

A Tree-based RAG Framework for Evidence-Intensive QA via Adaptive Planning and Topology-Aware Evidence Gathering

Songeun Lee^{1*} Kyungjin Min^{1,*} Injae Na^{2,*}
 Suyeong Lee³ Chiyong Kim³ Woohwan Jung^{1†}
 Korea University¹ Hanyang University² Hyundai Motor Company³
 {lse173, kjmin}@korea.ac.kr suhoij47@hanyang.ac.kr
 {lsy0620, cyoung.kim}@hyundai.com woohwan@korea.ac.kr

Abstract

Recent structured RAG methods leverage tree- or graph-based reasoning structures to improve multi-hop QA. However, they face key limitations in evidence-intensive QA, where answering a question requires synthesizing information scattered across dozens or even hundreds of documents: structural rigidity, which limits adaptive reasoning expansion, and topology-ignorant evidence gathering, which prevents effective integration of evidence across different reasoning nodes. To address these issues, we propose APT-RAG, an Adaptive Planning and Topology-aware evidence gathering RAG framework. Adaptive planning dynamically expands the reasoning structure based on question dependencies and evidence requirements, while topology-aware evidence gathering improves evidence coverage through sibling evidence reuse, direct retrieval, and evidence aggregation from child nodes. We further introduce evidence-guided batched answer generation to reduce significant generation overhead in evidence-intensive QA. In the experiments on evidence-intensive QA benchmarks, APT-RAG outperforms existing structured RAG methods. Our code is available at <https://github.com/hyudsl/APT-RAG>.

1 Introduction

Retrieval-Augmented Generation (RAG)-based reasoning approaches (Trivedi et al., 2023; Asai et al., 2024) have achieved remarkable success in multi-hop QA through iterative retrieval and reasoning. However, due to the linear reasoning path of these approaches, they often suffer from error propagation and limited exploration space. To address these limitations, recent structured RAG frameworks (Lee et al., 2024; Verma et al., 2025; Shi et al., 2026; Chen et al., 2026) utilize explicit rea-

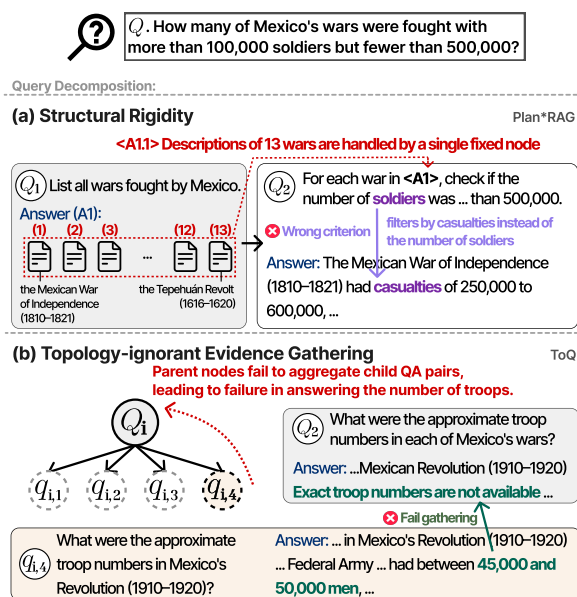


Figure 1: Failure cases of structured RAG frameworks caused by structural rigidity and topology-ignorant evidence gathering.

soning structures—such as trees or graphs—to effectively model the dependencies of sub-questions.

Despite their advantages, existing structured RAG frameworks face severe bottlenecks when transitioning to evidence-intensive QA (Wolfson et al., 2026; Amouyal et al., 2023). Unlike conventional multi-hop QA which requires only a few supporting pages on average (e.g., 2.0 pages in HotpotQA), evidence-intensive QA demands retrieving and synthesizing evidence distributed across dozens or hundreds of documents (e.g., 43.3 pages in MoNaCo). This unprecedented scale of evidence exposes two fundamental structural deficiencies in current structured RAG methods. First, existing methods, such as Plan*RAG (Verma et al., 2025), exhibit structural rigidity. Since they construct a fixed reasoning plan upfront, they cannot dynamically construct the reasoning structure. Consequently, a single node is often forced to process

*Equal contribution.

†Corresponding author.

a lot of evidence when the reasoning becomes complex. As illustrated in Figure 1(a), Plan*RAG (Verma et al., 2025) fails to answer question Q_2 since the node should synthesize 19 retrieved documents regarding 13 wars. Second, other method such as ToQ (Lee et al., 2024) suffer from topology-ignorant evidence gathering. Although these methods construct a reasoning architecture, they gather the evidence only in the root node and do not utilize the answers of the sub-questions in other nodes. As shown in Figure 1(b), ToQ fails to answer a sub-question q_2 even though it can be directly inferred from the answer of sub-question $q_{2,4}$.

To address these limitations, we propose APT-RAG (Adaptive Planning and Topology-aware evidence gathering) framework which answers evidence-intensive questions through a recursive depth-first process that constructs the reasoning tree. At each step of traversal, the planner decides whether to further decompose a node, answer it directly, or resolve it using previously gathered evidence. This planning allows the framework to flexibly adjust reasoning depth and breadth according to sub-problem complexity. APT-RAG also tightly couples tree traversal with hierarchical evidence gathering, preserving the reasoning topology through lateral gathering across sibling branches and vertical gathering from child to parent nodes. This topology-aware synthesis enables each node to leverage relevant evidence for answer generation within its reasoning context. Furthermore, APT-RAG incorporates evidence-guided answer batching to reduce redundant generation over overlapping evidence, substantially lowering LLM calls. On the evidence-intensive QA benchmarks MoNaCo (Wolfson et al., 2026) and QAMPARI (Amouyal et al., 2023), APT-RAG achieves higher answer F1 scores than existing structured RAG methods. Additionally, we observed that APT-RAG adaptively expands its reasoning tree and retrieves more supporting evidence as requirements increase while baseline methods retrieve a nearly constant number of documents regardless of requirements.

Our contributions are summarized as follows:

- We identify structural rigidity and topology-ignorant evidence gathering as two key limitations of existing structured RAG methods in evidence-intensive QA.
- We propose APT-RAG, a recursive framework that adaptively expands the reasoning tree and

gathers evidence along its topology.

- We introduce evidence-guided batched answer generation to reduce inference latency.
- We demonstrate that APT-RAG consistently outperforms structured RAG baselines on evidence-intensive QA benchmarks.

2 Related Work

Evidence-intensive QA. Evidence-intensive QA is a question answering task that requires substantially more evidence and reasoning steps than conventional multi-hop QA benchmarks, including HotpotQA and MuSiQue (Yang et al., 2018; Trivedi et al., 2022). Recent evidence-intensive QA benchmarks such as MoNaCo (Wolfson et al., 2026), QAMPARI (Amouyal et al., 2023), and DeepSearchQA (Gupta et al., 2026) require collecting and synthesizing evidence distributed in dozens to hundreds of documents, making it difficult to retrieve sufficient evidence through a single retrieval step using only the initial input question.

Structured RAG for Multi-Hop QA. Structured RAG improves multi-hop QA performance by decomposing complex questions into sub-questions and organizing the reasoning process using tree- or graph-based structures. Tree-based approaches, such as Lee et al. (2024); Shi et al. (2026); Jiao et al. (2026), decompose questions into sub-questions, construct a tree-structured reasoning architecture, and perform reasoning over the resulting tree structure. Graph-based approaches utilize Directed Acyclic Graphs to control retrieval and reasoning flows. In particular, Verma et al. (2025) constructs a fixed reasoning graph based on atomic sub-questions, while Chen et al. (2026) optimizes retrieval order using a logical dependency graph and topological sorting. However, although these methods effectively structure retrieval and reasoning at the sub-question level, they do not sufficiently address the dynamic expansion and management of evidence throughout the reasoning process, nor the hierarchical and lateral integration and synthesis of evidence required for evidence-intensive QA.

3 Method

We first formulate the evidence-intensive QA, then introduce APT-RAG, and finally present a method to improve its inference efficiency.

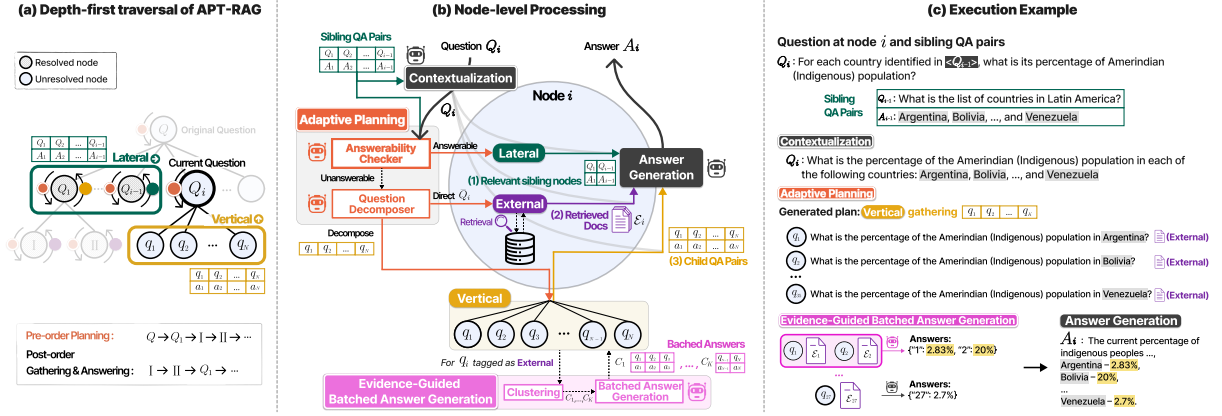


Figure 2: Overview of APT-RAG. (a) Depth-first traversal with pre-order planning and post-order evidence gathering. (b) Node-level processing. (c) An example of node-level processing.

3.1 Task Formulation

Given a question Q and a corpus $\mathcal{D} = \{d_1, \dots, d_{|\mathcal{D}|}\}$, the goal of evidence-intensive QA is to generate answer A by synthesizing information distributed in dozens to hundreds of documents. Similar to conventional multi-hop QA, the question Q is decomposed into a set of sub-questions $\mathcal{Q} = \{Q_1, Q_2, \dots, Q_N\}$, and a relevant evidence set $\mathcal{E}_{Q_i} \subset \mathcal{D}$ is retrieved for each sub-question Q_i . However, unlike conventional multi-hop QA, where the required evidence is often limited to a small number of paragraphs, evidence-intensive QA requires substantially larger evidence sets. Consequently, the model must aggregate evidence and intermediate reasoning results across many reasoning steps to construct the final evidence set $\mathcal{E}_Q$, from which the answer A is generated.

3.2 Overview of APT-RAG Framework

Figure 2 shows an overview of APT-RAG. Given an original question Q and a corpus $\mathcal{D}$, APT-RAG generates the final answer A by recursively constructing and resolving a reasoning tree in a depth-first manner. As shown in Figure 2(a), APT-RAG constructs a reasoning tree rooted at Q during execution, where each node represents a sub-question that should be resolved to answer its parent. Rather than constructing a complete reasoning structure upfront, APT-RAG interleaves planning, evidence gathering, and answer generation during recursive traversal. This process is performed repeatedly on all nodes in the tree and the final answer A to question Q is generated once the root node is resolved. At each node i , APT-RAG follows a four-step node-level procedure as shown in Figure 2(b): contextualization, adaptive planning, topology-aware evi-

dence gathering, and answer generation.

Contextualization. Following the previous structured RAG frameworks (Verma et al., 2025; Chen et al., 2026; Shi et al., 2026; Lee et al., 2024), APT-RAG can generate a sub-question Q_i that depends on the answers of preceding sibling nodes when the later sub-question is intended to use entities, values, or constraints identified by earlier ones. We represent the dependency with a place holder with question IDs such as $\langle Q_j \rangle$. In the contextualization step, we resolve the reference by using the input questions with sibling question-answer pairs. For example, in Figure 2(c), question Q_2 includes a reference to the answer of Q_1 , which contains a list of countries. Contextualizer produces a self-contained question based on the given sibling pair (Q_1, A_1) by using a prompt in Appendix F.1. Note that if there is no dependency, the contextualization step does not change the input questions Q_i .

Adaptive Planning. Next, APT-RAG determines how evidence should be gathered to resolve the input question Q_i . Depending on the dependency structure and evidence requirements of the question, the planner selects one of the following evidence gathering strategies: ① reusing QA pairs from preceding sibling nodes, ② directly retrieving evidence from the corpus, or ③ decomposing the question into sub-questions for fine-grained evidence exploration.

Topology-aware Evidence Gathering. Based on the planning result, we gather evidence necessary to answer the question Q_i through one of three topology-aware strategies: ① lateral gathering from preceding sibling QA pairs, ② external gathering through direct retrieval, and ③ vertical gathering

through aggregation of child QA pairs.

Answer Generation. Finally, each node generates an answer A_i to the input question Q_i based on the evidence $\mathcal{E}_{Q_i}$. The generated answer can subsequently serve as evidence for sibling or parent nodes during later reasoning steps, allowing APT-RAG to progressively accumulate and integrate information throughout the reasoning process. This recursive resolution process is repeatedly performed across the entire reasoning tree, ultimately producing the final answer A to the question Q .

Scope of the Evidence. Given a node i , every other node in the reasoning tree is in one of three regions: (i) its own subtree, (ii) its siblings’ subtrees, or (iii) its ancestors and their siblings’ subtrees. APT-RAG draws evidence from (i) and (ii)—via *vertical* and *lateral* gathering, respectively. Vertical gathering propagates each child’s answer along the tree topology rather than ingesting the full subtree at once, while lateral gathering reuses information already produced within the tree, preventing extra retrieval or decomposition at i . Region (iii) is delegated to the ancestors of i , which incorporate this evidence during their own answer generation and thereby avoid redundant aggregation across the tree. This topology-aware evidence flow allows each node to use only locally relevant evidence while avoiding redundant aggregation across the tree.

3.3 Adaptive Planning

Since the granularity and source of required evidence vary with the structural characteristics of each question, APT-RAG adaptively determines an appropriate evidence gathering strategy during the planning step. This stage consists of an answerability checker and a decomposer.

Given the input question Q_i , the answerability checker first determines whether it can be resolved solely using QA pairs from previously processed sibling nodes. If so, the lateral gathering is performed subsequently without further retrieval or decomposition, following the answerable path in Figure 2(b). Otherwise, Q_i is passed to the decomposer, which determines whether to keep the question as is or decompose it into sub-questions. If the question is atomic and sufficient to search evidence through a single retrieval step, it is rewritten into a concise retrieval query $\bar{Q}_i$; otherwise, it is decomposed into sub-questions, which are then organized as child nodes of Q_i . The prompts for the planners

are in Appendix F.2

3.4 Topology-aware Evidence Gathering

Lateral Gathering. When question Q_i is determined to be answerable using preceding sibling QA pairs during the planning stage, APT-RAG gathers sibling QA pairs relevant to Q_i as evidence. Such questions often depend on information already collected or generated from preceding nodes at the same depth, making them solvable without additional retrieval or decomposition. Therefore, by reusing information obtained from preceding sibling nodes, lateral gathering effectively suppresses redundant tree expansion caused by repetitive retrieval and decomposition over already explored information (Jiao et al., 2026). This forms the first base case of the recursive construction.

External Gathering. When the evidence required to answer Q_i can be directly obtained through retrieval without question decomposition, APT-RAG performs cosine similarity-based semantic retrieval between embedding vectors as $\text{score}(\bar{Q}_i, d_j) = \frac{\mathbf{e}(\bar{Q}_i)^\top \mathbf{e}(d_j)}{\|\mathbf{e}(\bar{Q}_i)\| \|\mathbf{e}(d_j)\|}$, using the retrieval query $\bar{Q}_i$ generated during the planning stage. The retrieved top- k documents are then directly used as evidence for the input question. This forms the second base case of the recursive construction.

Vertical Gathering. Vertical gathering is performed when the question Q_i is decomposed into sub-questions. A single retrieval query may fail to sufficiently cover the broadly dispersed evidence required by complex questions. To address this, APT-RAG decomposes the question into multiple sub-questions and gathers evidence at the child-node level, allowing each child node to focus on a narrower evidence scope. The QA pairs generated from the child nodes are then aggregated to expand the evidence coverage required to resolve the parent question. This forms the recursive step of the construction. When vertical gathering creates multiple independent child nodes that are planned for external gathering, APT-RAG can further batch their answer generation using the evidence-guided clustering method described in Section 3.5.

Example. Figure 3 illustrates how the three gathering strategies operate within a single reasoning tree. The initial question is first decomposed into multiple child nodes, where q_1 identifies candidate big cat species and q_2 is further decomposed into fine-grained sub-questions to retrieve population

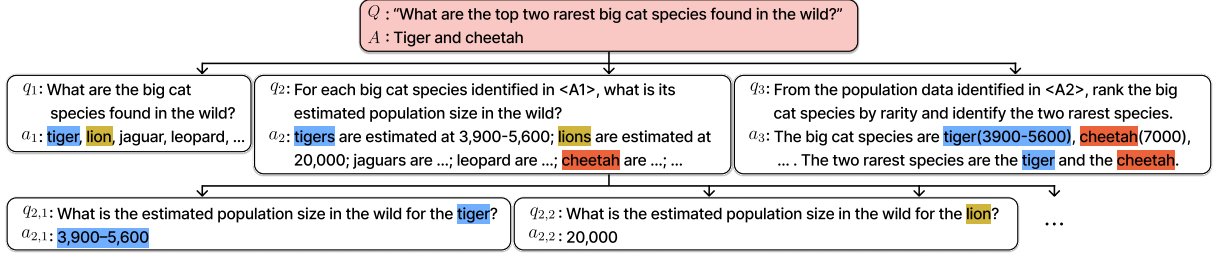


Figure 3: An example of the reasoning process of APT-RAG

statistics for each species through external gathering. The resulting lower-level QA pairs are aggregated through vertical gathering and reused as evidence for higher-level nodes. Meanwhile, q_3 uses the QA pair generated at q_2 through lateral gathering to rank species by rarity, producing the final answer to the initial question.

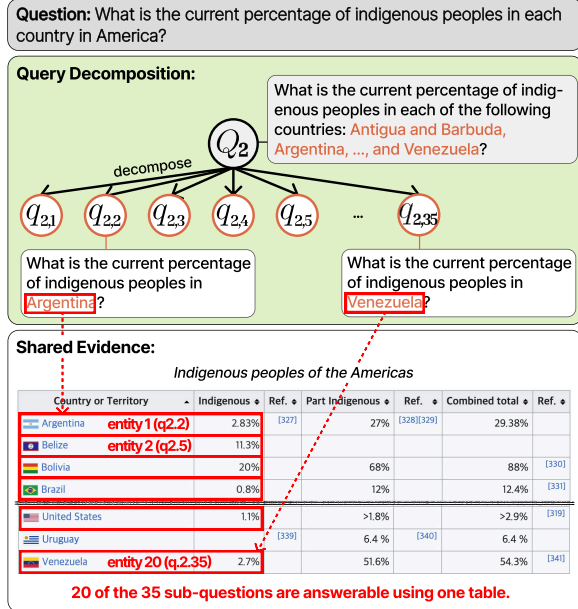


Figure 4: An example of entity-specific sibling nodes from recall-oriented decomposition retrieving highly overlapping evidence from a shared Wikipedia page.

3.5 Evidence-Guided Batched Answer Generation

When vertical gathering decomposes a node into multiple child sub-questions, many of the resulting sibling sub-questions often share the same evidence. This often occurs when a parent question asks for the same attribute across multiple entities. For example, as shown in Figure 4, the parent question is decomposed into 35 sub-questions asking about the percent of indigenous peoples in 35 different the Americas. Among them, 20 sub-questions can be answered by using a single shared evidence.

In that case, instead of independently answering each sub-question, APT-RAG jointly answers sub-questions that share highly overlapping evidence. It substantially reduces the number of LLM calls as well as the number of input tokens, thereby lowering latency for evidence intensive questions that require broad decomposition.

To do this, we slightly modify our pre-order planning schedule for the optimization. When APT-RAG decomposes a node into multiple child sub-questions, it first performs adaptive planning for all child nodes before starting evidence gathering for first child node. It then identifies the child nodes that are independent of preceding sibling answers and assigned to external gathering. These nodes can be answered jointly because they do not rely on one another and use only externally retrieved evidence. We cluster these candidate sub-questions based on the evidence overlap and answer the questions together with the union of their retrieved evidence. The prompt for the batched answer generation is provided in Appendix F.4. In the rest of this section, we first formulate the sub-question clustering problem and then propose an approximate algorithm by reducing a relaxed clustering problem to the minimum clique partition problem.

Problem Formulation. Let $\mathcal{B} = \{q_1, \dots, q_n\}$ be the set of sub-questions selected for batching where each $q_i \in \mathcal{B}$ has a set of retrieved documents $\mathcal{E}_i$. We aim to partition $\mathcal{B}$ into the minimum number of clusters $\{C_k\}_{k=1}^K$ while satisfying a context length constraint and an evidence-similarity constraint.

First, we limit the length of the batched prompt of each cluster by the following length constraint: $\ell\left(\bigcup_{i \in C_k} \mathcal{E}_i\right) < \tau^L$ where $\ell(\cdot)$ denotes token length and L is the maximum evidence length allowed in a batch. Second, sub-questions in the same cluster should have sufficiently overlapping evidence. We measure the evidence similarity between two sub-questions q_i and q_j using the Jaccard similarity of their evidence sets: $s_{i,j} = \frac{|\mathcal{E}_i \cap \mathcal{E}_j|}{|\mathcal{E}_i \cup \mathcal{E}_j|}$. Ac-

cordingly, when sub-questions are grouped into a cluster C_k , the evidence fed to the language model is the deduplicated union $\bigcup_{i \in C_k} \mathcal{E}_i$. Greater overlap produces a more compact union, allowing more sub-questions to be handled per call while reducing redundant evidence processing. By putting these together, the constrained sub-question clustering problem is formulated as:

$$\text{minimize } K \quad (1a)$$

$$\text{subject to} \quad (1b)$$

$$\bigcup_{k=1}^K C_k = \mathcal{B}, C_i \cap C_j = \emptyset (\forall i \neq j), \quad (1c)$$

$$s_{ij} > \tau^S \quad \forall i, j \in C_k, \forall k \quad (1d)$$

$$\ell \left(\bigcup_{i \in C_k} \mathcal{E}_i \right) < \tau^L \quad \forall k \quad (1e)$$

where the constraint in Eq. (1c) ensures every sub-question belongs to exactly one cluster.

Relaxation, Reduction, and Approximation.

We reduce the constrained sub-question clustering problem to the *Clique Covering Problem (CCP)* (Chalupa, 2013), which seeks a minimum clique partition of the graph, by relaxing the context-window constraint. Since the goal of evidence-guided clustering is to reduce LLM inference latency, the clustering step itself should be lightweight. We therefore further reduce CCP to a graph coloring problem on the complement graph $\bar{G}$, and solve it with a modified greedy graph coloring algorithm with largest-first ordering (Welsh and Powell, 1967). The modification extends the original verification to additionally enforce the context-window constraint $\ell \left(\bigcup_{i \in C_k \cup \{v\}} \mathcal{E}_i \right) < \tau^L$ which yields a greedy solution $\hat{K}$ satisfying $\tilde{K}^* \leq \hat{K}^* \leq \hat{K}$. Details of the approximation algorithm and proofs for the problem reduction are provided in Appendix E.

Batched Answer Generation. For each cluster C_k , APT-RAG concatenates the corresponding questions and the deduplicated documents into a single prompt and jointly generates answers:

$$\{a_i\}_{i \in C_k} = \text{Generator} \left(\{q_i\}_{i \in C_k} \parallel \bigcup_{i \in C_k} \mathcal{E}_i \right).$$

This module does not modify the reasoning tree structure or evidence semantics. Instead, it improves inference efficiency by reducing redundant

generation calls among sibling leaves that share overlapping retrieved evidence.

Statistic	MoNaCo	QAMPARI
Num. Questions	1,315	1,000
Avg. Question Length (Chars)	84.3	55.8
Avg. Answer Length (Chars)	241.2	274.3
Avg. Gold Document	43.3	13.0
Size. Corpus (Document)	784,776	6,311,809
Size. Corpus (Chunks)	1,212,878	25,856,230

Table 1: Data statistics.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate APT-RAG on evidence-intensive QA benchmarks with different evidence scales: (1) MoNaCo (Wolfson et al., 2026), a benchmark of complex information-seeking questions requiring synthesis across dozens to hundreds of evidence pages (43.3 pages on average); and (2) QAMPARI (Amouyal et al., 2023), a multi-answer open-domain QA benchmark where answers are distributed across multiple passages, requiring aggregation over 13.0 evidence pages on average. Statistics of the datasets are summarized in Table 1. Dataset examples are presented in Appendix A, and detailed corpus construction and indexing are described in Appendix B.

Baselines. We compare our method with two categories of baselines: (1) basic baselines, including **LLM-Only** and **Naïve RAG**; and (2) structured RAG baselines, including tree-based approaches such as **ToQ** (Lee et al., 2024) and **RT-RAG** (Shi et al., 2026), and graph-based approaches such as **Plan*RAG** (Verma et al., 2025) and **Logi-cRAG** (Chen et al., 2026). Due to the high cost of inference RT-RAG, we selected a representative 300-example subset of each benchmark for evaluating to keep computational costs manageable. We report the results in Appendix G.

Evaluation Measures. We evaluate answer quality using macro-averaged answer precision, recall, and F1, following the evaluation protocols of MoNaCo and QAMPARI. We also report retrieval recall (Ret. R), defined as the fraction of retrieved gold documents, to measure evidence coverage. We further evaluate inference cost by measuring the number of LLM calls, input and output tokens, and end-to-end latency per question, as detailed in Appendix H.

Models	Methods	MoNaCo				QAMPARI			
		Ans. P	Ans. R	Ans. F1	Ret. R	Ans. P	Ans. R	Ans. F1	Ret. R
QWEN3-4B-INST	LLM-Only	31.19	28.47	28.08	—	9.32	4.25	4.97	—
	NaiveRAG	40.36	34.53	34.98	31.68	<u>31.65</u>	16.44	18.58	11.74
	Plan*RAG	30.88	26.55	27.02	31.96	12.41	4.19	5.40	5.70
	LogicRAG	<u>43.83</u>	34.09	35.75	27.11	29.45	7.05	9.87	10.23
	ToQ	41.31	<u>36.22</u>	<u>36.32</u>	<u>33.07</u>	31.18	<u>16.78</u>	<u>18.76</u>	<u>11.96</u>
	APT-RAG (Ours)	46.75	39.95	40.69	40.12	34.12	18.78	20.91	14.08
QWEN3-30B-INST	LLM-Only	46.43	41.92	41.82	—	19.76	7.35	9.40	—
	NaiveRAG	51.98	<u>46.56</u>	46.64	31.68	33.43	18.46	20.46	11.74
	Plan*RAG	52.40	<u>46.54</u>	<u>47.03</u>	<u>39.28</u>	33.08	15.31	17.67	<u>16.54</u>
	LogicRAG	55.70	44.62	46.41	21.64	34.60	7.90	11.23	8.47
	ToQ	52.19	46.51	46.66	31.91	33.85	<u>19.66</u>	<u>21.37</u>	11.42
	APT-RAG (Ours)	<u>55.32</u>	50.75	50.84	50.79	<u>34.51</u>	22.08	23.28	19.61

Table 2: Performance comparison on evidence-intensive QA benchmarks.

Implementation details. To ensure fair comparison, all methods use the same retrieval setup and backbone LLMs. We use Qwen3-Embedding-0.6B as the embedding model in the dense passage retrieval and retrieve the top $k = 20$ passages for all experiments. Please refer to Appendix B for the detailed corpus construction and indexing. As backbone language models, we use Qwen3-4B¹ and Qwen3-30B². For MoNaCo, which requires LLM-as-a-judge evaluation, we use GPT-5.4 as the evaluator for the main results. Gemini-3.5-Flash yields comparable results, which are reported in Appendix J. Unless otherwise specified, all subsequent analyses are conducted on MoNaCo using Qwen3-30B-Inst as the backbone model. For the evidence-guided clustering in EC, we set the similarity threshold to $\tau^S = 0.0$, requiring at least one overlapping passage between subquestions in the same cluster, to maximize the latency reduction. We set the context-window budget to $\tau^L = 150,000$ tokens for Qwen3-4B-Inst and $\tau^L = 50,000$ tokens for Qwen3-30B-Inst. All prompts used in our experiments are provided in Appendix F. All experiments were conducted on an NVIDIA RTX PRO 6000 GPU.

4.2 Main Results

Table 2 shows the answer quality and retrieval performance of APT-RAG and the baselines. APT-RAG achieves the best overall performance across all benchmarks and model scales, demonstrating its effectiveness for evidence-intensive QA. All answer F1 gains of APT-RAG over the baselines are

¹Qwen3-4B-Instruct-2507

²Qwen3-30B-A3B-Instruct-2507

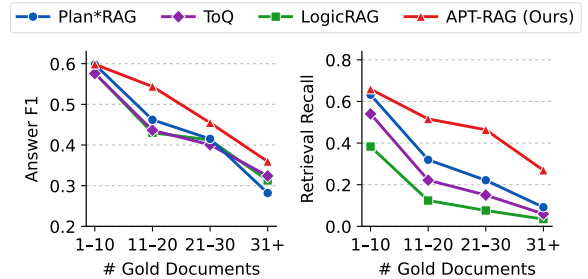


Figure 5: Comparison of answer F1 and retrieval recall across gold documents scale on MoNaCo ($\uparrow$ is better).

statistically significant ($p \leq 0.001$). Appendix I reports each gain with its 95% confidence interval. On MoNaCo under the 30B setting, APT-RAG outperforms the baseline, Plan*RAG, by 8% in answer F1, suggesting more effective evidence gathering for large-scale evidence synthesis. Specifically, APT-RAG achieves a retrieval recall of 50.79, while Plan*RAG reaches only 39.28. In contrast, although LogicRAG achieves the highest answer precision, its limited evidence coverage results in the lowest retrieval recall and ultimately the lowest answer F1 among non-LLM-only methods. Under the 4B setting, most baselines show substantial degradation compared to their 30B counterparts, whereas APT-RAG maintains relatively stable performance. A similar trend is observed on QAMPARI.

One key challenge in evidence-intensive QA is maintaining reasoning performance as the number of required evidence documents increases. Figure 5 compares performance trends with respect to the number of required gold evidence pages. Although all methods degrade as evidence requirements grow,

APT-RAG consistently maintains the highest performance with a smaller drop than the baselines. In particular, APT-RAG achieves an answer F1 in the 21–30 page range comparable to the baselines in the 11–20 page range, while maintaining stable retrieval recall even in the 31+ range. These results suggest that APT-RAG progressively expands the reasoning tree during inference, broadening retrieved evidence coverage and improving performance in evidence-intensive QA.

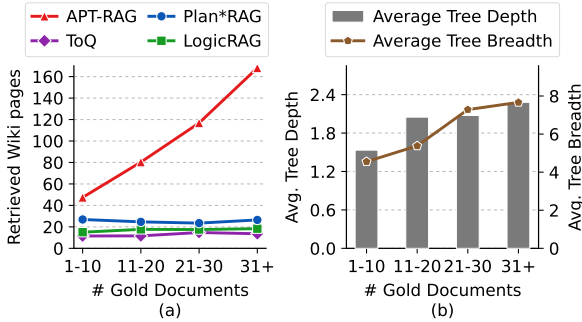


Figure 6: Adaptive retrieval across the evidence scale on MoNaCo. (a) Number of retrieved documents for each method. (b) Average tree depth and branching factor of APT-RAG.

4.3 Analysis

Adaptive Retrieval Scaling. Figure 6(a) shows the number of retrieved documents with varying gold evidence scales. While all baselines retrieve a nearly constant number of documents regardless of the required evidence volume, APT-RAG increases the number of retrieved documents as the number of gold documents grows. This limitation stems from the structural rigidity of methods such as Plan*RAG and LogicRAG, whose fixed reasoning structures restrict further exploration, while ToQ prematurely terminates retrieval based on inaccurate sufficiency judgments (Appendix C.1). Figure 6(b) further shows that the average tree depth and breadth of APT-RAG increase together with the required evidence scale. In contrast, the baselines’ depth and breadth remain nearly constant across the same range (Appendix G). It indicates that APT-RAG adaptively expands both its retrieval space and reasoning structure according to query complexity.

4.4 Ablation Studies

Table 3 presents a component-wise ablation of APT-RAG. We organize the ablations into two groups: performance-oriented modules, which affect evidence coverage and answer quality, and efficiency-

Method	F1 $\uparrow$	Ret. R $\uparrow$	Latency $\downarrow$
w/o Performance-oriented modules			
w/o AP ($B = 2$)	49.61	37.48	46.56
w/o AP ($B = \text{Avg.}$)	49.79	51.60	314.97
w/o VG	46.64	31.68	1.90
w/o Efficiency-oriented modules			
w/o EC & LG	53.96	53.29	183.95
w/o LG	52.24	54.56	178.55
w/o EC	52.34	50.69	113.90
Full framework			
APT-RAG (Ours)	50.84	50.79	104.59

Table 3: Component-wise ablation of APT-RAG. AP, VG, LG, and EC denote adaptive planning, vertical gathering, lateral gathering, and evidence-guided clustering, respectively.

oriented modules, which reduce inference cost.

Performance-oriented Modules. We evaluate adaptive planning by comparing APT-RAG with fixed-breadth variants. Fixing the tree breadth to 2 yields low latency but fails to capture varying evidence requirements across queries, resulting in the lowest answer F1 and retrieval recall. In contrast, fixing the tree breadth to the average breadth of 6 unnecessarily expands the tree for simple queries, substantially increasing latency. Ours achieves the highest answer F1 and comparable retrieval recall to the average-breadth variant, while using less than half of its latency. Additionally, removing vertical gathering, which restricts the framework to a single node and reduces it to NaiveRAG, decreases answer F1 by 4.20 points. These results suggest that APT-RAG effectively adapts the search structure to query-specific evidence requirements, balancing retrieval coverage and inference efficiency. Appendix J further reports the same ablation under fine-grained evidence criterion.

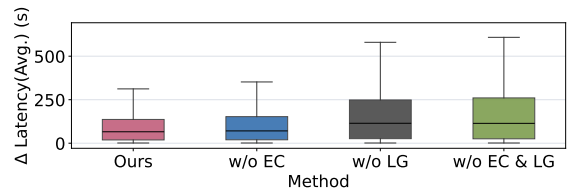


Figure 7: Effectiveness of evidence-guided clustering (EC) and lateral evidence gathering (LG).

Efficiency-oriented Modules. We analyze the impact of lateral gathering and evidence-guided clustering on inference cost. As shown in Table 3, lateral gathering and evidence-guided clustering reduce average inference latency by 41.4% and 8.2%, respectively. For lateral gathering, Figure 7

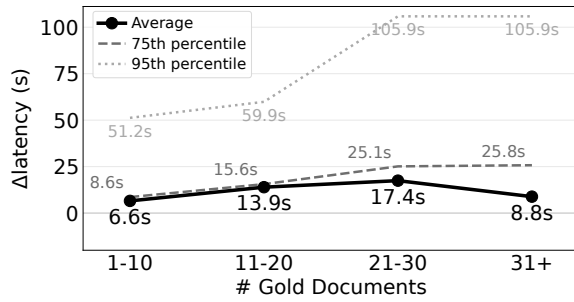


Figure 8: Effect of gold documents scale on latency reduction between APT-RAG and *w/o EC*.

shows that it controls worst-case latency by reducing redundant tree expansion caused by repetitive retrieval and decomposition over already explored information. For evidence-guided clustering, Figure 8 shows its latency reduction across varying gold document scales. While the average latency reduction increases relatively gradually,³ the latency reduction at the 75th and 95th percentiles becomes substantially larger as the number of gold documents increases. In particular, at the 95th percentile, it achieves large latency reductions even for queries exhibiting extreme latency exceeding 100 seconds. It shows that evidence-guided clustering effectively reduces long-tail inference latency.

5 Conclusion

We propose APT-RAG, an adaptive planning and topology-aware evidence gathering RAG framework for evidence-intensive QA. By dynamically adjusting the search space and utilizing the reasoning topology, APT-RAG overcomes the structural rigidity and topology-ignorant evidence gathering issues of existing structured RAG methods. Experiments show that APT-RAG consistently improves answer quality and retrieval recall on evidence-intensive QA benchmarks. These results suggest that recursive, topology-aware evidence gathering is a promising direction for answering questions over large, dispersed document spaces.

6 Limitations

In this study, we focus on evidence-intensive QA tasks that require large-scale evidence aggregation, which can incur substantial inference cost and latency. Although evidence-aware batched answer

³The average reduction decreases in the 31+ range because the larger merged evidence in this regime raises long-context processing cost, which outweighs the latency saved by clustering.

generation reduces this overhead, the cost still increases as the amount of evidence and reasoning branches grows. This could be further mitigated through techniques such as tree pruning to control unnecessary branches and evidence expansion.

In addition, the performance of APT-RAG depends on the quality of the planning process that constructs the reasoning tree. In particular, inaccurate output from the query decomposer or answerability checker may generate unnecessary reasoning branches or propagate errors. Future work could further improve performance by training planning-related modules.

Acknowledgments

This work was supported by Hyundai Motor Group through the 2025 Future Technology Research Program; the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2025-16070246); the National R&D Program for Cancer Control through the National Cancer Center (NCC) funded by the Ministry of Health & Welfare, Republic of Korea (RS-2025-02264000); the IITP(Institute of Information & Communications Technology Planning & Evaluation)-ITRC(Information Technology Research Center) grant funded by the Korea government(Ministry of Science and ICT)(IITP-2026-RS-2024-00436857); the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2019-II190079, Artificial Intelligence Graduate School Program (Korea University)).

References

- Samuel Amouyal, Tomer Wolfson, Ohad Rubin, Ori Yoran, Jonathan Herzig, and Jonathan Berant. 2023. [QAMPARI: A benchmark for open-domain questions with many answers](#). In *Proceedings of the Third Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)*, pages 97–110, Singapore. Association for Computational Linguistics.
- Akari Asai, Zequi Wu, Yizhong Wang, Avi Sil, and Hananeh Hajishirzi. 2024. [Self-rag: Learning to retrieve, generate, and critique through self-reflection](#). In *International Conference on Learning Representations*, volume 2024, pages 9112–9141.
- David Chalupa. 2013. [An analytical investigation of block-based mutation operators for order-based stochastic clique covering algorithms](#). In *Proceedings of the 15th Annual Conference on Genetic*

- and Evolutionary Computation, GECCO '13, page 495–502, New York, NY, USA. Association for Computing Machinery.
- Shengyuan Chen, Chuang Zhou, Zheng Yuan, Qinggang Zhang, Zeyang Cui, Hao Chen, Yilin Xiao, Jiannong Cao, and Xiao Huang. 2026. [You Don't Need Pre-Built Graphs for RAG: Retrieval Augmented Generation with Adaptive Reasoning Structures](#). *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(36):30270–30278.
- Nikita Gupta, Riju Chatterjee, Lukas Haas, Connie Tao, Andrew Wang, Chang Liu, Hidekazu Oiwa, Elena Gribovskaya, Jan Ackermann, John Blitzer, Sasha Goldshtein, and Dipanjan Das. 2026. [DeepSearchQA: Bridging the Comprehensiveness Gap for Deep Research Agents](#). *Preprint*, arXiv:2601.20975.
- Shuguang Jiao, Xinyu Xiao, Yunfan Wei, Shuhan Qi, Chengkai Huang, Quan Z. Sheng, and Lina Yao. 2026. [Prunerag: Confidence-guided query decomposition trees for efficient retrieval-augmented generation](#). In *Proceedings of the ACM Web Conference 2026*, WWW '26, page 1923–1934, New York, NY, USA. Association for Computing Machinery.
- 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.
- Dongyub Lee, Younghun Jeong, Hwa-Yeon Kim, Hongyeon Yu, Seunghyun Han, Taesun Whang, Seungwoo Cho, Chanhee Lee, Gunsu Lee, and Youngbum Kim. 2024. [Tree-of-Question: Structured Retrieval Framework for Korean Question Answering Systems](#). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 406–418, Mexico City, Mexico. Association for Computational Linguistics.
- Yuling Shi, Maolin Sun, Zijun Liu, Mo Yang, Yixiong Fang, Tianran Sun, and Xiaodong Gu. 2026. [Reasoning in Trees: Improving Retrieval-Augmented Generation for Multi-Hop Question Answering](#).
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. [MuSiQue: Multi-hop Questions via Single-hop Question Composition](#). *Transactions of the Association for Computational 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)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.
- Prakhar Verma, Sukruta Prakash Midigeshi, Gaurav Sinha, Arno Solin, Nagarajan Natarajan, and Amit Sharma. 2025. [Plan*RAG: Efficient Test-Time Planning for Retrieval Augmented Generation](#).
- D. J. A. Welsh and M. B. Powell. 1967. [An upper bound for the chromatic number of a graph and its application to timetabling problems](#). *The Computer Journal*, 10(1):85–86.
- Douglas B. West. 2001. *Introduction to Graph Theory*, second edition. Prentice Hall, Upper Saddle River, N.J.
- Tomer Wolfson, Harsh Trivedi, Mor Geva, Yoav Goldberg, Dan Roth, Tushar Khot, Ashish Sabharwal, and Reut Tsarfaty. 2026. [MoNaCo: More Natural and Complex Questions for Reasoning Across Dozens of Documents](#). *Transactions of the Association for Computational Linguistics*, 14:23–46.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William 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, Brussels, Belgium. Association for Computational Linguistics.

A Datasets

MoNaCo

Question. How did each of the Roman Emperors meet his end? How many years did they reign?

Answer.

- 'Basil I', 'a fever contracted after a serious hunting accident when his belt was caught in the antlers of a deer...', '18'
- 'Titus', 'fever', '2'
- 'Romanos III Argyros', 'poison administered by his wife', '5'
- 'Saloninus', 'Postumus killed Saloninus', '0'
- ...

QAMPARI

Question. What are the stage names of some famous drag queens?

Answer.

- 'Acid Betty'
- 'Adore Delano'
- 'Aja'
- 'A'keria Chanel Davenport'
- 'Akihiro Miwa'
- ...

Method	Structure	Intermediate Synthesis Scope	Final Synthesis Scope
Plan*RAG	DAG	Retrieved docs	All nodes' QA pairs
LogicRAG	DAG	N/A	Accumulated summary
Tree-of-Question	Tree	Retrieved docs	Validated nodes' QA pairs
RT-RAG	Binary tree	Retrieved docs; child QA pairs	Root's child QA pairs
APT-RAG (Ours)	Tree	Retrieved docs; sibling QA pairs; child QA pairs	Root's child QA pairs

Table 4: Structural comparison of evidence synthesis across structured RAG methods. *Intermediate Synthesis Scope* lists the evidence each intermediate node consumes when generating its sub-answer; *N/A* indicates that the method does not generate sub-answers. *Final Synthesis Scope* lists the evidence aggregated for final answer generation.

B Corpus and Indexing

Corpus. We use English Wikipedia as the retrieval source for both benchmarks. For QAMPARI, we adopt the official pre-chunked Wikipedia corpus released with the benchmark. For MoNaCo, we construct the corpus by including all benchmark gold pages and sampling additional non-gold pages from the January 2026 English Wikipedia dump. We then chunk the selected pages using a rule-based chunker that preserves document structure: narrative text is split at sentence boundaries, list and infobox entries at line boundaries, and tables into row-level chunks with the header row retained. Each passage is capped at 512 tokens using tiktoken and prepended with its article title and enclosing section heading, resulting in a collection of more than one million passages.

Indexing. We adopt a bi-encoder dense retriever (Karpukhin et al., 2020), using Qwen3-Embedding-0.6B without additional training. All passage embeddings are indexed with faiss IndexHNSWSQ under SQ8 scalar quantization ($M=64$, efConstruction=400, efSearch=128, L2 distance), with the SQ8 quantizer trained on 100,000 sampled vectors. For MoNaCo, we additionally apply a section-level context expansion. Upon retrieving a chunk, we collect additional chunks that share the same Wikipedia page, section heading, and content type (i.e., sentence, table, list, or infobox), and concatenate them with the retrieved chunk up to a maximum of 2,560 tokens.

C Baselines

C.1 Method Details

Table 4 compares the decomposition structure and evidence synthesis policy of each structured RAG baseline. The comparison describes the synthesis behavior used when the input question is decomposed into sub-questions; without decomposition,

all methods generate the answer directly from retrieved documents.

Plan*RAG. Plan*RAG (Verma et al., 2025) represents the decomposition as a directed acyclic graph, where nodes are sub-questions and edges encode answer dependencies. Each intermediate node generates its sub-answer from retrieved documents. The final answer is generated from the QA pairs of all nodes in the graph.

LogicRAG. LogicRAG (Chen et al., 2026) also uses a directed acyclic graph, but it does not generate a sub-answer for each node. It traverses nodes in topological order and incrementally refines a single information summary from the documents retrieved at each node. The final answer is generated from this accumulated summary.

Tree-of-Question. Tree-of-Question (Lee et al., 2024) organizes sub-questions into a tree. Each node generates its sub-answer from retrieved documents, and an evaluator filters unreliable nodes during traversal. The final answer aggregates the QA pairs of all validated nodes.

RT-RAG. RT-RAG (Shi et al., 2026) recursively splits each question into two children, forming a binary tree of bounded depth. Each intermediate node generates its answer from retrieved documents and child QA pairs. The final answer is generated from the QA pairs of the root's direct children. In its official setting, RT-RAG builds multiple candidate trees for each question, samples each tree several times to select a consensus structure, and repeatedly rewrites sub-questions during retrieval. This repeated tree construction, sampling, and rewriting increases the number of LLM calls, making RT-RAG slow in our evidence-intensive setting.

C.2 Reproduction Details

We re-implement Plan*RAG and ToQ using the algorithms and prompts described in their origi-

Algorithm 1: APT-RAG without Batched Answer Generation

```

function APT-RAG( $Q_i, \mathcal{P}_i^S$ ):
     $Q_i \leftarrow \text{Contextualizer}(Q_i, \mathcal{P}_i^S)$ 
    Evidence  $\mathcal{E}_{Q_i} \leftarrow \emptyset$ 
    if AnswerabilityCheck( $Q_i, \mathcal{P}_i^S$ ) then
        // Lateral gathering
         $\mathcal{E}_{Q_i} \leftarrow \mathcal{P}_i^S$ 
    else
        Subquestions  $\mathcal{Q}_i \leftarrow \text{Decomposer}(Q_i)$ 
        if  $|\mathcal{Q}_i| = 0$  then
            // External gathering
             $\mathcal{E}_{Q_i} \leftarrow \text{Retriever}(Q_i)$ 
        else
            // Vertical gathering
            for  $k \leftarrow 1$  to  $|\mathcal{Q}_i|$  do
                 $a_k \leftarrow \text{APT-RAG}(q_k, \mathcal{P}_{i, < k}^C)$ 
                Child QA Pairs  $\mathcal{P}_i^C \leftarrow \langle (q_k, a_k) \rangle_{k=1}^{|\mathcal{Q}_i|}$ 
             $\mathcal{E}_{Q_i} \leftarrow \mathcal{P}_i^C$ 
        // Answer generation
         $A_i \leftarrow \text{Generator}(Q_i, \mathcal{E}_{Q_i})$ 
    return  $A_i$ 

```

nal papers. To isolate the effect of the decomposition structure from the answer-generation policy, both use the APT-RAG final-answer prompt (Appendix F.4); it outperformed the original Plan*RAG prompt, and ToQ does not specify one. We run LogicRAG⁴ and RT-RAG⁵ using the official implementations released by the authors. We keep their original reasoning algorithms and final-answer prompts.

D Algorithms

We present the pseudocode for the three algorithms used in APT-RAG.

D.1 Inference without Batched Answer Generation

Algorithm 1 formalizes the recursive depth-first traversal introduced in §3.2. Each call $\text{APT-RAG}(Q_i, \mathcal{P}_i^S)$ resolves one node by executing the four-step node-level procedure: contextualization, adaptive planning (§3.3), topology-aware evidence gathering (§3.4), and answer generation. The planner directs the node to the lateral, external, or vertical branch, and the vertical branch expands the tree by recursing on each generated subquestion. The algorithm uses four symbols that are specific to the pseudocode: $\mathcal{P}_i^S$ denotes the preceding sibling QA pairs supplied to node i , $\mathcal{D}_i$ the documents returned by the retriever, $\mathcal{P}_i^C$ the child QA pairs aggregated from the recursive calls, and

⁴<https://github.com/chensyCN/LogicRAG>

⁵<https://github.com/sakura20221/RT-RAG>

Algorithm 2: APT-RAG with Batched Answer Generation

```

function APT-RAG( $Q_i, \mathcal{P}_i^S$ ):
     $Q_i \leftarrow \text{Contextualizer}(Q_i, \mathcal{P}_i^S)$ 
    Evidence  $\mathcal{E}_{Q_i} \leftarrow \emptyset$ 
    if AnswerabilityCheck( $Q_i, \mathcal{P}_i^S$ ) then
        // Lateral gathering
         $\mathcal{E}_{Q_i} \leftarrow \mathcal{P}_i^S$ 
    else
        Subquestions  $\mathcal{Q}_i \leftarrow \text{Decomposer}(Q_i)$ 
        if  $|\mathcal{Q}_i| = 0$  then
            // External gathering
             $\mathcal{E}_{Q_i} \leftarrow \text{Retriever}(Q_i)$ 
        else
            // Vertical gathering
            for  $k \leftarrow 1$  to  $|\mathcal{Q}_i|$  do
                 $\rho_k \leftarrow \text{PlanningMode}(q_k)$ 
                 $\mathcal{B} \leftarrow \emptyset$ 
                for  $k \leftarrow 1$  to  $|\mathcal{Q}_i|$  do
                    if  $\rho_k = \text{EXTERNAL}$  then
                         $\mathcal{B} \leftarrow \mathcal{B} \cup \{k\}$ 
                    else
                         $\mathcal{B} \leftarrow \text{BatchedAnswer}(\mathcal{B})$ 
                         $a_k \leftarrow \text{APT-RAG}(q_k, \mathcal{P}_{i, < k}^C)$ 
                 $\mathcal{B} \leftarrow \text{BatchedAnswer}(\mathcal{B})$ 
                Child QA Pairs  $\mathcal{P}_i^C \leftarrow \langle (q_k, a_k) \rangle_{k=1}^{|\mathcal{Q}_i|}$ 
             $\mathcal{E}_{Q_i} \leftarrow \mathcal{P}_i^C$ 
        // Answer generation
         $A_i \leftarrow \text{Generator}(Q_i, \mathcal{E}_{Q_i})$ 
    return  $A_i$ 

function BatchedAnswer( $\mathcal{B}$ ):
    if  $\mathcal{B} \neq \emptyset$  then
         $\{\mathcal{E}_{q_j}\}_{j \in \mathcal{B}} \leftarrow \text{Retriever}(\{q_j\}_{j \in \mathcal{B}})$ 
         $\mathcal{C}_{\mathcal{B}} \leftarrow \text{Cluster}(\{\mathcal{E}_{q_j}\}_{j \in \mathcal{B}})$  // Alg. 3
        foreach  $C \in \mathcal{C}_{\mathcal{B}}$  do
             $\langle a_j \rangle_{j \in C} \leftarrow \text{Generator}(\{q_j\}_{j \in C}, \bigcup_{j \in C} \mathcal{E}_{q_j})$ 
    return  $\emptyset$ 

```

$\mathcal{E}_i$ the evidence consumed by the answer generator, which is set to $\mathcal{P}_i^S$, $\mathcal{D}_i$, or $\mathcal{P}_i^C$ depending on the gathering mode.

D.2 Inference with Batched Answer Generation

Algorithm 2 extends this recursive inference procedure with the batched answer generation described in §3.5. After a node is decomposed into child sub-questions, it first computes $\rho_k \leftarrow \text{PlanningMode}(q_k)$ for every child, where ρ_k denotes the gathering mode planned for q_k . The procedure then scans the children in order, accumulates consecutive children with $\rho_k = \text{EXTERNAL}$ into a batch $\mathcal{B}$, and flushes the batch before recursing on a child that requires non-external gathering. Each flushed batch is handled by BatchedAnswer, which retrieves evidence for the

Algorithm 3: Constrained Greedy Graph Coloring with Largest-First Ordering

Input: Sub-questions vertices $V = \{v_1, \dots, v_n\}$,
retrieved evidence $\mathcal{S} = \{\mathcal{E}_1, \dots, \mathcal{E}_n\}$
similarity threshold τ^S , context budget τ^L

Output: Cluster set $\mathcal{C} = \{C_1, \dots, C_K\}$

// Construct compatibility graph
foreach $i < j$ **do**
 $s_{ij} \leftarrow \text{Jaccard}(\mathcal{E}_i, \mathcal{E}_j)$
 $\text{compat}(i, j) \leftarrow \mathbf{1}[s_{ij} > \tau^S]$

// Largest-first ordering on complement graph $\bar{G}$
foreach $v_i \in V$ **do**
 $\deg_{\bar{G}}(v_i) \leftarrow |\{v_j : \neg \text{compat}(i, j)\}|$
Sort V in descending order of $\deg_{\bar{G}}$: $v_{\pi(1)}, \dots, v_{\pi(n)}$

Function $\text{ISVALID}(C_k, v)$:
 if $\exists u \in C_k$ s.t. $\neg \text{compat}(u, v)$ **then**
 return False
 if $\ell(\bigcup_{u \in C_k \cup \{v\}} \mathcal{E}_u) \geq \tau^L$ **then**
 return False
 return True

// Greedy coloring
 $\mathcal{C} \leftarrow \emptyset$
for $i \leftarrow 1$ **to** n **do**
 $\text{placed} \leftarrow \text{False}$
 foreach $C_k \in \mathcal{C}$ **do**
 if $\text{ISVALID}(C_k, v_{\pi(i)})$ **then**
 $C_k \leftarrow C_k \cup \{v_{\pi(i)}\}$
 $\text{placed} \leftarrow \text{True}$
 break
 if $\neg \text{placed}$ **then**
 $\mathcal{C} \leftarrow \mathcal{C} \cup \{\{v_{\pi(i)}\}\}$
return $\mathcal{C}$

batched sub-questions, invokes Cluster procedure in Algorithm 3 to group evidence-overlapping sub-questions under the context-window budget, and generates their answers with the union of the clustered evidence. This preserves the original sibling-order dependencies for recursive children while jointly answering independent external children.

D.3 Evidence-Guided Subquestion Clustering

Algorithm 3 details the greedy approximation used in evidence-guided batched answer generation (§3.5). In graph coloring, adjacent vertices must be assigned different colors. We exploit this property by constructing the complement graph $\bar{G}$, where an edge connects two subquestion vertices that have *insufficient* evidence overlap—precisely the pairs that should not share a cluster. Applying graph coloring to $\bar{G}$ then ensures that incompatible subquestions are always assigned to different clusters. Vertices are sorted in descending order of their degree in $\bar{G}$ (largest-first ordering), and a greedy coloring is applied in that order: each vertex is assigned to the first existing cluster that passes

the ISVALID check—requiring both pairwise evidence compatibility and the context-window budget τ^L —or placed into a new singleton cluster if no such cluster exists. See Appendix E.3 for a detailed description of the full algorithm.

E Clustering: Problem Reduction and Approximation Analysis

E.1 Constrained Clustering Formulation

We formalize evidence-guided subquestion clustering as a constrained optimization problem. Given a set of subquestions $\{q_i\}_{i=1}^n$ with retrieved evidence $\mathcal{E}_i$, our objective is to partition the subquestions into the minimum number of clusters while satisfying the context-window limit of the language model.

Graph Construction. To capture pairwise evidence compatibility, we construct a graph

$$G = (V, E),$$

where each vertex corresponds to a subquestion and an edge indicates sufficient evidence overlap:

$$(i, j) \in E \iff \text{Jaccard}(\mathcal{E}_i, \mathcal{E}_j) > \tau^S, \quad (2)$$

for a similarity threshold $\tau^S \geq 0$.

Constrained Clustering Objective. We seek the minimum number of clusters

$$\mathcal{C} = \{C_1, \dots, C_{K^*}\}$$

such that each cluster contains mutually compatible subquestions and satisfies the context-window limit:

$$K^* = \min K \quad (3a)$$

$$\text{s.t.} \quad \bigcup_{k=1}^K C_k = V, \quad (3b)$$

$$C_a \cap C_b = \emptyset, \quad (\forall a \neq b), \quad (3c)$$

$$\text{Jaccard}(\mathcal{E}_i, \mathcal{E}_j) > \tau^S, \quad \forall i \neq j \in C_k, \quad \forall k, \quad (3d)$$

$$\ell\left(\bigcup_{i \in C_k} \mathcal{E}_i\right) < \tau^L, \quad \forall k. \quad (3e)$$

Here, K denotes the number of clusters in a feasible solution, K^* its minimum value, L the context-window size limit, and $\ell(\cdot)$ the token count of the merged evidence.

Relaxation to Standard CCP. If constraint (3e) is removed, the problem reduces to the standard clique covering problem (CCP) problem (Chalupa, 2013):

$$\begin{aligned} \tilde{K}^* &= \min \tilde{K} \\ \text{s.t. } &\bigcup_{i=1}^{\tilde{K}} C_i = V, \\ &C_a \cap C_b = \emptyset \quad (\forall a \neq b), \\ &C_i \text{ induces a clique in } G \quad (\forall 1 \leq i \leq \tilde{K}). \end{aligned} \quad (4)$$

The optimal value of this problem is the clique partition number of G , denoted by

$$\tilde{K}^* = \theta(G).$$

Since the constrained formulation introduces an additional context-window constraint, the feasible set of the constrained problem is a subset of that of the standard CCP. Hence, the constrained optimum is at least as large:

$$\tilde{K}^* \leq K^*. \quad (5)$$

E.2 Reduction to Graph Coloring

The CCP problem on G is equivalent to graph coloring on its complement graph $\bar{G} = (V, \bar{E})$, in which adjacent vertices are assigned different colors, where

$$(i, j) \in \bar{E} \iff (i, j) \notin E.$$

In particular,

$$\theta(G) = \chi(\bar{G}), \quad (6)$$

where $\theta(G)$ and $\chi(\bar{G})$ denote the clique partition number of G and the chromatic number of $\bar{G}$, respectively. This equivalence follows because a set of vertices forms a clique in G if and only if it forms an independent set in $\bar{G}$. Therefore, partitioning V into cliques of G is equivalent to a proper vertex coloring of $\bar{G}$ (West, 2001).

Under this equivalence, the compatibility condition in constraint (3d) corresponds to an independent-set condition on the complement graph

$$\bar{G} = (V, \{(i, j) : \text{Jaccard}(\mathcal{E}_i, \mathcal{E}_j) \leq \tau^S\})$$

Therefore, our problem reduces to a graph coloring problem on $\bar{G}$ in which each color class must additionally satisfy the context-window constraint

$$\ell\left(\bigcup_{i \in C_k} \mathcal{E}_i\right) < \tau^L.$$

E.3 Greedy Approximation

Since the constrained clustering problem remains NP-hard, we adopt a greedy graph-coloring based on the largest-first ordering (Welsh and Powell, 1967). The procedure is summarized in Appendix D.3.

Greedy approximation algorithm. Given the complement graph

$$\bar{G} = (V, \bar{E}),$$

the coloring is represented by a function

$$f : V \rightarrow \{1, 2, \dots, k\},$$

where $f(v)$ denotes the color assigned to vertex v and k denotes the number of colors used.

[1 Initialization]. Set $i = 0$ and construct the complement graph $\bar{G} = (V, \bar{E})$ by adding an edge $(u, v) \in \bar{E}$ whenever

$$\text{Jaccard}(\mathcal{E}_u, \mathcal{E}_v) \leq \tau^S.$$

Then compute the degree

$$\deg_{\bar{G}}(v) = |\{u \in V : (u, v) \in \bar{E}\}|$$

for each vertex $v \in V$. The vertices are then sorted in descending degree order as

$$\deg_{\bar{G}}(v_{\pi(1)}) \geq \deg_{\bar{G}}(v_{\pi(2)}) \geq \dots \geq \deg_{\bar{G}}(v_{\pi(n)}),$$

where π denotes the resulting vertex ordering and ties are broken arbitrarily. Finally, the color classes

$$C_1, C_2, \dots \subseteq V$$

are initialized as empty sets.

[2 Next vertex]. Increment i , and consider the next vertex $v_{\pi(i)}$. If $i = n + 1$, terminate with f as the resulting coloring.

[3 Find admissible colors]. Compute the set

$$F_i = \{f(v_{\pi(j)}) : j < i, (v_{\pi(j)}, v_{\pi(i)}) \in \bar{E}\},$$

that is, the set of colors already assigned to the neighbors of $v_{\pi(i)}$ in $\bar{G}$.

[4 Assign the smallest available color]. Assign $v_{\pi(i)}$ to the first color class C_r with

$$r \notin F_i$$

satisfying

$$\ell\left(\bigcup_{u \in C_r \cup \{v_{\pi(i)}\}} \mathcal{E}_u\right) < \tau^L.$$

The vertex $v_{\pi(i)}$ is then assigned color r , i.e.,

$$f(v_{\pi(i)}) = r.$$

If no admissible color class satisfies the constraint, a new singleton color class is created.

Including the largest-first sorting step, the overall running time of the greedy procedure is

$$O(m + n \log n),$$

where $n = |V|$ and $m = |\bar{E}|$.

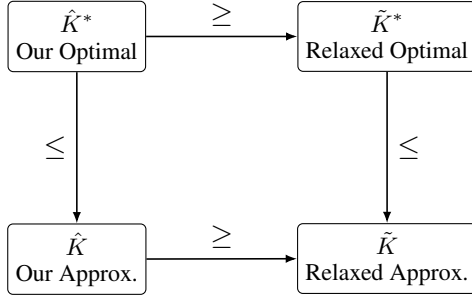


Figure 9: Relationship among optimal and approximate solutions under the constrained and relaxed formulations.

E.4 Approximation Analysis

Let $\hat{K}$ and $\tilde{K}$ denote the number of clusters produced by our greedy approximation algorithm and its relaxed version in Equation (4), respectively, and let $\hat{K}^*$ and $\tilde{K}^*$ denote their corresponding optimal values. Figure 9 summarizes their relationships.

Approximation Hierarchy. The following inequalities hold:

$$\tilde{K}^* \leq \hat{K}^* \leq \hat{K}, \quad \tilde{K}^* \leq \tilde{K} \leq \hat{K}. \quad (7)$$

Each inequality follows directly:

- $\tilde{K}^* \leq \hat{K}^*$: the relaxed problem admits more feasible solutions, so its optimal is smaller or equal.
- $\hat{K}^* \leq \hat{K}$: $\hat{K}^*$ is the optimal value, so $\hat{K}$ is an upper bound on it by definition.
- $\tilde{K}^* \leq \tilde{K}$: $\tilde{K}^*$ is the optimal value, so $\tilde{K}$ is an upper bound on it by definition.

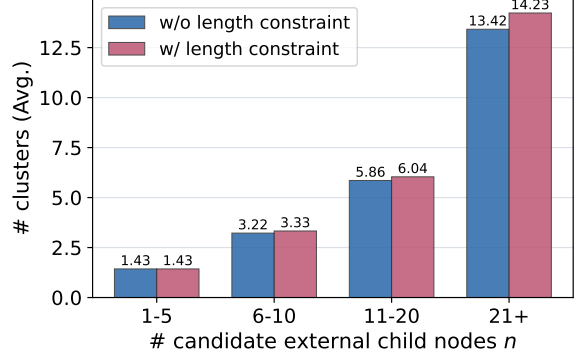


Figure 10: Effect of the context-window constraint on the number of clusters. The x-axis denotes the number of candidate external child nodes $n = |B|$, while the y-axis shows the average number of clusters for the relaxed and constrained settings.

- $\tilde{K} \leq \hat{K}$: since the unconstrained greedy does not enforce the context-window constraint, it can merge clusters that the constrained greedy cannot, yielding fewer or equal clusters.

Together, these inequalities confirm that $\tilde{K}^*$ serves as a lower bound and $\hat{K}$ as an upper bound on the constrained optimum $\hat{K}^*$.

Figure 10 evaluates the effect of introducing the context-window constraint into the greedy clustering algorithm. As the number of candidate external child nodes increases, the difference between the relaxed and constrained settings remains small; even at the 21+ point, the gap is less than one cluster. The result suggests that the context-window constraint preserves clustering behavior close to the relaxed solution.

F Prompt Templates

We provide the prompt templates used by the LLM-based components of APT-RAG introduced in §3. The templates are organized according to the four stages of the node-level procedure: contextualization(Appendix F.1), adaptive planning(Appendix F.2), evidence gathering(Appendix F.3), and answer generation(Appendix F.4). For each component, we explain the role its prompt plays in the recursive traversal, how the instruction is structured to enforce that role, and the design decisions that motivated this structure.

F.1 Contextualization

Contextualization fills $\langle Q_j \rangle$ in a sub-question using entities or values from the referenced sib-

ling’s answer (Figure 13; §3.2). The prompt uses a four-step sequence: analyze the question, identify needed conditions, extract items from prior answers, and then rewrite. Listing the items in Step 3 before the rewrite in Step 4 avoids leaving the placeholder verbatim or vaguely paraphrased.

F.2 Adaptive Planning

Adaptive planning runs two prompts in sequence (§3.3): it first checks answerability, then performs decomposition if external evidence is required.

Answerability Checker. The check uses two main criteria: *factual sufficiency*(are the entity and attributes explicit in the context?) and *operational feasibility*(does the question only compare or aggregate given facts?). The prompt(Figure 14) produces a four-step trace and ends in a Boolean. This split prevents the common conflation between "relevant information appears in the context" and "the question is fully answerable", which would otherwise trigger redundant retrieval on simple aggregations over sibling values.

Decomposer. A *maintain/split* decision is produced along with new sub-questions (Figure 15). Mutual dependencies between these sub-questions are written as the placeholder $< Q_j >$ and are resolved during contextualization. The prompt does not use a fixed depth or branching factor. Instead, it explicitly lists three conditions that prompt a split: when multiple bundled facts are present, when a stepwise dependency exists, or when operations such as sorting, ranking, or comparison are needed. This way, the tree’s granularity reflects the input’s structure rather than being governed by a global hyperparameter.

F.3 Evidence Gathering

External gathering is the only mode that issues a new retrieval call and therefore requires a dedicated query. We isolate query rewriting from sub-question generation so the search query can be tuned for the retriever while the sub-question retains the verbosity required for downstream reasoning.

Single-Query Rewriting. A single sub-question is compressed into a compact keyword query for dense retrieval (Figure 20). Two short in-context examples specify the intended transformation — removing filler while preserving key entities and attributes. This single sub-question rewriting path

is used only when retrieval is dispatched from a non-decomposed question, typically the root. After decomposition occurs—when a node splits into siblings—the query rewriting process shifts to the batched variant described below.

Multi-Query Rewriting. In the batched rewriting process, a group of sibling sub-questions is rewritten in a single call (Figure 21), returning a JSON dictionary keyed by question index. The instructions match those for single queries. Only the input and output formats change, ensuring rewriting stays consistent across batches. The batch is formed at planning time: siblings routed to external gathering are rewritten together. As a result, the rewriting cost scales with the number of decomposition levels rather than the number of sibling nodes.

F.4 Answer Generation

Each node produces an answer using a dedicated prompt for each gathering mode: lateral, external, and vertical(Figures 16, 17, 18). There is a unique prompt at the root(Figure 19), and a batched version for EC(Figure 22). The prompts share the same system instruction and differ only in their input fields and integration rules.

Lateral Answer Generation. The input consists of the question and the preceding sibling QA pairs (Figure 16). The prompt instructs the model to use only relevant QA pairs from siblings. One template handles both cases: when only one sibling is useful, and when many siblings are present but only a few are helpful. The output must be concise and self-contained so that the parent can use it as evidence for downstream reasoning.

External Answer Generation. Three inputs are required (Figure 17): the question, the retrieved documents, and the search query that produced them. Including the search query helps the model focus on the retrieval intent.

Vertical Answer Generation. Child QA pairs are integrated into a single parent answer (Figure 18). The prompt prohibits bullet-point or list-style output, since downstream generators consume the answer as a single evidence field.

Final Answer Generation. The root’s child QA pairs are consolidated into the benchmark-formatted answer (Figure 19). The system instruction comes from the MoNaCo answer-generation

Models	Methods	MoNaCo				QAMPARI			
		# LLM Calls	# Ret. Calls	Input Tok.	Output Tok.	# LLM Calls	# Ret. Calls	Input Tok.	Output Tok.
QWEN3-4B-INST	RT-RAG	95.0	10.9	921,219	40,558	29.4	2.8	183,320	12,269
	APT-RAG (Ours)	16.4	7.1	69,982	1,699	4.9	1.7	8,366	227
QWEN3-30B-INST	RT-RAG	115.6	14.1	1,130,788	52,319	59.0	7.1	354,931	29,299
	APT-RAG (Ours)	32.1	13.5	134,923	4,238	11.5	4.3	21,447	1,072

Table 5: Per-example inference cost of RT-RAG and APT-RAG on the 300-example subset of MoNaCo and QAMPARI. # LLM Calls: average number of LLM invocations per question; # Ret. Calls: retriever invocations; Input/Output Tok.: average input and output tokens per question.

Models	Methods	MoNaCo				QAMPARI			
		Ans. P	Ans. R	Ans. F1	Ret. R	Ans. P	Ans. R	Ans. F1	Ret. R
QWEN3-4B-INST	LLM-Only	31.08	28.59	28.07	–	12.73	5.38	6.56	–
	NaiveRAG	42.68	35.90	37.10	32.26	36.19	18.77	21.37	12.13
	Plan*RAG	31.77	28.61	29.14	33.52	16.60	5.94	7.52	6.87
	LogicRAG	43.11	33.98	35.90	28.33	33.95	8.87	12.10	10.32
	RT-RAG	46.78	38.13	39.86	21.90	37.90	14.93	17.86	13.21
	ToQ	38.58	33.39	34.17	34.03	35.21	19.78	21.75	12.57
	APT-RAG (Ours)	44.16	38.27	39.14	41.23	38.32	22.05	24.44	14.99
QWEN3-30B-INST	LLM-Only	44.22	40.35	39.94	–	22.97	9.60	11.74	–
	NaiveRAG	54.33	47.70	48.27	32.26	38.14	20.72	23.19	12.13
	Plan*RAG	53.25	48.09	48.59	40.71	34.09	15.80	18.21	17.00
	LogicRAG	56.41	45.16	46.95	21.53	38.64	9.42	13.19	9.39
	RT-RAG	56.31	45.02	46.87	16.47	39.74	14.72	17.78	10.20
	ToQ	53.37	47.05	47.60	32.44	39.47	22.31	24.77	11.79
	APT-RAG (Ours)	56.04	52.39	52.15	52.18	36.92	24.60	25.83	22.58

Table 6: Performance comparison on the 300-example subset of MoNaCo and QAMPARI used to evaluate RT-RAG. Ans. P/R/F1 denote answer precision, recall, and F1; Ret. R denotes retrieval recall. Best values per metric, backbone, and benchmark are in **bold**.

prompt. It includes the rule: "prefer fewer, correct answers over a long list that mixes correct and incorrect items." This instruction is used for both benchmarks. We append a benchmark-conditional output block that retains MoNaCo’s native answer-with-referent format and specifies a comma-separated list of short entity strings for QAMPARI.

Batched Answer Generation in EC. Answers for all sub-questions of an EC cluster are produced within a single LLM call (Figure 22; §3.5). Each question is linked to its allowed documents. The prompt tells the model to use only those documents. Without this rule, the model might cross-reference documents between questions and confuse entities that share attributes. The output is a JSON dictionary keyed by question index, so the cluster’s answers can be redistributed to their corresponding nodes without ambiguity.

G Subset Evaluation

We compare APT-RAG against RT-RAG on a 300-example subset randomly sampled from each benchmark. Table 5 details inference cost, while Table 6 gives answer accuracy and retrieval coverage.

Inference Cost. We restrict the comparison to a subset because RT-RAG incurs substantially higher costs than the other baselines. On MoNaCo with the 30B backbone, RT-RAG requires 115.6 LLM calls and consumes 1.13M input tokens per question, which is approximately $3.6\times$ more LLM calls and $8.4\times$ more input tokens compared to APT-RAG. This cost difference between RT-RAG and APT-RAG is consistent across model scales and benchmarks. This overhead stems from RT-RAG’s repeated tree construction, consensus sampling, and sub-question rewriting, as detailed in Appendix C.1.

Models	Methods	MoNaCo				QAMPARI			
		# LLM Calls	Input Tok.	Output Tok.	Latency (s)	# LLM Calls	Input Tok.	Output Tok.	Latency (s)
QWEN3-4B-INST	LLM-Only	1.00	122	138	1.79	1.00	167	383	5.07
	NaiveRAG	1.00	10,246	81	1.92	1.00	3,135	45	0.81
	Plan*RAG	5.25	28,759	499	27.51	2.55	4,028	168	2.48
	LogicRAG	5.76	21,656	803	12.83	5.08	6,435	688	9.67
	ToQ	28.11	40,063	1,332	71.69	9.40	7,391	522	7.19
	APT-RAG (Ours)	14.96	62,515	1,630	29.62	4.92	8,252	229	3.60
QWEN3-30B-INST	LLM-Only	1.00	122	71	1.45	1.00	167	28	0.64
	NaiveRAG	1.00	10,246	49	1.90	1.00	3,135	40	1.08
	Plan*RAG	7.66	42,538	875	22.20	5.25	10,172	398	8.69
	LogicRAG	5.52	18,348	828	17.77	5.20	6,302	770	15.38
	ToQ	8.20	18,428	329	81.27	6.25	5,710	208	4.89
	APT-RAG (Ours)	33.61	138,920	4,596	104.59	11.69	21,898	1,153	24.29

Table 7: Per-question inference cost on the full MoNaCo and QAMPARI benchmarks.

Answer Quality. Even with this increased cost, RT-RAG does not consistently outperform APT-RAG. APT-RAG leads in Answer F1 across most settings; the only exception is the 4B backbone on MoNaCo, where RT-RAG slightly surpasses APT-RAG in F1 but achieves only about half of APT-RAG’s Retrieval Recall—a gap that widens to nearly $3\times$ on MoNaCo with the 30B backbone. We attribute this asymmetry to RT-RAG’s iterative question refinement, which narrows the search space at the cost of evidence coverage. Notably, scaling the backbone from 4B to 30B improves APT-RAG’s MoNaCo F1 by $+13.0$, whereas RT-RAG’s F1 improves by only $+7.0$ under the same conditions. This result suggests that, for evidence-intensive QA, enhancement from a stronger backbone is more effectively achieved through broader evidence coverage—as in APT-RAG—than through the narrower refinement approach used by RT-RAG.

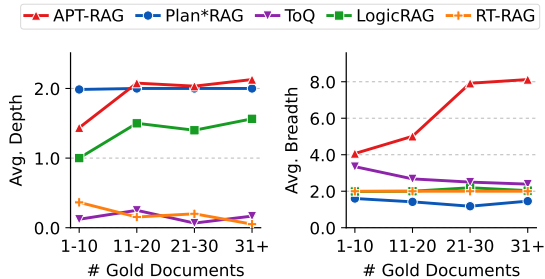


Figure 11: Average depth (left) and breadth (right) on the 300-example subset, grouped by the number of required gold documents.

Structural Adaptation. Figure 11 expands the comparison of depth and breadth metrics from Figure 6(b). It includes all baselines on the same subset. For each example, tree depth is the maximum root-to-node distance in the constructed reasoning structure, and tree breadth is the average number

of generated child sub-questions over decomposed parent nodes. While the tree depth and breadth of APT-RAG increase with the required evidence volume, those of the baselines remain nearly flat. This difference reflects how each baseline builds its reasoning structure (see Appendix C.1): Plan*RAG and LogicRAG fix the graph before retrieval. RT-RAG builds a binary tree with limited depth. ToQ stops expansion early when sufficiency checks are met. Because of this rigid structure, each query gets limited reasoning space. This leads to the retrieval coverage gap seen in Table 6.

H Full Evaluation

Inference Cost. Table 7 shows that APT-RAG’s inference cost depends on the evidence a question requires, rather than on a fixed budget. On MoNaCo with the 30B backbone, where each question needs 43.3 gold pages on average, APT-RAG is the most expensive method: it issues 33.61 LLM calls and takes 104.59s per question, while the structured RAG baselines issue 5.52 to 8.20 calls and take 17.77s to 81.27s. On QAMPARI, where each question needs 13.0 pages, APT-RAG’s LLM calls drop by $2.9\times$ and its latency by $4.3\times$, while the baselines’ calls drop by at most $1.5\times$. With the smaller 4B backbone on QAMPARI, APT-RAG is faster than LogicRAG and ToQ, at 3.60s versus 9.67s and 7.19s. Therefore, the baselines are less costly on MoNaCo not because of stricter budget constraints, but because the fixed structures discussed in §4.3 do not scale with increasing evidence requirements.

I Statistical Significance

An aggregate F1 gap does not distinguish systematic advantage from within-set variation. We therefore test each comparison for every question using a two-sided paired permutation test on the average

Models	Methods	MoNaCo		QAMPARI	
		Ans. F1	Δ Ans. F1 (95% CI)	Ans. F1	Δ Ans. F1 (95% CI)
QWEN3-4B-INST	LLM-Only	28.08	12.61 [10.37, 14.86]	4.97	15.94 [14.45, 17.48]
	NaiveRAG	34.98	5.71 [3.82, 7.66]	18.58	2.33 [1.37, 3.30]
	Plan*RAG	27.02	13.67 [11.55, 15.82]	5.40	15.51 [13.95, 17.03]
	LogicRAG	35.75	4.94 [2.92, 7.02]	9.87	11.04 [9.72, 12.41]
	ToQ	36.32	4.37 [2.46, 6.28]	18.76	2.15 [1.33, 3.00]
	APT-RAG (Ours)	40.69	–	20.91	–
QWEN3-30B-INST	LLM-Only	41.82	9.02 [6.71, 11.32]	9.40	13.87 [12.35, 15.42]
	NaiveRAG	46.64	4.20 [2.06, 6.28]	20.46	2.82 [1.63, 4.02]
	Plan*RAG	47.03	3.81 [1.83, 5.85]	17.67	5.60 [4.23, 6.98]
	LogicRAG	46.41	4.43 [2.38, 6.52]	11.23	12.05 [10.62, 13.49]
	ToQ	46.66	4.18 [2.11, 6.24]	21.37	1.90 [0.81, 3.00]
	APT-RAG (Ours)	50.84	–	23.28	–

Table 8: Answer F1 differences between APT-RAG and each baseline on the full benchmarks (Δ Answer F1 = APT-RAG – baseline), with 95% confidence intervals.

F1 difference per question, with a 95% confidence interval based on percentile bootstrap with 10,000 resamples. All confidence intervals in Table 8 exclude zero, and every comparison yields $p \leq 0.001$; all results remain significant after Bonferroni correction. Consequently, the improvements reported in Table 2 are statistically significant and not attributable to evaluation-set variation.

Method	Fine-grained Ret. R $\uparrow$
Fixed Breadth: 2	8.37
Fixed Breadth: Avg.	10.89
APT-RAG (Ours)	11.22

Table 9: Fine-grained retrieval recall for the adaptive expansion ablation.

J Additional Ablation Results

Table 9 reports the adaptive expansion ablation under MoNaCo’s fine-grained evidence matching criterion. MoNaCo gold evidence is annotated not only with the document title or URL, but also with the specific section within the document and the content type of the evidence, such as sentence, list, or table. Since our corpus is constructed using these fine-grained evidence units, this analysis evaluates whether each method retrieves evidence at the same granularity. Under this stricter criterion, APT-RAG achieves the highest retrieval recall. Together with the higher Answer F1 in Table 3, this result suggests that adaptive expansion improves answer quality partly by recovering more complete evidence at the corpus construction granularity.

Models	MoNaCo			QAMPARI		
	LG	VG	EG	LG	VG	EG
Qwen3-4B-Inst	4.0%	17.3%	78.7%	1.6%	12.0%	86.4%
Qwen3-30B-Inst	6.2%	16.6%	77.2%	3.7%	19.8%	76.5%

Table 10: Usage statistics for topology-aware evidence gathering.

Topology-Aware Gathering Usage. Table 10 reports the proportion of nodes resolved by each evidence-gathering strategy across datasets and model scales. Lateral gathering (LG) accounts for a small fraction (1.6-6.2%) of the total nodes generated, while most nodes rely on external or vertical gathering. Thus, the majority of sibling nodes remain independent and can be processed in parallel. Despite its limited usage, LG substantially reduces

Model	Method	Gemini-3.5-Flash	GPT-5.4
Qwen3-4B	LLM-Only	27.08	28.08
	NaiveRAG	34.19	34.98
	PlanRAG	26.40	27.02
	LogicRAG	35.31	35.75
	ToQ	35.39	36.32
	APT-RAG (Ours)	40.04	40.69
Qwen3-30B	LLM-Only	41.16	41.82
	NaiveRAG	45.49	46.64
	PlanRAG	46.60	47.03
	LogicRAG	46.16	46.41
	ToQ	45.91	46.66
	APT-RAG (Ours)	50.15	50.84

Table 11: Answer F1 on MoNaCo using Gemini-3.5-Flash and GPT-5.4 as LLM-as-a-judge evaluators.

latency by avoiding redundant retrieval and tree expansion, as demonstrated by the ablation in Figure 7.

Evaluator Robustness. To assess the robustness of the LLM-as-a-judge evaluation on MoNaCo, we additionally evaluate all methods using Gemini-3.5-Flash as an independent evaluator. As shown in Table 11, the two evaluators yield comparable F1 scores across all methods and backbone scales. Importantly, APT-RAG consistently achieves the highest F1 under both evaluators, indicating that our main performance conclusions are robust to the choice of evaluator.

K Case Study

Figure 12 presents a qualitative inference trace from MoNaCo, illustrating how APT-RAG operates through adaptive planning and topology-aware evidence gathering. This is the same question used in Figure 1, where we illustrate the limitations of existing structured RAG methods. The example requires identifying Mexico’s wars whose soldier counts are greater than 100,000 and fewer than 500,000.

Inference Case: Evidence-Intensive Counting over Mexico's Wars

Question Q : How many of Mexico's wars were fought with more than 100,000 soldiers but fewer than 500,000?

Evidence: $q_1, a_1; \dots; q_4, a_4$

Final answer A : 3 ✓

Inference trace:

Sub-question q_1 : What are the wars fought by Mexico throughout its history?

Evidence: $q_{1,1}, a_{1,1}; q_{1,2}, a_{1,2}$

Sub-answer a_1 : Mexico's wars span independence, territorial, intervention, revolutionary, religious, global-war, and internal conflicts, including the Mexican War of Independence, ..., and the Dirty War.

Sub-question $q_{1,1}$: What are the major wars fought by Mexico throughout its history?

Evidence: d_1 : Mexican War of Independence, ..., d_{20} : Dirty War, ...

Sub-answer $a_{1,1}$: The major wars include the Mexican War of Independence, Texas–Indian Wars, Mexican–American War, French Intervention in Mexico, Mexican Revolution, Cristero War, World War II involvement, ..., and the Dirty War.

Sub-question $q_{1,2}$: For each war in $\langle q_{1,1} \rangle$, provide a brief description of its causes, key events, and outcomes.

Contextualized question: For each of the following wars fought by Mexico throughout its history—Mexican War of Independence, ..., and the Dirty War—provide a brief description of its causes, key events, and outcomes.

Answerability: Unanswerable

Evidence: $q_{1,2,1}, a_{1,2,1}; \dots; q_{1,2,10}, a_{1,2,10}$

Sub-answer $a_{1,2}$: The wars are described by their main causes and outcomes: independence from Spain, Indigenous displacement, territorial loss, French intervention, constitutional reform, church–state conflict, World War II mobilization, ...

Sub-question $q_{1,2,1}$: What were the causes, key events, and outcomes of the Mexican War of Independence (1810–1821)?

Evidence: d_1 : Mexican War of Independence, ..., Cry of Dolores, Miguel Hidalgo, Army of the Three Guarantees, ...; d_{20} : Treaty of Cordoba, casualties, ...

Sub-answer $a_{1,2,1}$: Napoleon's invasion of Spain created a legitimacy crisis; the Cry of Dolores, Hidalgo, Morelos, Plan of Iguala, and Treaty of Cordoba, ..., led to independence.

Sub-question $q_{1,2,2}$: What were the causes, key events, and outcomes of the Texas–Indian Wars (1820–1875)?

Evidence: d_1 : Texas–Indian Wars, ..., Comanche, Apache, Anglo-American settlement, ...; d_{20} : Texas Revolution, Comanche–Mexico Wars, ...

Sub-answer $a_{1,2,2}$: Anglo-American settlement conflicted with Comanche and Apache sovereignty; raids, the Great Raid of 1840, ..., ended in Indigenous displacement.

⋮

Sub-question $q_{1,2,10}$: What were the causes, key events, and outcomes of the Dirty War in Mexico (1960s–1980s)?

Evidence: d_1 : Mexican Dirty War, ..., PRI, state repression, left-wing groups, disappearances, ...; d_{20} : Tlatelolco, Corpus Christi, guerrilla groups, ...

Sub-answer $a_{1,2,10}$: Under PRI rule, the Dirty War suppressed left-wing students and guerrillas through Tlatelolco, Corpus Christi, torture, disappearances, and extrajudicial violence.

Inference Case: Continued

Sub-question q_2 : For each war in $\langle q_1 \rangle$, what was the number of soldiers involved?

Contextualized question: For each war throughout Mexico's history, what was the number of soldiers involved: Mexican War of Independence (1810–1821), . . . , Dirty War (1960s–1980s)?

Answerability: **Unanswerable**

Evidence: $q_{2,1}, a_{2,1}; \dots; q_{2,10}, a_{2,10}$

Sub-answer a_2 : For the Mexican War of Independence (1810–1821), exact troop counts are not stated, but casualties are estimated at 250,000–600,000; . . . ; for the Dirty War (1960s–1980s), the evidence reports about 784,300 personnel.

Sub-question $q_{2,1}$: What was the number of soldiers involved in the Mexican War of Independence (1810–1821)?

Evidence: d_1 : Mexican War of Independence, . . . , insurgent forces, Spanish Empire army, casualties 250,000–600,000, . . . ; d_{20} : Treaty of Cordoba, Plan of Iguala, . . .

Sub-answer $a_{2,1}$: Exact troop counts are not given; the evidence mentions insurgent forces, the Spanish Empire's army, and casualties around 250,000–600,000.

Sub-question $q_{2,2}$: What was the number of soldiers involved in the Texas–Indian Wars (1820–1875)?

Evidence: d_1 : Texas–Indian Wars, . . . , Comanche, Apache, Republic of Texas, U.S. Army, . . . ; d_{20} : Texas Revolution, Apache–Mexico Wars, . . .

Sub-answer $a_{2,2}$: No single total is given; the evidence mentions the Republic of Texas, U.S. Army, Comanche, Kiowa, Apache, . . . , across many engagements.

⋮

Sub-question $q_{2,10}$: What was the number of soldiers involved in the Dirty War (1960s–1980s)?

Evidence: d_1 : Mexican Dirty War, . . . , military, police, National Guard, . . . ; d_{20} : Mexican drug war, . . . , 277,000 soldiers, 368,000 police, and 107,000 National Guard personnel, . . .

Sub-answer $a_{2,10}$: The evidence mentions about 784,300 total security personnel, including 277,000 soldiers, 368,000 police officers, and 107,000 National Guard personnel.

Sub-question q_3 : From the wars in $\langle q_2 \rangle$, which ones had more than 100,000 soldiers but fewer than 500,000 soldiers?

Contextualized question: From the wars in the following war-count evidence, which ones had more than 100,000 soldiers but fewer than 500,000 soldiers: Mexican War of Independence (250,000–600,000 casualties), . . . , Dirty War (784,300 personnel)?

Answerability: **Answerable**

Evidence: q_2, a_2

Sub-answer a_3 : Mexican War of Independence, Reform War, and Cristero War satisfy the condition. Wars with missing totals remain uncertain, smaller rebellions fall below 100,000, and the Dirty War exceeds the upper bound.

Sub-question q_4 : How many wars satisfy the condition in the final candidate set?

Contextualized question: How many wars satisfy the condition among the final candidate set: Mexican War of Independence, Reform War, and Cristero War?

Answerability: **Answerable**

Evidence: q_3, a_3

Sub-answer a_4 : **3**

Figure 12: Inference case for the MoNaCo dataset.

Prompt template for Contextualization

You are a Contextualizer who extracts the key keywords needed to specify a vague query and rewrites the question in a more specific form.

[Input Information]

- Target question: A question that requires specification. This question may contain reference expressions such as <Qn>.
- <Qn> refers to the result (answer) of the nth question.
- Prior QA information: Previous questions and their corresponding answers that are necessary to specify the target question.

[Process]

- Step 1. Analyze the target question.
- Step 2. Identify required search conditions.
- Step 3. Extract complete structured items from prior answers.
- Step 4. Rewrite the question by replacing vague references with concrete information.

[Output Format]

```
{
  "step 1": "...",
  "step 2": "...",
  "step 3": ["keyword1", "keyword2", ...],
  "step 4": "Rewritten question"
}
```

Target question: {target_question}

Prior QA information: {prior_qa_information}

Figure 13: Prompt template for contextualizing question.

Prompt template for Answerability Check

You are an answerability checker who determines whether external search is required to answer the current question.

[Input]

- input question: The question for which you must determine whether search is necessary.
- context: Previous question-answer (QA) pairs that provide context relevant to the current question.

[Decision Rules]

- Skip search if the core entity and required attributes are already explicit in the context.
- Perform search if the target entity or required attributes are missing or uncertain.
- Skip search if the question only compares, summarizes, or reorganizes information already available in the context.
- Perform search if the question requires new information.

[Output Format]

```
{
  "step 1": "...",
  "step 2": "...",
  "step 3": "...",
  "step 4": true or false
}
```

input question: {input_question}

context: {context}

Figure 14: Prompt template for the Answerability Checker (§3.3).

Prompt template for Decomposition

You are a Decomposer who analyzes an input question and determines whether it should be handled as a single search (maintain) or decomposed into sub-queries (split).

[Maintain / Split Decision Criteria]
A question should be split when multiple pieces of data must be assembled, when the answer requires stepwise dependency, or when the task requires sorting, ranking, comparison, or other complex operations.
A question should be maintained when the requested information is likely to be found as a single bundled fact.

[Decomposition Rules]
- If a sub-query depends on the result of a previous query, indicate this dependency using <Qn>, where n refers to the required prior query.

[Output Format]
{
 "decision": "maintain" or "split",
 "subqueries": [] or ["sub-query 1", "sub-query 2", ...]
}

input question: {input_question}

Figure 15: Prompt template for the Decomposer (§3.3).

Prompt template for Lateral Answer Generation

[System prompt]
You are a helpful question-answering assistant. Your task is to answer a complex question provided by the user. You may generate a brief explanation before presenting the final answer if necessary. Each sentence must be concise and clear. Since the answer will be passed to another agent for follow-up tasks, preserve sufficient context while remaining concise.
Your response must strictly follow the format below:
Answers: {ANSWERS}

[Input prompt]
Question: {question}

QA pairs generated along the path leading to the current question:
Q1: {previous_question_1}
A1: {previous_answer_1}
...

Above are related question-answer pairs generated along the path leading to the current question. Some QA pairs may contain information relevant to answering the current question. Use relevant QA pairs and ignore irrelevant ones.

Answer:

Figure 16: Prompt template for answer generation under lateral gathering (§3.4).

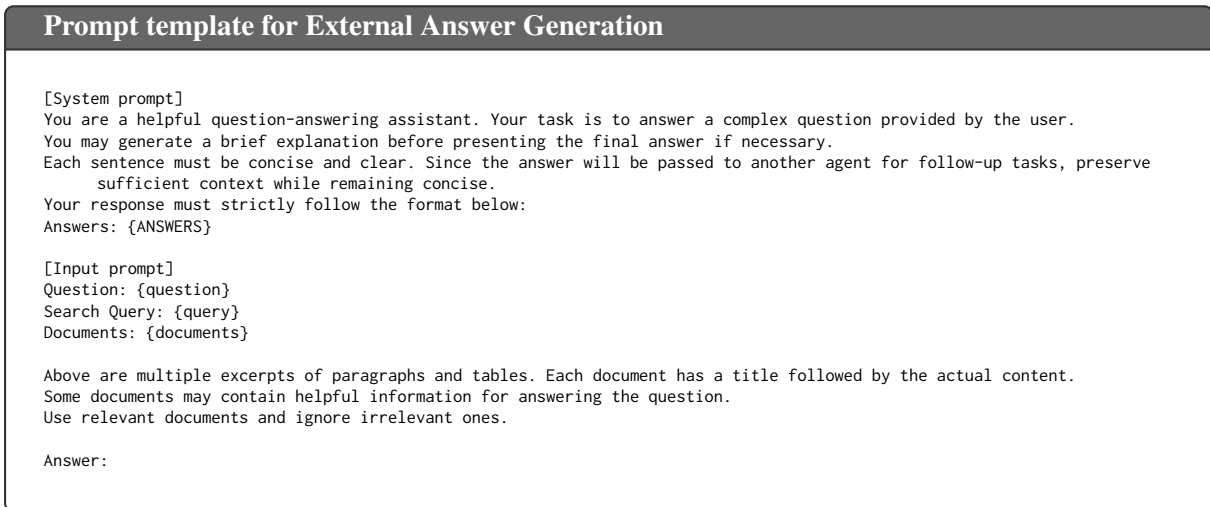


Figure 17: Prompt template for answer generation under external gathering (§3.4).

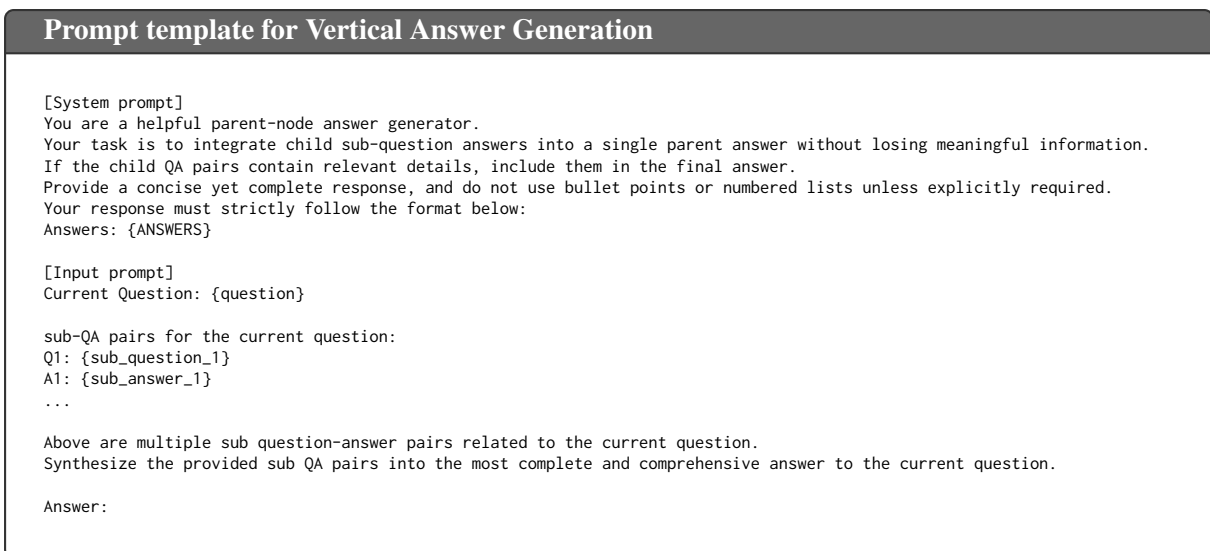


Figure 18: Prompt template for answer generation under vertical gathering (§3.4).

Prompt template for Final Answer Generation

```
[System prompt]
You are a helpful question answering assistant. Your task is to answer the question with high precision.
- List only answers that directly and clearly satisfy what the question asks.
- Prefer fewer, correct answers over a long list that mixes correct and incorrect items. When in doubt, omit the item.

[For QAMPARI Only]
- Each answer must be a short answer (entity or short phrase), not a full sentence.
- If multiple answers exist, separate them using commas.
- List each distinct answer only once.
- Do not include any explanation or extra text before or after the answer line.
- Your response must use the following format:
Answers: answer1, answer2, ..., answerN

[For MoNaCo Only]
- Preserve relevant context so each part of the answer is clearly tied to what it refers to.
- Your response must use the following format:
Answers: {ANSWERS}

[Input prompt]
Current Question: {question}

sub-QA pairs for the current question:
Q1: {sub_question_1}
A1: {sub_answer_1}
...

Using the provided sub QA pairs, generate a concise and direct answer to the current question.
Include only what is needed to answer the question; omit filler and redundant phrasing.

Answer:
```

Figure 19: Prompt template for final answer generation at the root.

Prompt template for Single-Query Rewriting

```
- You are a model that rewrites users' questions into search engine queries.
- Rewrite the question into one optimal search query.
- Examples:
  Please recommend a national park in California with hiking trails similar to Yosemite.
  Query: California national park hiking trails similar to Yosemite
  Please find the admission fee for the Louvre Museum.
  Query: Louvre Museum admission fee

Question: {question}
Query:
```

Figure 20: Prompt template for single-query rewriting before external retrieval.

Prompt template for Multi-Query Rewriting

```
- You are a model that rewrites users' questions into search engine queries.
- Rewrite each question into one optimal search query.
- Use the question numbers as JSON keys. Each value should contain only the rewritten search query corresponding to its question.

Question:
{questions}

Output format:
{
  "1": "query",
  "2": "query",
  ...
}

Query:
```

Figure 21: Prompt template for multi-query rewriting before external retrieval.

Prompt template for Batched Answer Generation

Each question includes an explicit list of associated documents. Use only the documents in that list for the question.

Rules:

- Answer each question independently.
- For each question, use only the documents specified in its list and no others.
- Use the question numbers as JSON keys. Each value should contain only the answer to its corresponding question.

Question-document ID pairs:

[Question 1]

Documents: [D1, D2, ...]

Question: {sub_question_1}

Search Query: {search_query_1}

[Question 2]

Documents: [D2, D3, ...]

Question: {sub_question_2}

Search Query: {search_query_2}

Documents:

{documents}

Output format:

```
{
  "1": "answer",
  "2": "answer",
  ...
}
```

Answer:

Figure 22: Prompt template for generating answers over sub-question clusters in Evidence-Guided Batched Answer Generation (see Section 3.5).