Multi-view Content-aware Indexing for Long Document Retrieval

Anonymous ACL submission

Abstract

Long document question answering (DocQA) aims to answer questions from long documents over 10k words. They usually contain *content structures* such as sections, sub-sections, and paragraph demarcations. However, the indexing methods of long documents remain under-explored, while existing systems generally employ fixed-length chunking. As they *do not consider content structures*, the resultant chunks can exclude vital information or include irrelevant content. Motivated by this, we propose the **M**ulti-view **C**ontent-aware **indexing** (**MC-indexing**) for more effective long DocQA via (i) segment structured document into content chunks, and (ii) represent each content chunk in raw-text, keywords, and summary views. We highlight that MC-indexing *requires neither training nor fine-tuning*. Having plug-and-play capability, it can be seamlessly integrated with any retrievers to boost their performance. Besides, we propose a long DocQA dataset that includes not only question-answer pair, but also *document structure* and *answer scope*. When compared to state-of-art chunking schemes, MC-indexing has significantly increased the recall by **42.8%**, **30.0%**, **23.9%**, and **16.3%** via top $k = 1.5$, 3, 5, and 10 respectively. These improved scores are the average of 8 widely used retrievers (2 sparse and 6 dense) via extensive experiments.[1]

Question (a): HOW TO BAKE A CHOCOLATE CAKE?

Desired Reference Text: You can bake a chocolate cake by following procedures: 1.Preparation: ... 2.Gather Ingredients: ... 3.Dry Ingredients Mixture: ... 4.Wet Ingredients Mixture: ... 5.Combine Mixtures: ... 6.Bake the Cake: ... (500 words)

Actual Chunks Retrieved: ... You can bake a chocolate cake by following procedures: 1.Preparation: ... (100 words)

(a) The whole section (approx. 500 words) is required to answer the question. The retrieved chunk only has 100 words.

Question (b): WHAT IS THE HARDWARE SPECIFICATIONS (CPU, DISPLAY, BATTERY, ETC) OF DELL XPS 13?

Desired Reference Text: ... 11th Gen Intel Core i7 processor ... a 13.4-inch FHD InfinityEdge display ... battery life ... backlit keyboard ... with Thunderbolt 4 ports ... (250 words)

Actual Chunks Retrieved:
1. ... an 11th Gen Intel Core i7 processor ... 13.4-inch FHD InfinityEdge display ... (Content: **Dell XPS 13**, 100 words)
2. ... new M1 Pro chip ... 14-inch Liquid Retina XDR display showcases ... (Content: **MacBook Pro**, 100 words)
3. ... a powerful Intel Core M processor ... 13.3-inch 4K UHD touch display ... (Content: **Dell XPS 12**, 100 words)

(b) The whole section (approx. 250 words) is required to answer the given question related to Dell XPS 13. Missing information (e.g, model name) leads to conflicting information.

Figure 1: Bad cases from fixed-length chunking due to relevant text missing and inclusion of irrelevant text.

1 Introduction

Document question answering (DocQA) is a pivotal task in natural language processing (NLP) that involves responding to questions using textual documents as the reference answer scope. Conventional DocQA systems comprise three key components: (i) an indexer that segments the document into manageable text chunks indexed with embeddings, (ii) a retriever that identifies and fetches the most relevant chunks to the corresponding question, and (iii) a reader that digests the retrieved answer scope and generates an accurate answer. Unlike the retriever (Robertson and Zaragoza, 2009; Karpukhin et al., 2020; Khattab and Zaharia, 2020a) and reader (Nie et al., 2019; Lewis et al., 2020; Izacard and Grave, 2021) that are vastly studied, the indexer received relatively less attention.

Existing indexing schemes *overlook the importance of content structures* when dealing with long documents, as they are usually organized into chapters, sections, subsections, and paragraphs (Yang et al., 2020; Buchmann et al., 2024), *i.e.,* structured. The widely used fixed-length chunking strategy can easily break the contextual relevance between text chunks for long documents. Such chunking errors can be further aggravated by the retriever and the reader. Moreover, determining the boundary

[1]We will release dataset and code upon paper acceptance.

between chunks can be tricky, requiring delicate design to prevent contextual coherence disruption. Ideally, each chunk should represent a coherent and content-relevant textual span. Otherwise, it can lead to the exclusion of relevant information or the inclusion of irrelevant text, as exemplified in Figure 1. Our empirical study on fixed-length chunking reveals that setting the chunk length to 100 results in over 70% of long answers/supporting evidence being truncated, *i.e.,* incomplete. Such incompleteness still exists at 45%, despite an increase of chunk length to 200.[2]

Meanwhile, most existing retrieval systems *rely solely on the raw text of chunks to determine relevance to a query*. While raw-text-based semantic embeddings effectively address queries seeking specific short-form details, they often fail to capture complete semantic essence of the text. When inquiring high-level information, such as event summaries or comparisons, raw-text embeddings may fall short. Additionally, reliance on raw text poses practical constraints, as models *e.g.,* DPR (Karpukhin et al., 2020), E5 (Wang et al., 2022), BGE (Xiao et al., 2023) based on BERT (Devlin et al., 2019) typically have a token limit of 512. This leads to potential truncation and loss of information during the indexing process. Zhang et al. (2022) attempt to embed the entire document with multiple representations, however, these embeddings are not applicable to individual chunks.

To mitigate aforementioned gaps, we present **M**ulti-view **C**ontent-aware **Indexing**, termed **MC-indexing**, for more effective retrieval over long documents. Our method involves content-aware chunking of structured long documents, whereby, instead of employing naïve fixed-length chunking, the document is segmented into section chunks. The content-aware chunking can effective eliminate chunking errors. Each of these section chunks is then indexed in three different views, representing each chunk with raw-text, a list of keywords, and a summary. The keyword and summary view can provide richer but more concise representation of section chunks, thereby significantly enhancing the semantic richness of each chunk. For retrieval, we aggregate the top relevant chunks from each view. Note that the entire process of MC-indexing is unsupervised. We leverage on the strength of existing retrievers for the embedding generation of raw-text, keyword, and summary views.

[2]More statistics of chunking errors are in Appendix A.

To our best knowledge, existing DocQA datasets do not provide content structure. Hence, we transform an existing long documents dataset, namely WikiWeb2M (Burns et al., 2023), into a QA dataset, by adding annotations to the documents. In addition, we complement Natural Questions dataset (Kwiatkowski et al., 2019) with content structure, and filter only long documents for our experiment. Distinct from other QA datasets, our documents are longer (averaging at 15k tokens) and contain detailed content structure. Our contributions are in fourfold:

- We propose a long document QA dataset annotated with question-answer pair, document content structure, and scope of answer.
- We propose **M**ulti-view **C**ontent-aware **indexing** (**MC-indexing**), that can (i) segment the long documents according to their content structures, and (ii) represent each chunk in three views, *i.e.,* raw-text, keywords, and summary.
- MC-indexing requires neither training nor fine-tuning, and can seamlessly act as a plug-and-play indexer to enhance any existing retrievers.
- Through extensive experiments and analysis, we demonstrate that MC-indexing can significantly improve retrieval performance of **eight** commonly-used retrievers (2 sparse and 6 dense) on two long DocQA datasets.

2 Related Work

Chunking Methods. Chunking is a crucial step in either QA or Retrieval-Augmented Generation (RAG). When dealing with ultra-long text documents, chunk optimization involves breaking the document into smaller chunks. Existing systems focus on how to retrieve relevant chunks, but neglecting how text content is chunked. In practice, fixed-length chunking is a commonly used method that is easy to be implemented. It chunks text at a fixed length, *e.g.,* 200 words. Sentence chunking involves dividing textual content based on sentences. Recursive chunking employs various delimiters, such as paragraph separators, newline characters, or spaces, to recursively segment the text. Raina and Gales (2024) propose to represent each chunk as a set of atomic pieces of information. However, these methods often fail to preserve semantic integrity of critical content. In contrast, content-aware chunking (Section 3.2) chunk the text by the smallest subdivision according to the document's content structure. This ensures each chunk to be semantically coherent, thus reducing chunking error.

Long Document Question Answering. Traditional retrieval methods such as BM25 and DPR only retrieve short consecutive chunks from the retrieval corpus, limiting the overall understanding of the context of long documents. To overcome this drawback, several methods focusing on long document retrieval have been proposed. Nie et al. (2022) propose a compressive graph selector network to select question-related chunks from the long document and then use the selected short chunks for answer generation. AttenWalker (Nie et al., 2023) addresses the task of incorporating long-range information by employing a meticulously crafted answer generator. Chen et al. (2023) convert the long document into a tree of summary nodes. Upon receiving a question, LLM navigates this tree to find relevant summaries until sufficient information is gathered. Sarthi et al. (2024) utilize recursive embedding, clustering, and summarizing chunks of text to build a tree with different levels of summarization. However, existing methods only consider the retrieval of long documents from one view, limiting the semantic completeness and coherence.

3 Methodology

3.1 Overview of MC-indexing

As shown in Figure 2b, MC-indexing consists of two stages. **(1) Indexing:** given a input document, we first chunk the document into *content-aware chunks* (Section 3.2). We then represent each section chunks with three distinct views: raw-text, keywords, and summary view (Section 3.3). **(2) Retrieval and Question Answering:** Given a user query, we use existing retriever to fetch top-k relevant chunks constructed by our MC-indexing. The query along with retrieved results are fed into LLM to generate the final answer.

3.2 Content-aware Chunking

We elaborate how Content-Aware chunking is performed in order to obtain *section chunks*. Given a piece of structured document (*e.g.,* Markdown, Latex, and HTML), we first extract the table of contents of the document (or header information, in the event where the table of content is not readily available). Upon acquiring this information, we identify the smallest division in the document, such as a section, subsection, or sub-subsection, depending on the structure of the content. It is reasonable to assume that these smallest divisions function as atomic, coherent semantic units within the document. The text present in each smallest division is the desired *section chunk*.

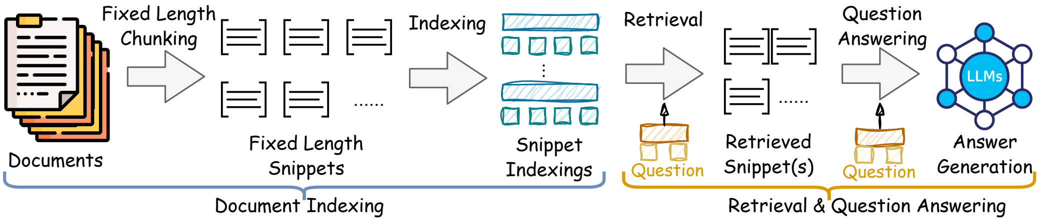
Chunking text based on the smallest division, as opposed to fixed length chunking, ensures that information in each chunk cannot contain information across two different sections. Most importantly, we preserve the semantic integrity during the chunking process, leading to each section chunk to be an atomic and coherent semantic unit. Note that different sections may have a hierarchical relationship between them. We ignore them for now and assume a flat structure between different chunks.

3.3 Multi-View Indexing and Retrieval

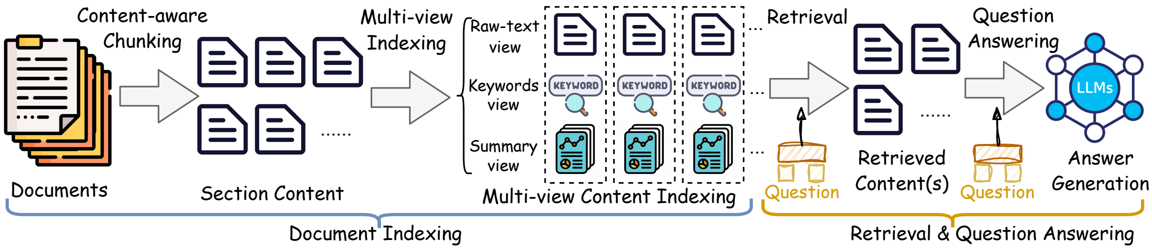
Most dense retrieval methods primarily use raw text from each chunk to determine the relevancy of each chunk with respect to a given query. However, raw-text alone may not fully represent the semantic meaning of each chunk. Hence, we propose using the ***summary*** view and the ***keyword*** view for richer but more concise representation of section chunks.

The ***summary*** view represents each section chunk with a succinct summary. It captures the key information of each section. The summary can be more easily fits within the dense retrieval model's maximum input limit. To compensate for the potential omission of critical details in the generated summaries, we introduce a ***keyword*** view. This view characterizes each section chunk by a list of essential keywords, including significant concepts, entities, and terms from the section. The detailed generation process of summary and keywords are discussed in Section 5.7.

Finally, we describe the procedure for utilizing multi-view indexing to retrieve top-k relevant sections with respect to a given question. For each of the views, *e.g.,* raw-text, summary, keywords, we simply rank the sections using each view to first retrieve the top-$k^{'}$ results. Setting $k^{'} \approx 2k/3$ works since empirically we expect on average a total of $3k^{'}/2$ unique results after deduplication (see more details in Appendix E). Thereafter we feed the retrieved results along with the given question to LLM for answer generation (see Figure 10 for prompt details). Note that MC-indexing is independent of retriever selection. MC-indexing can utilize the strengths of any existing retrievers, and further improve their retrieval performance. Moreover, as a plug-and-play boost for retrievers, MC-indexing requires no additional training or fine-tuning to integrate effectively.



(a) Conventional DocQA system: document → fixed length snippets → retrieved snippets → answer



(b) MC-indexing: document → section content → multi-view content indexing → retrieved sections → answer

Figure 2: Comparison between conventional fixed length chunking and our proposed MC-indexing.

4 Dataset Construction

In our work, we focus on long and structured document, thus we collect dataset corpus based on the following two factors. **(1) Presence of structured information:** The content of long documents is usually divided into multiple sections. For example, a research paper is organized into various sections such as Abstract, Introduction, Methodology and Conclusion. Structured documents have explicitly labelled sections along their corresponding text. Most of the existing QA datasets (*e.g.,* SQuAD (Rajpurkar et al., 2016), TriviaQA (Joshi et al., 2017), Ms Macro (Bajaj et al., 2018)) do not include the content structure of source documents. Due to the absence of structure information, they are not considered in our work. **(2) Sufficiently Long Document:** The main focus of our study is on context retrieval in long documents. Short documents, being within the LLM's capacity, do not necessitate a structured layout for question answering. Hence, to ensure the challenge of our dataset, we select only documents with at least 15k words.

According to these criteria, we select Wikipedia Webpage 2M (WikiWeb2M) (Burns et al., 2023) and Natural Questions (NQ) (Kwiatkowski et al., 2019) datasets. We discuss dataset processing and annotations on these datasets in finer detail.

4.1 Wikipedia Webpage 2M (WikiWeb2M)

WikiWeb2M is designed for multimodal webpage understanding rather than QA. The dataset stores individual sections within each Wikipedia article. Thus, on top of the structured information, we annotate additional question-answer pairs and their answer scope. We utilize GPT-4 to construct questions for selected articles (over 10k tokens) in WikiWeb2M. To ensure that the questions rely on long answer scope span, we define the 8 types of questions.[3] For each section given, we request GPT-4 (using prompt shown in Figure 6) to generate (i) three questions, (ii) the corresponding answers to the each question, and (iii) the answer scope for each answer. We then evaluate the retrieval efficiency and answer quality of MC-indexing by utilizing the constructed data.

Using this approach we have generated questions for 83,625 sections from 3,365 documents. For evaluation, in order to demonstrate the effectiveness of our method in long DocQA, we only use questions generated from documents with 28k to 30k tokens, resulting in 30 documents for evaluation. The remaining questions not used in evaluation are intended for training / fine-tuning.

4.2 Natural Questions (NQ)

The NQ dataset provides rendered HTML of Wikipedia articles alongside the questions and answer scope. By parsing the rendered HTML, we are able to extract the section name and the corresponding texts in each section of the document. We augment the NQ dataset with our extracted structured information. We omit sections such as 'See Also', 'Notes', and 'References', which refer as references for the main content, to reduce noise during retrieval. We follow NQ's train/test split setting in our work. However, we only retain the

[3] Refer to Appendix B.1 for more details about the type, definition, and statistics of question annotations.

Statistics	NQ		WikiWeb2M	
	Test	Train	Test	Train
questions	586	36.8k	3027	82.6k
sections/doc	34.1	33.2	75.0	42.7
tokens/doc	17.4k	17.4k	28.1k	15.2k
tokens/sec	510	525	375	356
tokens/ans	827	581	109	104

Table 1: Document statistics for NQ and WikiWeb2M.

question whose corresponding document has more than 10k tokens. For dev set, there exists multiple annotations. We only retain questions where all annotations reside within the same section. After filtering, we obtain 36,829 and 586 question-article pairs for train/test respectively. Again, we emphasise that our approach does not require fine-tuning and solely utilises the test-set.

5 Experiment

5.1 Baseline Systems

Chunking and Indexing. Our experiment consists of 5 chunking/indexing methods as follows: **(i)** Fixed-length chunking (FLC), **(ii)** Recursive Fixed-length chunking, known as RAPTOR (Sarthi et al., 2024), **(iii)** Atomic chunking (Raina and Gales, 2024), **(iv)** Content-aware chunking, and **(v)** our proposed MC-indexing. Refer to Appendix C for more implementation details.

Retrieval. We apply MC-indexing and baselines on 2 sparse (TF-IDF and BM25) and 6 dense (DPR, ColBERT, Contriever, E5, BGE, and GTE) retrievers. The description and implementation details of these retrievers are written on Appendix D.

5.2 Evaluation Metrics

We evaluate the performance of MC-indexing and other baselines based on (i) recall of retrieval and (ii) quality of answer generation.

Recall of Retrieval. The retriever scores each chunk in the document based on its relevance to the question, and returns the top k chunks with the highest scores. We define recall as the proportion of the ground truth answer scope that is successfully retrieved by retriever. For instance, if each of three retrieved chunks overlaps with 10%, 50% and 0% of the ground truth answer scope, the recall is the sum of all individual scores to be 0.6. The recall gives us a clear indication of how effective our chunking strategy has boosted the retriever.

Answer Generation. As the final goal of DocQA is to generate accurate answer, it is essential for us to evaluate the quality of final answer based on retrieved chunks. We evaluate the answers via pairwise evaluation using GPT-4 as evaluator. Specifically, we provide prompt for GPT-4 (see Figure 11) to score each answer. To avoid any positional bias, which may cause the GPT-4 model to favor the initial displayed answer, we switch answer positions in two evaluation rounds. The winning answer is determined based on scores in two rounds.

For **Score-based evaluation**, each answer's scores from the two rounds are combined. The answer with higher overall score is the winner. The result is a tie if both answers have same score. For **Round-based evaluation**, the scores from each round are compared, and the winner of each round is determined by the higher score. The overall winner is the one that wins both rounds. In cases where each answer wins a round, or answers tie in both rounds, the result is marked as a tie.

5.3 Main Results

We display our main result in Table 2 and summarise the our analysis with several key observations as follows: **(1)** The size of chunk significantly impacts the recall. As shown in Table 2, the improvement from FLC-100 to FLC-300 is around 10-15%. We believe that larger chunks are able to retain more information of the answer scope in a single chunk, which lead to better prediction from the retrieval. **(2)** Each view of multi-view strategy tends to help retrieval achieves a higher recall than FLC. Among each individual view, utilizing summary view generate the best results, while raw-text view generate the second best results. Despite keywords view down-performs overall due to text having poor semantic structure, we observe that keyword is able to solve some tasks which the other two view unable. This contributes to a positive impact (see Section 5.5). **(3)** The multi-view strategy, which consolidates top-ranked results of raw-text, keywords, and summary views, can substantially all baselines. We believe the improvement is mainly contributed by the content-aware chunking and multi-view indexing strategy. Different views are able to rank the relevance of sections to question from different perspectives, thus providing complimentary information.

5.4 Evaluation of Answer Generation

We compare the performance of MC-indexing against FLC-300 via the relevance of generated answers. For our experiments, we employ various retrieval methods, including BM25, DPR, Col-

	Chunking Scheme	Sparse Retrieval				Dense Embedding Retrieval												Avg
		TF-IDF		BM25		DPR		ColBERT		Contriever		E5		BGE		GTE		
		2M	NQ	2M	NQ	2M	NQ	2M	NQ	2M	NQ	2M	NQ	2M	NQ	2M	NQ	
Top $k = 1.5$	FLC: 100 tokens	47.8	14.6	45.8	7.8	35.3	25.1	54.2	27.4	54.2	22.9	57.7	33.0	55.8	27.9	56.3	29.8	37.2
	FLC: 200 tokens	51.1	19.4	56.1	11.7	40.6	35.7	62.0	37.1	61.9	29.8	67.0	41.9	63.2	37.3	63.7	38.1	44.8
	FLC: 300 tokens	60.9	20.8	61.6	13.9	41.5	41.3	64.0	37.5	64.4	35.0	68.1	47.9	64.6	41.1	65.1	41.8	48.1
	RAPTOR	15.1	20.2	16.3	13.5	14.1	21.0	22.8	37.8	23.4	38.6	25.3	38.0	25.2	38.3	25.6	36.7	25.7
	Atomic Unit	51.0	30.1	49.9	38.1	28.1	39.1	45.5	36.7	48.2	35.7	48.0	42.2	46.8	38.5	45.9	43.8	41.7
	Atomic Unit: Plus	73.3	**47.1**	75.6	**51.2**	54.0	54.5	65.1	51.9	71.1	51.6	73.4	58.5	69.6	55.6	71.5	60.8	61.6
	Content: raw-text	59.0	22.5	66.7	19.6	49.0	39.6	67.1	43.2	72.1	34.5	76.3	43.5	72.7	45.9	74.0	47.8	52.1
	Content: keyword	47.4	16.7	57.8	12.8	46.5	31.3	69.2	38.9	67.0	30.4	70.0	44.2	65.8	39.8	68.3	41.0	46.7
	Content: summary	66.2	24.4	72.2	17.6	54.3	43.3	74.0	42.7	72.8	37.0	73.3	53.2	71.8	47.4	73.3	45.6	54.3
	MC-indexing	**79.2**	40.9	**83.7**	36.9	**67.7**	**58.4**	**85.1**	**62.3**	**83.8**	**52.2**	**87.0**	**69.6**	**83.7**	**63.1**	**84.0**	**62.3**	**68.7**
Top $k = 3$	FLC: 100 tokens	58.3	21.2	58.7	12.9	46.9	35.4	64.4	39.2	65.0	35.2	69.5	46.3	69.4	41.1	69.5	43.0	48.5
	FLC: 200 tokens	67.7	30.2	70.2	21.9	55.0	48.7	70.9	50.8	73.5	43.6	77.8	56.7	75.7	52.9	77.5	54.2	58.0
	FLC: 300 tokens	70.7	32.3	74.9	23.7	58.4	54.4	73.8	50.0	75.6	51.7	81.2	62.1	77.7	57.6	78.2	59.2	61.3
	RAPTOR	30.1	34.8	34.2	26.3	27.1	34.3	41.4	52.1	43.0	54.5	45.0	55.2	47.8	56.2	46.1	56.1	42.8
	Atomic Unit	64.4	47.1	65.6	51.2	43.1	54.5	56.6	51.9	60.8	51.6	62.4	58.5	60.0	55.6	61.6	60.8	56.6
	Atomic Unit: Plus	79.8	**60.7**	81.7	**64.7**	63.9	70.2	72.5	64.7	79.0	67.6	80.1	73.2	77.7	69.0	79.2	74.1	72.4
	Content: raw-text	75.2	46.8	81.4	41.6	66.5	69.5	80.0	68.9	86.1	62.6	88.1	77.3	85.6	73.9	86.4	74.4	72.8
	Content: keyword	69.5	39.9	73.8	30.7	64.9	59.7	84.2	65.5	82.5	63.3	83.6	75.6	83.3	70.1	84.5	70.3	68.8
	Content: summary	83.1	51.9	86.1	39.1	71.1	72.4	86.8	71.1	86.6	64.5	88.1	81.6	86.9	76.9	87.3	76.3	75.6
	MC-indexing	**86.6**	54.1	**89.3**	47.6	**77.2**	**75.1**	**91.0**	**77.1**	**90.5**	**70.8**	**92.8**	**85.3**	**90.6**	**78.8**	**90.8**	**77.8**	**79.7**
Top $k = 5$	FLC: 100 tokens	65.5	28.4	65.2	19.2	54.8	45.4	70.6	46.7	70.9	43.3	77.7	55.2	75.8	50.8	76.8	52.0	56.1
	FLC: 200 tokens	74.1	39.2	77.2	30.1	64.9	60.2	76.1	59.5	78.9	54.0	83.6	66.3	81.6	61.6	82.4	63.9	65.9
	FLC: 300 tokens	76.7	42.5	80.8	34.9	65.7	66.8	78.8	60.3	81.9	62.8	85.9	73.1	83.1	68.6	84.1	70.0	69.8
	RAPTOR	47.0	46.1	48.9	36.6	37.9	47.8	56.8	62.5	60.4	64.3	60.6	63.3	62.6	69.1	60.8	70.0	55.9
	Atomic Unit	71.4	59.3	73.6	61.0	51.4	66.5	62.7	60.5	69.2	64.3	71.3	70.1	67.4	64.8	69.1	70.4	65.8
	Atomic Unit: Plus	83.5	**72.9**	85.7	**71.9**	71.3	79.4	77.8	75.4	83.6	77.8	84.9	82.3	82.3	78.3	84.0	81.7	79.6
	Content: raw-text	80.0	63.5	85.3	53.8	74.2	80.7	84.5	78.2	90.2	74.2	91.3	87.9	89.2	82.6	89.7	84.1	80.6
	Content: keyword	76.5	53.8	80.2	43.3	73.0	75.1	89.0	76.6	87.5	75.8	87.8	85.8	87.8	82.8	88.9	82.0	77.9
	Content: summary	88.1	66.5	89.5	51.9	78.2	84.8	90.7	81.9	90.8	78.1	91.7	90.9	90.7	86.4	91.2	86.5	83.6
	MC-indexing	**90.5**	67.6	**93.6**	60.1	**81.9**	**87.5**	**93.4**	**85.2**	**92.8**	**82.1**	**94.5**	**91.8**	**93.0**	**89.2**	**93.1**	**88.0**	**86.5**
Top $k = 10$	FLC: 100 tokens	73.3	38.8	73.0	29.2	65.7	60.9	77.8	60.3	80.0	55.9	83.8	68.6	83.2	63.6	83.9	64.8	66.4
	FLC: 200 tokens	81.1	52.4	83.5	44.2	74.9	73.8	82.5	70.8	85.5	69.8	88.4	78.7	88.2	75.2	88.5	75.8	75.8
	FLC: 300 tokens	82.7	60.8	86.9	52.1	75.6	79.7	85.7	75.8	87.9	77.6	89.9	85.1	89.0	83.3	89.9	81.1	80.2
	RAPTOR	67.8	63.9	69.2	63.9	56.9	67.5	74.9	78.0	79.1	81.0	79.0	79.7	81.2	83.3	79.4	83.7	74.3
	Atomic Unit	78.1	72.9	79.9	71.9	60.9	79.4	70.9	75.4	77.1	77.8	78.3	82.3	76.0	78.3	77.0	81.7	76.1
	Atomic Unit: Plus	88.9	85.5	90.2	**85.7**	81.4	88.5	85.2	87.7	90.3	88.7	90.3	92.1	89.0	89.8	89.6	92.0	88.4
	Content: raw-text	85.3	82.4	89.3	74.2	83.5	89.9	90.2	90.6	93.6	88.7	94.3	96.2	92.6	93.7	93.7	93.0	89.5
	Content: keyword	84.5	76.6	86.8	67.2	82.3	89.8	92.9	90.8	91.9	89.2	93.0	94.4	93.0	92.2	93.7	92.5	88.2
	Content: summary	92.9	84.5	93.3	76.8	86.9	94.2	94.3	92.2	94.4	90.9	95.2	96.4	94.1	94.5	94.6	94.5	91.8
	MC-indexing	**94.5**	**85.7**	**95.3**	78.2	**88.8**	**95.0**	**96.0**	**94.8**	**95.8**	**92.7**	**96.5**	**97.2**	**95.3**	**95.4**	**96.0**	**95.4**	**93.3**

Table 2: Main results: recall of ground truth span. The best score is in **boldface** and second best score is underlined.

BERT, and BGE. For each of MC-indexing and FLC-300, we first use these retrievers to sample the sections related to the question. Given the retrieved sections, we proceed to generate answers using the prompt provided in Figure 10. The generated answers are then compared using pairwise comparison (see Section 5.2).

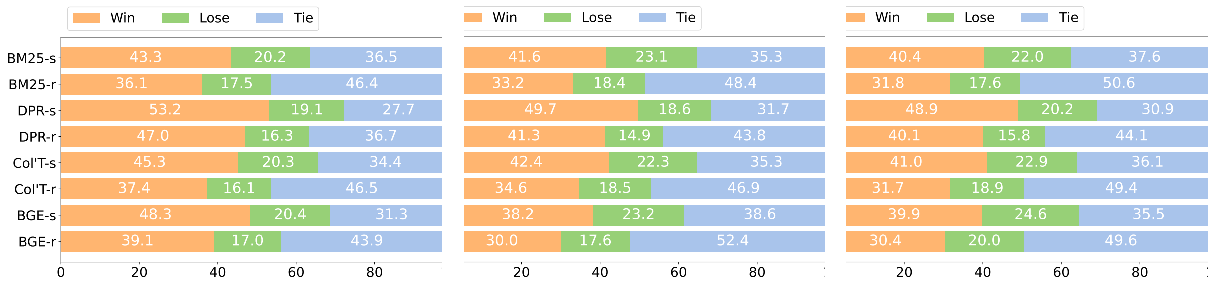
The results of this comparative assessment are displayed in Figure 3. We find that MC-indexing consistently demonstrates higher win rates than loss rates against FLC-300 across all retrievers and both evaluation metrics.

Positional bias in GPT-4 may cause it to assign higher scores to the first answer in the prompt. Unlike score-based evaluation, which takes into account the magnitude of score differences, round-based evaluation is purely predicated on the number of rounds won by each answer. Consequently, we anticipate that the round-based evaluation will yield more ties than the score-based evaluation.

5.5 Ablation Study

We conducted an in-depth study by ablating each view from our multi-view indexing strategy and measuring the performance by recall. From the results presented in Table 3, we observe that: **(1)** Removing the summary view leads to the most significant decrease in performance, ranging between 2 and 8%. **(2)** Eliminating the raw-text view results in the second-most considerable performance drop, varying between 2 and 5%. **(3)** Disregarding the keywords view contributes to a decrease of performance ranging from 1 to 4%.

Thus, we infer that the impact of each view on



(a) Win, lose, tie rates for top $k = 1.5$ (b) Win, lose, tie rates for top $k = 3$ (c) Win, lose, tie rates for top $k = 5$

Figure 3: The evaluation results of answer generation.

	Chunk Scheme	Top1.5	Top3	Top5	Top10	Δ
TF-IDF	MC-indexing	**79.2**	**86.6**	**90.5**	**94.5**	-
	- w/o raw text	71.2	82.6	87.4	93.3	-4.1
	- w/o keyword	76.8	85.6	89.1	93.8	-1.4
	- w/o summary	68.2	77.8	82.1	87.9	-8.7
BM25	MC-indexing	**83.7**	**89.3**	**93.6**	**95.3**	-
	- w/o raw text	78.2	85.9	91.0	93.8	-3.2
	- w/o keyword	81.6	87.8	92.1	94.0	-1.6
	- w/o summary	74.9	83.8	88.4	91.5	-5.8
DPR	MC-indexing	**67.7**	**77.2**	**81.9**	**88.8**	-
	- w/o raw text	61.3	72.0	77.6	86.1	-4.7
	- w/o keyword	63.6	73.9	79.2	86.7	-3.0
	- w/o summary	59.3	69.9	75.6	84.2	-6.7
ColBERT	MC-indexing	**85.1**	**91.0**	**93.4**	**96.0**	-
	- w/o raw text	82.3	89.5	91.8	95.3	-1.7
	- w/o keyword	82.0	88.6	91.3	94.4	-2.3
	- w/o summary	78.4	86.3	90.1	94.1	-4.2
Contriever	MC-indexing	**83.8**	**90.5**	**92.8**	**95.8**	-
	- w/o raw text	79.1	87.4	90.4	94.7	-2.8
	- w/o keyword	81.5	89.0	91.5	95.0	-1.5
	- w/o summary	78.9	87.3	90.6	94.4	-2.9
E5	MC-indexing	**87.0**	**92.8**	**94.5**	**96.5**	-
	- w/o raw text	80.6	89.0	92.1	95.4	-3.4
	- w/o keyword	84.6	91.3	93.3	96.0	-1.4
	- w/o summary	83.9	90.3	92.8	95.5	-2.1
BGE	MC-indexing	**83.7**	**90.6**	**93.0**	**95.3**	-
	- w/o raw text	78.3	87.0	90.5	94.1	-3.2
	- w/o keyword	81.0	89.0	91.3	94.3	-1.8
	- w/o summary	79.7	88.1	91.1	94.2	-2.4
GTE	MC-indexing	**84.0**	**90.8**	**93.1**	**96.0**	-
	- w/o raw text	79.6	87.7	90.6	94.5	-2.9
	- w/o keyword	81.8	89.2	91.8	94.7	-1.6
	- w/o summary	80.4	88.5	91.4	94.5	-2.3

Table 3: Ablation study of recall on WikiWeb2M, Δ refers to the average decrease of top 1.5, 3, 5, and 10.

	Chunk Scheme	Top1.5	Top3	Top5	Top10	Δ
TF-IDF	FLC: 300 tokens	60.9	70.7	76.7	82.7	-
	- w/ content	64.5	76.2	80.3	85.2	+3.8
	- w/ multi-view	69.5	75.2	82.6	88.8	+6.3
BM25	FLC: 300 tokens	61.6	74.9	80.8	86.9	-
	- w/ content	66.3	76.4	81.1	85.4	+1.3
	- w/ multi-view	69.9	79.3	84.3	89.2	+4.6
DPR	FLC: 300 tokens	41.5	58.4	65.7	75.6	-
	- w/ content	48.8	61.8	69.4	78.5	+4.3
	- w/ multi-view	50.1	60.8	70.0	79.0	+4.7
ColB'	FLC: 300 tokens	64.0	73.8	78.8	85.7	-
	- w/ content	73.0	82.5	87.1	91.8	+8.0
	- w/ multi-view	72.7	81.7	85.7	91.9	+7.4
Cont'	FLC: 300 tokens	64.4	75.6	81.9	87.9	-
	- w/ content	73.5	85.0	89.0	93.0	+7.7
	- w/ multi-view	69.3	80.0	86.6	91.1	+4.3
E5	FLC: 300 tokens	68.1	81.2	85.9	89.9	-
	- w/ content	75.9	86.9	90.4	93.7	+5.5
	- w/ multi-view	74.2	83.7	88.8	93.5	+3.8
BGE	FLC: 300 tokens	64.6	77.7	83.1	89.0	-
	- w/ content	75.1	85.5	89.5	92.8	+7.1
	- w/ multi-view	69.3	79.7	86.5	92.2	+3.3
GTE	FLC: 300 tokens	65.1	78.2	84.1	89.9	-
	- w/ content	75.7	87.1	91.2	95.1	+8.0
	- w/ multi-view	70.4	81.8	87.7	93.2	+4.0

Table 4: Using MC-indexing on FLC 300 tokens, Δ refers to the average increase of top 1.5, 3, 5, and 10.

the recall performance of retrieval, from the most to the least significant, is as follows: summary view, raw-text view, and keywords view. In conclusion, each view plays a crucial role in improving recall performance. More ablation results on NQ dataset are shown in Appendix F.

5.6 Does MC-indexing improve FLC?

MC-indexing improves the performance of FLC by (i) incorporating document structures and (2) using multi-view indexing. In this section, we discuss results (Table 4) of applying MC-indexing on FLC (300 tokens). More results of MC-indexing impact on FLC (200 tokens) are shown in Table 8.

Content-awareness. We evaluate the *capability of content awareness in boosting FLC*. We first segment the document into section chunks, and further apply FLC on each section. Hence, a section may have multiple chunks but each chunk is only be associated with a section. In this way, content-aware chunking reduces possibility of the ground truth answer scope being split, *i.e.,* chunking error (see Appendix A). As shown in Table 4, given same chunk length, FLC improves by 3-8% after content information is incorporated.

Multi-view Indexing. We evaluate if *multi-view indexing improves FLC, given the absence of content structure*. In this case, each FLC is additionally indexed with summary and keywords view for more efficient retrieval. We observe that the multi-view indexing significantly improves the performance of FLC by 3-7%, as shown in Table 4.

Multi-view via		Top1.5	Top3	Top5	Top10	Avg
BM25	GPT4	83.7	89.3	93.6	95.3	90.5
	Llama2-7B	79.7	87.4	89.3	93.1	87.4
	Mistral-7B	80.3	89.3	93.6	93.6	89.2
DPR	GPT4	67.7	77.2	81.9	88.8	78.9
	Llama2-7B	69.1	77.1	82.1	89.4	79.4
	Mistral-7B	68.1	76.0	82.3	89.4	78.9
ColB'	GPT4	85.1	91.0	93.4	96.0	91.4
	Llama2-7B	84.7	89.6	93.1	96.0	90.9
	Mistral-7B	83.6	88.5	92.1	95.8	90.0
E5	GPT4	87.0	92.8	94.5	96.5	92.7
	Llama2-7B	87.6	91.9	94.1	96.2	92.4
	Mistral-7B	86.9	91.8	94.2	96.2	92.3
GTE	GPT4	84.0	90.8	93.1	96.0	91.0
	Llama2-7B	84.6	90.7	93.0	95.7	91.0
	Mistral-7B	84.2	90.1	92.3	95.7	90.6

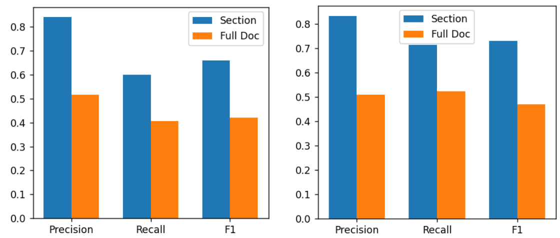
Table 5: Using different LLMs for summary generation and keywords extraction during multi-view indexing.

5.7 Multi-view Indexing using different LLMs

Multi-view indexing involves two well-studied NLP tasks: text summarization and keywords extraction. In this section, we elaborate on using different LLMs for summary and keywords generation. Firstly, we apply the proprietary model (GPT-4) to generate summary and keywords. We acknowledge that using such approach on larger scale of long documents could be cost-intensive. Hence, we have attempted using a far less cost-intensive open-sourced models (*e.g.,* Llama2-7B and Mistral-7B) instead. Our findings suggest that open-sourced models are capable of generating reliable summary and keywords. The final results, as shown in Table 5, indicate that using Llama2-7B and Mistral-7B for multi-view indexing is nearly as effective as using GPT-4 model.

5.8 Can Long-context LLM resolve Long Document QA?

Recently, there is a growing interest in utilizing LLMs for QA tasks (Chen et al., 2023; Sarthi et al., 2024). However, feeding LLM directly with long documents are infeasible due to its token limit constraints. For instance, LLaMA (Touvron et al., 2023a), LLaMA 2 (Touvron et al., 2023b), and Mistral (Jiang et al., 2023) have token limit of to 2k, 4k, and 8k, respectively, which is too less for long documents. Furthermore, Liu et al. (2023) indicates that LLMs struggle in retaining and referencing information from earlier portions of long documents. In this section, we test if advanced LLMs (*e.g.,* GPT-3.5 and 4), can effectively understand long documents. We have opted for Span-QA setting to simplify the process, where gold answer



(a) Span-QA using GPT3.5 (b) Span-QA using GPT4

Figure 4: GPT on span-QA using Full Doc vs Section

is a span of raw text from the input document. We then measure the precision, recall, and F_1 score of the retrieved span based on gold answer.

GPT-3.5 takes in document with 15k tokens as context, while GPT-4 taking longer documents with 30k tokens. They are given 2,000 questions to answer, which questions are all sourced from our Wiki-2M dataset. On the other hand, we use only the section (370 tokens in average) containing gold answers as context to GPT, to observe if GPT performs more proficiently on shorter answer scope. As depicted in Figure 4, our research indicates that the performance of GPT-3.5 and GPT-4 in span-based QA deteriorates substantially when given long documents as compared to a specific section. When GPT-4 is applied to documents of around 30k words, the recall is a mere 52.3%. This score is far lower than that of the existing index-then-retrieve systems, which can yield a recall of 90-97%.

6 Conclusion

In this paper, we propose a new approach: **Multi**-view **C**ontent-aware **indexing** (**MC-indexing**) for more effective long document question answering. Specially, we propose a long document QA dataset which annotates not only the question-answer pair, but also the document structure and the document scope to answer this question. We propose a content-aware chunking method to segment the document into content chunks according to its organizational content structure. We design a multi-view indexing method to represent each content chunk in raw-text, keywords, and summary views. Through extensive experiments, we demonstrate that content-aware chunking can eliminate chunking errors, and multi-view indexing can significantly benefit long DocQA. For future work, we would like to explore how to use the hierarchical document structure for more effective retrieval. Moreover, we would like to train or finetune a retriever that can generate more fine-grained or nuanced embeddings across multiple views.

Limitations

The limitations of our method MC-indexing, can be evaluated from two primary perspectives.

Firstly, our method considers the structured format of a document. When the document lacks clear indications of content structure, applying our content-aware chunking technique becomes challenging. However, we would like to emphasize that our work focuses on structured indexing and retrieval of long documents, and long documents usually have structured content to be utilised. It is unusual to encounter lengthy and poorly structured documents in which the authors have written tens of thousands of words without providing clear document section or chapter demarcations.

To study the usability of our method to unstructured documents, we apply the multi-view indexing on fixed-length chunking (FLC) documents, as mentioned in Section 5.6. We observe that multi-view indexing significantly improves FLC by 3-7%. Hence we believe our proposed MC-indexing will benefit existing FLC, even when content structure of the document is not available.

Secondly, short documents, being within the Large Language Model's (LLM) capacity, which means structured layout might not be required for the model to perform Question Answering (QA) tasks efficiently. Hence, we clarify that our method does not aim to enhance retrieval performance on unstructured short document. In contrast, our method can significantly benefit the retrieval of structured long documents.

Potential Risks

In this work, we utilize two existing datasets: Wikipedia Webpage 2M (WikiWeb2M) (Burns et al., 2023) and Natural Questions (NQ) (Kwiatkowski et al., 2019) datasets. Both datasets are from public resource, Wikipedia, which we believe the potential risk of malicious or unintended harmful content is minimal.

References

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2018. Ms marco: A human generated machine reading comprehension dataset.

Jan Buchmann, Max Eichler, Jan-Micha Bodensohn, Ilia Kuznetsov, and Iryna Gurevych. 2024. Document structure in long document transformers. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1056–1073, St. Julian's, Malta. Association for Computational Linguistics.

Andrea Burns, Krishna Srinivasan, Joshua Ainslie, Geoff Brown, Bryan A. Plummer, Kate Saenko, Jianmo Ni, and Mandy Guo. 2023. Wikiweb2m: A page-level multimodal wikipedia dataset.

Howard Chen, Ramakanth Pasunuru, Jason Weston, and Asli Celikyilmaz. 2023. Walking down the memory maze: Beyond context limit through interactive reading.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning. *Trans. Mach. Learn. Res.*, 2022.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. Association for Computational Linguistics.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Omar Khattab and Matei Zaharia. 2020a. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 39–48, New York, NY, USA. Association for Computing Machinery.

Omar Khattab and Matei Zaharia. 2020b. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 39–48, New York, NY, USA. Association for Computing Machinery.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA. Curran Associates Inc.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts.

Yixin Nie, Songhe Wang, and Mohit Bansal. 2019. Revealing the importance of semantic retrieval for machine reading at scale. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2553–2566, Hong Kong, China. Association for Computational Linguistics.

Yuxiang Nie, Heyan Huang, Wei Wei, and Xian-Ling Mao. 2022. Capturing global structural information in long document question answering with compressive graph selector network. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5036–5047, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yuxiang Nie, Heyan Huang, Wei Wei, and Xian-Ling Mao. 2023. AttenWalker: Unsupervised long-document question answering via attention-based graph walking. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13650–13663, Toronto, Canada. Association for Computational Linguistics.

Vatsal Raina and Mark Gales. 2024. Question-based retrieval using atomic units for enterprise rag.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994*, volume 500-225 of *NIST Special Publication*, pages 109–126. National Institute of Standards and Technology (NIST).

Stephen E. Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Gerard Salton, Edward A. Fox, and Harry Wu. 1983. Extended boolean information retrieval. *Commun. ACM*, 26(11):1022–1036.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu,

Chunk	Dataset	FLC	FLC-content	Content-aware
N=100	Wiki-NQ	66.4	50.8	0.0
	Wiki-2M	75.3	60.9	0.0
N=200	Wiki-NQ	41.4	23.2	0.0
	Wiki-2M	46.6	28.7	0.0
N=300	Wiki-NQ	26.4	13.5	0.0
	Wiki-2M	32.2	15.0	0.0

Table 6: Chunking Error for each chunking method.

Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training.

Shitao Xiao, Zheng Liu, Yingxia Shao, and Zhao Cao. 2022. Retromae: Pre-training retrieval-oriented language models via masked auto-encoder.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding.

Liu Yang, Mingyang Zhang, Cheng Li, Michael Bendersky, and Marc Najork. 2020. Beyond 512 tokens: Siamese multi-depth transformer-based hierarchical encoder for long-form document matching. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, page 1725–1734, New York, NY, USA. Association for Computing Machinery.

Shunyu Zhang, Yaobo Liang, Ming Gong, Daxin Jiang, and Nan Duan. 2022. Multi-view document representation learning for open-domain dense retrieval. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5990–6000, Dublin, Ireland. Association for Computational Linguistics.

A Chunking Error

As previously discussed in Section 1, FLC tends to cause significant chunking errors. Such chunking errors can significant affect the performance of the quality of final answer. In this section, we elaborate the chunking errors from two fixed-length chunking strategies on two datasets.

Firstly, the existing FLC method is content-agnostic. This is due to the fact the method divides the entire document into fixed-length chunks, which may inadvertently break a coherent section into separate parts. Alternatively, we recommend a different FLC approach that segments each section of the document into fixed-length chunks. This would ensure that a chunk doesn't span across two different sections, thereby more robust to chunking errors. In summary, our proposed content-aware chunking strategy ensures that no chunk extends over two sections, effectively reducing chunking errors. Results shown in Table 6 highlight the impact of content-aware chunking on chunking error.

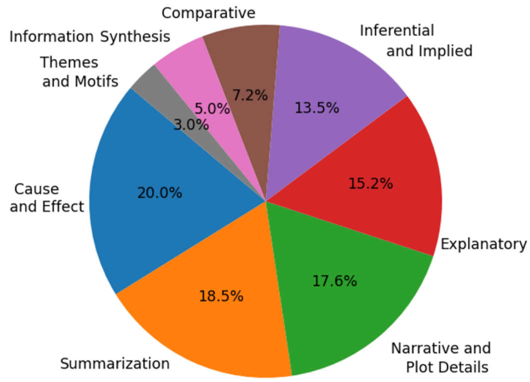


Figure 5: Pie chart of question type distribution.

B WikiWeb2M: More Annotation Details

B.1 Question Generation for WikiWeb2M

We aim to generate question that tends to rely on a long answer scope. Typically, the length of answer scope ranges from 50 to 500 tokens. We define questions of the following 8 types:

- *Narrative and Plot Details*: inquire specific details or sequence of events in a narrative (*e.g.*, a story, movie, or historical account).
- *Summarization*: require the summarization of a long passage, argument, or complicated process.
- *Inferential and Implied*: depend on understanding subtleties and reading across a long passage.
- *Information Synthesis*: inquire the synthesis of information dispersed across a long passage.
- *Cause and Effect*: understand the causal relationship between events in a long passage.
- *Comparative*: ask for comparisons between different ideas, characters, or events within a text.
- *Explanatory*: ask for explanations of complex concepts or processes that are described in detail.
- *Themes and Motifs*: consider entire text to identify patterns and conclude on central messages.

The distribution of generated question types is shown in Figure 5.

B.2 Question Answer Annotation for WikiWeb2M

For each given section, we request GPT-4 to generate 3 questions, the corresponding answers and identify the raw text that maps to the answer. In our prompt from Figure 6, we provide GPT-4 the raw text of the given section, the description of the 8 question types from Appendix B.1 and our designed prompt instruction. Our prompt instruction ensures GPT-4 to generate the continuous context sentences to sufficiently answer the question. The answer scope is then used to evaluate the retrieval efficiency of MC-indexing.

C Implementation Details of Chunking/Indexing Baselines

C.1 Fixed-length chunking (FLC)

We firstly segment the document into individual sentences using NLTK library [4]. This is to avoid the first and last sentence in each chunk being truncated. Subsequently, we merge consecutive sentences into fixed length chunks, with approximately 100, 200 or 300 tokens. Note that in order to prevent chunking sentences in the middle, the number of tokens per chunk is not exactly same to the predefined length.

C.2 Recursive Fixed-length chunking

We follow Sarthi et al. (2024) to implement *RAPTOR* scheme, which consist of the document indexing process (recursive fixed-length chunking) and retrieval process (hierarchical tree traversal). The implementation is based on the source code, which is available on GitHub.[5].

Document Indexing. The document is divided into chunks of 300 tokens. The chunks are then used to construct RAPTOR tree construction, which the procedures are as follows: the chunks are initialised as the leaf nodes of the tree. Each node is embedded using a chosen dense embedding model, and clustered based on Gaussian Mixture Models (GMMs). The nodes in each cluster are summarised using large language model and re-embedded. The summarised text and embedding of the each cluster is initialised as node, a layer above the leaf node. The clustering and embedding process are repeated until the number of nodes are too less to be clustered. For ColBERT, tree construction is not possible. This is due to the fact ColBERT relies on post interactions between the embedding of both query and chunk. In other words, the embedding of the chunk is dependent to query and could not be constructed standalone. Sparse retrieval does not have embedding model, hence making tree construction not possible. For these three experiments, we used text-embedding-ada-002, which is the same encoder provided from the GitHub[6] to embed the chunks and construct the tree.

Chunk Retrieval. For tree retrieval, there are two methods available, namely tree traversal and collapsed tree respectively. We choose the tree traversal approach as it allows retrieving a fixed number of leaf nodes, which is required to calculate recall of retrieval for each top-k (see Section 5.2). Given that our top-k sampling is k, and the tree has n layers, the steps for tree traversal are as follows : the query is embedded with the same embedding model used for tree construction. The cosine similarity between the embedding of query and nodes are computed. k nodes are sampled in the root layer based to form set S_i. The cosine similarity for each child node in S_i are calculated and k nodes are sampled to form set S_{i+1}. The iteration continues until it reaches the last layer of the tree, which S_n consists of k number of leaf nodes. We calculate the recall of retrieval based on the original token positions of the corresponding chunk of the retrieved leaf nodes. For $k = 1.5$, we set k as 1 for half of the query and k as 2 for the other half. As it is not possible to embed the query using sparse retrieval, we modify the sampling procedure of every layer based on the retrieval relevance score of the text in each nodes given the query.

C.3 Atomic Unit Chunking

The *atomic unit chunking* scheme loosely follows text chunking ideas described in (Raina and Gales, 2024), with some modification to ensure fair comparison with our models and various baseline methodologies. The procedures of atomic unit chunking are as follows: we first split each long text documents into 2000-token segments using the NLTK library. Then a LLM is instructed to split each 2000-token segment into atomic chunks, where the prompt template is given in Figure 12.

[4] https://www.nltk.org/api/nltk.tokenize.html
[5] https://github.com/parthsarthi03/raptor
[6] https://github.com/parthsarthi03/raptor

Atomic Unit: Plus. Since the lengths of atomic unit chunking is usually much shorter than the section length in NQ and WikiWeb2M, for abalation purposes controlling for chunk length, we also increased number of passages to be retrieved under the *Atomic Unit: Plus* such that the number of tokens retrieved is close to (top-k retrieved $\times$ average number of token per section). Note that since the average length of chunks produced by atomic chunking is 94 and 233 for WikiWeb2M and NQ respectively, and average number of tokens in each section produced by raw-text chunking is 375 and 510 for WikiWeb2M and NQ respectively, the number of chunks retrieved in *Atomic Unit : Plus* is 4 times and 2 times in WikiWeb2M and NQ respectively the number of chunks retrieved in *Atomic Unit* chunking scheme.

Atomic Chunking Details. Since the LLM might not faithfully reproduce sentences in each section (e.g. leaving out certain words, sentences; paraphrasing content etc.), we map contiguous sentences, where each sentence is tokenized using NLTK, from the original document to corresponding sections produced by the LLM. These contiguous subsequence of sentences would form the passages to be retrieved. We describe the procedures as follows: Let the i-th section generated by the designated LLM be denoted by S_i and the j-th original sentence in the original text be denoted by y_j where the indices are ordered according to their order of appearance. We first breakdown each section S_i into sentences using NLTK where the k-th sentence from the generated section S_i is denoted by $s_{i,k}$. For each section S_i, we define the distance between a sentence y_j and the section generated by the LLM to be

$$D(y_j, S_i) = \min_{s_{i,k} \in S_i} d(y_j, s_{i,k})$$

where d is the Levenshtein Distance[7] function between two strings (note the abuse of notation here for S_i is not *strictly* a set of sentences). Starting from $i, j = 1$, we find the first j_1 such that $D(y_{j_1}, S_1) > D(y_{j_1}, S_2)$. All sentences y_1 to $y_{j_1 - 1}$ will first be mapped to S_1. Similarly, we recursively define $j_i \geq j_{i-1}$ to be the first index such that $D(y_{j_i}, S_i) > D(y_{j_i}, S_{i+1})$. Thus the contiguous sequence of sentences $y_{j_i}, \dots y_{j_{i+1}-1}$ forms the $i+1$-th section which we concatenate to form a atomic semantic unit to be retrieved for *atomic chunking*.

[7] https://en.wikipedia.org/wiki/Levenshtein_distance

C.4 Content-aware chunking.

The content-aware chunking methods are variants of our proposed MC-indexing. We first split the long documents as section chunks. Hence, the chunking process is content-aware, and each chunk is a semantic coherent unit. Differing from MC-indexing, we utilize only a single view from raw-text, keywords, and summary views for retrieval.

D Retrieval Models

In our experiments (section 5), we implement 2 sparse retrievers and 6 dense retrievers on our proposed MC-indexing and other chunking/indexing baselines. To facilitate understanding of these retrieval models, we first introduce the background of these commonly used retrievers in Appendix D.1. We then elaborate the implementation details in Appendix D.2.

D.1 Introduction of Retrievers

Current approaches to content retrieval are primarily classified into sparse and dense retrieval. There are two widely-used sparse retrieval methods, namely TF-IDF (Salton et al., 1983) and BM25 (Robertson et al., 1994). TF-IDF calculates the relevance of a word to a document in the corpus by multiplying the word frequency with the inverse document frequency. BM25 is an advancement of TF-IDF that introduces nonlinear word frequency saturation and length normalization to improve retrieval accuracy.

Recently, dense retrieval methods have shown promising results, by encoding content into high-dimensional representations. DPR (Karpukhin et al., 2020) is the pioneering work of dense vector representations for QA tasks. Similarly, ColBERT (Khattab and Zaharia, 2020b) introduces an efficient question-document interaction model, enhancing retrieval accuracy by allowing fine-grained term matching. Contriever (Izacard et al., 2022) further leverages contrastive learning to improve content dense encoding. E5 (Wang et al., 2022) and BGE (Xiao et al., 2023) propose novel training and data preparation techniques to enhance retrieval performance, *e.g.,* consistency-filtering of noisy web data in E5 and the usage of RetroMAE (Xiao et al., 2022) pre-training paradigm in BGE. Moreover, GTE (Li et al., 2023) integrates graph-based techniques to enhance dense embedding.

Model	Dimension	Base Model	HuggingFace Checkpoint
DPR	768	bert-base	https://huggingface.co/facebook/dpr-ctx_encoder-multiset-base https://huggingface.co/facebook/dpr-question_encoder-multiset-base
ColBERT	768	bert-base	https://huggingface.co/colbert-ir/colbertv2.0
Contriever	768	bert-base	https://huggingface.co/facebook/contriever-msmarco
E5	1024	bert-large	https://huggingface.co/intfloat/e5-large-v2
BGE	1024	RetroMAE	https://huggingface.co/BAAI/bge-large-en-v1.5
GTE	1024	bert-large	https://huggingface.co/thenlper/gte-large

Table 7: Implementation details for Dense Models

D.2 Implementation Details of Retrievers

Sparse Retrievers. In our experiments (section 5), we implement 2 sparse retrievers that are BM25 and TF-IDF (Term Frequency - Inverse Document Frequency). Note that when calculating scores for BM25 and TF-IDF for each question, we restrict the set of corpus to chunks appearing in the sole relevant Wikipedia article. For BM25, we use the code from github repository https://github.com/dorianbrown/rank_bm25. For TF-IDF we use the TF-IDF Vectorizer from scikit-learn library [8]. We briefly describe how we rank document using the TF-IDF vectorizer here. First, given the corpus (i.e. the chunks appearing in the sole relevant Wikipedia article) we convert each chunk into a sparse vector with each entry indicating the TF-IDF score of each word appearing in the chunk. Next, we convert the question into a sparse vector. Finally to rank each chunk, we calculate the cosine similarity between the question sparse vector and sparse vectors of each individual chunk.

Dense Retrievers. In our experiments (section 5), we implement 6 types of dense embedding retrievers. The dense retrieval models deployed are namely DPR (Dense Passage Retriever), ColBERT, Contriever, E5, BGE and GTE. These models use the WordPiece tokenizer from BERT and also inherit the maximum input length of 512 tokens from BERT (Devlin et al., 2019). We use pre-trained checkpoints available on HuggingFace [9]; the specific checkpoint information can be found in Table 7 alongside other configuration details. Additionally, we make use of the sentence-transformer library[10] when deploying E5, BGE and GTE.

[8]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[9]https://huggingface.co/

[10]https://www.sbert.net/

	Chunk Scheme	Top1.5	Top3	Top5	Top10	Δ
TF-IDF	FLC: 200 tokens	51.1	67.7	74.1	81.1	-
	- w/ content	58.9	72.9	77.8	82.7	+4.6
	- w/ multi-view	64.1	74.3	80.1	85.7	+7.5
BM25	FLC: 200 tokens	56.1	70.2	77.2	83.5	-
	- w/ content	60.6	71.7	77.2	82.4	+1.2
	- w/ multi-view	64.3	74.9	80.1	86.0	+4.6
DPR	FLC: 200 tokens	40.6	55.0	64.9	74.9	-
	- w/ content	45.5	61.6	69.5	78.6	+4.9
	- w/ multi-view	49.2	58.9	66.1	76.7	+3.9
ColB’	FLC: 200 tokens	62.0	70.9	76.1	82.5	-
	- w/ content	71.0	81.8	85.9	90.7	+9.5
	- w/ multi-view	68.9	79.2	85.2	90.0	+8.0
Cont’	FLC: 200 tokens	61.9	73.5	78.9	85.5	-
	- w/ content	70.1	83.4	87.6	90.6	+7.9
	- w/ multi-view	66.1	77.0	83.6	89.4	+4.1
E5	FLC: 200 tokens	67.0	77.8	83.6	88.4	-
	- w/ content	73.6	84.3	89.1	92.9	+5.8
	- w/ multi-view	70.9	81.4	87.4	91.9	+3.7
BGE	FLC: 200 tokens	63.2	75.7	81.6	88.2	-
	- w/ content	71.9	82.7	87.1	91.3	+6.1
	- w/ multi-view	67.6	77.8	84.9	92.0	+3.4
GTE	FLC: 200 tokens	63.7	77.5	82.4	88.5	-
	- w/ content	72.4	85.2	89.5	93.4	+7.1
	- w/ multi-view	67.9	80.5	86.0	91.1	+3.4

Table 8: Using MC-indexing on FLC 200 tokens, Δ refers to the average increase of top 1.5, 3, 5, and 10.

E Top *k* Selection of MC-indexing

Due to the fact MC-indexing combines the results from three views, we reduce the number of chunks retrieved from each view to have a fair comparison with single-view baselines. We describe the procedure for utilizing multi-view indexing to retrieve top-k relevant chunks with respect to a given question in Section 3.3. For each of the views, *e.g.,* raw-text, summary, keywords, we first retrieve the top-k' chunks, where $k' \approx 2k/3$. In this way, we empirically obtain an average a total of $3k'/2 \approx k$ unique chunks after deduplication.

Specifically, when comparing with top $k = 3$ single-view baselines, MC-indexing will only retrieve top $k = 1$ or 2 from each view. By combining the chunks from each view and remove overlapping ones, MC-indexing manages to retrieve an approximate of 3 chunks in total. Similarly for top $k = 5$, our method retrieves only 3 chunks form

Retriever	Chunk Scheme	Top1.5	Top3	Top5	Top10	Δ
TF-IDF	MC-indexing	40.9	54.1	67.6	85.7	-
	- w/o raw text	32.4	49.5	63.5	83.8	-4.8
	- w/o keyword	34.5	51.2	64.5	84.3	-3.4
	- w/o summary	32.4	47.6	60.1	82.6	-6.4
BM25	MC-indexing	36.9	47.6	60.1	78.2	-
	- w/o raw text	25.9	41.6	52.0	72.9	-7.6
	- w/o keyword	30.4	43.2	55.1	74.2	-5.0
	- w/o summary	27.6	41.6	54.4	72.7	-6.6
DPR	MC-indexing	58.4	75.1	87.5	95.0	-
	- w/o raw text	53.1	71.0	81.7	93.5	-4.2
	- w/o keyword	52.7	71.2	82.6	93.3	-4.0
	- w/o summary	49.8	69.1	81.2	90.5	-6.4
ColBERT	MC-indexing	62.3	77.1	85.2	94.8	-
	- w/o raw text	54.8	71.7	81.4	93.5	-4.5
	- w/o keyword	55.8	72.5	81.1	93.7	-4.1
	- w/o summary	55.6	72.4	81.2	93.2	-4.2
Contriever	MC-indexing	52.2	70.8	82.1	92.7	-
	- w/o raw text	46.9	65.5	79.4	89.2	-4.2
	- w/o keyword	46.1	64.7	78.5	88.7	-4.9
	- w/o summary	45.1	65.0	77.6	91.6	-4.6
E5	MC-indexing	69.6	85.3	91.8	97.2	-
	- w/o raw text	63.3	81.4	90.3	95.9	-3.2
	- w/o keyword	62.8	80.0	91.3	96.4	-3.3
	- w/o summary	60.9	80.3	91.1	96.7	-3.7
BGE	MC-indexing	63.1	78.8	89.2	95.4	-
	- w/o raw text	58.0	74.9	86.2	94.0	-3.3
	- w/o keyword	57.5	73.7	85.7	94.9	-3.7
	- w/o summary	56.7	74.4	85.8	94.4	-3.8
GTE	MC-indexing	62.3	77.8	88.0	95.4	-
	- w/o raw text	55.5	73.0	85.8	94.5	-3.7
	- w/o keyword	57.3	74.7	86.1	94.8	-2.7
	- w/o summary	57.7	74.0	85.0	94.0	-3.2

Table 9: Ablation study of recall on NQ, Δ refers to the average decrease of top 1.5, 3, 5, and 10.

each view. For top $k = 10$, our method retrieves 6 or 7 chunks from each view. To evaluate the performance of our method in greedy ranking, our method retrieves exactly 1 chunk from each view, while other baselines retrieves 1.5 chunks in average. This is achieved by retrieving 1 chunk for half of the questions and 2 chunks for the other half.

F Extended Ablation Study on NQ

In this section, we reported the ablation results of MC-indexing on NQ dataset, serving as the extension of Section 5.5. From the data in Table 9, it's evident that: **(1)** Removing the raw-text view leads to the most significant performance drop, ranging between 3.2 and 7.6%. **(2)** Eliminating the summary view results in the second-most considerable performance drop, varying between 3.2 and 6.6%. **(3)** Disregarding the keywords view contributes to a performance drop between 2.7 and 5%.

G Prompt Design

In this paper, we utilize the following prompts on GPT models to facilitate the respective process:

- The generation of WikiWeb2M question, question type, answer, and answer contextual sentences. The prompt is shown in Figure 6.
- The contextual sentences retrieval when provided with a long document or a section of the document. This is used to evaluate if GPT-3.5 or GPT-4 can directly cope with long document. The prompt is shown in Figure 7.
- The generation of summary for the sections consisting of more than 200 tokens. The generated summary is used as additional view for document indexing. The prompt is shown in Figure 8.
- The generation of the list of keywords for each section. The generated keywords list is used as additional view for document indexing. The prompt is shown in Figure 9.
- The generation of atomic chunks are shown in Figure 12. We further process these results in the procedures described in Appendix C.3 under **Atomic chunking**.
- The answer generation when provided with retrieved top k chunks or sections. The prompt is shown in Figure 10.
- The automatic answer evaluation of two answers, given the ground truth answer. This is used to evaluate the answer quality. This prompt is shown in Figure 11.

```
You are a sophisticated question generator. You need to use the reference text to generate a question,
with its question type, and the supporting context sentences, and the short answer.

The generation should strictly follow the following guidelines:
(1) The question must be sufficiently answered by the reference text only;
(2) The question need to be short and accurate;
(3) All supporting context sentences must be the original text from the reference text;
(4) The question should need long context (more than 5 sentences) to answer accurately;
(5) The type of each question needs to be ONE from the following eight types:
1. **Questions about Narrative and Plot Details**: inquire about specific details or the sequence of events
   in a narrative (such as a story, movie, or historical account) require understanding the entire context
   to provide an accurate answer.
2. **Summarization Questions**: require the summarization of a long passage, argument, or a complicated
   process rely on understanding the full context to capture the essence of the content without omitting
   crucial details.
3. **Inferential and Implied Questions**: depend on understanding subtleties and reading between the lines.
   They may involve inferring the author's intent, the mood of the characters in a story, or the
   implications of certain actions, which can't be answered with a direct quote from the text.
4. **Questions Requiring Synthesis of Information**: necessitate the synthesis of information dispersed
   across a long passage or multiple passages, requiring an understanding of the broader context to
   answer correctly.
5. **Cause and Effect Questions**: to understand the causal relationship between events in a text, one
   often needs to consider a substantial portion of the context to identify the factors that led to
   a particular outcome.
6. **Comparative Questions**: ask for comparisons between different ideas, characters, or events within
   a text often require a comprehensive understanding of each element being compared.
7. **Explanatory Questions**: ask for explanations of complex concepts or processes that are described
   in detail within the text. Answering these questions accurately requires a deep understanding of the
   entire explanation as presented.
8. **Questions about Themes and Motifs**: when asked about the overarching themes or motifs in a text, one
   must consider the entire work to identify patterns and draw conclusions about the central messages.

**Reference text**:
$text

Return the question and answer in the following json format:
{question:"...", type:"...", answer:"...", answer_context:"..."}
```

Figure 6: GPT-4 Prompt used for question and answer generation.

```
You are helpful question answering assistant. Given a question and the reference text, you need to find
sufficient context to answer this question. The context sentences must be the original text of reference
text. Note that you must not answer these question.

**Question**: $question

**Reference Text**: $reference

Return the result in json format: {"context": ..., "}
```

Figure 7: GPT prompt template designed to find the relevant answer scope given the question and section text.

```
You are a helpful summarization assistant. Please help me summarize the following section into no more
than 10 sentences or 200 words.

**Section Name**:
$section_name

**Section Text**:
$section_text
```

Figure 8: Prompt template designed to provide summary for section given its corresponding name and text.

```
You are a helpful keyword extractor. You need to extract keywords from the following section. The keywords
should consist of concepts, entities, or important descriptions that are related to the section text, which
could be used to answer any questions from users.

**Section Name**:
$section_name

**Section Text**:
**Beginning of text**
$section_text$
**End of text**

Please output format in list format: [...]. Do not output anything else aside from this list.
```

Figure 9: Prompt template designed to provide keywords for section given its corresponding name and text.

```
You are a helpful question answering assistant. You are good at answering question based on provided contents.

**Contents**: $quotes

**Question**: $question

**Instruction:**
Assume you do not have any background and internal knowledge about this given contents and question.
You need to answer the question using the given contents only. The answer need to be short and accurate.
```

Figure 10: Prompt template designed to answer question based on the retrieved results.

```
You are a helpful assistant for evaluating answers. Given a question and ground truth answer, there will be
two possible answers. Provide a score from 0-10 for each answer.

**Question**: $question

**Ground truth answer**: $ground_truth_answer

**Answer 1**: $answer_1
**Answer 2**: $answer_2


**Instruction:**
Assume you do not have any background and internal knowledge about this given contents and question. You
need to evaluate each answer and give a score based on the ground truth answer.
You must write out your reasoning of the score based on relevance to the answer. If both answers are
exactly similar, you must ensure the scores and reasoning for both answers are the same.
Finally in a new line, you must return the scores and nothing else. The scores must be returned in the
following json format:
{"answer_1_score":"...", "answer_2_score":"..."}
```

Figure 11: GPT prompt template designed to provide score for each answer in pair-wise evaluations.

```
You are a helpful text chunking assistant that can divide a piece of text into sections.
Given a piece of text, your task is to partition the sentences in the given text into sections according
to the following guidelines:

1. The sentences in each section should make up one stand-alone atomic fact.

2. Each section should be a contiguous chunk of text from the given text. The text in each section should be faithful
and unchanged from the given text.

3. No sentences in the given text should be divided across two different sections.

Return each section on a new line.

Please breakdown the following text into sections:
$text
```

Figure 12: Prompt template designed to provide summary for section given its corresponding name and text.