Mitigating Lost-in-Retrieval Problems in Retrieval Augmented Multi-Hop Question Answering

Anonymous ACL submission

Abstract

In this paper, we identify a critical problem, "lost-in-retrieval", in retrieval-augmented multi-hop question answering (QA): the key entities are missed in LLMs' sub-question decomposition. "Lost-in-retrieval" significantly degrades the retrieval performance, which disrupts the reasoning chain and leads to the incorrect answers. To resolve this problem, we propose a progressive retrieval and rewriting method, namely ChainRAG, which sequentially handles each sub-question by completing missing key entities and retrieving relevant sentences from a sentence graph for answer generation. Each step in our retrieval and rewriting process builds upon the previous one, creating a seamless chain that leads to accurate retrieval and answers. Finally, all retrieved sentences and sub-question answers are integrated to generate a comprehensive answer to the original question. We evaluate ChainRAG on three multi-hop QA datasets—MuSiQue, 2Wiki, and HotpotQA—using three large language models: GPT4o-mini, Qwen2.5-72B, and GLM-4-Plus. Empirical results demonstrate that ChainRAG consistently outperforms baselines in both effectiveness and efficiency.

1 Introduction

Large language models (LLMs) (OpenAI, 2023; Zeng et al., 2024; Yang et al., 2024; Li et al., 2024a) have exhibited promising performance on a wide range of natural language processing tasks, such as machine translation (Zhu et al., 2023), text summarization (Wu et al., 2021), question answering (QA) (Daull et al., 2023). While LLMs possess strong reasoning abilities, they still face challenges such as outdated knowledge and lack of domain-specific expertise (Mousavi et al., 2024), which lead to incorrect outputs (Xu et al., 2024b).

To fill the gap between LLMs' memory and real-world knowledge, retrieval-augmented generation (RAG) (Lewis et al., 2020; Gao et al., 2023) is widely used to retrieve the knowledge that is relevant to the user's question to improve the LLMs' QA performance. When answering multi-hop questions, LLMs generally utilize a question decomposition strategy in the chain-of-thought outputs, which decomposes the input question into multiple simpler sub-questions. However, in this decomposition process, we find that when a sub-question lacks a clear entity and instead uses demonstrative pronouns, the retrieval performance drops sharply. We refer to this phenomenon as "*lost-in-retrieval*".

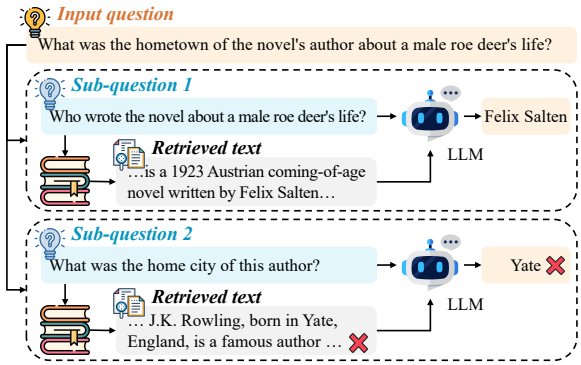

Figure 1: Example of the "lost in retrieval" issue where the second sub-question retrieves irrelevant text due to the unclear key entity, leading to an incorrect answer.

Figure 1 presents a real-world example in solving a multi-hop question using RAG combined with question decomposition. In this example, the second sub-question, "What was the home city of this author?", lacks a clear entity of the author. As a result, it leads to retrieval errors, which ultimately cause the final answer to be incorrect. The right answer is Vienna. To give the correct answer, we need to identify the specific key entities in the sub-questions so as to improve the retrieval performance.

To analyze the retrieval performance of different sub-questions, we have conducted an empirical study with 300 randomly sampled QA examples from each of the three datasets: MuSiQue (Trivedi

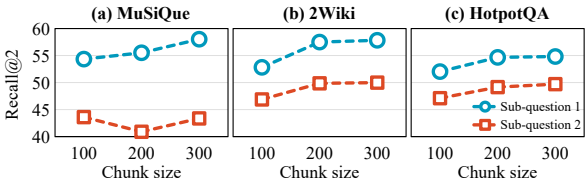

Figure 2: Analysis of "lost in retrieval". We evaluate the Recall@2 (%) of different sub-questions.

et al., 2022), 2WikiMultiHopQA (2Wiki) (Ho et al., 2020), and HotpotQA (Yang et al., 2018). Since most questions are two-hop reasoning problems, we calculate the Recall@2 scores for the first two sub-questions. As shown in Figure 2, under different chunk size settings, the Recall@2 of the second sub-question is noticeably lower than that of the first sub-question, with an average decrease of 18.29% across the three datasets. We analyze the results and find that the first sub-question typically contains a specific key entity, whereas the second sub-question often lacks one. *The ambiguity of key entities in sub-questions causes the "lost-in-retrieval" problems, which further disrupts the chain of reasoning for multi-hop QA.*

To mitigate the "lost-in-retrieval" problems and improve multi-hop QA, we propose a progressive retrieval framework called ChainRAG. It involves an iterative process of sentence retrieval, sub-question answering and subsequent sub-question rewriting. We first construct a sentence graph with named entity indexing from texts, which is used to facilitate entity completion in sub-question rewriting and to structure the knowledge scattered across different texts. Next, given an input question, we employ the LLM to decompose it into several sub-questions and retrieve relevant sentences for the first sub-question. Then, our iterative process operates as follows until all sub-questions are addressed. We prompt an LLM to answer the current sub-question. The answer is then used to rewrite the next sub-question by completing any missing key entities, if possible. The updated sub-question is subsequently used for retrieval. Finally, all retrieved sentences and sub-question answers are integrated to answer the original question.

We conduct a series of experiments using three LLMs on three multi-hop QA datasets from LongBench (Bai et al., 2024), evaluating the performance and efficiency of our method. The results suggest that our method consistently outperforms the baselines across the three datasets. It also demonstrates stable performance across different LLMs, reflecting a certain degree of robustness. In summary, our contributions are outlined as follows:

- We investigate the "lost-in-retrieval" problems of RAG for multi-hop QA. We identify that the reason is the absence of key entities in sub-questions by empirical studies.
- To resolve this issue, we propose ChainRAG, a progressive retrieval and sub-question rewriting framework. We construct a sentence graph based on the similarities and entities within the texts to support our retrieval and the completion of missing entities in sub-questions.
- We evaluate our ChainRAG on three multi-hop QA datasets. Our experimental results and analysis show that it outperforms the baselines in both effectiveness and efficiency.

2 Related Work

In this section, we review related work and discuss how our method differs from them.

2.1 Retrieval-Augmented Generation

RAG (Lewis et al., 2020) is a widely-used technique for addressing knowledge-intensive tasks. It enables LLMs to fetch relevant information from external knowledge bases, enhancing their effectiveness for complex QA, such as multi-hop KBQA. The biggest challenge of RAG lies in how to retrieve relevant and comprehensive information. One solution is to perform multiple rounds of retrieval to gather relevant passages (Trivedi et al., 2023; Shao et al., 2023). The other solution seeks to remove irrelevant information from retrieved texts (Jiang et al., 2024). Besides, recent work pays more attention to the effective utilization of retrieved texts using techniques like gist memory (Mu et al., 2023), text summarization (Xu et al., 2024a), reranking (Glass et al., 2022; Wang et al., 2024), context compression (Liu et al., 2024).

Despite the aforementioned methods, RAG still encounters challenges when handling long texts or complex questions, since relevant information is usually scattered across different parts of the text. To address this issue, many studies incorporate graph structures to organize the text. RAPTOR (Sarthi et al., 2024) structures the text into a tree structure. GraphRAG (Edge et al., 2024), GraphReader (Li et al., 2024b), and HippoRAG (Gutiérrez et al., 2024) use LLMs to extract entities and relations from the text, constructing

knowledge graphs (KGs). While effective, these methods rely on LLMs for entities and facts extraction, which increases costs.

Our work investigates a subtle problem in RAG for multi-hop QA, i.e., "lost in retrieval", caused by the missing topic entities in sub-questions. Our method resolves the problem by iteratively completing the missing entities and retrieving relevant sentences containing these entities. It eliminates the need for a complex reasoning process or an expensive KG construction pipeline.

2.2 Multi-hop QA

Multi-hop QA is an ideal scenario for evaluating RAG systems, since it requires strong capabilities from both the knowledge retriever and the answer generator. RoG (Luo et al., 2024) adopts a planning-retrieval-reasoning paradigm, using relation paths in KGs to guide the retrieval of effective reasoning paths. EfficientRAG (Zhuang et al., 2024) fine-tunes the DeBERTa-v3-large model (He et al., 2021) to construct a labeler and a filter for handling multi-round queries, reducing the frequency of LLM calls. OneGen (Zhang et al., 2024) unifies generation and retrieval by fine-tuning the model to perform both tasks simultaneously in a single-step inference. Many existing multi-hop QA methods use LLMs for query decomposition (Gao et al., 2023). However, as demonstrated in the empirical study in Section 1, this strategy suffers from "lost-in-retrieval". Our work resolves this issue without fine-tuning and frequent API calls.

3 Methodology

Figure 3 provides an overview of ChainRAG. We first construct a sentence graph from texts. Given a question, we use an LLM to decompose it into several sub-questions. Then, we design an iterative process includes sentence retrieval, sub-question answering, and subsequent sub-question rewriting. This process continues until all sub-questions are addressed. Finally, we integrates all retrieved sentences and sub-question answers to produce a comprehensive answer to the original question.

3.1 Sentence Graph with Entity Indexing

In our method, the completion of missing key entities in sub-questions is a crucial step. To facilitate entity completion, it is essential to identify all named entities retrievable from the given texts. Therefore, for efficiency consideration, we first extract named entities from texts s_i using spaCy, resulting in an entity set $\mathcal{E}_i$. In this process, we also store the mappings between each entity and all its sentences. Then, to conveniently obtain all the information of an entity from scattered texts for the following knowledge retrieval, we propose to construct a sentence graph with named entities as edge labels, where each node represents a sentence and the edge between nodes indicates that the two sentences describe the same entity. We denote the node (i.e., sentence) set as $C = \{s_1, s_2, \ldots, s_n\}$, which has already been obtained in the previous entity extract process. As for the edge, the entity-sentence mappings can be used to mine edges. However, relying solely on entity co-occurrence edges is insufficient for effective knowledge retrieval. Besides, the sentence-level retrieval is too fine-grained. We should enhance the associations between sentences for border and comprehensive retrieval. Finally, we consider the following three types of edges in our sentence graph:

- **Entity co-occurrence (EC)**. If two sentences describe the same key entity, they will be linked. A sentence may contain multiple entities, but not all of them are the key entities. We calculate the importance score, i.e., BM25, for each entity $e \in \mathcal{E}_i$, and retain only the top-$\alpha\%$ entities as key entities, denoted by $\mathcal{K}_i \subseteq \mathcal{E}_i$. This process reduces redundancy of the following construction steps. Two sentences s_i and s_j would be linked with an edge labeled "EC" if $\mathcal{K}_i \cap \mathcal{K}_j \neq \emptyset$ holds.

- **Semantic similarity (SS)**. If two sentences have a high embedding similarity, they will be linked. We encode each sentence s_i into a dense vector $\mathbf{v}_i$ using OpenAI text-embedding-3-small embeddings for computing pairwise similarities of sentences. For sentence s_i, we maintain a set $\mathcal{R}_i$ containing its top-m most similar sentences. Two sentences s_i and s_j would be linked with an edge labeled "SS" if $s_j \in \mathcal{R}_i \ \vee \ s_i \in \mathcal{R}_j$ holds.

- **Structural adjacency (SA)**. If two sentences are adjacent in texts, they will be linked. In this work, we consider a span of three sentences. If two sentences s_i and s_j are within three sentences of each other, i.e., $|i - j| \leq 3$, we add an edge labeled "SA" between them. This type of edges can helps us reconstruct the overall structure of text for a wider retrieval.

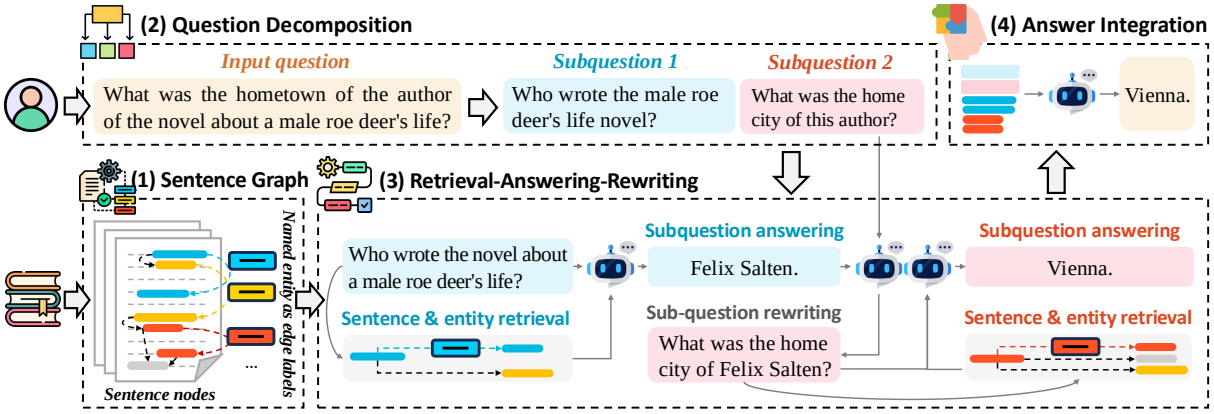

Figure 3: Framework overview of ChainRAG. It first constructs a sentence graph, where the edges between sentence nodes are labeled by their common named entities. Given a question, it is decomposed into sub-questions. Then, our iterative process involves retrieval, answering, and rewriting the unclear sub-question by filling in missing entities. Finally, it integrates all retrieved sentences and answers to produce a comprehensive answer.

The sentence graph plays a crucial role in mitigating the "lost-in-retrieval" problems. It connects sentences through shared entities and semantic associations, organizing the knowledge to ensure that even when a sub-question lacks clear entities, the necessary context can still be retrieved.

3.2 Sentence and Entity Retrieval

Before the retrieval process begins, we first utilize LLM to decompose multi-hop questions into sub-questions. The detailed prompt used for decomposing multi-hop questions can be found in Appendix D. Our retrieval method deals with the sub-questions in turn, which is a progressive retrieval process and constructs a complete inference chain through entity expansion. It involves the following two retrieval steps for each sub-question.

Seed sentence retrieval. Given a sub-question, we first calculate its embedding similarity (e.g., inner product) with all sentences in the sentence graph. This can be done quickly by matrix multiplication. We then filter our the sentences with low similarity to narrow down the retrieval candidates. Next, we use a cross-encoder to assess the relevance of each candidate sentence within the context of the sub-question, and finally, we select the top-k sentences as seed sentences.

Retrieval expansion on sentence graph. Starting from seed sentences, we iteratively explore their neighbors in the sentence graph. After each expansion, we use a LLM to assess whether the gathered sentences contain sufficient information to answer the question. If validated, the expansion is terminated. Otherwise, we continue to explore higher-order neighbors. To ensure both efficiency and quality of the retrieval, we implement multiple optimization mechanisms. To reduce the number of LLM calls, the initial neighbor exploration fetches all 1-hop neighbors of the seed sentences. Additionally, to prevent the context from becoming too lengthy, we introduce a length limit. Once the total length of the retrieved sentences reaches this limit, the retrieval process is stopped. The LLM is finally promoted to answer the sub-question based on the retrieved sentences.

3.3 Sub-question Rewriting

As we have mentioned and validated in Sect. 1 that the retrieval performance degrades if sub-questions lack necessary named entities. To resolve this problem, we propose to rewrite sub-questions. First, we determine if a sub-question needs rewriting by checking for the presence of pronouns (such as "this", "it", "they", etc.). If these pronouns exist, we feed both the current sub-question and the previous sub-questions along with their answers into an LLM to rewrite the current sub-question. When the previous sub-questions were not adequately answered, we cannot rewrite the sub-questions. In this case, we summarize their corresponding context and incorporate the summary into the context of the current sub-question. Our sub-question rewriting method serves two purposes. First, it mitigates the degradation in retrieval performance caused by missing entities. Second, since the previous sub-question is closely related to the current one, preserving its key information supports the reasoning process of the current sub-question.

3.4 Answer and Context Integration

After obtaining the answers to all sub-questions, we have two different integration methods to generate a comprehensive answer to the original question: sub-answer integration and sub-context integration.

Sub-answer integration. This method generates the answer to the original question by utilizing each sub-question and its answer. It relies solely on the information from the sub-questions' answers, without external interference. Since the decomposition of sub-questions can be regarded as a reasoning process, this method enables the LLM to infer the original question's answer. Moreover, this method ensures that the LLM processes the relevant text of only one sub-question at a time, avoiding performance degradation caused by the LLM's weak long-context processing capabilities when processing multiple sub-questions simultaneously. However, if a sub-question is answered incorrectly or left unanswered, it can significantly impact the final result. To reduce such impact, we consider the following method that integrates all retrieved sentences to enrich the context.

Sub-context integration. We remove duplicate contexts from all retrieved sentences (without using sub-questions) and use a cross-encoder to rerank the sentences for generating the final answer. This method is similar to traditional RAG, which uses only relevant text to generate the answer. It helps mitigate the impact of errors in sub-question decomposition or answers by focusing on the retrieved sentences, rather than relying solely on the sub-question answers. However, this method requires the LLM to have strong long-context processing capabilities. Compared to answering each sub-question individually, the merged context may contain more noisy information, which can negatively impact the final answer.

4 Experiments

In this section, we report the experimental results and analysis to evaluate the effectiveness and efficiency of our method for multi-hop QA. Our source code is in the attachments.

4.1 Setup

Dataset and metrics. We use the following three challenging multi-hop QA datasets in our experiments: MuSiQue (Trivedi et al., 2022), 2Wiki (Ho et al., 2020), and HotpotQA (Yang et al., 2018). Instead of using raw data, we follow the same data setting as in LongBench (Bai et al., 2024). Detailed statistics of the used datasets are provided in the Appendix A. Following convention, we assess the multi-hop QA performance using the F1-score and exact match (EM) score.

Baselines. To ensure the fairness of our evaluation, we standardize the embedding model, i.e., OpenAI's text-embedding-small-v3,[1] and the cross-encoder reranker, i.e., BGE-Reranker (Chen et al., 2024), across both our method and the baselines. We conduct experiments with three popular LLMs as the answer generator: GPT4o-mini,[2] Qwen2.5-72B (Yang et al., 2024) and GLM-4-Plus (Zeng et al., 2024). For comparison, we select NaiveRAG and three advanced train-free RAG methods as baselines: Iter-RetGen (Shao et al., 2023), LongRAG (Jiang et al., 2024), and a combination of HippoRAG (Gutiérrez et al., 2024) with IRCoT (Trivedi et al., 2023). Except for the main experiments and the efficiency analysis, all subsequent experiments are conducted exclusively on GPT4o-mini, as similar results have been observed with other LLMs. Our method has two variants, namely ChainRAG (AnsInt) and ChainRAG (CxtInt), which use the sub-answer integration and sub-context integration strategies, respectively.

Implementation details. In all experiments, we set the word limit to 3000. For sentence graph construction, α is set to 60 for the entity filter and m is set to 10 for selecting the most similar sentences. During the seed sentence retrieval phase, the number of candidate sentences selected in the first round is 100, with the top-k sentences chosen as seed sentences, where $k = 3$ in the main experiment. Further implementation details are provided in Appendix B.

4.2 Main Results

We hereby provide a detailed comparison and analysis of the overall results shown in Table 1. In general, our method has performed better compared to baselines. Compared to NaiveRAG, ChainRAG achieves significant improvements across all datasets, especially on MuSiQue, where the average F1 score has improved by approximately 60%. When compared to three advanced RAG methods,

[1] https://platform.openai.com/docs/guides/embeddings

[2] https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

LLMs	Methods	MuSiQue		2Wiki		HotpotQA	
		F1	EM	F1	EM	F1	EM
GPT4o-mini	NaiveRAG	29.82	19.00	50.61	42.50	56.92	42.00
	NaiveRAG w/ QD	37.49	26.00	56.88	38.50	60.00	43.50
	ITER-RETGEN Iter3 (Shao et al., 2023)	38.41	33.00	58.43	50.50	57.77	42.00
	LongRAG (Jiang et al., 2024)	44.88	32.00	62.39	49.00	**64.74**	**51.00**
	HippoRAG w/ IRCoT (Gutiérrez et al., 2024)	46.50	28.50	62.38	48.00	56.12	40.00
	Ours (AnsInt)	**50.54**	37.00	**62.55**	**52.00**	60.73	46.00
	Ours (CxtInt)	47.87	**38.50**	56.54	50.50	64.59	50.00
Qwen2.5-72B	NaiveRAG	27.08	16.50	39.82	28.50	50.25	34.50
	NaiveRAG w/ QD	33.91	20.50	53.84	37.00	52.14	34.50
	ITER-RETGEN Iter3 (Shao et al., 2023)	40.15	31.50	53.59	41.50	58.41	45.00
	LongRAG (Jiang et al., 2024)	40.89	29.50	62.00	51.50	60.29	46.50
	HippoRAG w/ IRCoT (Gutiérrez et al., 2024)	44.64	31.50	64.19	52.00	55.21	41.00
	Ours (AnsInt)	47.75	37.00	64.23	54.00	60.55	46.00
	Ours (CxtInt)	**49.37**	**39.00**	**65.85**	**55.50**	**64.54**	**52.00**
GLM-4-Plus	NaiveRAG	37.86	28.00	57.78	45.50	58.42	44.00
	NaiveRAG w/ QD	30.33	22.50	60.93	48.00	59.05	44.00
	ITER-RETGEN Iter3 (Shao et al., 2023)	**52.57**	**44.50**	66.56	56.50	61.03	48.50
	LongRAG (Jiang et al., 2024)	40.44	29.00	62.27	54.50	61.60	48.50
	HippoRAG w/ IRCoT (Gutiérrez et al., 2024)	44.70	29.50	67.55	55.50	63.91	48.00
	Ours (AnsInt)	51.66	40.00	67.35	58.00	55.40	42.00
	Ours (CxtInt)	49.40	38.00	**70.58**	**61.50**	**64.22**	**50.00**

Table 1: Performance (%) on MuSiQue, 2Wiki, and HotpotQA. QD refers to question decomposition. AnsInt refers to generating answers using only sub-questions and their corresponding answers, while CxtInt refers to generating answers using only the contexts retrieved by sub-questions.

ChainRAG consistently outperforms them, achieving the highest average performance across all three datasets. This is most pronounced when using Qwen2.5-72B as the LLM, where the average F1 score of CxtInt is 59.92. This surpasses the second-best method, HippoRAG w/ IRCoT, which has an average F1 score of 54.68, representing a 9.6% improvement. This advantage demonstrates the effectiveness of our proposed progressive retrieval and query entity completion strategy.

ChainRAG has also demonstrated stable performance across all three LLMs, reflecting its robustness. We further observe some differences in the results of each LLM. For example, when using GPT4o-mini, our AnsInt variant outperformed CxtInt, while the opposite is true for Qwen2.5-72B and GLM-4-Plus. This difference may primarily stem from the varying capabilities of these LLMs. GPT4o-mini exhibits strong reasoning abilities, while the other two LLMs are better at processing long-context. Our two answering strategies can each leverage these two advantages separately.

We also find that adding question decomposition to NaiveRAG sometimes brings only limited improvements. For example, on the HotpotQA dataset, the average F1 score increases by just 1.87, while the average EM score improves by only 0.5. In addition, when using GLM-4-Plus as the LLM, the performance of NaiveRAG on the MuSiQue dataset decreases after incorporating question decomposition. These cases also suggest the presence of the "lost-in-retrieval" problems. In contrast, ChainRAG consistently benefits from question decomposition with sub-question rewriting, showing its robustness in resolving the "lost-in-retrieval" problems.

4.3 Ablation Study

To validate the effectiveness of each technical module in our method, we carry out an ablation study. We explore the effects of removing the sub-question rewriting method, ablating edges in the sentence graph by type, and completely removing the graph (i.e., segmenting texts into chunks like NaiveRAG). Figures 4 and 5 show the F1 and EM scores using the AnsInt strategy, respectively. The complete results can be found in Appendix C. We observe similar results when using CxtInt.

Sub-question rewriting is one of the core components of our method. As shown in the experimental

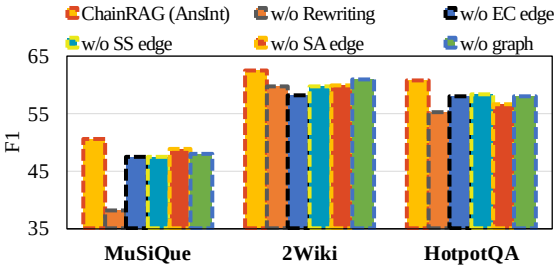

Figure 4: F1 (%) comparison of ablation study.

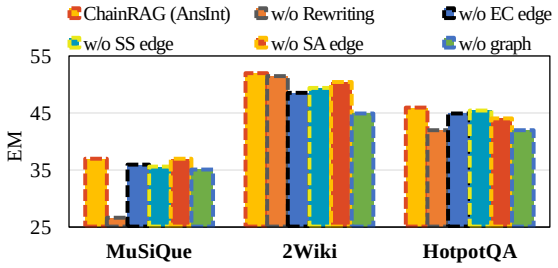

Figure 5: EM (%) comparison of ablation study.

	MuSiQue	2Wiki	HotpotQA
Sub-question 1	55.52	57.50	54.67
Sub-question 2	40.91	49.87	49.17
Modified sub-question 2	58.81	54.32	61.83

Table 2: Recall@2 (%) results of text retrieval for sub-questions. The modified sub-question 2 is rewritten by our entity completion method.

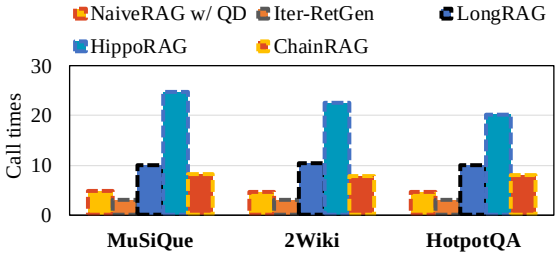

Figure 6: Comparison of GPT4o-mini call times.

results, removing this component leads to a significant decline in QA performance. On the MuSiQue dataset, both F1 and EM scores drop by approximately 30%. These results further confirm the existence of the "lost-in-retrieval" problems.

Ablating different edge types in our sentence graph leads to performance declines, demonstrating the rationality of our edge design for graph construction. Moreover, the impact of ablating specific edge types varies across datasets. For example, removing SA edges has the greatest impact on the HotpotQA dataset, while the situation is reversed on the MuSiQue dataset. This indicates that different datasets have distinct dependencies on edge types. Future work can dynamically adjust edge construction strategies for each dataset to further improve performance.

To evaluate the effectiveness of our sentence graph, we remove the entire graph and adopt a method similar to NaiveRAG, where we index the texts in chunks. The results show a decline in both F1 and EM scores, with the decrease in EM being notably more pronounced. We speculate that this is because finer-grained textual information can better guide LLM to generate more accurate answers. This outcome further confirms the rationality and effectiveness of our sentence graph.

4.4 Retrieval Performance Analysis

To validate whether our method effectively addresses the "lost-in-retrieval" problems, we conduct additional retrieval experiments. Since our method divides the context into sentences, which differs rom NaiveRAG in granularity, we standardize the chunk size in ChainRAG to match NaiveRAG for fair comparison. From Table 2, we observe that after rewriting, the Recall@2 of sub-question 2 shows a marked improvement. Notably, on the MuSiQue and HotpotQA datasets, the Recall@2 of the rewritten sub-question 2 even exceeds that of sub-question 1, as the rewriting process uses information from sub-question 1 to retrieve more relevant content.

4.5 Efficiency Comparison

We measure the efficiency of each method by counting the number of LLM calls. Figure 6 compares the average number of calls of different methods across three datasets. HippoRAG, which constructs knowledge graphs based on LLM, requires several times more calls than other methods and is significantly influenced by the dataset. This is especially true for datasets with longer context lengths, where the number of calls increases further. Compared to

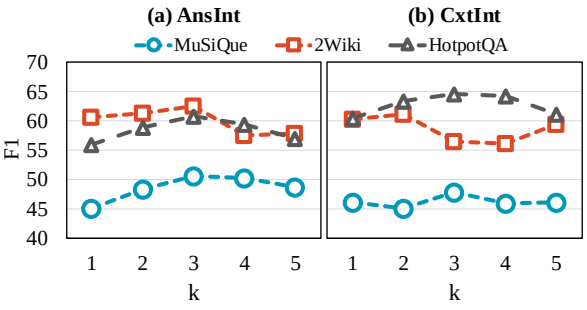

Figure 7: F1 (%) results w.r.t. different k values.

	Question	Gold facts	Sub-question 1	Sub-question 2	Final answer
NaiveRAG w/ QD	In what region of the country of S-Fone is The place of birth of John Phan located?	⇒ John Phan born October 10, 1974, in Da Nang, Vietnam. ⇒ South Central Coast (sometimes called South Central Region) consists of the independent municipality of Da Nang.	**Question**: What is the place of birth of John Phan? **Context**: John Phan born October 10, 1974, in Da Nang, Vietnam. **Answer**: Da Nang, Vietnam.	**Question**: In which region of S-Fone is this place located? context: S-Fone was a mobile communication operator in Vietnam that used the CDMA technology. Founded on 1 July 2003 in Ho Chi Minh City, Vietnam. **Answer**: Ho Chi Minh City, Vietnam	Ho Chi Minh City, Vietnam.
ChainRAG	In what region of the country of S-Fone is The place of birth of John Phan located?	⇒ John Phan born October 10, 1974, in Da Nang, Vietnam. ⇒ South Central Coast (sometimes called South Central Region) consists of the independent municipality of Da Nang.	**Question**: What is the place of birth of John Phan? **Context**: John Phan born October 10, 1974, in Da Nang, Vietnam. **Answer**: Da Nang, Vietnam.	**Question**: In which region of S-Fone is Da Nang, Vietnam located? **Context**: South Central Coast (sometimes called South Central Region) consists of the independent municipality of Da Nang and seven other provinces. **Answer**: South Central Coast	South Central Coast.

Table 3: Examples of the RAG process of NaiveRAG w/ QD and our ChainRAG from the MuSiQue dataset. Blue text represents correct and relevant information or answers, while red text indicates incorrect information.

LongRAG, ChainRAG shows a notable improvement in efficiency across all datasets, with an average reduction of about 17.3% in the number of calls. Although ITER-RETGEN is the most efficient method (second only to NaiveRAG), considering the overall performance of the three models across the three datasets, ChainRAG achieves a better balance between efficiency and performance.

4.6 Top-k Study

The selection of k sentences as seed sentences in the retrieval step is crucial, as it significantly influences the subsequent retrieval process. Given the fixed word limit, a smaller k value tends to retrieve more sentences from higher-order neighbors of the seed sentences, while a larger k value retrieves more sentences from lower-order neighbors. To determine the optimal value of k, we conduct comparative experiments. As shown in Figure 7, except for CxtInt's performance on the 2Wiki dataset, all other cases achieve the best results when $k = 3$. Considering the overall performance, we choose $k = 3$ as the default value.

4.7 Case Study

We present an example question from MuSiQue and compare the RAG process of NaiveRAG w/ QD and ChainRAG in Table 3. As NaiveRAG w/ QD does not rewrite the sub-question, its second sub-question lacks a clear entity, leading to retrieval errors and causing the "lost-in-retrieval" problems. This ultimately results in an incorrect answer. In the example, the context comes from the first chunk retrieved, which is also the source of the final answer. For ChainRAG, by using the answer from the first sub-question to complete the entity in the second sub-question, the retrieved information becomes more accurate, on which the LLM then provides the correct answer.

5 Conclusion and Future Work

In this paper, we investigate the "lost-in-retrieval" problems of RAG that occurs during multi-hop QA. We propose a progressive retrieval framework involving sentence graph construction, question decomposition, retrieval, and sub-question rewriting, which effectively enhances the retrieval performance, especially for sub-questions lacking clear entities. Our experiments and analysis on three challenging datasets—MuSiQue, 2Wiki, and HotpotQA—demonstrate that our method consistently outperforms baselines. Additionally, it demonstrates robustness and efficiency across different LLMs, showcasing its adaptability and potential for broader applications.

For future work,we plan to optimize efficiency by exploring lightweight graph traversal and adaptive termination strategies, reducing LLM calls and resource consumption. We also plan to enhance dynamic adaptability by developing dataset-specific edge construction policies to better align with diverse text structures.

Limitations

While our proposed ChainRAG framework has demonstrated significant improvements in resolving the "lost-in-retrieval" problems, several limitations should be acknowledged. First, although ChainRAG outperforms LongRAG and HippoRAG in efficiency, our iterative process of retrieval, subquestion rewriting increases the computational resources and time required compared to NaiveRAG. This may pose challenges for deployment in resource-constrained environments. Second, although robust across existing datasets, adapting ChainRAG to highly specialized domains requires further validation and potential adjustments to indexing strategies. Third, the accuracy of entity recognition and completion is critical for the success of ChainRAG. Errors in entity recognition can propagate through the retrieval and reasoning process, affecting the overall performance.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024*, pages 3119–3137.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. BGE m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *CoRR*, abs/2402.03216.

Xavier Daull, Patrice Bellot, Emmanuel Bruno, Vincent Martin, and Elisabeth Murisasco. 2023. Complex QA and language models hybrid architectures, survey. *CoRR*, abs/2302.09051.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph RAG approach to query-focused summarization. *CoRR*, abs/2404.16130.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *CoRR*, abs/2312.10997.

Michael R. Glass, Gaetano Rossiello, Md. Faisal Mahbub Chowdhury, Ankita Naik, Pengshan Cai, and Alfio Gliozzo. 2022. Re2g: Retrieve, rerank, generate. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022*, pages 2701–2715.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically inspired long-term memory for large language models. *CoRR*, abs/2405.14831.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: decoding-enhanced bert with disentangled attention. In *9th International Conference on Learning Representations, ICLR 2021*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing A multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020*, pages 6609–6625.

Ziyan Jiang, Xueguang Ma, and Wenhu Chen. 2024. LongRAG: Enhancing retrieval-augmented generation with long-context llms. *CoRR*, abs/2406.15319.

Patrick S. H. 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 *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020,*.

Jiawei Li, Yizhe Yang, Yu Bai, Xiaofeng Zhou, Yinghao Li, Huashan Sun, Yuhang Liu, Xingpeng Si, Yuhao Ye, Yixiao Wu, Yiguan Lin, Bin Xu, Ren Bowen, Chong Feng, Yang Gao, and Heyan Huang. 2024a. Fundamental capabilities of large language models and their applications in domain scenarios: A survey. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024*, pages 11116–11141.

Shilong Li, Yancheng He, Hangyu Guo, Xingyuan Bu, Ge Bai, Jie Liu, Jiaheng Liu, Xingwei Qu, Yangguang Li, Wanli Ouyang, Wenbo Su, and Bo Zheng. 2024b. Graphreader: Building graph-based agent to enhance long-context abilities of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12758–12786.

Zheng Liu, Chenyuan Wu, Ninglu Shao, Shitao Xiao, Chaozhuo Li, and Defu Lian. 2024. Lighter and better: Towards flexible context adaptation for retrieval augmented generation. *CoRR*, abs/2409.15699.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024*.

Seyed Mahed Mousavi, Simone Alghisi, and Giuseppe Riccardi. 2024. Is your LLM outdated? benchmarking llms & alignment algorithms for time-sensitive knowledge. *CoRR*, abs/2404.08700.

Jesse Mu, Xiang Li, and Noah D. Goodman. 2023. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. RAPTOR: recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations, ICLR 2024*. OpenReview.net.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 9248–9274.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Trans. Assoc. Comput. Linguistics*, 10:539–554.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023*, pages 10014–10037.

Ziting Wang, Haitao Yuan, Wei Dong, Gao Cong, and Feifei Li. 2024. CORAG: A cost-constrained retrieval optimization system for retrieval-augmented generation. *CoRR*, abs/2411.00744.

Jeff Wu, Long Ouyang, Daniel M. Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul F. Christiano. 2021. Recursively summarizing books with human feedback. *CoRR*, abs/2109.10862.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024a. RECOMP: improving retrieval-augmented lms with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations, ICLR 2024*.

Ziwei Xu, Sanjay Jain, and Mohan S. Kankanhalli. 2024b. Hallucination is inevitable: An innate limitation of large language models. *CoRR*, abs/2401.11817.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. 2024. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools. *CoRR*, abs/2406.12793.

Jintian Zhang, Cheng Peng, Mengshu Sun, Xiang Chen, Lei Liang, Zhiqiang Zhang, Jun Zhou, Huajun Chen, and Ningyu Zhang. 2024. Onegen: Efficient one-pass unified generation and retrieval for llms. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4088–4119.

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Lingpeng Kong, Jiajun Chen, Lei Li, and Shujian Huang. 2023. Multilingual machine translation with large language models: Empirical results and analysis. *CoRR*, abs/2304.04675.

Ziyuan Zhuang, Zhiyang Zhang, Sitao Cheng, Fangkai Yang, Jia Liu, Shujian Huang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2024. Efficientrag: Efficient retriever for multi-hop question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024*, pages 3392–3411.

A Dataset Statistics

Dataset statistics are presented in Table 4. To meet the requirements of LongRAG and HippoRAG, we have segmented the context within the dataset. Since these contexts are originally composed of

multiple integrated paragraphs, the original segmentation structure of the data can be accurately restored by regular expression matching.

Datasets	**MuSiQue**	**2Wiki**	**HotpotQA**
No. of Samples	200	200	200
No. of p	2,212	1,986	1,722
Avg. Length	11,214	4,887	9,151

Table 4: Statistics of the datasets used in our work. "Avg. Length" denotes the average word count.

B Implementation Details

We strictly follow the process outlined in the IterRetGen (Shao et al., 2023) paper and the prompts provided in its appendix for our reproduction. Additionally, we refer to the results in the paper and the EfficientRAG (Zhuang et al., 2024), selecting the third iteration results as the baseline, as the third iteration often produces results close to the optimal and exhibits a clear edge effect. For LongRAG and HippoRAG, we use their open-source code, follow the default settings, and conduct testing after switching to the unified model. Regarding the chunking process, LongRAG follows the default settings and segments based on word count, with a chunk size of 200. HippoRAG, on the other hand, does not apply any chunking.

We utilize the "en_core_web_sm" model from the spaCy library for entity extraction. This model is a small-scale model with approximately millions of parameters, designed for lightweight and efficient natural language processing tasks.

C Detailed Results of Ablation Study

Tables 5 and 6 present the complete results of our ablation study. After removing certain processes from ChainRAG, we observe varying degrees of performance degradation in QA tasks. When the rewriting phase is removed, the performance on MuSiQue and HotpotQA drops significantly, while the performance on 2Wiki dataset sees a minor decrease. This is mainly because many of the questions in the 2Wiki dataset, after being decomposed, result in two parallel sub-questions with no dependency, which avoids the "lost-in-retrieval" problems. For MuSiQue, removing the SS edges has the greatest impact on performance, while for 2Wiki, it is the removal of the EC edges that has the most significant effect. In HotpotQA, the SA edges have the largest impact when removed. When we remove the entire graph and implement the indexing process according to NaiveRAG, we observe a noticeable decline in performance, with the CxtInt method being more significantly affected. This indicates the effectiveness of our method, which involves building an index using sentences and entities and performing retrieval on the sentence graph.

Methods	MuSiQue	2Wiki	HotpotQA
ChainRAG (AnsInt)	50.54	62.55	60.73
w/o Rewriting	38.18	59.69	55.19
w/o EC edge	47.52	58.28	57.99
w/o SS edge	47.40	59.78	58.40
w/o SA edge	48.83	59.87	56.64
w/o Graph	47.97	60.96	58.05
ChainRAG (CxtInt)	47.87	56.54	64.59
w/o Rewriting	36.60	56.36	61.57
w/o EC edge	45.34	55.12	61.73
w/o SS edge	43.39	55.68	60.85
w/o SA edge	47.36	55.08	61.32
w/o Graph	40.18	54.76	60.19

Table 5: F1 results of ablation study.

Methods	MuSiQue	2Wiki	HotpotQA
ChainRAG (AnsInt)	37.00	52.00	46.00
w/o Rewriting	26.50	51.50	42.00
w/o EC edge	36.00	48.50	45.00
w/o SS edge	35.50	49.50	45.50
w/o SA edge	37.00	50.50	44.00
w/o Graph	35.00	45.00	42.00
ChainRAG (CxtInt)	38.50	50.50	50.00
w/o Rewriting	25.00	47.00	47.00
w/o EC edge	33.00	47.00	47.00
w/o SS edge	34.00	46.00	48.00
w/o SA edge	36.50	46.50	47.00
w/o Graph	27.00	39.50	44.00

Table 6: EM results of ablation study.

D Prompt Example

Figure 8 shows our prompt to guide LLMs in decomposing complex multi-hop questions into a series of minimal and necessary sub-questions. The goal of this decomposition is to ensure that each sub-question targets a specific and essential piece of information, thereby facilitating more accurate and efficient retrieval and reasoning processes. The *ex.* here refers to an example of a question decomposition that follows this logical progression.

You are a helpful AI assistant that helps break down questions into minimal necessary sub-questions.

Guidelines:
1. Only break down the question if it requires finding and connecting multiple distinct pieces of information.
2. Each sub-question should target a specific, essential piece of information.
3. Avoid generating redundant or overlapping sub-questions.
4. For questions about impact/significance, focus on:
   - What was the thing/event.
   - What was its impact/significance.
5. For comparison questions between two items (A vs B):
   - First identify the specific attribute being compared for each item.
   - Then ask about that attribute for each item separately.
   - For complex comparisons, add a final question to compare the findings.
6. Please consider the following logical progression:
   - Parallel: Independent sub-questions that contribute to answering the original question. Example: {*ex.*}.
   - Sequential: Sub-questions that build upon each other step-by-step. Example: {*ex.*}.
   - Comparative: Questions that compare attributes between items. Example: {*ex.*}.
7. Keep the total number of sub-questions minimal (usually 2 at most). Output format should be a JSON array of sub-questions. For example: {*examples of sub-questions*}.

Remember:
Each sub-question must be necessary and distinct. Do not create redundant questions. For comparison questions, focus on gathering the specific information needed for the comparison in the most efficient way.

Figure 8: Prompt for instructing LLMs to decompose the input question into several sub-questions.