evaluated for all queries, while Answer Correctness skips queries where the model decided not to provide an answer (insufficient context). Highest scores per column are marked in bold; "LC" means LangChain. Retrieval Completeness differences are statistically significant (Wilcoxon signed-rank, $p < 0.05$).

Step	Time (mm:ss)
Document Contextual Coherence	15:58
Intrachunk Cohesion	03:29
Size Compliance	00:07
References Completeness	00:02
Block Integrity	00:01
Entity-pronoun pairs extraction	13:13
Chunk embeddings	03:04
Chunk token metrics	00:17
Total	**36:11**

Table 6: Runtime per evaluation component. All embeddings were computed using `jina-embeddings-v3` sentence transformer, and the entity-pronoun pairs were extracted using Sapienza NLP `maverick-mes-ontonotes`.

Our metric calculations assume contiguous chunks matching the parsed Markdown text exactly, enabling precise character offset mapping but limiting evaluation of methods that assemble non-contiguous chunks or modify content. Block Integrity requires parser-generated block span annotations; users starting with pre-parsed text in a non-markdown format must provide equivalent annotations.

Practical Deployment Chunking strategies are selected once at indexing time and do not adapt to query characteristics or task requirements. Our evaluation uses a fixed hybrid search pipeline ; while this isolates the impact of chunking, performance with alternative embedding models, retrieval methods, or reranking strategies remains unexplored. Running multiple candidate chunkers per document may challenge high-throughput production pipelines, though this cost is amortized at indexing and could be reduced through pre-screening mechanisms.

7. Bibliographical References

Anthropic. 2024. Introducing contextual retrieval. Blog post, accessed 30 Aug. 2025.

Confident AI. 2025a. Answer correctness metric | deepeval – the open-source LLM evaluation framework. Documentation page, last updated 28 Aug. 2025; accessed 31 Aug. 2025.

Confident AI. 2025b. Deepeval: The LLM evaluation framework. GitHub repository, accessed 31 Aug. 2025.

IBM Developer. 2025. Enhancing RAG performance with smart chunking strategies. Article, accessed 31 Aug. 2025.

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

Michael Günther, Isabelle Mohr, Daniel James Williams, Bo Wang, and Han Xiao. 2025. Late chunking: Contextual chunk embeddings using long-context embedding models.

Hugging Face. 2025. Mteb: Massive text embedding benchmark — leaderboard. Leaderboard website, accessed 31 Aug. 2025.

Arihant Jain, Purav Aggarwal, and Anoop Saladi. 2025. AutoChunker: Structured text chunking and its evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 983–995, Vienna, Austria. Association for Computational Linguistics.

LangChain. 2025a. Recursivecharactertextsplitter. API documentation page, accessed 31 Aug. 2025.

LangChain. 2025b. Semanticchunker (experimental). API documentation page, accessed 31 Aug. 2025.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

LlamaIndex. 2024. Semantic chunker. https://developers.llamaindex.ai/python/examples/node_parsers/semantic_chunking/. Accessed: 2025-10-16.

Giuliano Martinelli, Edoardo Barba, and Roberto Navigli. 2024. Maverick: Efficient and accurate coreference resolution defying recent trends. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13380–13394, Bangkok, Thailand. Association for Computational Linguistics.

Carlo Merola and Jaspinder Singh. 2025. Reconstructing context: Evaluating advanced chunking strategies for retrieval-augmented generation.

Microsoft. 2025. Azure AI document intelligence. Product page, accessed 31 Aug. 2025.

Tongke Ni, Yang Fan, Junru Zhou, Xiangping Wu, and Qingcai Chen. 2025. Crossformer: Cross-segment semantic fusion for document segmentation. *arXiv preprint arXiv:2503.23671*.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 101–108, Online. Association for Computational Linguistics.

Renyi Qu, Ruixuan Tu, and Forrest Bao. 2025. Is semantic chunking worth the computational cost? In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2155–2177.

Brandon Smith and Anton Troynikov. 2024. Evaluating chunking strategies for retrieval. Technical report, Chroma Research. Accessed 31 Aug. 2025.

Saba Sturua, Isabelle Mohr, Mohammad Kalim Akram, Michael Günther, Bo Wang, Markus Krimmel, Feng Wang, Georgios Mastrapas, Andreas Koukounas, Nan Wang, and Han Xiao. 2024. jina-embeddings-v3: Multilingual embeddings with task lora.

Zhitong Wang, Cheng Gao, Chaojun Xiao, Yufei Huang, Shuzheng Si, Kangyang Luo, Yuzhuo Bai, Wenhao Li, Tangjian Duan, Chuancheng Lv, Guoshan Lu, Gang Chen, Fanchao Qi, and Maosong Sun. 2025. Document segmentation matters for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8063–8075, Vienna, Austria. Association for Computational Linguistics.

Jihao Zhao, Zhiyuan Ji, Zhaoxin Fan, Hanyu Wang, Simin Niu, Bo Tang, Feiyu Xiong, and Zhiyu Li. 2025. Moc: Mixtures of text chunking learners for retrieval-augmented generation system.

Appendix A. Example of Generated Markdown Output

```
# The Hamburg Commissioner for Data protection and freedom of information

## Discussion Paper: Large Language Models and Personal Data

This discussion paper reflects the current state of knowledge and understanding
at the Hamburg Com- missioner for Data Protection and Freedom of Information (
HmbBfDI) regarding the applicability of the General Data Protection Regulation (
GDPR) to Large Language Models1 (LLMs). This paper aims to stim- ulate further
debate. It is intended to support companies and public authorities in better
navigating com- plex data protection issues surrounding this subject matter. To
this end, this paper explains relevant tech- nical aspects of LLMs, assesses
them in light of case law regarding personal data from the Court of Jus- tice of
 the European Union (CJEU) and highlights the resulting practical implications.
From this, three principle theses can be derived:

1. The mere storage of an LLM does not constitute processing within the meaning
of article 4 (2) GDPR. This is because no personal data is stored in LLMs.
Insofar as personal data is processed in an LLM-supported AI system, the
processing must comply with the require- ments of the GDPR. This applies in
particular to the output of such an AI system.

2. Given that no personal data is stored in LLMs, data subject rights as defined
 in the GDPR cannot relate to the model itself. However, claims for access,
erasure or rectification can certainly relate to the input and output of an AI
system of the responsible provider or de- ployer.

3. The training of LLMs using personal data must comply with data protection
regulations. Throughout this process, data subject rights must also be upheld.
However, potential vio- lations during the LLMs training phase do not affect the
 lawfulness of using such a model within an AI system.

\* 1 This refers exclusively to models as an important, but not sole, component
of a comprehensive AI system (e.g. an LLM-based chatbot).

<!-- PageFooter: www.datenschutz-hamburg.de E-mail: mailbox@datenschutz.hamburg.
de Ludwig-Erhard-Strasse 22 - D-20459 Hamburg - Tel .: 040 - 4 28 54 - 40 40 -
Fax: 040 - 4 28 54 - 40 00 Confidential information should only be sent to us
electronically in encrypted form. Our public PGP key is available on the
Internet (fingerprint: 0932 579B 33C1 8C21 6C9D E77D 08DD BAE4 3377 5707). -->
<!-- PageBreak -->

<!-- PageHeader: The Hamburg Commissioner for Data protection and freedom of
information -->

### I. Introduction

When an LLM, functioning as a component of an AI system, processes2 prompts (so-
called "in- ference"), the LLM's output may contain information relating to
natural persons, especially if the prompt specifically asks for it. This raises
the question of whether personal data is stored in an LLM.
```

Appendix B. LLM Regex Prompt

```
<Task>
Your task is to split a long document into self-contained and logically complete
 chunks to be used in a Retrieval Augmented Generation (RAG) system. Given a
document text, choose the best **unique** regular-expression to be used as a *
delimiter* to split it into small chunks using the Python `re` engine and the `
re.split` function.
</Task>

<Output requirements>
You **must** return only the answer in this format:
    <regex>regex pattern here</regex>
</Output requirements>

<Splitting guidelines>
    - The regex pattern **must** be valid.
    - The chunks should be self-contained, logically complete and not too large.
    - Do not split paragraphs.
    - Do not split tables, marked between <Table> </Table> tags.
    - Do not split figures, marked between <Figure> </Figure> tags.
    - Do not split lists of short elements.
    - Do not split titles from the text that follows them.
    - Do not split footnotes from their parent text.
</Splitting guidelines>

<Splitting example>
    <Example of input text>
{example["input"]}
    </Example of input text>

    <Expected answer>
        <regex>{example["output"]}</regex>
    </Expected answer>
</Splitting example>

Now, please apply this method to the following text between <Input> and </Input>
 markers:
<Input>{document_context_str}</Input>
```

Appendix C. Question-Answer Pairs Generation Prompt

```
<Task>
Create exactly {qa_pairs_per_document} high-quality, mutually distinct question-
answer pairs based on the provided Document Lines context.
</Task>

<Instructions>
1. Create each question-answer pair based *only* on the text provided in the
Document Lines section. Do not use any external knowledge.
2. Each question should be concise with less than 20 words and sound like a
natural question a curious person would ask.
3. Each question should target a key concept or relationship, not a trivial
detail.
4. Each question must be self-contained. They should still make sense if read
alone. Avoid vague references (e.g., "the report/paper/it/they/this/that/above").

5. Answers must be strictly grounded in the document lines.
6. Write both questions and answers in English, even if the document lines are
in another language.
7. Produce **{qa_pairs_per_document}** questions that are **meaningfully
different** from one another: no rephrasing, duplicates, or overlapping focus.
Cover different aspects of the content when possible.
</Instructions>

<Document Lines>
{document_context}
</Document Lines>
```

Appendix D. Retrieval Completeness Evaluation Prompt

```
<Task>
You are an expert fact-checker. Your task is to evaluate how completely the
provided Context supports a Reference Answer.
</Task>

<Instructions>
1.  Read the Context thoroughly.
2.  Read the Reference Answer carefully.
3.  Compare the Reference Answer against the Context.
4.  Classify the completeness level:
    - 0 = Incomplete: Context does not support key claims.
    - 1 = Partially Complete: Context supports some but not all claims.
    - 2 = Complete: Context fully supports all claims.
5.  Provide a brief one-sentence reason.
</Instructions>

<Context>
{context}
</Context>

<Reference Answer>
{reference_answer}
</Reference Answer>
```

Appendix E. Answer Generation Prompt for RAG

```
<Task>
You are a grounded QA assistant. Answer the question strictly using the provided
 context. Read all the documents provided as context before answering the
question. Do not hallucinate. If the context is insufficient, reply exactly: "I
don't know based on the provided context." and nothing else. Keep the answer
concise. Answer only in English even if the context or question is in another
language.
</Task>

<Context>
{context_str}
</Context>

<Question>
{question_str}
</Question>
```

Appendix F. LLM Regex Splitter Diagram

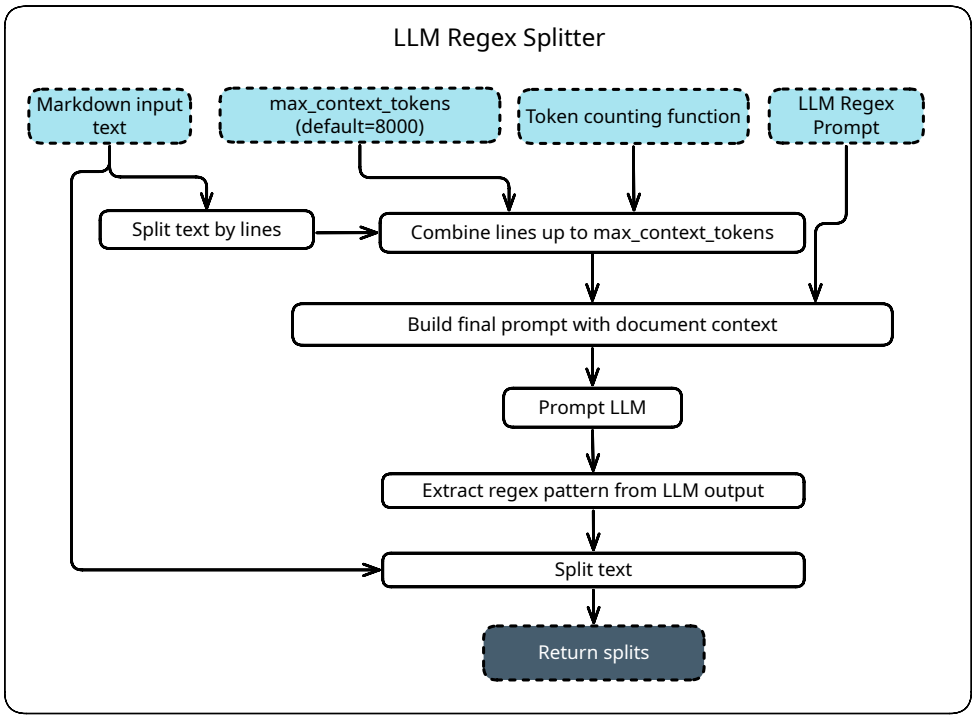

Figure 3: LLM Regex splitter pipeline. From markdown input and a token budget, the system builds a context-aware prompt, obtains a delimiter regex from the LLM, and applies it to split the document into chunks.

Appendix G. Separators list for recursive splitters

Regular Expression	Description
`(?<=\n)#{1}\s`	1st level titles
`(?<=\n)#{2}\s`	2nd level titles
`(?<=\n)#{3}\s`	3rd level titles
`(?<=\n)#{4}\s`	4th level titles
`(?<=\n)#{5}\s`	5th level titles
`(?<=\n)#{6}\s`	6th level titles
`(?<=\n)\s*\(?[A-Za-z0-9]{1,4}[.)]\s+`	Numbered list item (e.g., "1.", "(a)", "I. ")
`(?<=\n)\s*[-*·•●▪◦▸▸▹○○–––]\s+`	Bulleted list item
`\n{2,}`	Blank lines (2 or more newlines)
`\n`	Single newline (useful for tables)
`[.!?]\s+`	Sentence end
`,\s+`	Comma separator
`\s+`	Space separator
`"" (empty string)`	Character-level separator

Figure 4: Regex separators list for recursive splitters, adapted to our Markdown parser outputs. The list is sorted from highest to lowest priority.

Appendix H. Split-then-merge Recursive Splitter Diagram

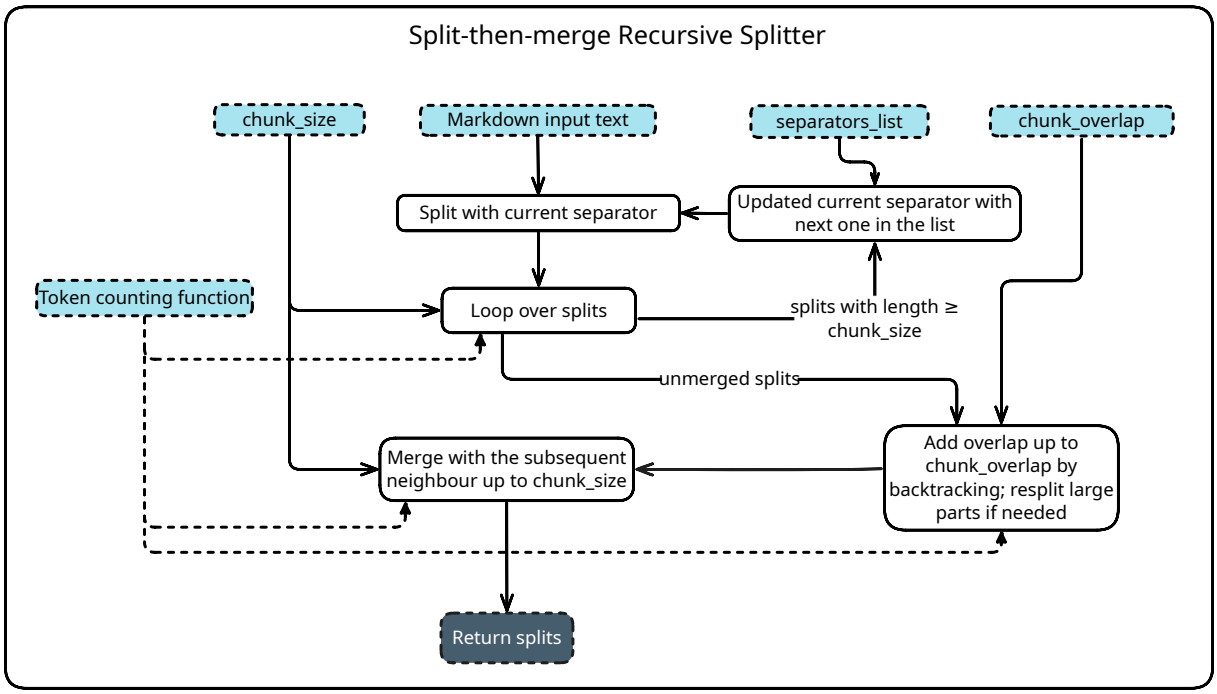

Figure 5: Split–then–merge recursive splitter pipeline. Text is recursively split following a priority list of separators until each piece is $\leq$ chunk size, then merged forward into chunks; when the next piece would exceed chunk size, a new chunk starts with up to chunk overlap tokens of backtracked overlap.

Appendix I. Post-Processing Effect Details

Table 7 reports the detailed effect of our two-stage post-processing pipeline on Size Compliance (SC) and overall mean chunking scores. The pipeline consists of: (1) *oversized re-split*, applied only to chunks exceeding 1100 tokens, and (2) *tiny-chunk merge*, applied only to chunks smaller than 100 tokens, with a maximum merged size of 1150 tokens. The "–" symbol indicates that a particular step was not required for that method (e.g., our recursive splitters with $s = 1100$ or $s = 600$ already produce well-sized chunks in the raw output).

For every method that required post-processing, both SC and the final mean score improved substantially. The largest improvements were observed for the LLM regex method (raw SC: 58.3% $\rightarrow$ final SC: 99.6%, mean: 80.0% $\rightarrow$ 89.8%) and the semantic chunker (raw SC: 48.1% $\rightarrow$ final SC: 99.9%, mean: 76.5% $\rightarrow$ 88.9%), demonstrating that post-processing is particularly critical for methods that initially produce highly variable chunk sizes. The final SC scores do not always reach 100% because we prioritize minimizing tiny chunks over strict adherence to the upper bound—specifically, we avoid merging fragments that would exceed 1150 tokens even if this leaves some chunks slightly below the 100-token minimum.

	Raw		Oversized re-split		Tiny-chunk merge	
Chunking method	**SC %**	**final mean %**	**SC %**	**final mean %**	**SC %**	**final mean %**
LLM regex (GPT-5)	58.3	80.00	71.4	82.55	99.6	89.80
our recursive ($s = 1100$)	100.0	90.67	–	–	100.0	90.68
our recursive ($s = 600$)	98.7	88.96	–	–	100.0	89.24
page	92.7	89.03	95.3	89.53	99.9	90.52
LC recursive ($s = 1100$)	93.3	88.37	–	–	99.4	89.93
LC recursive (default)	97.7	88.62	–	–	–	–
semantic	48.1	76.49	73.8	82.11	99.9	88.90
sentence	67.2	73.26	–	–	100.0	82.03

Table 7: Effect of post-processing on Size Compliance (SC) and the final mean chunking score. Columns follow the sequential pipeline *raw (no post-processing)* $\rightarrow$ *oversized re-split* $\rightarrow$ *tiny-chunk merge*. Oversized re-split is only applied to chunks with size > 1100 tokens. Tiny-chunk merge is only applied to chunks with size < 100 tokens with a maximum merging size of 1150 tokens.