CHROMA TECHNICAL REPORT

July 03, 2024

Evaluating Chunking Strategies for Retrieval

Brandon Smith Researcher in Residence - Chroma

Anton Troynikov Cofounder - Chroma

Despite document chunking being virtually ubiquitous as a pre-processing step, little work has been done to

investigate its impact on retrieval performance. This is partially due to the structure of commonly used information retrieval benchmarks, which are aimed at whole-document retrieval tasks.

In this work we present an evaluation approach which takes token-level relevance into account, and allows for the evaluation of several popular document chunking strategies. We demonstrate that the choice of chunking strategy can have a significant impact on retrieval performance, with some strategies outperforming others by up to 9% in recall.

Chunking	Size	Overlap	Recall	Precision	Precision_Ω	IoU
Recursive	800 (~661)	400	85.4 ± 34.9	1.5 ± 1.3	6.7 ± 5.2	1.5 ± 1.3
TokenText	800	400	87.9 ± 31.7	1.4 ± 1.1	4.7 ± 3.1	1.4 ± 1.1
Recursive	400 (~312)	200	88.1 ± 31.6	3.3 ± 2.7	13.9 ± 10.4	3.3 ± 2.7
TokenText	400	200	88.6 ± 29.7	2.7 ± 2.2	8.4 ± 5.1	2.7 ± 2.2
Recursive	400 (~276)	0	89.5 ± 29.7	3.6 ± 3.2	17.7 ± 14.0	3.6 ± 3.2
TokenText	400	0	89.2 ± 29.2	2.7 ± 2.2	12.5 ± 8.1	2.7 ± 2.2
Recursive	200 (~137)	0	88.1 ± 30.1	7.0 ± 5.6	29.9 ± 18.4	6.9 ± 5.6
TokenText	200	0	87.0 ± 30.8	5.2 ± 4.1	21.0 ± 11.9	5.1 ± 4.1
Kamradt	N/A (~660)	0	83.6 ± 36.8	1.5 ± 1.6	7.4 ± 10.2	1.5 ± 1.6
⋆ KamradtMod	300 (~397)	0	87.1 ± 31.9	2.1 ± 2.0	10.5 ± 12.3	2.1 ± 2.0
⋆ Cluster	400 (~182)	0	91.3 ± 25.4	4.5 ± 3.4	20.7 ± 14.5	4.5 ± 3.4
⋆ Cluster	200 (~103)	0	87.3 ± 29.8	**8.0 ± 6.0**	**34.0 ± 19.7**	**8.0 ± 6.0**
⋆ LLM	N/A (~240)	0	**91.9 ± 26.5**	3.9 ± 3.2	19.9 ± 16.3	3.9 ± 3.2

Evaluation of various popular chunking strategies on

our evaluation, as well as new (★) strategies we propose. We show that the choice of chunking strategy can have significant impact on retrieval performance, in terms of accuracy and efficiency. Size denotes chunk size in tokens, in brackets indicates mean chunk size where it may vary by chunking strategy. Overlap denotes the chunk overlap in tokens. Bold values highlight the best performance in each category. See metrics section for details of each metric.

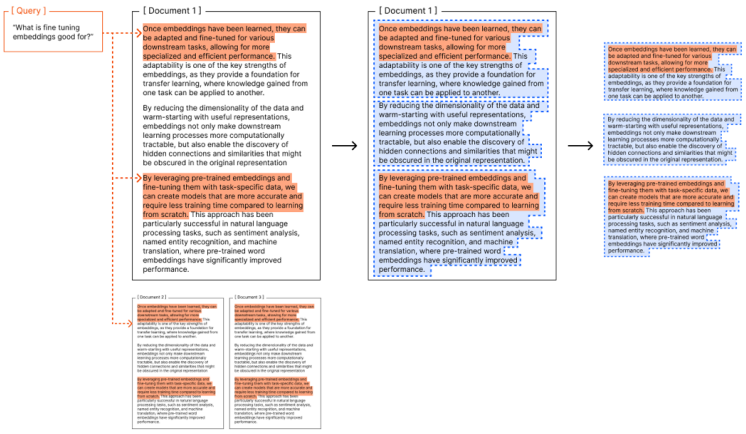
Chunking is a commonly used pre-processing step when ingesting documents for retrieval in the context of AI applications. Chunking serves to divide documents into units of information, with semantic content suitable for embeddings-based retrieval and processing by an LLM.

The purpose of this technical report is to evaluate the impact of the choice of chunking strategy on retrieval performance, in a way representative of how chunking and retrieval is used in the context of AI applications.

While LLM context lengths have grown, and it has become possible to insert entire documents, or even text corpora into the context window, in practice doing so is often inefficient, and can distract the model. For any given query, only a small portion of the text corpus is likely to be relevant, but all tokens in the context window are processed for each inference. Ideally, for each query, the LLM

would only need to process only the relevant tokens, and hence one of the primary goals of a retrieval system in AI applications is to identify and retrieve only the relevant tokens for a given query.

Commonly used benchmarks like **MTEB** take a traditional information retrieval (IR) approach, where retrieval performance is typically evaluated with respect to the relevance of entire documents, rather than at the level of passages or tokens, meaning they cannot take chunking into account.

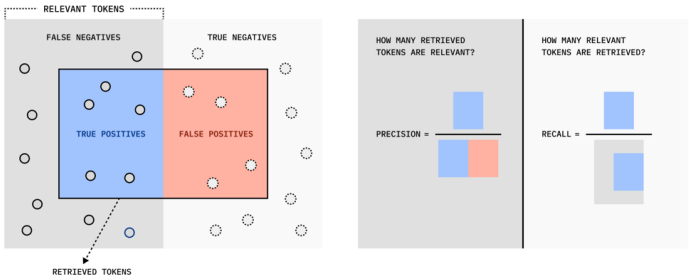


In AI applications, excerpts containing all tokens relevant to a query may be found within or across many documents. Chunks may contain both relevant and irrelevant tokens, and relevant excerpts may be split across chunks.

Traditional IR benchmarks also often focus on the relative ranking of the retrieved documents, however in practice, LLMs are relatively insensitive to the position of the relevant information within the context window. Additionally, the information relevant to a given query may be spread across multiple documents, making the evaluation of relative

ranking between documents ambiguous.

Motivated by these limitations, we propose a new evaluation strategy, evaluating retrieval performance at the token level. Our evaluation uses an LLM to generate, and subsequently filter, a set of queries and associated relevant exerpts from any given text corpus, and subsequently evaluates retrieval performance via precision, recall, and intersection-over-union (Jaccard index) on the basis of retrieved *tokens*.



Our evaluation takes a fine-grained, token-wise approach to evaluating retrieval accuracy and efficiency, which allows it to take components such as chunking into account. The presence or absence or relevant tokens is more important than their relative ranking in AI applications.

We generate a concrete dataset across various domains with varying data cleanliness, and use this to evaluate the performance of a several popular chunking strategies. We find that the choice of chunking strategy has significant impact on retrieval accuracy and efficiency. Notably, we find that default settings for certain popular chunking strategies can lead to relatively poor performance. We also find that heuristic chunking strategies such as the

popular **RecursiveCharacterTextSplitter** often perform well in practice when parametrized appropriately.

Finally, we propose and evaluate two novel chunking strategies; **ClusterSemanticChunker**, which uses embedding models directly to compose chunks up to a given size on the basis of semantic similarity, and **LLMChunker**, which prompts an LLM directly to perform chunking over a text corpus. We find that both produce competitive results on our evaluation.

We provide all code to replicate our results as a GitHub repo, along with useful utilities for similar experiments. We hope that this preliminary work will inspire further research into factors affecting real world performance of retrieval systems for AI applications.

Our in-depth technical report continues below. If you find our work useful, please consider citing us:

plaintext Copy Code

```
@techreport{smith2024evaluating,
  title = {Evaluating Chunking Strate
  author = {Smith, Brandon and Troynil
  year = {2024},
  month = {July},
  institution = {Chroma},
  url = {https://research.trychroma.c
}
```

Thanks to Douwe Kiela and Chris Manning for their feedback on this work.

Interested in working on improving retrieval for AI applications? **Chroma is Hiring**

Introduction

Besides answering questions and generating text [1], recently Large Language Models (LLMs) have emerged as a new type of computing primitive, capable of processing unstructured information in a "common sense" way, leading to the creation of AI applications. A key element of AI applications is so-called Retrieval-Augmented Generation (RAG)[2], wherein external data which is semantically relevant to the current task is retrieved for processing. In contrast to traditional IR, retrieval for AI applications often relies on storing and retrieving document parts (chunks), sometimes across several documents, in order to present the most relevant information to the LLM. Thus, in the AI application use-case, which *tokens* and *chunks* are returned is often as important as which *documents*.

Information Retrieval (IR) in general, and passage retrieval in particular, are long-studied problems in natural language processing, with a considerable literature [3][4][5]. However, benchmarks like the Massive Text Embedding

Benchmark (MTEB) [6] and MSMARCO[10] do not account for token efficiency, or chunking, and usually evaluate on the basis of the relevance of entire retrieved documents. Additionally, the normalized Discounted Cumulative Gain (nDCG@K) metric, commonly used in IR [7], is less useful in the context of RAG because the rank order of retrieved documents is less important. It has been shown that as long as the relevant tokens are present, and the overall context is of reasonable length, LLMs can accurately process the information regardless of token position [8].

To address these shortcomings, we propose a new evaluation designed to capture the essential details of retrieval performance in the AI application context, consisting of a generative dataset, and a new performance measure. We describe the pipeline that generates the dataset, allowing others to generate domain specific evaluations for their own data and use cases. Because the dataset is generated, in general it should not exist in the training set of any general-purpose embedding model. Along with the dataset, we introduce a new measure of performance, based on the Jaccard similarity coefficient [11] at the token level, which we refer to as Intersection over Union (IoU) for short, as well as evaluating recall and precision at the token rather than document level. By analogy to its use in computer vision, we can think of text chunks as bounding boxes, and the IoU as a measure of how well the bounding boxes of the retrieved

chunks overlap with the bounding boxes of the relevant tokens.

As an application of our new evaluation strategy, we compare the performance of popular document chunking methods, such as the **RecursiveCharacterTextSplitter**. We also develop and evaluate two new chunking methods, the **ClusterSemanticChunker** which takes the embedding model used for retrieval into account, and **LLMChunker**, which prompts an LLM directly to perform chunking over a text corpus.

Contributions

We present the following:

- A framework for generating domain specific datasets to evaluate retrieval quality in the context of AI applications, as well as a new measure of retrieval quality which takes chunking strategies into account.
- We use this framework to generate a concrete evaluation dataset for a limited set of popular domains, and subsequently we evaluate several commonly-used document chunking strategies according to the proposed measures.
- We present and evaluate new chunking strategies, including an embedding-model aware chunking strategy, the **ClusterSemanticChunker**, which uses

any given embedding model to produce a partition of chunks, as well as the **LLMChunker** which prompts an LLM to perform the chunking task directly.

- Finally, we provide the **complete codebase** for this project.

Related Work

Current popular Information Retrieval (IR) benchmarks include the Massive Text Embedding Benchmark (MTEB) [6] and Benchmarking-IR (BEIR) [7]. MTEB evaluates text embedding models across 58 datasets and 8 tasks. BEIR includes 18 datasets across 9 retrieval tasks. The MTEB text retrieval datasets contain all publicly available BEIR retrieval datasets. Their primary metric is normalized Discounted Cumulative Gain at rank 10 (nDCG@10), and they both provide Mean Average Precision at rank k (MAP@k), precision@k and recall@k additionally.

The nDCG@K metric evaluates the relevance of K retrieved documents, giving more weight to higher-ranked documents, penalizing relevant results that appear lower in the ranking. Similarly, MAP@K takes the average of the

precision at each position in the top K retrieved documents. As mentioned in the introduction, the order of retrieved documents is less important in the context of RAG. Additionally, existing benchmarks evaluate retrieval system performance at the document level, while for AI applications, only a fraction of the tokens in any document may be relevant to a particular query, and relevant tokens may be found across an entire corpus. Our proposed evaluation metrics aim to take this token-level relevance into account.

Along with information retrieval in general, there has been extensive work on passage retrieval [9], including at large scales and in a machine learning context as with the MS MARCO dataset [10]. However, this work has mostly been considered as either an intermediate step to document retrieval, or in the general context of search and retrieval, rather than in the specific context of retrieval for AI applications, and without reference to token efficiency.

More recently, LLMs have been used to directly evaluate retrieval performance, for example by prompting the LLM for a binary classification of relevancy as in ARAGOG [12]. LLMs have also been used to generate synthetic data from a text corpus, as well as evaluating the final generated output of a RAG pipeline as in RAGAS [13]. In contrast, our proposed evaluation focuses only on retrieval, with a limited step to synthesize queries from document corpora, rather than complex multi-

step synthesis and evaluation pipelines which may be sensitive to model particulars and prompting. We evaluate retrieval directly, without relying on an LLM. In principle, our proposed approach can be composed with others to produce a more complete evaluation.

Despite the fact that chunking is often the first step of data ingestion in RAG pipelines, the literature on evaluating chunking strategies is sparse. Greg Kamradt's work on semantic chunking was incorporated by LangChain [14], and Aurelio AI developed its own version [15]. Additionally, Unstructured explored chunking in financial contexts, focusing on metadata about chunks based on their document position rather than optimal text partitioning [16]. These efforts highlight the emerging interest evaluations of chunking for RAG, yet there remains a significant gap in comprehensive evaluation, which our works is intended to address.

Evaluating Retrieval for AI Applications

We introduce our evaluation framework for retrieval in the context of AI applications, consisting of a generative evaluation dataset from a supplied corpus of documents, as well as a new metric based on the well-known Jaccard similarity coefficient, token-wise Intersection over Union (IoU) which takes

chunking strategies into account, and corresponds to the token efficiency of the retrieval system. We use this metric alongside other measures such as recall to evaluate the effectiveness of retrieval systems in the AI application context.

This approach to partially synthetic data generation ensures that data generated this way was not available at training time for any general-purpose embedding model, preventing possible bias. Our approach also allows for domain-specific evaluation, making it suitable for evaluating retrieval quality on any dataset. Finally, our approach can be used in combination with other evaluation approaches, including the use of human labels.

Dataset Generation

For a given text corpus consisting of a set of documents, we sample from an LLM to generate a query relevant to the documents, as well as excerpts from the corpus which are relevant to the generated query. We present the LLM with a prompt consisting of documents from the evaluation corpus, instructions to generate factual queries about the corpus, and to provide excerpts corresponding to the generated questions.

To ensure the queries are unique, we include a random sample of up to 50 previously generated queries in the prompt. We accept only excerpts which have full-text matches

within the corpus as valid. We reject any query and all associated excerpts, if any excerpt is invalid.

```
Query: What were the main characteristics
of the armor used on the ship Atlanta?
Excerpts:
 1. "Her armor was backed by 3 inches ( 76
    mm ) of oak , vertically oriented ,
    and two layers of 7 @.@ 5 inches ( 191
    mm ) of pine , alternating in
    direction "
 2. "The upper portion of Atlanta 's hull
    received two inches of armor "
```

In order to reduce the instance of compound queries, e.g. "What was the date and significance of the Gettysburg Address?", which may complicate evaluation, we prompt the LLM to avoid using the connecting form of 'and', allowing it only when used as part of a proper noun.

For details of our prompt, see **the appendix**.

Dataset Prefiltering

To ensure high quality queries and excerpts we filter the outputs of the initial generation step.

Despite prompting to avoid duplicate generated queries, the LLM will occasionally still generate duplicates or near-duplicates. We de-duplicate the generated dataset by embedding each query, and computing the cosine similarity between the embeddings of all pairs of queries. Subsequently, we filter all queries and

associated excerpts where the cosine similarity is above a certain threshold value.

We note that the LLM will occasionally produce excerpts which are out-of-context or irrelevant to the query. In order to mitigate this, we embed each query and its associated excerpts, and compute the cosine similarity from the query to each of them. We filter any query and all its associated excerpts, where the query's cosine similarity with any of its associated excerpts is less than a set threshold.

The choice of threshold values for both duplicate queries and excerpt relevance is dataset dependent. We provide details of the threshold values used in **the details of our concrete dataset**. We use OpenAI's text-embedding-3-large as the embedding model.

We observe that the latter threshold sets a minimum cosine similarity between a query and its related excerpts. Although this might introduce a sampling bias, it is important to note that each excerpt is part of one or more text chunks, with each chunk being one among many from a given corpus. Additionally, because we perform our evaluation over multiple embedding models, the impact of this bias is negligible in practice.

We note also that although we filter aggressively to ensure generated excerpts are relevant to each query, we do not search for other exceprts in the corpus which might be relevant to the query but were not generated. This is a limitation of our current approach,

which future work may mitigate by using supervised or semi-supervised methods.

Metrics

For a given query related to a specific corpus, only a subset of tokens within that corpus will be relevant. Ideally, for both efficiency and accuracy, the retrieval system should retrieve exactly and only the relevant tokens for each query across the entire corpus.

In practice, the unit of retrieval for AI applications is usually a text chunk containing the relevant excerpt. This chunk will often contain superfluous tokens which require additional compute to process, and which may contain irrelevant distractors which may reduce overall performance of the RAG application [17]. Additionally, where a chunking strategy uses overlapping chunks, the retriever may return redundant tokens, for example if tokens from a relevant excerpt are in more than one chunk due to overlap. Finally, a given retriever may not retrieve all necessary excerpts for a given query.

We therefore seek a metric which can take into account not only whether relevant excerpts are retrieved, but also how many irrelevant, redundant, or distracting tokens are also retrieved. Inspired by a similar metric in computer vision and data mining, the Jaccard similarity [11], we propose the token-wise Intersection over Union (IoU) metric for

evaluating the efficiency of a retrieval system [18].

We compute IoU for a given query and chunked corpus $\mathbf{C}$ as follows. For a query q, we denote the set of tokens among all relevant excerpts t_e. The set of all retrieved chunks consists of tokens t_r.

We compute IoU for a given query in the chunked corpus as:

$$\text{IoU}_q\left(\mathbf{C}\right) = \frac{|t_e \cap t_r|}{|t_e| + |t_r| - |t_e \cap t_r|}$$

Note that in the numerator, we count each t_e among t_r only once, while counting *all* retrieved tokens in $|t_r|$ in the denominator. This accounts for the case where redundant excerpt tokens appear in multiple retrieved chunks, for example where an overlapping chunking strategy is used.

Along with IoU, we propose to use precision and recall metrics in a similar way as they are traditionally used in information retrieval, but at the token rather than the document level.

We define precision as:

$$\text{Precision}_q\left(\mathbf{C}\right) = \frac{|t_e \cap t_r|}{|t_r|}$$

And recall as:

$$\text{Recall}_q\left(\mathbf{C}\right) = \frac{|t_e \cap t_r|}{|t_e|}$$

When performing our evaluations, we report the mean of IoU, Precision, and Recall, across all queries and all corpora in our dataset. Additionally, we report precision for the case that all chunks containing excerpt tokens are successfully retrieved as Precision_Ω. This gives an upper bound on token efficiency given perfect recall.

We note that we could also compute the F-score (F_1) as:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{|t_e \cap t_r|}{|t_e| + |t_r|}$$

But in practice, IoU provides similar information while preserving a more intuitive 'bounding box' analogy.

Chunking Evaluation Dataset

We use our evaluation framework to generate a dataset for evaluating chunking strategies for retrieval for AI applications.

Corpora

We selected five diverse corpora for our dataset, ensuring a mix of both clean and messy text sources. Clean corpora contain well-structured text, while messy corpora reflect the unstructured nature of text typically obtained from web scraping. Each corpus is briefly described below. We provide the length of each corpus in tokens, measured with Tiktoken for "cl100k_base" (the standard used by OpenAI's GPT-4). All Hugging Face corpora are subsets of the full corpora, and the lengths provided refer to the subsets we used. These subsets always start from the beginning of the text as it is on Hugging Face.

While these corpora are relatively small in comparison to most large-scale retreival benchmarks, our aim is to demonstrate the utility of our evaluation framework on representative data, provide easily reproduced results, and to provide a starting point for future work. We do not claim that the concrete dataset we present is comprehensive, but rather that it demonstrates the utility of our evaluation framework.

State of the Union Address 2024 [19]

This is a plain transcript of the State of the Union Address in 2024. It is well-structured and clear. This corpus is 10,444 tokens long.

Wikitext [20]

The WikiText language modeling dataset consists of over 100 million tokens from verified

Good and Featured articles on Wikipedia. Our subset is the first 26,649 tokens of this text as it appears on Hugging Face.

Chatlogs [21]

The UltraChat 200k dataset is a high quality filtered subset of UltraChat consisting of 1.4M dialogues generated by ChatGPT. Our corpus includes all surrounding JSON syntax, making it a more accurate representation of real-world raw text. Our subset is the first 7,727 tokens of this text.

Finance [22]

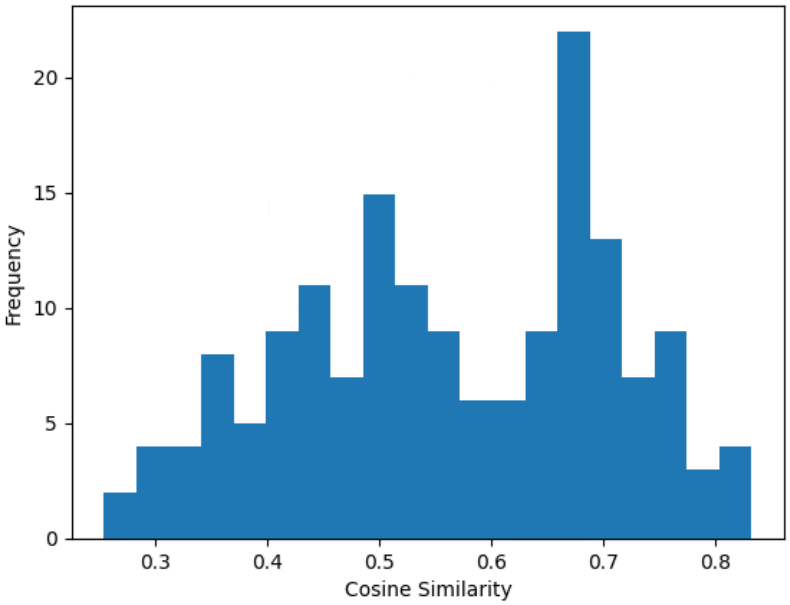
The ConvFinQA dataset is designed to study numerical reasoning in conversational question answering within the finance domain. This corpus includes complex multi-turn questions and answers based on financial reports, focusing on modeling long-range numerical reasoning paths. Our subset is the first 166,177 tokens of this text.

Pubmed [23]

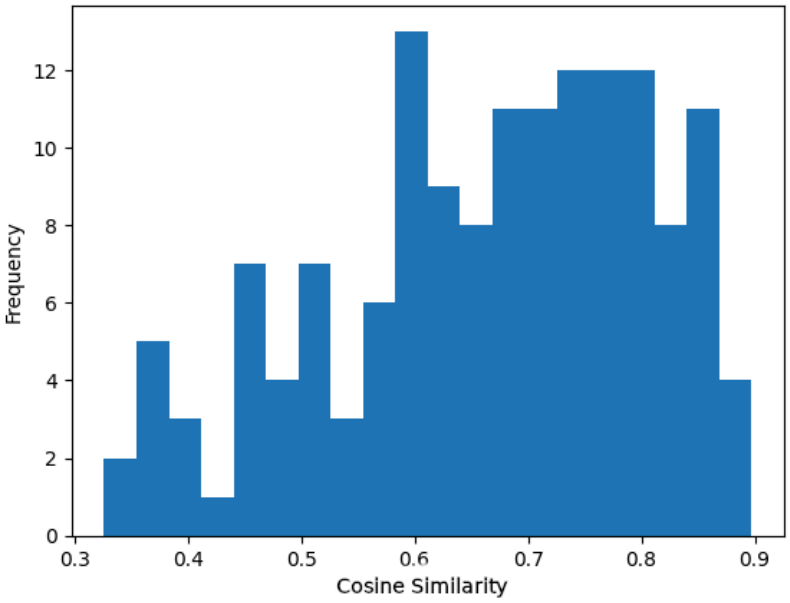
The PMC Open Access Subset is a collection of biomedical and life sciences journal literature from the National Library of Medicine. Our subset is the first 117,211 tokens of this text.

Generation & Prefiltering

To determine the duplicate threshold, we perform a binary search over threshold values by sampling from pairs of generated queries for a given threshold, manually examine the sampled pairs, and adjusting the threshold according to whether the queries in each pair are sufficiently distinct.



Distributions of cosine similarity between queries and their associated excerpts, Pubmed corpus.



Distribution of cosine similarity between all query pairs, Pubmet corpus.

A similar approach is used for the excerpt filter. Future work could automate this process. We note also that in the adopted threshold values are similar across corpora, suggesting a heuristic value could be used in practice.

Corpus	No. of Tokens	No. of Queries	Ex. Thresh.	
State of the Union	10,444	76	0.40	
Wikitext	26,649	144	0.42	
Chatlogs	7,727	56	0.43	
Finance	166,177	97	0.42	
Pubmed	117,211	99	0.40	
Total	328,208	472		

Number of tokens, number of queries, excerpt threshold and duplicate query threshold for each corpus.

All text excerpts and queries are available in the GitHub repository.

Time & Cost Analysis

The time required for query generation varied between 3 to 16 seconds per question, averaging around 10 seconds. The longer times were primarily due to the need to disqualify questions based on invalid excerpts, which required additional processing to generate valid questions.

For our base prompt, we used 703 tokens (measured with OpenAI's cl100k Tokenizer). During query generation, we included a random

4,000-character subset from the original corpus and 50 sampled queries from the dataset, resulting in an average prompt size of 2,000 tokens. The average response length was 200 tokens, giving an overall input length of 2,000 tokens and an output length of 200 tokens per question.

At the time of writing, the cost of generating a single question using GPT-4 is approximately $0.01. Consequently, creating a dataset of 1,000 questions would cost around $10. Note that this estimate does not account for the subsequent filtering steps, which would reduce the final question count.

Chunking Algorithms

Below we present the chunking methods used in this report. We evaluate general-purpose, commonly-used chunkers, as well as novel approaches . Any mentions of tokens in this section refer to that within the context of OpenAI's cl100k Tokenizer. The symbol ★ indicates that the subsequent algorithm is a new chunking method we developed for this study.

RecursiveCharacterTextSplit & TokenTextSplitter [24]

`RecursiveCharacterTextSplitter` and `TokenTextSplitter` are some of the most popular chunking methods, and the default used by many RAG systems. These chunking methods are insensitive to the semantic content of the corpus, relying instead on the position of character sequences to divide documents into chunks, up to a maximum specified length. We follow the implementation of the popular Langchain library.

We found that it was necessary to alter some defaults to achieve fair results. By default, the `RecursiveCharacterTextSplitter` uses the following separators: `["\n\n", "\n", " ", ""]`. We found this would commonly result in very short chunks, which performed poorly in comparison to `TokenTextSplitter` which produces chunks of a fixed length by default. Therefore we use `["\n\n", "\n", ".", "?", "!", " ", ""]` as the set of separators.

KamradtSemanticChunker [14]

Greg Kamradt proposed a novel semantic chunking algorithm, which was later incorporated into LangChain. The chunker works by first splitting the corpus by sentence. It functions by computing the embedding of a

sliding window of tokens over a document, and searching for discontinuities in the cosine distances between consecutive windows. By default, the threshold for detecting a discontinuity is set to any distance above the 95th percentile of all consecutive distances. This is a relative metric and can lead to larger chunks in bigger corpora.

KamradtModifiedChunker

It is desirable for users to be able to set the chunk length directly, in order to ensure that chunks fit within the context window of the embedding model. To address this, we modify the `KamradtSemanticChunker` by implementing a binary search over discontinuity thresholds such that the largest chunk is shorter than the specified length.

ClusterSemanticChunker

We further extend the principle of embeddings-based semantic chunking by observing that the `KamradtSemanticChunker` algorithm is greedy in nature, which may not produce the optimal chunking under the objective of preserving chunks with high internal similarity. We propose a new algorithm, the `ClusterSemanticChunker`, which aims to produce globally optimal chunks by maximizing the sum of cosine similarities

within chunks, up to a user-specified maximum length. Intuitively, this preserves as much similar semantic information as possible across the document, while compacting relevant information in each chunk.

Documents are divided into pieces of at most 50 tokens using the `RecursiveCharacterTextSplitter`, and pieces are individually embedded. Subsequently, we use a dynamic programming approach to maximize the pairwise cosine similarity between all pairs of pieces within any chunk. Code for this algorithm is available in the **GitHub repository**

A limitation to this method is that the globally optimal packing relies on the global statistics of the corpus, requiring chunks to be re-computed as data is added. This limitation may be improved upon by maintaining an appropriate data-structure over the smaller pieces as data is added or removed; we leave this to future work.

★

LLMSemanticChunker

In the spirit of "just [ask] the model", we experimented with directly prompting an LLM to chunk the text. We found that a naive approach where the LLM is prompted to repeat the corpus with a `<|split|>` token was costly, slow, and suffered from hallucinations in the produced text.

To resolve this we broke the text into chunks of size 50 in tokens using a `RecursiveCharacterTextSplitter`. Then we re-joined the text but surrounded by chunk tags resulting in text like the following:

```
<start_chunk_0>On glancing over my notes
of the seventy odd cases in which I have
during the last eight years <end_chunk_0>
<start_chunk_1>studied the methods of my
friend Sherlock Holmes, I find many
tragic, some comic, a large number
<end_chunk_1><start_chunk_2> . . .
```

We then instruct the LLM to return the indexes of chunks to split on, in the form

```
split_after: 3, 5
```

We found both GPT-4o and Llama 3 8B Instruct consistently adhered to this format. Complete prompts and code for this algorithm are available in the **GitHub repository**

Results

We present the mean of Recall, Precision, Precision_Ω, and IoU scores for each chunking method, over all queries and corpora in our evaluation. ± indicates the standard deviation.

We report the results for n=5 retrieved chunks, using OpenAI's `text-embedding-3-large` as the embedding model.

Chunking	Size	Overlap	Recall	
Recursive	800 (~661)	400	85.4 ± 34.9	
TokenText	800	400	87.9 ± 31.7	
Recursive	400 (~312)	200	88.1 ± 31.6	
TokenText	400	200	88.6 ± 29.7	
Recursive	400 (~276)	0	89.5 ± 29.7	
TokenText	400	0	89.2 ± 29.2	
Recursive	200 (~137)	0	88.1 ± 30.1	
TokenText	200	0	87.0 ± 30.8	

Kamradt	N/A (~660)	0	83.6 ± 36.8
★ KamradtMod	300 (~397)	0	87.1 ± 31.9
★ Cluster	400 (~182)	0	91.3 ± 25.4
★ Cluster	200 (~103)	0	87.3 ± 29.8
★ LLM (GPT4o)	N/A (~240)	0	**91.9 ± 26.5**

Results for text-embedding-3-large. Size denotes the chunk size in tokens (cl100k tokenizer). Size in brackets indicates mean chunk size where it may vary by chunking strategy. Overlap denotes the chunk overlap in tokens. Bold values highlight the best performance in each category.

We find that the heuristic `RecursiveCharacterTextSplitter` with chunk size 200 and no overlap performs well. While it does not achieve the best result, it is consistently high performing across all evaluation metrtics. We note that it outperforms `TokenTextSplitter` across all metrics for chunk size of 400 or less with no chunk overlap. Interestingly, this does not hold for larger chunk sizes and overlap. Unsurprisingly, reducing chunk overlap improves IoU scores,

as this metric penalizes redundant information.

OpenAI Assistants can use file search to improve their response, following the standard Retrieval-Augmented Generation (RAG) process. According to their documentation, the default chunking strategy uses a chunk size of 800 tokens with an overlap of 400 tokens [25]. Assuming the `TokenTextSplitter` method is employed, we observe that this setting results in slightly below-average recall and the lowest scores across all other metrics, suggesting particularly poor recall-efficiency tradeoffs.

The `ClusterSemanticChunker`, with a max chunk size set to 400 tokens, achieves the second highest recall of 0.913. Dropping the max chunk size to 200 results in average recall but the highest precision, Precision_Ω, and IoU. The `LLMSemanticChunkers` achieves the highest recall of 0.919 while having average scores on the remaining metrics, suggesting LLMs are relatively capable at this task.

The `KamradtSemanticChunker` with the default settings scores slightly below average across all metrics, with a recall of 0.836. Recall rises to 0.871 with a similar boost in the remaining metrics when the modifications of the `KamradtModifiedChunker` are applied. We note that recall improved despite the mean chunk length dropping.

We speculate that recall could reach a maximum before relevant information is diluted within chunks, making it more difficult to retrieve, while chunks which are too small fail

to capture necessary context within a single unit.

We repeat our experiments with the embedding model changed to the Sentence Transformers 'all-MiniLM-L6-v2'[26].

Chunking	Size	Overlap	Recall	P
Recursive	250 (~180)	125	78.7 ± 39.2	4
TokenText	250	125	**82.4 ± 36.2**	3
Recursive	250 (~162)	0	78.5 ± 39.5	5
TokenText	250	0	77.1 ± 39.3	3
Recursive	200 (~128)	0	75.7 ± 40.7	6
TokenText	200	0	76.6 ± 38.8	4
✒ Cluster	250 (~168)	0	77.3 ± 38.6	6
✒	200		75.2 ±	**7**

Cluster	(~102)	0	39.9	**6**

Results for all-MiniLM-L6-v2. Size denotes the chunk size in tokens (cl100k tokenizer). Size in brackets indicates mean chunk size where it may vary by chunking strategy. Overlap denotes the chunk overlap in tokens. Bold values highlight the best performance in each category.

We find that the `ClusterSemanticChunker` achieves the highest precision, Precision_Ω and IoU however has slightly below average recall likely due to its small average chunk size. The `TokenTextSplitter` achieves the highest recall of 0.824 when configured with chunk size 250 and chunk overlap of 125. This then drops to 0.771 when chunk overlap is set to zero, suggesting that for smaller context, overlapping chunks are necessary for high recall.

Limitations & Future Work

One limitation in our synthetic evaluation pipeline is that LLMs tend to generate a specific style of question. We attempt to mitigate this by providing it with its previous questions to

encourage new ones, but this sometimes exacerbates its lack of creativity. The metric will be more accurate with more realistic and diverse questions. Future work may experiment with various temperature settings and other prompting techniques.

A limitation of the concrete evaluation we present is its relatively small size, and limited diversity. Future work can easily expand on the evaluation we provide by generating larger datasets from larger corpora. Future work may also explore generating excerpt datasets from benchmarks containing existing queries, which are considered to be diverse and widely representative of many retrieval domains. Future work may also blend human-annotated data with synthetic data to improve the evaluation quality.

Finally, our work does not take into account the time required to run these chunking methods, which can vary from almost instantaneous to tens of minutes in the case of the `LLMChunker`, which is an important consideration in practical retrieval settings.

Conclusion

In this work, we introduced a comprehensive framework for generating domain and dataset-specific evaluations that accurately capture critical properties, such as information density via IoU (Intersection over Union), for retreival as used in practical AI applications. Using this framework, we created a concrete evaluation across several popular domains, enabling robust comparisons of various chunking methods.

Our evaluation of popular chunking strategies demonstrate that this evaluation effectively captures variations in retrieval performance due to the selected chunking strategye. We developed the `ClusterSemanticChunker`, an embedding-model aware chunking strategy which produced consistently strong results across the evaluation. We also present a second novel chunking strategy, the `LLMSemanticChunker`, which also achieves good performance on this evaluation.

References

[1] Jacob Devlin Maarten Bosma Gaurav Mishra Adam Roberts Paul Barham Hyung Won Chung Charles Sutton Sebastian Gehrmann et al. Aakanksha Chowdhery, Sharan Narang. Palm:

Scaling language modeling with pathways. Journal of Machine Learning Research, 24(240):1–113, 2023.

[2] 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. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021

[3] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In Proceedings of the seventh international conference on World Wide Web 7, pages 107–117. Elsevier, 1998.

[4] C.J. van Rijsbergen. A probability ranking principle in information retrieval. Journal of Documentation, 33(4):334– 354, 1977.

[5] S. E. Robertson, S. Walker, M. Beaulieu, M. Gatford, and A. Payne. Okapi at trec-3. In Proceedings of the Third Text REtrieval Conference (TREC-3), pages 109–126. NIST, 1995.

[6] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark, 2022.

[7] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models, 2021.

[8] Weiyun Wang, Shuibo Zhang, Yiming Ren, Yuchen Duan, Tiantong Li, Shuo Liu, Mengkang Hu, Zhe Chen, Kaipeng Zhang, Lewei Lu, Xizhou Zhu, Ping Luo, Yu Qiao, Jifeng Dai, Wenqi Shao, and Wenhai Wang. Needle in a multimodal haystack, 2024.

[9] Marcin Kaszkiel and Justin Zobel. Passage retrieval revisited. In ACM SIGIR Forum, volume 31, pages 178–185. ACM New York, NY, USA, 1997

[10] Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., Rosenberg, M., Song, X., Stoica, A., Tiwary, S., & Wang, T. (2018). MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *arXiv preprint arXiv:1611.09268*. https://arxiv.org/abs/1611.09268

[11] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive data sets. Cambridge university press, 2020.

[12] Matouš Eibich, Shivay Nagpal, and Alexander Fred-Ojala. Aragog: Advanced rag output grading, 2024.

[13] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. Ragas: Automated evaluation of retrieval augmented generation, 2023.

[14] Greg Kamradt. 5 levels of text splitting. https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfText

Splitting.ipynb, 2024. Implementation of Semantic Chunking.

[15] Aurelio AI Labs. Semantic chunkers. https://github.com/aurelio-labs/semantic-chunkers, 2024. Library for semantic text chunking.

[16] Antonio Jimeno Yepes, Yao You, Jan Milczek, Sebastian Laverde, and Renyu Li. Financial report chunking for effective retrieval augmented generation, 2024.

[17] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context, 2023.

[18] Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Fatih Porikli, Sandra Skaff, Alireza Entezari, Jianyuan Min, Daisuke Iwai, Amela Sadagic, Carlos Scheidegger, and Tobias Isenberg, editors, Advances in Visual Computing. ISVC 2016. Lecture Notes in Computer Science, volume 10072, pages 234–244. Springer, Cham, 2016.

[19] The White House. State of the union 2024, 2024. Accessed: 2024-05-02.

[20] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[21] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023.

[22] Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. Convfinqa: Exploring the chain of numerical reasoning in conversational finance question answering. In EMNLP, pages 6279–6292. Association for Computational Linguistics, 2022.

[23] National Library of Medicine. Pmc open access subset. Dataset retrieved from Hugging Face Datasets (2023). PubMed Central Open Access dataset (Version 2023-06-17.commercial). Available from https://huggingface.co/datasets/pmc/open_access 2003–. [cited 2024 May 08].

[24] LangChain. Recursive text splitter, 2023. Accessed: 2024-04-16.

[25] OpenAI. Vector stores. https://platform.openai.com/docs/assistants/tools/search/ vector-stores, 2024. Accessed: 2024-06-27.

[26] HuggingFace. Sentence Transformers, all-MiniLM-L6-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2 2024. Accessed: 2024-06-27.

[27] Kristian Georgiev Aleksander Madry

Benjamin Cohen-Wang, Harshay Shah. Contextcite: Attributing model generation to context. https://github.com/MadryLab/context-cite/tree/main, 2024.

Appendices

Synthetic Dataset Prompt

```
You are an agent that generates questions from provided text. Your job is to generate a question and provide the relevant sections from the text as references.

Instructions:
1. For each provided text, generate a question that can be answered solely by the facts in the text.
2. Extract all significant facts that answer the generated question.
3. Format the response in JSON format with two fields:
   - 'question': A question directly related to these facts, ensuring it can only be answered using the references provided.
   - 'references': A list of all text sections that answer the generated
```

question. These must be exact copies from the original text and should be whole sentences where possible.

Notes: Make the question more specific. Do not ask a question about multiple topics. Do not ask a question with over 5 references.

Example:

Text: "Experiment A: The temperature control test showed that at higher temperatures, the reaction rate increased significantly, resulting in quicker product formation. However, at extremely high temperatures, the reaction yield decreased due to the degradation of reactants.

Experiment B: The pH sensitivity test revealed that the reaction is highly dependent on acidity, with optimal results at a pH of 7. Deviating from this pH level in either direction led to a substantial drop in yield.

Experiment C: In the enzyme activity assay, it was found that the presence of a specific enzyme accelerated the reaction by a factor of 3. The absence of the enzyme, however, led to a sluggish reaction with an extended completion time.

Experiment D: The light exposure trial demonstrated that UV light stimulated the reaction, making it complete in half the time compared to the absence of light. Conversely, prolonged light exposure led to unwanted side reactions that contaminated the final product."

Response: { 'oath': "I will not use the word 'and' in the question unless it is

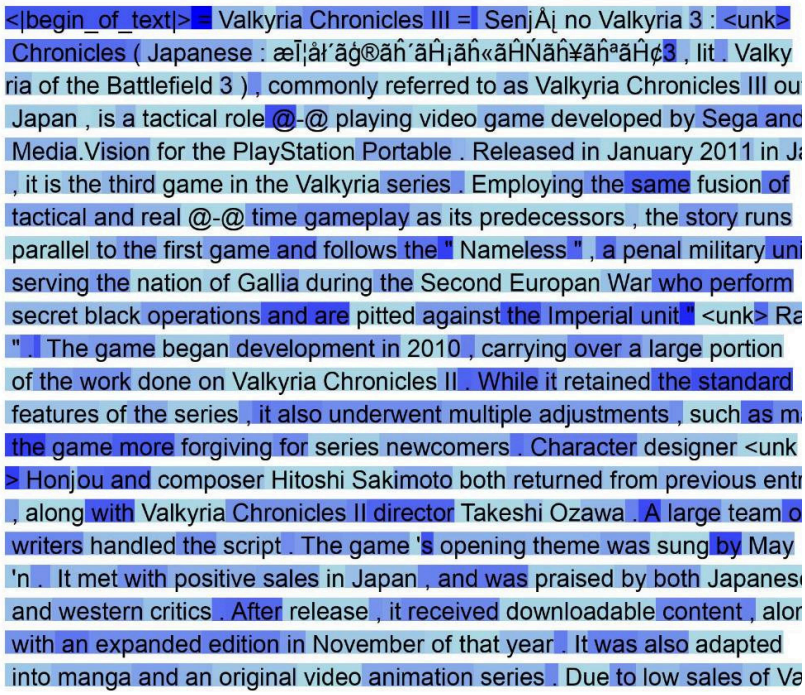
```
part of a proper noun. I will also make
sure the question is concise.",
'question': 'What experiments were done in
this paper?', 'references': ['Experiment
A: The temperature control test showed
that at higher temperatures, the reaction
rate increased significantly, resulting in
quicker product formation.', 'Experiment
B: The pH sensitivity test revealed that
the reaction is highly dependent on
acidity, with optimal results at a pH of
7.', 'Experiment C: In the enzyme activity
assay, it was found that the presence of a
specific enzyme accelerated the reaction
by a factor of 3.', 'Experiment D: The
light exposure trial demonstrated that UV
light stimulated the reaction, making it
complete in half the time compared to the
absence of light.'] }

DO NOT USE THE WORD 'and' IN THE QUESTION
UNLESS IT IS PART OF A PROPER NOUN. YOU
MUST INCLUDE THE OATH ABOVE IN YOUR
RESPONSE. YOU MUST ALSO NOT REPEAT A
QUESTION THAT HAS ALREADY BEEN USED."
```

An interesting observation from the above is the inclusion of an oath. We require these queries to be specific so that the excerpts answer only that query. This generally means the query shouldn't include 'and' unless it is part of a proper noun. Despite this requirement being part of the prompt at many points, GPT would not comply until we required it to include an 'oath': "I will not use the word 'and' in the question unless it is part of a proper noun. I will also make sure the question is concise." in the JSON response. It then consistently included the oath and followed our requirements.

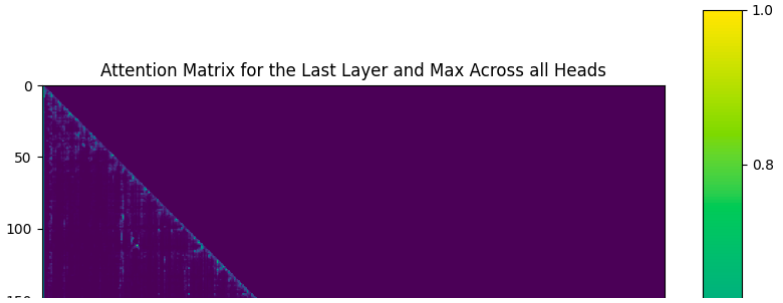
Further Algorithms

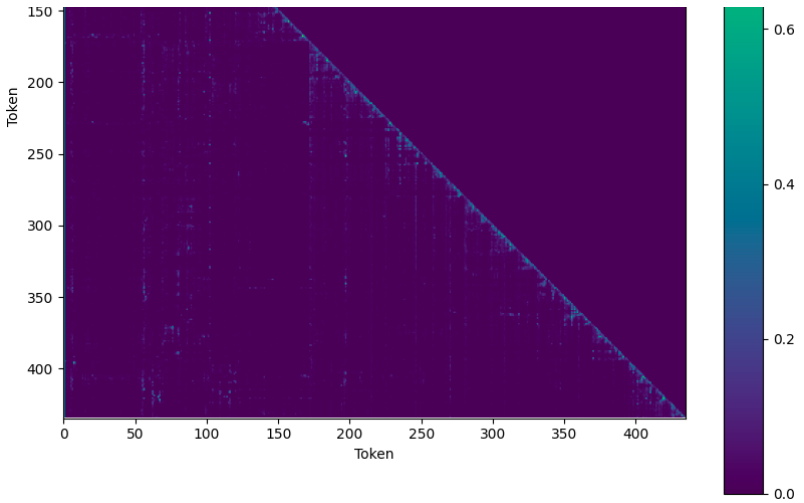
Logit & Attention Based Chunking

We experimented with next token prediction probabilities and attention values to determine if any signals could be used for chunking. All these experiments were conducted using Llama 3 8B, which provides access to next token logits and attention values.



Token entropy visualization over Wikitext corpus using Llama 3 8B. The color gradient from light blue (under 1 bit) to dark blue (8 bits and over) indicates the entropy in the next token prediction at that token.





Heatmap of attention values from the Llama 3 8B model on the wikitext corpus. This plot shows the max attention values in the last layer across all 32 heads, with x and y axes representing token positions.

We calculate token entropy according to:

$$H(Y) = -\sum_{t} P(y_t) \log P(y_t)$$

Where Y is a random variable representing the next token, y_t is the t-th possible token and $P(y_t)$ is the probability assigned to token (y_t). We observe that in general, token entropy rises at the beginning and towards the end of sentences, as well as after a verb after which any noun or adjective could be expected. We initially expected that the entropy would rise as the theme of the text changes, however, we find that the LLM quickly refocuses between topics and so we did not extract a clear signal for semantic boundaries.

We also did not find that there was any clear structure in the attention values which could be used for chunking.

ContextCite Inspired Chunking

The final algorithm was inspired by ContextCite [27]. Their work leverages next-token prediction to observe how the probability of an LLM generating a response changes as different parts of the input prompt are masked. This allows them to fit weights to each part of the input prompt, determining how much it influences the likelihood of generating a specific response. Essentially, this results in a weighted importance of each part of the input prompt relative to the response.

For our work, we split the Wikitext corpus into 50-token chunks using a `RecursiveCharacterTextSplitter` and applied the same algorithm, but with each chunk as the 'response' and all previous chunks as the input prompt. This allowed us to fit weights between each chunk. Unfortunately, no clear structure emerged, and we were unable to derive a useful chunking algorithm.

All Results

We present the mean of Recall, Precision, $\text{Precision}_{\Omega}$, and IoU scores for each chunking method, over all queries and corpora in our evaluation. ± indicates the standard deviation. We vary the number of chunks retrieved: 'Min' retrieves the minimum number of chunks containing excerpts (usually 1-3), while 5 and 10 represent fixed retrieval numbers for all queries. Captions indicate corpus and

embedding model.

Chunking	Size	Overlap	Retrieve	R
Recursive	800	400	Min	6 4
TokenText	800	400	Min	7 4
Recursive	400	200	Min	7 4
TokenText	400	200	Min	**7 ± 3**
Recursive	400	0	Min	6 4
TokenText	400	0	Min	5 4
Recursive	200	0	Min	7 4
TokenText	200	0	Min	6 4
Cluster	400	0	Min	6 4

Cluster	200	0	Min	64
LLM	N/A	0	Min	64
Recursive	800	400	5	83
TokenText	800	400	5	83
Recursive	400	200	5	83
TokenText	400	200	5	83
Recursive	400	0	5	82
TokenText	400	0	5	82
Recursive	200	0	5	82
TokenText	200	0	5	8

				3
Cluster	400	0	5	**9 ± 2**
Cluster	200	0	5	8 2
LLM	N/A	0	5	9 2
Recursive	800	400	10	9 2
TokenText	800	400	10	9 2
Recursive	400	200	10	9 2
TokenText	400	200	10	9 2
Recursive	400	0	10	9 2
TokenText	400	0	10	9 2

Recursive	200	0	10	9 2
TokenText	200	0	10	9 2
Cluster	400	0	10	**9 ± 1**
Cluster	200	0	10	9 2
LLM	N/A	0	10	9 1

All corpora, text-embedding-3-large

Chunking	Size	Overlap	Retrieve	R
Recursive	800	400	Min	8 3
TokenText	800	400	Min	8 2
Recursive	400	200	Min	**9 ± 2**
				8

TokenText	400	200	Min	3
Recursive	400	0	Min	7 4
TokenText	400	0	Min	6 4
Recursive	200	0	Min	8 3
TokenText	200	0	Min	7 3
Cluster	400	0	Min	7 3
Cluster	200	0	Min	8 3
LLM	N/A	0	Min	7 4
Recursive	800	400	5	9 2
TokenText	800	400	5	9 2

Recursive	400	200	5	9 2
TokenText	400	200	5	8 3
Recursive	400	0	5	**1** **±**
TokenText	400	0	5	9 2
Recursive	200	0	5	9 2
TokenText	200	0	5	8 3
Cluster	400	0	5	1 ±
Cluster	200	0	5	9 3
LLM	N/A	0	5	9 1
Recursive	800	400	10	9 2

TokenText	800	400	10	9 9
Recursive	400	200	10	**1** **±**
TokenText	400	200	10	8 3
Recursive	400	0	10	1 ±
TokenText	400	0	10	9 1
Recursive	200	0	10	1 ±
TokenText	200	0	10	8 3
Cluster	400	0	10	1 ±
Cluster	200	0	10	9 0
LLM	N/A	0	10	1 ±

Chatlogs corpus, text-embedding-3-large

Chunking	Size	Overlap	Retrieve	R
Recursive	800	400	Min	5 4
TokenText	800	400	Min	6 4
Recursive	400	200	Min	6 4
TokenText	400	200	Min	**7 ± 4**
Recursive	400	0	Min	5 4
TokenText	400	0	Min	5 4
Recursive	200	0	Min	6 4
TokenText	200	0	Min	6 4
Cluster	400	0	Min	5 4

Cluster	200	0	Min	6 4
LLM	N/A	0	Min	4 4
Recursive	800	400	5	7 4
TokenText	800	400	5	7 4
Recursive	400	200	5	8 3
TokenText	400	200	5	**8 ± 3**
Recursive	400	0	5	8 3
TokenText	400	0	5	8 3
Recursive	200	0	5	7 3
TokenText	200	0	5	8

				3
Cluster	400	0	5	83
Cluster	200	0	5	83
LLM	N/A	0	5	83
Recursive	800	400	10	83
TokenText	800	400	10	83
Recursive	400	200	10	92
TokenText	400	200	10	92
Recursive	400	0	10	92
TokenText	400	0	10	83

Recursive	200	0	10	83
TokenText	200	0	10	92
Cluster	400	0	10	**91**
Cluster	200	0	10	83
LLM	N/A	0	10	92

Finance corpus, text-embedding-3-large

Chunking	Size	Overlap	Retrieve	R
Recursive	800	400	Min	54
TokenText	800	400	Min	64
Recursive	400	200	Min	54
				6

TokenText	400	200	Min	**± 4**
Recursive	400	0	Min	5 4
TokenText	400	0	Min	4 4
Recursive	200	0	Min	5 4
TokenText	200	0	Min	4 4
Cluster	400	0	Min	5 4
Cluster	200	0	Min	6 4
LLM	N/A	0	Min	5 4
Recursive	800	400	5	8 3
TokenText	800	400	5	8 3

Recursive	400	200	5	8 3
TokenText	400	200	5	8 3
Recursive	400	0	5	8 3
TokenText	400	0	5	7 3
Recursive	200	0	5	8 3
TokenText	200	0	5	7 3
Cluster	400	0	5	8 3
Cluster	200	0	5	8 3
LLM	N/A	0	5	**8 ± 3**
Recursive	800	400	10	8 3

TokenText	800	400	10	9 2
Recursive	400	200	10	8 2
TokenText	400	200	10	8 2
Recursive	400	0	10	8 2
TokenText	400	0	10	9 2
Recursive	200	0	10	8 2
TokenText	200	0	10	9 2
Cluster	400	0	10	8 2
Cluster	200	0	10	9 2
LLM	N/A	0	10	**9 ± 1**

Pubmed corpus, text-embedding-3-large

Chunking	Size	Overlap	Retrieve	R
Recursive	800	400	Min	8 3
TokenText	800	400	Min	8 3
Recursive	400	200	Min	**9 ± 2**
TokenText	400	200	Min	8 3
Recursive	400	0	Min	6 4
TokenText	400	0	Min	7 4
Recursive	200	0	Min	8 3
TokenText	200	0	Min	8 3
Cluster	400	0	Min	8 3

Cluster	200	0	Min	83
LLM	N/A	0	Min	83
Recursive	800	400	5	92
TokenText	800	400	5	91
Recursive	400	200	5	91
TokenText	400	200	5	92
Recursive	400	0	5	92
TokenText	400	0	5	**9 ± 1**
Recursive	200	0	5	91
TokenText	200	0	5	91

Cluster	400	0	5	9 1
Cluster	200	0	5	9 1
LLM	N/A	0	5	9 1
Recursive	800	400	10	9 2
TokenText	800	400	10	9 1
Recursive	400	200	10	9 1
TokenText	400	200	10	9 1
Recursive	400	0	10	9 1
TokenText	400	0	10	**1** **±**

Recursive	200	0	10	9 1
TokenText	200	0	10	9 1
Cluster	400	0	10	9 1
Cluster	200	0	10	9 1
LLM	N/A	0	10	9 1

State of the Union corpus, text-embedding-3-large

Chunking	Size	Overlap	Retrieve	R
Recursive	800	400	Min	7 4
TokenText	800	400	Min	**7 ± 4**
Recursive	400	200	Min	7 4
TokenText	400	200	Min	7

				4
Recursive	400	0	Min	64
TokenText	400	0	Min	54
Recursive	200	0	Min	74
TokenText	200	0	Min	64
Cluster	400	0	Min	74
Cluster	200	0	Min	64
LLM	N/A	0	Min	64
Recursive	800	400	5	83
TokenText	800	400	5	92

Recursive	400	200	5	8 3
TokenText	400	200	5	8 3
Recursive	400	0	5	9 2
TokenText	400	0	5	9 1
Recursive	200	0	5	9 2
TokenText	200	0	5	9 2
Cluster	400	0	5	**9 ± 1**
Cluster	200	0	5	9 2
LLM	N/A	0	5	9 2
Recursive	800	400	10	9 1

TokenText	800	400	10	9 1
Recursive	400	200	10	9 2
TokenText	400	200	10	9 2
Recursive	400	0	10	9 1
TokenText	400	0	10	**9** **±**
Recursive	200	0	10	9 2
TokenText	200	0	10	9 1
Cluster	400	0	10	9 9
Cluster	200	0	10	9 2
LLM	N/A	0	10	9 2

Wikitext corpus, text-embedding-3-large

Chunking	Size	Overlap	Retrieve	R
Recursive	250	125	Min	6 4
TokenText	250	125	Min	**7 ± 4**
Recursive	250	0	Min	5 4
TokenText	250	0	Min	5 4
Recursive	200	0	Min	4 4
TokenText	200	0	Min	5 4
Cluster	250	0	Min	5 4
Cluster	200	0	Min	5 4
Recursive	250	125	5	7

				3
TokenText	250	125	5	**8 ± 3**
Recursive	250	0	5	7 3
TokenText	250	0	5	7 3
Recursive	200	0	5	7 4
TokenText	200	0	5	7 3
Cluster	250	0	5	7 3
Cluster	200	0	5	7 3
Recursive	250	125	10	8 3
TokenText	250	125	10	**8 ± 3**

Recursive	250	0	10	8 3
TokenText	250	0	10	8 3
Recursive	200	0	10	8 3
TokenText	200	0	10	8 3
Cluster	250	0	10	8 3
Cluster	200	0	10	8 3

All corpora, all-MiniLM-L6-v2

Chunking	Size	Overlap	Retrieve	R
Recursive	250	125	Min	9 2
TokenText	250	125	Min	**9 ± 2**
Recursive	250	0	Min	7

				4
TokenText	250	0	Min	73
Recursive	200	0	Min	64
TokenText	200	0	Min	73
Cluster	250	0	Min	73
Cluster	200	0	Min	64
Recursive	250	125	5	91
TokenText	250	125	5	**1 ±**
Recursive	250	0	5	91
TokenText	250	0	5	91

Recursive	200	0	5	9 1
TokenText	200	0	5	9 7
Cluster	250	0	5	9 1
Cluster	200	0	5	9 2
Recursive	250	125	10	9 1
TokenText	250	125	10	**1** **±**
Recursive	250	0	10	9 1
TokenText	250	0	10	9 1
Recursive	200	0	10	1 ±
TokenText	200	0	10	1 ±

Cluster	250	0	10	9 4
Cluster	200	0	10	9 9

Chatlogs corpus, all-MiniLM-L6-v2

Chunking	Size	Overlap	Retrieve	R
Recursive	250	125	Min	5 4
TokenText	250	125	Min	**7 ± 4**
Recursive	250	0	Min	5 4
TokenText	250	0	Min	5 4
Recursive	200	0	Min	4 4
TokenText	200	0	Min	5 4
				5

Cluster	250	0	Min	4
Cluster	200	0	Min	5 4
Recursive	250	125	5	6 4
TokenText	250	125	5	**7 ± 3**
Recursive	250	0	5	6 4
TokenText	250	0	5	7 4
Recursive	200	0	5	7 4
TokenText	200	0	5	7 4
Cluster	250	0	5	7 4
Cluster	200	0	5	6 4

Recursive	250	125	10	7 4
TokenText	250	125	10	**8 ± 3**
Recursive	250	0	10	7 4
TokenText	250	0	10	7 4
Recursive	200	0	10	7 3
TokenText	200	0	10	7 3
Cluster	250	0	10	7 3
Cluster	200	0	10	7 3

Finance corpus, all-MiniLM-L6-v2

Chunking	Size	Overlap	Retrieve	R

Recursive	250	125	Min	4 4
TokenText	250	125	Min	**6 ± 4**
Recursive	250	0	Min	3 4
TokenText	250	0	Min	4 4
Recursive	200	0	Min	3 4
TokenText	200	0	Min	3 4
Cluster	250	0	Min	4 4
Cluster	200	0	Min	3 4
Recursive	250	125	5	6 4
TokenText	250	125	5	**6 ± 4**

Recursive	250	0	5	64
TokenText	250	0	5	64
Recursive	200	0	5	64
TokenText	200	0	5	64
Cluster	250	0	5	64
Cluster	200	0	5	54
Recursive	250	125	10	74
TokenText	250	125	10	**74 ± 4**
Recursive	250	0	10	64
TokenText	250	0	10	74

Recursive	200	0	10	64
TokenText	200	0	10	73
Cluster	250	0	10	74
Cluster	200	0	10	73

Pubmed corpus, all-MiniLM-L6-v2

Chunking	Size	Overlap	Retrieve	R
Recursive	250	125	Min	74
TokenText	250	125	Min	**7 ± 4**
Recursive	250	0	Min	64
TokenText	250	0	Min	54

Recursive	200	0	Min	54
TokenText	200	0	Min	54
Cluster	250	0	Min	64
Cluster	200	0	Min	54
Recursive	250	125	5	**92**
TokenText	250	125	5	83
Recursive	250	0	5	83
TokenText	250	0	5	83
Recursive	200	0	5	83
TokenText	200	0	5	83

Cluster	250	0	5	83
Cluster	200	0	5	83
Recursive	250	125	10	**91**
TokenText	250	125	10	92
Recursive	250	0	10	83
TokenText	250	0	10	92
Recursive	200	0	10	83
TokenText	200	0	10	82
Cluster	250	0	10	92
Cluster	200	0	10	82

State of the Union corpus, all-MiniLM-L6-v2

Chunking	Size	Overlap	Retrieve	R
Recursive	250	125	Min	5 4
TokenText	250	125	Min	**7 ± 4**
Recursive	250	0	Min	5 4
TokenText	250	0	Min	4 4
Recursive	200	0	Min	5 4
TokenText	200	0	Min	5 4
Cluster	250	0	Min	5 4
Cluster	200	0	Min	5 4

Recursive	250	125	5	83
TokenText	250	125	5	83
Recursive	250	0	5	**83**
TokenText	250	0	5	73
Recursive	200	0	5	74
TokenText	200	0	5	73
Cluster	250	0	5	73
Cluster	200	0	5	73
Recursive	250	125	10	92
TokenText	250	125	10	82

Recursive	250	0	10	9 2
TokenText	250	0	10	**9 ± 2**
Recursive	200	0	10	8 2
TokenText	200	0	10	9 2
Cluster	250	0	10	8 2
Cluster	200	0	10	8 2

Wikitext, all-MiniLM-L6-v2

An example query and set of excerpts generated from the Wikitexts corpus, one of the 474 queries in our generated dataset, sampled from GPT-4-Turbo in JSON mode. Excerpts are exact matches from the Wikitexts corpus.

Try Chroma Cloud

Chroma is the open-source search a
retrieval database for AI
applications.

About

Logos

Careers

Contact Us

Privacy

Terms of Use