Graph2Tree-RAG: Bridging Knowledge Graphs and Tree-Structured Evidence for Multi-Hop Question Answering

Anonymous EMNLP submission

Abstract

Retrieval-Augmented Generation (RAG) has emerged as a promising paradigm to enhance large language models (LLMs) with external knowledge. However, existing RAG methods face two fundamental limitations: (1) handling complex multi-hop questions that require reasoning across disparate evidence, and (2) addressing summarization-oriented queries that demand synthesis rather than factual retrieval. In this paper, we propose **Graph2Tree-RAG** (GTRAG), a novel RAG framework that bridges the gap between structured knowledge retrieval and hierarchical answer generation. Our method constructs a knowledge graph (KG) via prompt engineering, retrieves relevant entities and relations, and applies multi-stage pruning with indexing to filter noise. This process iteratively expands to incorporate multi-hop neighbors, organizing retrieved chunks into a tree-structured representation. A divide-and-conquer strategy then recursively synthesizes intermediate answers into a coherent final response. We evaluate GTRAG on two task types. On multi-hop QA benchmarks, our method achieves state-of-the-art results on three public benchmarks, significantly outperforming existing RAG baselines. On summarization-oriented queries, GTRAG excels by producing more faithful and well-synthesized summaries. These results underscore the generalizability of our approach, highlighting its potential as a unified framework for diverse RAG tasks.

1 Introduction

Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) enhances Large Language Models (LLMs) (Zhao et al., 2023) with external knowledge, enabling access to up-to-date or domain-specific information. However, naive RAG methods face two fundamental limitations: they struggle with complex multi-hop questions that require reasoning across multiple pieces of evidence, and they are inherently ill-suited for summarization-oriented queries that demand synthesis across documents.

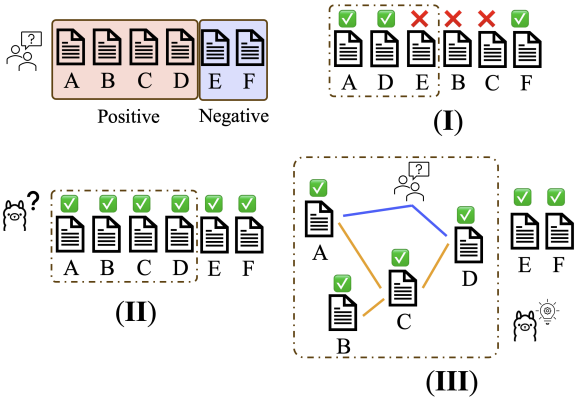


Figure 1: Illustration of the limitations of naive RAG methods. Given a query and six text chunks, suppose chunks A, B, C, and D contain the key information needed to answer it. **(I)** Relying solely on chunk-level semantic similarity does not guarantee that all necessary chunks will be retrieved. **(II)** Even when all necessary chunks are correctly selected, the lack of explicit relational exploration among them may prevent the LLM from focusing on critical information distributed across different chunks. **(III)** When chunk relationships are explicitly provided, the LLM is better equipped to perform complex reasoning tasks.

Figure 1 illustrates these limitations, which serve as the primary motivation for our work.

Motivation 1: Relational Reasoning for Multi-Hop Questions. Traditional RAG methods typically rely on semantic similarity-based retrieval (Reimers and Gurevych, 2019), which embeds queries and chunks into a vector space and retrieves chunks with the highest cosine similarity. This approach suffers from two critical flaws. First, it treats each chunk as isolated evidence, failing to explicitly explore relational connections among entities across different documents (Yang et al., 2024). Second, it prioritizes overall semantic similarity, often missing nuanced but critical information—a single entity name, a specific date, or a subtle relational clue—that may constitute only a small fraction of a chunk's content yet is essential for correct

reasoning (Liu et al., 2024c). Consider a multi-hop question: *"Which river flows through the European city that is home to the oldest university in Germany?"* Answering this requires correctly chaining multiple relations: the oldest German university (Heidelberg University) → its host city (Heidelberg) → the river flowing through that city (Neckar River). This chain cannot be resolved by simply retrieving chunks semantically similar to the query. To address this, recent work has incorporated offline knowledge graphs (KGs) into RAG frameworks (Gutiérrez et al., 2024; Baek et al., 2023; Fang et al., 2024), performing structured retrieval by explicitly capturing relations (Liu et al., 2024a; Shen et al., 2025; Gao et al., 2025) or condensing relevant subgraphs (Hu et al., 2025; Cai et al., 2025).

Motivation 2: Cross-Document Synthesis for Summarization-Oriented Queries. Consider a summarization-oriented query: *"How do different states enforce cybersecurity compliance among local government agencies with limited IT resources?"* Answering this requires retrieving policy documents from multiple states, comparing their enforcement mechanisms (e.g., waivers, tiered requirements, and state-provided technical assistance), and synthesizing the differences into a coherent comparative summary. Standard RAG pipelines (Arslan et al., 2024; Fan et al., 2024) retrieve relevant chunks independently but lack mechanisms to compare, contrast, or synthesize information across sources. Even when KGs are leveraged to capture entity relationships, existing approaches typically feed retrieved information to the LLM in a flat manner (Liu et al., 2025; Edge et al., 2024), without a well-structured organization that mirrors the logical dependencies inherent in comparative or abstractive reasoning. Recent work has explored two mainstream directions to address this challenge: (1) iterative consolidation approaches (Wang et al., 2026; Shen et al., 2025) that index selected chunks and recursively generate summaries to produce the final answer, and (2) structure-aware reasoning frameworks (Sarthi et al., 2024; Fatehkia et al., 2024; Zhang et al., 2025; Cai et al., 2025; Wu et al., 2024) that organize selected chunks into tree or graph structures to enable hierarchical reasoning.

In this paper, we introduce **Graph2Tree-RAG** (GTRAG), a novel RAG framework that bridges structured knowledge retrieval with hierarchical answer generation. Our method directly addresses the two motivations outlined above. During preprocessing, a knowledge graph is constructed from the text corpus, explicitly capturing entities and their interconnections. After that, when a query is given, relevant entities are extracted and iteratively traversed on the graph to retrieve higher-order neighbors. A multi-stage pruning mechanism, coupled with a chunk indexing strategy, is then applied to build a set of relation chains. Based on these chains, the retrieved chunks are organized into a tree-structured representation that reflects the logical dependencies among evidence pieces. Finally, a divide-and-conquer strategy is employed to recursively generate intermediate answers from subtrees and synthesizes them into a coherent final response, aligning the reasoning process with the inherent hierarchy of complex questions.

We evaluate GTRAG on six benchmarks spanning two task types: MuSiQue, HotpotQA, and 2WikiMultihopQA for factoid QA, and TraCR, QA-Expert-multi-hop-qa, and MultiHop-RAG for open-ended QA. Experimental results on factoid QA datasets demonstrate that our method significantly outperforms existing baselines, achieving substantial improvements in both retrieval efficiency and answer accuracy. Ablation studies on open-ended benchmarks further confirm that each additional hop brings measurable gains, underscoring the effectiveness of each component. Our main contributions are summarized as follows:

- We propose GTRAG, a novel framework that bridges graph-based knowledge retrieval with tree-structured recursive reasoning, directly addressing the identified limitations.
- We introduce a multi-stage pruning mechanism combined with chunk indexing that effectively filters irrelevant information while preserving nuanced critical details.
- We develop a divide-and-conquer strategy that recursively generates answers from tree-structured evidence, enabling coherent multi-hop reasoning.
- Extensive experiments on 6 public benchmarks demonstrate that GTRAG achieves state-of-the-art performance, with significant improvements over strong baselines.

2 Related Work

2.1 Retrieval-Augmented Generation

Early research (Lewis et al., 2020) proposed basic pipelines for incorporating external knowledge into

pre-trained language models. Initial approaches typically employed sparse retrieval methods such as BM25 (Robertson and Walker, 1994; Roberts et al., 2020) and TF-IDF (Qaiser and Ali, 2018) to retrieve relevant documents. More recent work leverages deep learning techniques for dense retrieval. For instance, Contriever (Izacard et al., 2021), SentenceBERT (Reimers and Gurevych, 2019), and ColBERT (Khattab and Zaharia, 2020) compute similarities using semantic embeddings. Atlas (Izacard et al., 2023) applies a joint training strategy within a text-to-text framework to retrieve documents. After selecting relevant chunks, answers can be either directly extracted from text (Hu et al., 2022) or generated by LLMs (Mansurova et al., 2024; Liu et al., 2024b).

2.2 Knowledge Graph-Enhanced Retrieval

Incorporating knowledge graphs (KGs) into reasoning systems has been extensively explored. KGs can be constructed from external data sources such as ConceptNet (Speer et al., 2017) or built internally from specific corpora (Wei et al., 2023; Edge et al., 2024). When performing retrieval on KGs, multiple strategies exist for exploring neighbors and expanding subgraphs. KAPING (Baek et al., 2023) directly retrieves relevant relations based on semantic similarity. SimGRAG (Cai et al., 2025) first extracts a pattern from the query and then explores a subgraph from the KG that topologically aligns with the query pattern. HippoRAG (Gutiérrez et al., 2024) applies the Personalized PageRank algorithm on offline KGs to retrieve passages and locate chunks. GEAR (Shen et al., 2025) introduces a diverse triple beam search for relation expansion. GRAG (Hu et al., 2025) employs specifically designed prompts to adaptively determine query depth and retrieval strategies based on query complexity. TRACE (Fang et al., 2024) utilizes an LLM-based triple ranker to iteratively retrieve relations. Additionally, several LLM-related techniques have been integrated into KG retrieval, including chain-of-thought (Trivedi et al., 2023) and agent-based methods (Li et al., 2024; Luo et al., 2023).

2.3 Hierarchical and Recursive Reasoning

Various methods have been developed to enable multi-hop reasoning. GraphRAG (Edge et al., 2024) makes progress in query-focused summarization by applying entity clustering to organize evidence into hierarchical communities. RAPTOR (Sarthi et al., 2024) recursively clusters chunks based on semantic embeddings to form a tree-structured representation. T-RAG (Fatehkia et al., 2024) employs a custom named entity recognition module to identify entities in the query and retrieves hierarchical information from a pre-built knowledge tree to augment the context. Since retrieved knowledge is inherently structured as graphs, recent work (Hu et al., 2025; Mavromatis and Karypis, 2024) has naturally integrated Graph Neural Networks (GNNs) with LLMs, leveraging the inherent graph-processing capabilities of GNNs to mine retrieved subgraphs and thereby enhance overall reasoning capabilities.

3 Methodology

3.1 Problem Formulation

Given a set of text chunks $\mathcal{C}$ and a question q, our goal is to retrieve evidence from $\mathcal{C}$ that supports q and use it to generate a coherent response. We first construct a knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ from $\mathcal{C}$ using an LLM, where nodes are entities and edges are the relations between them. The task then proceeds in three steps: we retrieve a query-relevant subgraph $\mathcal{G}_q$ from $\mathcal{G}$, use $\mathcal{G}_q$ to select the chunks $\mathcal{C}_q \subseteq \mathcal{C}$ that contain useful evidence, and have an LLM produce the answer A from q, $\mathcal{G}_q$, and $\mathcal{C}_q$.

As illustrated in Figure 2, our framework consists of three stages: (1) preprocessing, where we construct a knowledge graph from the corpus (Section 3.2); (2) chunk selection, where we retrieve and organize relevant chunks into a tree through six steps (Section 3.3); and (3) answer generation, where we recursively process the tree from the bottom up using divide-and-conquer (Section 3.4). The graph captures the relational structure that conventional retrieval misses, while the tree reflects the hierarchical dependencies of multi-hop reasoning. After each round of chunk selection, when starting from a new selected chunk, higher-order neighbors in KG and nodes in the tree can be updated by iteratively repeating steps 1-6. A complete worked example of the full pipeline is illustrated in Appendix H.

3.2 Knowledge Graph Construction

Given a text corpus $\mathcal{C}$, we construct a knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ by prompting an LLM to extract entities and relations from each chunk.

For each chunk, the LLM returns a set of entities, each with a name, type, and short description, and a set of relations between them. Each relation has a source entity, a target entity, a natural language

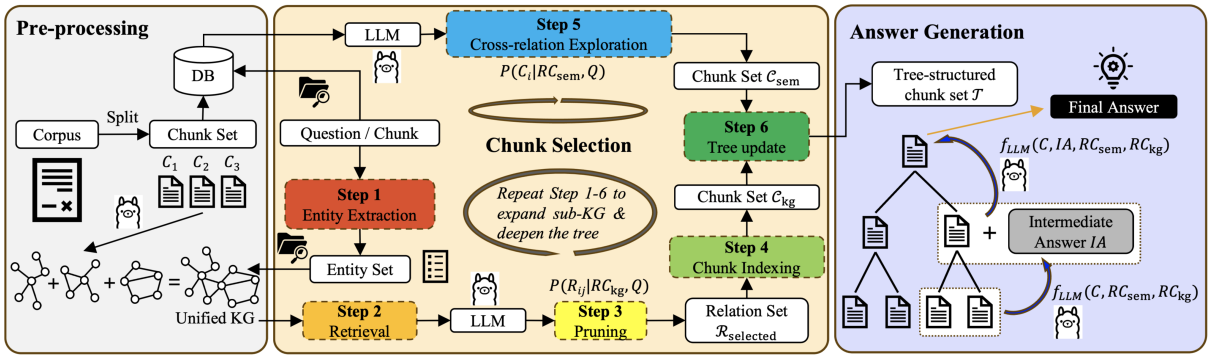


Figure 2: Graph2Tree-RAG consists of three main stages: pre-processing, chunk selection, and answer generation. First, a knowledge graph is constructed from the text corpus by extracting entities and relations from each chunk using an LLM, after which identical entities are merged via semantic similarity. When given a question, we retrieve relevant relations from the knowledge graph, apply LLM-based pruning based on the evolving reasoning chain, group relations by source chunk, and index chunks to build a tree-structured representation through iterative expansion. During answer generation, we recursively generate intermediate answers from the bottom up using a divide-and-conquer strategy, where each node synthesizes its content with the answers of its children to produce the final answer at the root.

description, the identifier of its source chunk, and a strength score $stg \in [0, 10]$ that reflects how confidently the relation holds in that chunk. Formally, we define it as the conditional probability of the relation R_i given the chunk c_j: $stg = P(R_i \mid c_j)$, where R_i denotes the relation occurrence and c_j is the corresponding text chunk.

Each chunk yields a local subgraph. We merge all local subgraphs into the unified graph $\mathcal{G}$. The merging procedure is in Appendix A.

3.3 Chunk Selection

Given a question q and the unified knowledge graph $\mathcal{G}$, chunk selection retrieves a query-relevant subgraph, identifies evidence chunks, and organizes them into a tree. It proceeds through six steps: entity extraction from the question, relation retrieval, and LLM-based semantic pruning (Steps 1 to 3); chunk indexing, where we score and rank the relevant chunks (Step 4); cross-relation exploration, where we pick chunk candidates by semantic similarity and add cross-relations (Step 5); and tree update, where we organize the retrieved chunks into a tree by their relation chains (Step 6). Steps 1 to 6 are then repeated with each new chunk as a new query, expanding the sub-knowledge graph and deepening the tree.

We define a **Relation Chain** (RC), which captures the cumulative set of relations retrieved along an exploration path. Formally, RC_{kg} is an ordered list $[r_1, r_2, \ldots, r_t]$, where each r_i denotes a relation selected as relevant in previous steps. The chain expands dynamically as the subgraph grows, and each chunk maintains its own unique RC reflecting its specific retrieval trajectory. Initially, RC_{kg} is empty. In addition, we maintain a second chain RC_{sem} that tracks semantic connections between the question and selected chunks, also initialized as empty.

Entity Extraction, Retrieval, and Pruning (Steps 1 to 3) : We extract entities from q using the same prompt as in Section 3.2 and map them to nodes in $\mathcal{G}$ via the entity resolution method in Appendix A. For each matched query entity, we run a top-down approach–Breadth First Search (BFS) over $\mathcal{G}$ to collect every relation in which it appears as source or target. These form the candidate relations.

The candidates often contain noise. We score each one with two pruning mechanisms to obtain a relevance score $\rho(r)$. The first uses an LLM: we prompt it to judge how relevant adding the candidate r to the current $\text{RC}_{\text{kg}} \cup \text{RC}_{\text{sem}}$ would be (prompt template in Appendix B.1). The second is semantic-based: we compute the cosine similarity between r and the current $\text{RC}_{\text{kg}} \cup \text{RC}_{\text{sem}}$. When RC is empty, we replace it with the query embedding.

We then apply a threshold θ to filter relations: defined as $\mathcal{R}_{\text{selected}} = \{r \in \mathcal{R}_{e_q} \mid \rho(r) \geq \theta\}$. Relations with scores below θ are discarded. The query-relevant subgraph $\mathcal{G}_q$ contains $\mathcal{R}_{\text{selected}}$ together with the source and target entities of these relations.

Chunk Indexing (Step 4): After pruning, we organize and rank the chunks that produced the selected relations.

We group the relations in $\mathcal{R}_{\text{selected}}$ by their source chunk. This yields a set of candidate chunks, each carrying the subset of relations that came from it.

We score each candidate chunk in two stages. For each of its relations r, the relation-level score is $\text{Rel}(r) = \sqrt{stg \times \rho(r)}$, where the square root smooths the two factors and prevents extreme values from dominating. The chunk score $\rho(c)$ then pools these scores via a function f that can be Sum, Average, or Max.

We sort the candidates by $\rho(c)$ and keep the top k as $\mathcal{C}_{\text{kg}}$. Ties are broken by the cosine similarity between the chunk embedding and the embedding of the current RC_{kg}.

For each selected chunk $c \in \mathcal{C}_{\text{kg}}$, we update its own relation chain by appending the relations originating from this chunk: $\text{RC}_{\text{kg},c} \leftarrow \text{RC}_{\text{kg},c} \oplus \mathcal{R}_c$, where $\oplus$ denotes concatenation.

Cross-relation Exploration (Step 5): While our graph-based retrieval effectively exploits structured knowledge, relying solely on relation triples has two inherent limitations. First, knowledge graphs derived from LLMs usually suffer from incompleteness. Second, relation descriptions are inherently local, focusing on specific entity pairs and their connections, which may not capture the overall theme or context of a chunk. Accordingly, we complement our structured retrieval with a semantic-guided expansion mechanism that captures broader semantic relevance.

We first retrieve the top k chunks most similar to the question by cosine similarity, denoted $\mathcal{C}_{\text{sem}}$. An LLM judges the relevance of each chunk in $\mathcal{C}_{\text{sem}}$. For each chunk c that passes, we extract cross-relations between the question and the chunk via a second prompt and append them to its own chain: $\text{RC}_{\text{sem, c}} \leftarrow \text{RC}_{\text{sem, c}} \oplus \mathcal{R}_{\text{cross}}(c, q)$. The prompt templates are in Appendix B.2.

The two chains complement each other: RC_{kg} captures structured reasoning paths through the knowledge graph, and RC_{sem} captures direct semantic bridges between the question and the chunks. If a chunk is selected in both $\mathcal{C}_{\text{kg}}$ (Step 4) and $\mathcal{C}_{\text{sem}}$ (Step 5), we merge its RC_{kg} and RC_{sem} so that the chunk carries a single combined chain into the tree update step.

Tree Update (Step 6): We organize the retrieved chunks and their relation chains into a hierarchical tree that mirrors the multi-hop reasoning process. The tree $\mathcal{T}$ starts with the query q as the root, and each chunk in $\mathcal{C}_{\text{kg}} \cup \mathcal{C}_{\text{sem}}$ is added as a child, which is the foundation for recursive answer generation.

To retrieve multi-hop neighbors and deepen the tree, each newly added chunk is then treated as a new 'query' and Steps 1 through 6 are repeated. The chunk inherits the current RC_{kg} and RC_{sem} and extends them with the newly selected relations, rather than starting from scratch. This iterative process progressively adds chunks and grows the tree. The pseudocode is in Appendix C.

3.4 Answer Generation

Given the tree $\mathcal{T}$, we employ a divide-and-conquer strategy to recursively generate answers from the bottom up. Concatenating retrieved chunks as a flat sequence does not capture the logical dependencies needed for multi-hop reasoning, and direct concatenation can exceed the LLM's input length limit when many chunks are retrieved. Our hierarchical strategy addresses both issues by synthesizing intermediate answers while staying within context limits.

For a node $n \in \mathcal{T}$, the intermediate answer $\mathcal{I}(n)$ distills the evidence from n itself, its relation chain RC_n, and all its descendants. For a leaf, $\mathcal{I}(n)$ is the node's content. For an internal node, the children have already been replaced by their intermediate answers, so the node combines its content and RC_n with the children's answers and feeds the result to the LLM: $\mathcal{I}(n) = f_{\text{LLM}}\big(n, \text{RC}_n, \{\mathcal{I}(u)\}_{u \in \text{children}(n)}\big)$. The recursion proceeds upward, and $\mathcal{I}$ at the root of $\mathcal{T}$ is the final answer. The pseudocode is in Appendix D.

4 Experiments on Factoid QA

4.1 Datasets and Experimental Settings

We evaluate our proposed method on three widely-used multi-hop question answering benchmarks: **MuSiQue** (Trivedi et al., 2022), **HotpotQA** (Yang et al., 2018), and **2WikiMultihopQA** (Ho et al., 2020). To ensure fair comparison, we adopt the same experimental settings as HippoRAG (Gutiérrez et al., 2024). Detailed dataset statistics are provided in Appendix E.

For text representation, we employ Sentence-BERT (Reimers and Gurevych, 2019) to generate semantic embeddings with a dimensionality of 768. To preserve semantic coherence during preprocessing, we utilize the RecursiveCharacterTextSplitter from LangChain (Chase, 2022). We use LLaMA3-8B (Touvron et al., 2023) with temperature set to 0 as our underlying LLM.

We assess retrieval performance using Recall@k ($R@k$) for $k \in \{2, 5\}$. To evaluate the quality

Table 1: Retrieval performance of single- and multi-step retrievers on MuSiQue, 2WikiMultihopQA, and HotpotQA. Results are reported using Recall@k ($R@k$) for $k \in \{2, 5\}$. The best and second-best results are highlighted in **bold** and underlined, respectively. The rightmost columns report the average performance across the three datasets.

Retriever	MuSiQue		2Wiki		HotpotQA		Average	
	R@2	R@5	R@2	R@5	R@2	R@5	R@2	R@5
Single-step Retrieval								
BM25	32.3	41.2	51.8	61.9	55.4	72.2	46.5	58.4
Contriever	34.8	46.6	46.6	57.5	57.2	75.5	46.2	59.9
GTR	37.4	49.1	60.2	67.9	59.4	73.3	52.3	63.4
ColBERTv2	37.9	49.2	59.2	68.2	64.7	79.3	53.9	65.6
RAPTOR	35.7	45.3	46.3	53.8	58.1	71.2	46.7	56.8
RAPTOR (Contriever)	36.9	46.5	57.3	64.7	63.1	75.6	52.4	62.3
Proposition	37.6	49.3	56.4	63.1	58.7	71.1	50.9	61.2
Proposition (Contriever)	37.8	50.1	55.9	64.9	63.9	78.1	52.5	64.4
HippoRAG (Contriever)	41.0	52.1	**71.5**	**89.5**	59.0	76.2	57.2	72.6
HippoRAG (ColBERTv2)	40.9	51.9	70.7	89.1	60.5	77.7	57.4	72.9
GTRAG	**43.8**	**55.2**	64.2	84.6	**68.2**	**81.3**	**58.7**	**73.7**
Multi-step Retrieval								
IRCoT + BM25	34.2	44.7	61.2	75.6	65.6	79.0	53.7	66.4
IRCoT + Contriever	39.1	52.2	51.6	63.8	65.9	81.6	52.2	65.9
IRCoT + ColBERTv2	41.7	53.7	64.1	74.4	67.9	82.0	57.9	70.0
HippoRAG (Contriever) w/ IRCoT	43.9	56.6	75.3	93.4	65.8	82.3	61.7	77.4
HippoRAG (ColBERTv2) w/ IRCoT	45.3	57.6	**75.8**	**93.9**	67.0	83.0	62.7	78.2
GTRAG	**49.7**	**62.4**	73.9	92.7	**71.5**	**91.3**	**65.0**	**82.1**

of the final generated answers, we adopt Exact Match (EM) and F1 score. Detailed definitions of all metrics are provided in Appendix F.

4.2 Baselines

We compare GTRAG with several representative baselines, including BM25 (Robertson and Walker, 1994), Contriever (Izacard et al., 2021), GTR (Ni et al., 2022), ColBERTv2 (Santhanam et al., 2022), GraphRAG (Edge et al., 2024), Propositionizer (Chen et al., 2024), RAPTOR (Sarthi et al., 2024), HippoRAG (Gutiérrez et al., 2024), IRCoT (Trivedi et al., 2023), and SiReRAG (Zhang et al., 2025). All baseline results are obtained either through our own reproduction or from the best publicly available results reported in prior work. In addition, to highlight the distinctiveness of our approach, we provide a detailed comparison between our framework and various baselines in Appendix G.

4.3 Retrieval Performance

Table 1 presents the retrieval performance of various baselines. Among single-step retrievers, GTRAG achieves state-of-the-art results, with an average $R@5$ of 73.7%. On MuSiQue, GTRAG outperforms HippoRAG (Contriever) by 6.8% at $R@2$ and 5.9% at $R@5$. On HotpotQA, it surpasses ColBERTv2 by 5.4% at $R@2$ and 2.5% at $R@5$. Notably, despite being a single-step retriever, GTRAG demonstrates even greater advantages in the multi-step retrieval setting. On MuSiQue, the most challenging dataset due to its complex compositional reasoning requirements, GTRAG achieves $R@2 = 49.7\%$ and $R@5 = 62.4\%$, surpassing HippoRAG (ColBERTv2) w/ IRCoT by 9.7% and 8.3%, respectively. On HotpotQA, GTRAG attains $R@2 = 71.5\%$ and $R@5 = 91.3\%$, outperforming all baselines by substantial margins.

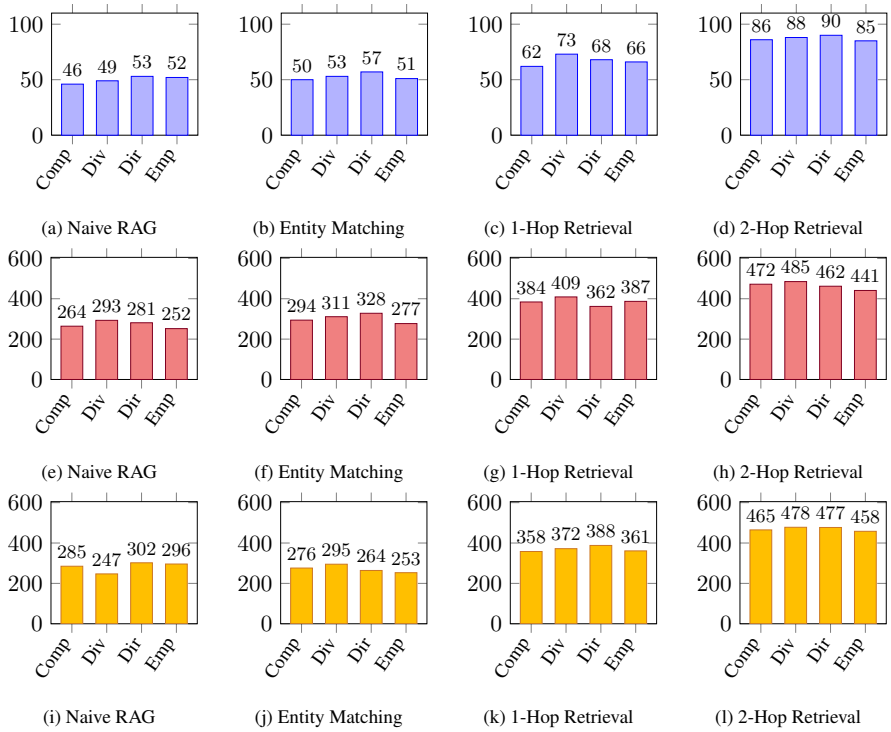
Furthermore, we observe that the improvement rate for $R@2$ is generally higher than that for $R@5$, suggesting that GTRAG is more effective at retrieving the required information with less noise. This also demonstrates the effectiveness of our relation chain-based pruning mechanism.

However, compared to HippoRAG, which also leverages knowledge graphs for retrieval, GTRAG's performance on 2WikiMultihopQA is not particularly outstanding. As shown in Appendix E, the density of the knowledge graph we constructed on this dataset is lower than that of the other two datasets, which may limit our method's ability to retrieve the necessary chunks. This observation also highlights the extent to which our approach relies on the quality of the constructed knowledge graph.

Table 2: QA performance of multiple baselines on MuSiQue, 2Wiki, and HotpotQA. Results are reported using EM and F1 score metrics. Best results are in **bold** and second best are underlined. The rightmost columns show the average performance across the three datasets.

Retriever	MuSiQue		2Wiki		HotpotQA		Average	
	EM	F1	EM	F1	EM	F1	EM	F1
BM25	13.8	21.5	40.3	44.8	41.2	53.2	31.8	39.9
ColBERTv2	15.5	26.4	33.4	43.3	43.4	57.7	30.8	42.5
HippoRAG	32.6	43.8	**66.4**	**74.0**	59.9	74.3	53.0	64.0
RAPTOR	36.4	49.1	53.8	61.5	58.0	73.1	49.4	61.2
GraphRAG	12.1	20.2	22.5	27.5	31.7	42.7	22.1	30.2
IRCoT (ColBERTv2)	19.1	30.5	35.4	45.1	45.5	58.4	33.3	44.7
HippoRAG (ColBERTv2) w/ IRCoT	21.9	33.3	47.7	62.7	45.7	59.2	38.4	51.7
SIRERAG	40.5	53.1	59.6	67.9	61.7	76.5	53.9	65.8
HopRAG	42.2	54.9	61.1	68.3	62.0	76.1	55.1	66.4
GTRAG	**43.5**	**56.8**	62.4	69.2	**63.1**	**77.3**	**56.3**	**67.8**

Figure 3: Open-ended QA results on TraCR, QA-Expert-multi-hop-qa-V1.0, and MultiHop-RAG. Subfigures (a)–(d) report results on TraCR, (e)–(h) on QA-Expert-multi-hop-qa-V1.0, and (i)–(l) on MultiHop-RAG. Each bar indicates the number of queries for which the corresponding method achieved the highest score (or tied for the highest score) among all variants in that category. For example, in subfigure (a), Naive RAG achieves a value of 46 on Comprehensiveness, meaning that across the 91 TraCR queries, Naive RAG was judged by the LLM to have the highest (or tied for the highest) score on 46 queries.



Note: Comp = Comprehensiveness, Div = Diversity, Dir = Directness, Emp = Empowerment.

4.4 Generated Answer Quality

We evaluate the QA performance of all methods on MuSiQue, 2WikiMultihopQA, and HotpotQA using Exact Match (EM) and F1 score. Results are reported in Table 2. All answers are generated using the top-20 retrieved chunks from each method.

GTRAG achieves the best average performance across all three datasets, with an average EM of 56.3% and F1 of 67.8%, outperforming the strongest baseline by 2.2% and 2.1%, respectively. This demonstrates the effectiveness of our answer generation strategy. On MuSiQue, GTRAG achieves EM = 43.5% and F1 = 56.8%, surpassing HopRAG by 3.1% and 3.5%, respectively. On

HotpotQA, GTRAG attains the highest EM of 63.1% and F1 of 77.3%, outperforming all baselines. On 2WikiMultihopQA, GTRAG ranks second behind HippoRAG, achieving EM = 62.4% and F1 = 69.2%.

As expected, higher retrieval recall generally leads to better answer quality. This trend is evident across most baselines: HippoRAG achieves the highest retrieval recall on 2WikiMultihopQA (R@5 = 93.9%) and correspondingly attains the best QA performance (EM = 66.4%) on that dataset. Similarly, GTRAG's strong retrieval results on HotpotQA (R@5 = 91.3%) translate into the highest QA performance (EM = 63.1%, F1 = 77.3%). These observations confirm that effective retrieval is a critical foundation for successful answer generation.

5 Experiments on Open-Ended QA

5.1 Datasets and Evaluation Protocol

To evaluate the effectiveness of GTRAG on open-ended QA, we conduct experiments on three additional datasets: **TraCR** (Thomas et al., 2025), **QA-Expert-multi-hop-qa-V1.0** (Mai, 2023), and **MultiHop-RAG** (Tang and Yang, 2024). TraCR contains 91 queries, while for the latter two datasets, we randomly sample 500 queries each to ensure computational feasibility. Unlike factoid QA datasets where answers are short entities or phrases, these datasets—particularly the first two—require generating free-form, abstractive responses. Consequently, traditional token-level metrics such as EM and F1 are less suitable for evaluating answer quality.

Following GraphRAG (Edge et al., 2024), we adopt an LLM-based evaluation strategy. Specifically, we prompt an LLM (LLama3-8B with temperature set to 0) to assess each generated answer along four dimensions: *Comprehensiveness*, *Diversity*, *Directness*, and *Empowerment*. Each dimension is scored on a scale of 1 to 10.

5.2 Ablation Study Design

We conduct an ablation study on the three QA datasets, and compare the following variants of our method: **Entity Matching**: A simplified version that retrieves chunks by matching query entities with KG nodes. Specifically, for each query entity e_q, we compute its semantic similarity with all nodes in $\mathcal{G}$ using the same encoder as in Section 3.2. The similarity score directly serves as the relevance score for the corresponding chunk. Retrieved chunks are then grouped and ranked according to their associated entity scores, and the top-k chunks are selected as the final retrieval result. **1-Hop Retrieval**: Our full retrieval pipeline restricted to first-hop neighbors only. No iterative expansion is performed. **2-Hop Retrieval**: The full GTRAG pipeline with iterative expansion up to two hops. **Naive RAG**: A traditional RAG baseline that retrieves chunks solely based on dense semantic similarity between the query and each chunk, without any graph-based or multi-hop reasoning.

5.3 Results & Analysis

To ensure fair comparison, the retrieval budget (top-k chunks) is kept consistent across all variants with $k = 6$. As shown in Figure 3, we evaluate four variants of GTRAG along four dimensions: Comprehensiveness, Diversity, Directness, and Empowerment.

From the results, we observe that the 2-hop variant (full GTRAG) achieves consistently superior results across all four dimensions on all three datasets. This clear and uncontested advantage demonstrates that multi-hop reasoning plays a decisive role in improving answer generation quality for open-ended QA tasks.

As expected, Entity Matching achieves performance comparable to Naive RAG, since it merely matches query entities against KG nodes without leveraging any structural relationships within the graph. In contrast, the 1-hop variant, which retrieves immediate neighbors of the matched entities, achieves substantially better results than both Entity Matching and Naive RAG. This performance gap highlights the critical importance of exploiting graph structure for effective open-ended answer generation.

6 Conclusion

In this paper, we proposed **Graph2Tree-RAG** (GTRAG), a novel RAG framework that bridges structured knowledge retrieval with hierarchical answer generation. GTRAG applies multi-stage pruning and reranking based on evolving relation chains, organizes retrieved chunks into a tree structure, and recursively generates answers from the bottom up. Experiments on factoid QA benchmarks demonstrate that GTRAG achieves state-of-the-art retrieval and generation performance. On open-ended QA tasks, our full pipeline consistently outperforms all ablation variants across four evaluation dimensions. Ablation studies further validate the contribution of each component.

7 Limitations

Despite its strong performance, GTRAG has two key limitations: it consumes substantial tokens due to multiple LLM calls, and its iterative expansion leads to high inference latency. Future work will focus on improving computational efficiency and reducing token consumption to make the framework more lightweight and scalable.

References

Muhammad Arslan, Hussam Ghanem, Saba Munawar, and Christophe Cruz. 2024. A survey on rag with llms. *Procedia computer science*, 246:3781–3790.

Jinheon Baek, Alham Fikri Aji, and Amir Saffari. 2023. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. In *Proceedings of the 1st Workshop on Natural Language Reasoning and Structured Explanations (NLRSE)*, pages 78–106.

Yuzheng Cai, Zhenyue Guo, Yiwen Pei, Wanrui Bian, and Weiguo Zheng. 2025. Simgrag: Leveraging similar subgraphs for knowledge graphs driven retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 3139–3158.

Harrison Chase. 2022. Langchain (2022). *URL https://github. com/langchain-ai/langchain*.

Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. 2024. Dense x retrieval: What retrieval granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15159–15177.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6491–6501.

Jinyuan Fang, Zaiqiao Meng, and Craig Macdonald. 2024. Trace the evidence: Constructing knowledge-grounded reasoning chains for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8472–8494.

Masoomali Fatehkia, Ji Kim Lucas, and Sanjay Chawla. 2024. T-rag: lessons from the llm trenches. *arXiv preprint arXiv:2402.07483*.

Zengyi Gao, Yukun Cao, Hairu Wang, Ao Ke, Yuan Feng, S Kevin Zhou, and Xike Xie. 2025. Frag: A flexible modular framework for retrieval-augmented generation based on knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6178–6192.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569.

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*, pages 6609–6625.

Yibo Hu, MohammadSaleh Hosseini, Erick Skorupa Parolin, Javier Osorio, Latifur Khan, Patrick Brandt, and Vito D'Orazio. 2022. Conflibert: A pre-trained language model for political conflict and violence. In *Proceedings of the 2022 conference of the north American chapter of the association for computational linguistics: human language technologies*, pages 5469–5482.

Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. 2025. Grag: Graph retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4145–4157.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2023. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Shilong Li, Yancheng He, Hangyu Guo, Xingyuan Bu, Ge Bai, Jie Liu, Jiaheng Liu, Xingwei Qu, Yangguang Li, Wanli Ouyang, et al. 2024. 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.

Hao Liu, Zhengren Wang, Xi Chen, Zhiyu Li, Feiyu Xiong, Qinhan Yu, and Wentao Zhang. 2025. Hoprag: Multi-hop reasoning for logic-aware retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 1897–1913.

Haochen Liu, Song Wang, Yaochen Zhu, Yushun Dong, and Jundong Li. 2024a. Knowledge graph-enhanced large language models via path selection. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 6311–6321.

Jingyu Liu, Jiaen Lin, and Yong Liu. 2024b. How much can rag help the reasoning of llm? *arXiv preprint arXiv:2410.02338*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024c. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2023. Reasoning on graphs: Faithful and interpretable large language model reasoning. *arXiv preprint arXiv:2310.01061*.

Khai Mai. 2023. Qa expert: Llm for multi-hop question answering. https://github.com/khaimt/qa_expert.

Aigerim Mansurova, Aiganym Mansurova, and Aliya Nugumanova. 2024. Qa-rag: Exploring llm reliance on external knowledge. *Big Data and Cognitive Computing*, 8(9):115.

Costas Mavromatis and George Karypis. 2024. Gnn-rag: Graph neural retrieval for large language model reasoning. *arXiv preprint arXiv:2405.20139*.

Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernandez Abrego, Ji Ma, Vincent Zhao, Yi Luan, Keith Hall, Ming-Wei Chang, et al. 2022. Large dual encoders are generalizable retrievers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9844–9855.

Shahzad Qaiser and Ramsha Ali. 2018. Text mining: use of tf-idf to examine the relevance of words to documents. *International journal of computer applications*, 181(1):25–29.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 3982–3992.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pages 5418–5426.

Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*, pages 232–241. Springer.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. Colbertv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734.

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*.

Zhili Shen, Chenxin Diao, Pavlos Vougiouklis, Pascual Merita, Shriram Piramanayagam, Enting Chen, Damien Graux, Andre Melo, Ruofei Lai, Zeren Jiang, et al. 2025. Gear: Graph-enhanced agent for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 12049–12072.

Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.

Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391*.

Ostonya Thomas, Muhaimin Bin Munir, Jean-Michel Tine, Mizanur Rahman, Yuchen Cai, Khandakar Ashrafi Akbar, Md Nahiyan Uddin, Latifur Khan, Trayce Hockstad, and Mashrur Chowdhury. 2025. Transportation cyber incident awareness through generative ai-based incident analysis and retrieval-augmented question-answering systems. *arXiv preprint arXiv:2508.02523*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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.

Juyuan Wang, Rongchen Zhao, Wei Wei, Yufeng Wang, Mo Yu, Jie Zhou, Jin Xu, and Liyan Xu. 2026. Comorag: A cognitive-inspired memory-organized rag for stateful long narrative reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 33557–33565.

Xiang Wei, Xingyu Cui, Ning Cheng, Xiaobin Wang, Xin Zhang, Shen Huang, Pengjun Xie, Jinan Xu, Yufeng Chen, Meishan Zhang, et al. 2023. Chatie: Zero-shot information extraction via chatting with chatgpt. *arXiv preprint arXiv:2302.10205*.

Junde Wu, Jiayuan Zhu, Yunli Qi, Jingkun Chen, Min Xu, Filippo Menolascina, and Vicente Grau. 2024. Medical graph rag: Towards safe medical large language model via graph retrieval-augmented generation. *arXiv preprint arXiv:2408.04187*.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. 2024. Do large language models latently perform multi-hop reasoning? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10210–10229.

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.

Nan Zhang, Prafulla Kumar Choubey, Alexander Fabbri, Gabriel Bernadett-Shapiro, Rui Zhang, Prasenjit Mitra, Caiming Xiong, and Chien-Sheng Wu. 2025. Sirerag: Indexing similar and related information for multihop reasoning. In *International Conference on Learning Representations*, volume 2025, pages 100413–100432.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2):1–124.

A Knowledge Graph Fusion

We construct a unified knowledge graph from a collection of local subgraphs by merging entities that refer to the same real-world object. To this end, we adopt a semantic similarity-based approach for entity resolution. For each entity $e \in \bigcup_{k=1}^{n} \mathcal{V}_k$, we first encode it using a BERT-based encoder to obtain its embedding: $\mathbf{e}_{\text{emb}} = \text{Encoder}(e)$. Then we compute the cosine similarity between $\mathbf{e}_{\text{emb}}$ and the embeddings of all other entities. Two entities e_i and e_j are initially considered candidates for merging if their similarity exceeds a predefined threshold τ:

$$\text{sim}(e_i, e_j) = \begin{cases} 1 & \text{if cosine}(\mathbf{e}_{i,\text{emb}}, \mathbf{e}_{j,\text{emb}}) \geq \tau \\ 0 & \text{otherwise} \end{cases}$$

For each pair of entities (e_i, e_j) with $\text{sim}(e_i, e_j) = 1$, we further leverage an LLM with a carefully designed prompt to determine whether they refer to the same entity. Only when the LLM outputs an affirmative decision are they considered identical. When entities are merged, their attributes and descriptions are aggregated, and their associated relations are consolidated under the unified entity.

Let $\sim$ be the equivalence relation on entities defined by the merging rule: $e_i \sim e_j$ if and only if e_i and e_j refer to the same real-world object. For each entity $e \in \bigcup_{j=1}^{n} \mathcal{V}_j$, let $[e] = \{e' \mid e' \sim e\}$ denote its equivalence class. The unified knowledge graph $\mathcal{G}$ is defined as the quotient graph: $\mathcal{G} = (\mathcal{V}^*, \mathcal{E}^*)$, where $\mathcal{V}^* = \{[e] \mid e \in \bigcup_{k=1}^{n} \mathcal{V}_k\}$, i.e., each node represents an equivalence class of merged entities, and $\mathcal{E}^* = \{([e_s], [e_t]) \mid (e_s, e_t) \in \bigcup_{k=1}^{n} \mathcal{E}_k \text{ with } [e_s] \neq [e_t]\}$, i.e., edges connect distinct equivalence classes, preserving all relations between their members.

B LLM Agent Prompts

B.1 Relation Pruning

To assign an importance score to each retrieved relation r, we leverage an LLM to evaluate how much r contributes to answering the question q, given the existing reasoning context captured by RC_{kg}. We formalize this as the conditional relevance score: $\rho(r) = P(r \mid \text{RC}_{\text{kg}}, q)$, which represents the probability, as assessed by the LLM, that relation r is beneficial for answering q when considered alongside the existing relation chain RC_{kg}. The prompt template used for this evaluation is presented in Fig 5.

B.2 Cross-Chunk Relation Extraction

For each candidate chunk $c_i \in \mathcal{C}_{\text{sem}}$, we design a prompt that instructs the LLM to perform two tasks. First, it computes $\text{Rel}(c_i, q) = P(c_i \mid \text{RC}_{\text{sem}}, q)$, which determines whether c_i is relevant to answering q. Second, if the chunk is deemed relevant,

the LLM extracts cross-relations between the question and the chunk. Specifically, for a relevant chunk c_i, the LLM extracts a set of cross-relations $\mathcal{R}_{\text{cross}}(c_i, q)$, where each cross-relation is a triple $(e_s, e_t, r_{\text{desc}})$. Each cross-relation connects an entity from the question with an entity from the chunk, thereby bridging the query context with the retrieved evidence. The prompt template is shown in Figure 6.

C Multi-Hop Retrieval Algorithm

To capture multi-hop reasoning, we iteratively expand the tree by exploring higher-order neighbors. At each iteration, we start from the current leaf nodes of the tree. For each leaf node n, the same retrieval and pruning mechanisms described in previous sections are applied to identify new relevant chunks. Any newly retrieved chunk that is not already present in the tree is added as a child node of n. This process is formally outlined in Algorithm 1.

D Answer Generation Algorithm

The recursive bottom-up answer generation procedure is formalized in Algorithm 2. Starting from leaf nodes, each node generates its intermediate answer by either directly outputting its content (for leaves) or combining its content and RC with the intermediate answers of its children (for internal nodes). This propagation ensures that intermediate answers are progressively refined as information flows upward, enabling the LLM to leverage hierarchical reasoning rather than processing all retrieved chunks in a flat, unstructured manner.

E Dataset Statistics

Details of evaluation datasets are shown in Table 3. The knowledge graph construction process is introduced in Section 3.2, and the entity and relation extraction prompt is adapted from GraphRAG (Edge et al., 2024).

F Evaluation Metrics

Here we provide detailed definitions of the evaluation metrics used in our experiments. Suppose $\mathcal{Q} = \{q_1, q_2, \ldots, q_m\}$ be the set of test queries.

Recall measures the proportion of ground-truth relevant documents that are successfully retrieved within the top-k results. For each query q_i, let $\mathcal{R}_i$ denote the set of ground-truth relevant text chunks, and let $\text{Top}_k(q_i)$ denote the set of top-k chunks retrieved by the model. The overall Recall@k across all queries is the average of individual scores:

$$\text{R@}k = \frac{1}{m} \sum_{i=1}^{m} \frac{|\mathcal{R}_i \cap \text{Top}_k(q_i)|}{|\mathcal{R}_i|}$$

where $|\cdot|$ denotes the cardinality of a set.

Algorithm 1: Graph2Tree-RAG Retrieval

Input: Query q, knowledge graph $\mathcal{G}$, maximum hop h
Output: Tree-structured data $\mathcal{T}$

Initialize RC $\leftarrow \emptyset$; queue $\leftarrow [\,]$; $\mathcal{T} \leftarrow [q]$;
Extract entity set $\mathcal{E}_q$ from q;
queue.add({'text' : q, 'entity set' : $\mathcal{E}_q$, 'RC' : RC, 'hop' : 0});
while *queue* $\neq \emptyset$ **do**
  $t \leftarrow$ queue.pop(0)['text'];
  $\mathcal{E}_t \leftarrow$ queue.pop(0)['entity set'];
  current_RC $\leftarrow$ queue.pop(0)['RC'];
  hop $\leftarrow$ queue.pop(0)['hop'];
  $\mathcal{R}_t \leftarrow \{r \in \mathcal{E} \mid \exists e \in \mathcal{E}_t, e \in \{e_s(r), e_t(r)\}\}$; // Subgraph Retrieval
  for *each* $r \in \mathcal{R}_t$ **do**
    $\rho(r) \leftarrow P(r \mid \text{current_RC}, t)$; // LLM-based or semantic-based pruning
  end
  $\mathcal{C}_{t0} \leftarrow \text{Top-}k(\{c_i \mid f_{pool}(\rho(r), r_{id} = c_i)\})$; // Chunk Indexing
  $\mathcal{C}_{t1} \leftarrow \text{Top-}k(\{c_i \mid \text{sim}(\mathbf{q}_{\text{emb}}, \mathbf{c}_{i,\text{emb}})\})$; // Semantic-Guided Chunk Expansion
  $\mathcal{R}_{\text{cross}} \leftarrow \{f_{LLM}(e_s, e_t, r_{\text{desc}}) \mid e_s \in \mathcal{E}_t, e_t \in \mathcal{E}_c, c \in \mathcal{C}_{t1}\}$; // Cross-Relation Extraction
  for *each* $c \in \mathcal{C}_{t0} \cup \mathcal{C}_{t1}$ **do**
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{(t, c)\}$; // Add c as a child node of t
    if *hop* $< h$ **then**
      $\text{RC}_t \leftarrow \text{current_RC} \oplus \{r \in \mathcal{R}_t \cup \mathcal{R}_{\text{cross}} \mid r_{id} = c\}$;
      $\mathcal{E}_c \leftarrow \{e \mid \exists r \in \mathcal{R}_t, r_{id} = c, e \in \{e_s(r), e_t(r)\}\}$;
      queue.add({'text' : c, 'entity set' : $\mathcal{E}_c$, 'RC' : RC_t, 'hop' : hop + 1});
    end
  end
end

Algorithm 2: Bottom-Up Recursive Answer Generation

Input: Tree-structured chunk set $\mathcal{T} = (\mathcal{N}, \mathcal{L})$ with root node n_{root}
Output: Final answer A

```
for i = (L − 1) to 1 step −1 do
    N_i = {n ∈ N | level(n) = i};
    // Process all nodes at depth i
    for each n ∈ N_i do
        child_answers ← [ ];
        for each child u ∈ children(n) do
            child_answers.append(I(u));
        end
        I(n) ←
          f_LLM(n, RC_n, child_answers);
    end
end
A ←
  f_LLM(n_root, RC_{n_root}, {I(u)}_{u∈children(n_root)});
```

Table 3: Statistical summary of the evaluation datasets.

Statistic	MuSiQue	HotpotQA	2Wiki
# Queries	1,000	1,000	1,000
Max Hops	4	2	2
# Chunks	11,656	9858	6273
# Entities	74863	69343	28861
# Relations	61315	56759	22078
Avg. # E/C	6.4	5.8	4.6
Avg. # R/C	5.3	7.0	3.5

Note: E/C = Entities per Chunk, R/C = Relations per Chunk.

Exact Match (EM) is a strict evaluation metric that measures the percentage of predictions that exactly match the ground truth answer. For each query q_i, let a_i be the ground truth answer and $\hat{a}_i$ be the model's predicted answer. Let $\text{norm}(\cdot)$ be a function that converts a string to a canonical form. The EM score is defined as:

$$\text{EM} = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}\left(\text{norm}(\hat{a}_i) = \text{norm}(a_i)\right)$$

where $\mathbb{I}(\cdot)$ is the indicator function that returns 1 if the condition is true and 0 otherwise.

F1 score is a more lenient evaluation metric than EM, measuring the token-level overlap between the predicted answer and the ground truth answer. Let A_i be the set of tokens in the ground truth answer, and let $\hat{A}_i$ be the set of tokens in the predicted answer. Precision and recall are defined as:

$$\text{Pr}_i = \frac{|A_i \cap \hat{A}_i|}{|\hat{A}_i|}, \qquad \text{Re}_i = \frac{|A_i \cap \hat{A}_i|}{|A_i|}$$

The F1 score for query q_i is then the harmonic mean of precision and recall:

$$\text{F1} = \frac{1}{m} \sum_{i=1}^{m} \frac{2 \times \text{Pr}_i \times \text{Re}_i}{\text{Pr}_i + \text{Re}_i}$$

G Baselines Comparison

We compare GTRAG with several state-of-the-art baselines, including GRAG (Hu et al., 2025), GEAR (Shen et al., 2025), FRAG (Gao et al., 2025), HopRAG (Liu et al., 2025), and SimGRAG (Cai et al., 2025). The comparison is conducted along the following dimensions: **Knowledge Graph Construction**: whether the method builds a KG from unstructured text (e.g., chunks or corpora) or relies on an existing KG. **Query Decomposition**: whether the method incorporates query decomposition. **Retrieval Strategy**: the mechanism used for retrieving relevant information from the KG. **Pruning Mechanism**: what pruning mechanism is applied to filter irrelevant information. **Subgraph Organization**: how the retrieved subgraph is structured and represented. **Augmentation Data**: what data is fed to the LLM as augmented context. **Prompting Strategy**: the design of prompts used for LLM interaction. **Answer Generation**: the strategy adopted for generating the final answer. The comparison results are summarized in Table 4.

H Example

Figure 4 illustrates the two-hop retrieval process of GTRAG on a single query.

The query is the root of the tree, and both relation chains RC_{kg} and RC_{sem} start empty. We extract the entity 'Distribution of Industry Act' from the query as the starting point for retrieval. To keep the example readable, we set $k = 1$, so each chunk indexing step and each semantic-guided expansion step keeps only the single top-ranked chunk, and we omit the detailed scoring and pruning steps, reporting only the chunks that are finally selected.

One-Hop Retrieval: Starting from the query, we run Steps 1 to 6 from Section 3 to obtain its children. The knowledge graph branch runs BFS from the query entity, followed by pruning and chunk

indexing, and finally selects chunk 1. Its RC_{kg} then records the relation linking the query entity 'Distribution of Industry Act' to 'Clement Attlee', which involves one entity from the query and one from chunk 1. The semantic-guided branch retrieves the chunks most similar to the query and prompts the LLM to extract cross-relations between the query and each chunk; suppose it selects chunk 2, whose RC_{sem} records a cross-relation that selects 'Harold Wilson'. The root thus has two children: chunk 1 from knowledge graph retrieval and chunk 2 from semantic-guided expansion.

Two-Hop Retrieval: Each new chunk is then treated as a new 'query', and Steps 1 to 6 are repeated on it. When expanding from a chunk, we no longer consider the original query; only the chunk and its inherited relation chain are used. Both the knowledge graph branch and the semantic-guided branch are applied to chunk 1 and chunk 2.

Expanding from chunk 1, we take its selected entity 'Clement Attlee' as the new starting point. BFS and pruning guided by the current RC_{kg} select chunk 3, whose RC_{kg} gains two new relations and two new entities. The semantic-guided branch then selects chunk 4. Chunk 1 therefore has two children, chunk 3 and chunk 4.

Expanding from chunk 2 follows the same procedure. The semantic-guided branch selects chunk 5, but the knowledge graph branch finds no relevant relation containing the entity 'Harold Wilson' selected from chunk 2 after retrieval and pruning, hence it returns nothing. Chunk 2 thus has only one child, chunk 5.

Answer Generation: The retrieval process stops after two hops, producing the final tree shown in Figure 4. Answer generation then proceeds bottom up over this tree: the subtree rooted at chunk 1 yields the intermediate answer $\mathcal{I}(1)$ from chunk 1, chunk 3, and chunk 4, while the subtree rooted at chunk 2 yields a second intermediate answer $\mathcal{I}(2)$ from chunk 2 and chunk 5. Note that both intermediate answers are generated by the LLM.

$\mathcal{I}(1)$: *The Distribution of Industry Act 1950 was passed by the Labour government, which was led by Clement Attlee. Clement Attlee was the Prime Minister of the United Kingdom from 1945 to 1951. The Act strengthened the powers of the Board of Trade in the Development Areas.*

$\mathcal{I}(2)$: *Harold Wilson was the Prime Minister when the Industry Act 1975 (c. 68) was passed. The Act was passed by his Labour government in 1975. Jim Callaghan, who served as Prime Minister from 1976 to 1979, is mentioned as a separate entity, but there is no direct connection to the Distribution of Industry Act.*

The LLM combines the root (query) with $\mathcal{I}(1)$ and $\mathcal{I}(2)$ to produce the final answer: **1945-1951**.

Table 4: Comparison of Advanced RAG Approaches. Our proposed model is highlighted in **bold**.

Baselines	KG	UDU	QD	Subgraph Retrieval	Pruning	Graph-to-Text	Aug. Materials	Prompts Design	Answer Gen.
LLM only	✗	✗	✗	✗	✗	✗	✗	Hard Prompts	One-step Output
Naive RAG	✗	✓	✗	✗	✗	✗	Unstructured Chunk Set	Hard Prompts	One-step Output
GRAG	✓	✗	✗	Ego-graph Ranking & Expansion	Textual Subgraph Soft Pruning	Hierarchical Description	Subgraphs	Hard Prompts + GNN-based Soft Prompts	One-step Output
GEAR	✓	✓	✓	Diverse Triple Beam Search	Base Retrieval Function	Reciprocal Rank Fusion	Unstructured Chunk Set	Hard Prompts	Iterative Output
FRAG	✓	✗	✗	BFS + Reasoning-aware Module	Multiple Ranking Mechanisms	Reasoning Path Construction	Filtered Reasoning Path Set	Hard Prompts	One-step Output
HopRAG	✓	✓	✗	BFS + LLM Reasoning	Helpfulness Metric	✗	Unstructured Chunk Set	Hard Prompts	One-step Output
SimGRAG	✓	✗	✓	Pattern-to-Subgraph Alignment	Graph Semantic Distance Sorting	Subgraph Verbalization	Subgraphs	Hard Prompts	One-step Output
GTRAG	✓	✓	✗	**BFS + Entity Alignment**	**Relation Chain Reasoning**	**Chunk Scoring & Indexing**	**Tree-structured Chunk Set**	**Hard Prompts**	**Recursive Multi-step Output**

Note: KG = Knowledge Graph, UDU = Unstructured Data Usage, QD = Query Decomposition, Aug. = Augmentation, Gen. = Generation. ✓=Yes/Supported, ✗=No/Not Supported.

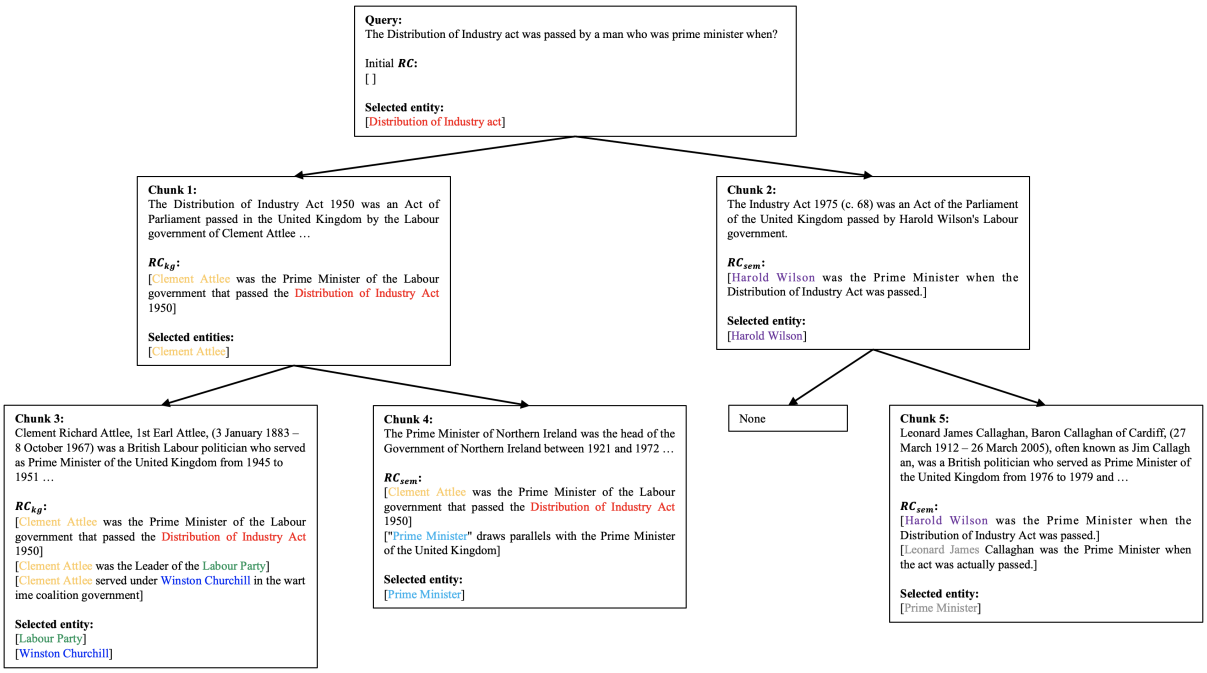


Figure 4: An example of GTRAG's two-hop retrieval process. Each entity is shown in a distinct color so its occurrences across the query, chunks, and relation chains can be traced. The arrow from chunk 2 to an empty placeholder marks that its knowledge graph branch returned no relation after pruning.

```
-Goal-
Given the QUERY, a CURRENT RELATION CHAIN, and a NEW RELATION, evaluate the NEW RELATION by giving a
↪  score.
Note that your explanation is NOT required, and the answer MUST only contain a score in the
↪  following format: {SEP}<your score>{SEP}.

-Steps-
(1). Identify the Connection Point. Definition:
- The connection point must be either the SOURCE ENTITY or TARGET ENTITY of the NEW RELATION.
- Additionally, the same entity must already exist in either:
    The original QUERY, or
    The CURRENT RELATION CHAIN
In other words: Find an entity from the NEW RELATION that matches an entity already present in the
↪  QUERY or anywhere in the existing CURRENT RELATION CHAIN. This matching entity becomes the
↪  connection point.
If there is no connection point (especially note that there can be different entities with similar
↪  or the same names), jump to step (5) and output 0.
Otherwise, go to step (2).

(2). Determine the relevance of the NEW RELATION.
Having a connection point does NOT guarantee that the NEW RELATION is relevant. The NEW RELATION may
↪  still be unrelated to the query or the existing relation chain.
Check Criteria:
- Does the NEW RELATION provide information that contributes to solving the query?
- Does it logically follow from or connect to the existing CURRENT RELATION CHAIN in a way that
↪  advances reasoning?
- Or is it merely topically adjacent but functionally irrelevant?
If no, jump to step (5) and output 0.
If yes, go to step (3).

(3). If there is an existing CURRENT RELATION CHAIN, apply critical thinking and do the proper
↪  reasoning to determine how well it can answer the QUERY without the NEW RELATION.

(4). Combine the analysis results in step (3) and the information (supporting and facts) in the NEW
↪  RELATION you extracted from step (2), and evaluate how helpful the NEW RELATION is to answer the
↪  QUERY.
Give a single score on a scale of 1-10 to represent the helpfulness and logical coherence of the NEW
↪  RELATION.
1-2: Nearly unrelated.
3-4: Loosely related or weak connection. The NEW RELATION shares few entities or concepts with the
↪  CURRENT RELATION CHAIN or introduces irrelevant topics.
5-6: Moderately related. The NEW RELATION has logical relevance or indirect evidence to answer the
↪  query.
7-8: Highly relevant. ONLY the NEW RELATION is highly logically relevant, providing clear evidence
↪  and filling a clear gap.
9-10: Perfect, necessary connection. It directly continues the logical flow from the last step and
↪  is crucial for answering the question.

(5). Give your score. Make sure that your output matches the following format: {SEP}<your
↪  score>{SEP}

-Notes-
Check that your output only contains a number.
Check that your output strictly follows the format: {SEP}<your score>{SEP}.
```

Figure 5: Relation Pruning Prompt

```
-Goal-
Given a query and a text, determine whether the text can help to answer the query or not.

-Steps-
1. Output {begin_delimiter}

2. Given a query and a list of entities from the query, determine whether the text contains further
↪  supporting facts to answer the query.
If no, answer ONLY a single letter N and jump to Step 6.
Otherwise, trying to extract helpful relationships ACROSS the query and text. The details are
↪  described in Step 3.

3. For each relation, (i) first extract a source entity from the query, (ii) extract a target entity
↪  from the text, (iii) establish a relation between them, and format it as:
"relationship"{tuple_delimiter}<source_entity>{tuple_delimiter}<target_entity>{tuple_delimiter}<r⌋
↪  elationship_description>{record_delimiter}
- <source_entity>: name of the source entity
- <target_entity>: name of the target entity
- <relationship_description>: explanation as to why you think the source entity and the target
↪  entity are related to each other based on the relevant information.
(i) Check that the source entity is from the query.
(ii) Check that the target entity is from the text.

4. Process all relations you created in step 3 one by one. Since the target entity in each relation
↪  must be extracted from the TEXT, you only need to output the target entity from TEXT (instead of
↪  the query) in that pair and format it as:
"entity"{tuple_delimiter}<entity_name>{tuple_delimiter}<entity_type>{tuple_delimiter}<entity_desc⌋
↪  ription>{record_delimiter}
- <entity_name>: Name of the target entity, capitalized.
- <entity_type>: Type of the entity, capitalized.
- <entity_description>: Comprehensive description of the entity's attributes based on the relevant
↪  information.
(i) Check that those entities MUST be extracted from the text.
(ii) Check the output MUST strictly adhere to the required format.
(iii) The entity type can be: NAMED ENTITIES (such as PERSON, ORGANIZATION, LOCATION, DATE, TIME,
↪  EVENT, and so on); CONCEPTS AND OBJECTS (such as PRODUCT, VEHICLE, FOOD, ANIMAL, PLANT, and so
↪  on); QUANTITIES AND MEASURES (such as NUMBER, MONEY, PERCENT, WEIGHT, DISTANCE, and so on);
↪  OTHER (such as URL, EMAIL, PHONE NUMBER, ISBN, CODE, and so on); DOMAIN-SPECIFIC ENTITIES (such
↪  as MEDICAL, FINANCIAL, SCIENTIFIC, POLITICAL, and so on).

5. Output {completion_delimiter}

-Input Format-
Query:
<specific query content>

Entity from the query: <specific entity content>

Text:
<specific text content>

Before you output the result,
(1) If there is no extracted relation, check that it contains only the capital letter N, nothing
↪  else.
(2) Check that all extracted relations should ONLY be between entities in the query and entities in
↪  the text
(3) Check the correct use of {begin_delimiter}, {completion_delimiter}, {tuple_delimiter}, and
↪  {record_delimiter}
(4) Check each entity only contains THREE {tuple_delimiter}
(5) Check each relation only contains THREE {tuple_delimiter}
```

Figure 6: Cross-chunk Relation Extraction Prompt