Think on Chunks: Knowledge Graph Chunk-based Retrieval for Question Answering

Anonymous ACL submission

Abstract

The integration of Knowledge Graphs (KGs) into Large Language Model (LLM) pipelines promises to combine structured, precise knowledge with powerful natural language reasoning. However, existing KGQA methods rely on brittle query translation or iterative traversal, limiting robustness and scalability on large KGs. We propose **TiC** (**Th**ink on **C**hunks), a simple yet effective framework that converts a KG from a traversable graph into a retrievable *subgraph-chunk corpus*. TiC comprises (1) an *edge-balanced knowledge-to-chunk transformation* that partitions the KG into semantically coherent, uniformly sized chunks while preserving key local structure for standard indexing and retrieval, and (2) a *planning-guided chunk reasoning* module that replaces multi-turn graph traversal with single- or few-batch subgraph-chunk retrieval driven by a global information plan. This design enables robust, schema-agnostic retrieval and, more importantly, seamless fusion with text retrieval sources. Experiments on three public KGQA benchmarks and a KG-Text hybrid QA benchmark demonstrate that TiC achieves *state-of-the-art or highly competitive performance* while substantially improving execution efficiency, offering a practical paradigm for large-scale LLM-based KGQA.

1 Introduction

Knowledge Graphs (KGs) are structured repositories of factual knowledge over entities and relations, and have long served as core resources for Knowledge Graph Question Answering (KGQA) due to their explicit semantics and support for multi-hop reasoning (Berant et al., 2013; Liang et al., 2024). Recent advances in LLMs have inspired their application to KGQA, leading to two dominant paradigms. Query-based methods use LLMs to translate natural-language questions into formal queries (e.g., SPARQL), which achieve high precision under stable schemas (Luo et al., 2024). More advanced agent-based methods treat the LLM as an interactive reasoning agent that explores the KG step by step, offering flexibility and adaptability (Sun et al., 2024; Jiang et al., 2023; Ma et al., 2025).

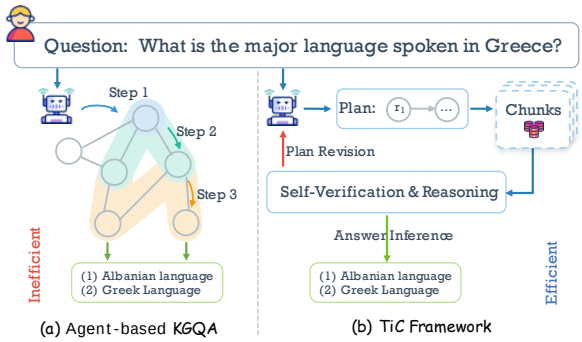

Figure 1: Comparison of (a) traditional agent-based KGQA and (b) our proposed TiC framework, highlighting the shift from multi-step, fine-grained retrieval to single-batch, coarse-grained retrieval.

Agent-based KGQA methods usually incorporate dynamic access to relevant information from the KG during QA. They rely on fine-grained and stepwise exploration over the KG. Moreover, increasingly complex traversal strategies have been introduced in recent years, from local subgraph expansion (Li et al., 2025; Kim et al., 2023), iterative relation traversal (Liu et al., 2024) to adaptive planning-based graph navigation (Guo et al., 2024). However, traversal-based strategies lead to high latency and limited robustness across different KGs. These methods are not as efficient and general as the widely adopted *chunk-based retrieval* paradigm of text corpus retrieval in practice, where external information is gathered in coarse-grained, parallelizable batches for efficient reasoning (Lewis et al., 2020; Shuster et al., 2021).

These limitations and differences motivate us to explore a practical alternative paradigm for KGQA

methods: rethinking *how KG evidence should be organized and retrieved* rather than introducing more complex traversal strategies. Specifically, we transform the original triple-level KG into semantically coherent, size-controlled *subgraph chunks* that can be indexed and retrieved like standard text passages, while preserving essential local structure. This representation enables efficient batch retrieval for KGQA and naturally supports hybrid question answering by fusing retrieved KG evidence with text chunks. *Figure 1* illustrates how our approach shifts from multi-step, traversal-heavy interaction with the graph to a more efficient and general single-batch, coarse-grained retrieval process.

Specifically, we propose **TiC**, a chunk-based KGQA framework that enables LLMs to reason over coarse-grained subgraph chunks instead of fine-grained graph traversal. TiC consists of two complementary modules: *Knowledge-to-Chunk Transformation (KCT)* and *Planning-Guided Chunk Reasoning (PCR)*. *KCT* reorganizes the KG into semantically meaningful, uniformly sized chunks via edge-balanced partitioning, while *PCR* replaces iterative graph traversal with efficient, plan-aligned batch retrieval and verification.

We evaluate TiC on three widely used KGQA benchmarks and further validate its cross-modal compatibility using SKA-Bench (Liu et al., 2025), which combines KGs with textual data. Across all datasets, TiC achieves strong accuracy while substantially improving execution efficiency, showing that it is both effective and practical for integrating structured and unstructured knowledge.

The main contributions of this paper are as follows:

- We propose **TiC**, a simple and scalable chunk-based KGQA framework that transforms KGs into semantically coherent, size-controlled subgraph chunks.
- We introduce *PCR*, a planning-guided chunk reasoning paradigm that replaces iterative graph traversal with efficient batch retrieval and verification, substantially reducing LLM calls.
- Experiments on three KGQA benchmarks demonstrate that TiC achieves competitive accuracy with significantly improved efficiency, and further generalizes effectively to hybrid KG-text retrieval on SKA-Bench.

2 Related Works

As our method transforms KGs into subgraph chunks for KGQA, we regard the existing KGQA and graph partitioning methods as our closely related works and introduce them in this section.

2.1 Knowledge Graph Question Answering

Knowledge Graph Question Answering (KGQA) leverages structured facts for multi-hop reasoning (Berant et al., 2013). Driven by recent advancements in LLMs, current approaches have largely converged into two dominant paradigms.

(i) **Query-based KGQA.** Query-based methods translate natural-language questions into executable queries (e.g., SPARQL), achieving high precision under well-defined schemas (Luo et al., 2024). However, these methods are brittle to schema variation and entity alignment, and often require substantial engineering effort to generalize across graphs.

(ii) **Agent-based KGQA.** Agent-based methods treat the LLM as a stepwise reasoning agent that iteratively explores the KG, avoiding strict query construction and offering more flexible reasoning (Sun et al., 2024; Jiang et al., 2023; Ma et al., 2025). While effective for complex multi-hop queries, these methods rely on *stateful, multi-round* interactions, leading to high latency and cost.

Recent work retrieves dynamic subgraphs at inference time to reduce traversal cost (Li et al., 2025; Kim et al., 2023; Liu et al., 2024; Guo et al., 2024). While improving efficiency, these approaches often remain sensitive to schema design and fine-grained entity modeling, limiting robustness and cross-source fusion.

Rather than fine-grained traversing during inference, we transform KGs into semantically coherent, size-controlled *subgraph chunks* offline, simplifying the online path and yielding more stable latency/cost and improved robustness across graphs.

2.2 Graph Partitioning

Graph partitioning methods generally fall into two categories: *balanced k-way partitioning* and *community detection*, emphasizing balance efficiency and semantic cohesion, respectively.

Balanced k-way partitioning. METIS (Karypis and Kumar, 1998) efficiently produces well-balanced partitions with minimal edge cuts, but its purely structural objective often fragments semantically related regions in knowledge graphs.

Community detection. Leiden (Traag et al., 2019), building on Louvain (Blondel et al., 2008), produces cohesive and stable communities via modularity optimization, but lacks explicit control over partition balance and size uniformity.

METIS and Leiden illustrate two complementary graph partitioning philosophies: one driven by structural balance and efficiency, the other by modularity and community integrity. Our Edge-Balanced Subgraph Partitioning (ESP) strategy bridges these paradigms by preserving knowledge connectivity while achieving a balanced partition structure suitable for downstream reasoning tasks.

3 Preliminaries

Knowledge Graph (KG). A knowledge graph $\mathcal{G}$ is a structured repository that stores factual knowledge in the form of triples:

$$\mathcal{G} = \{(u, r, v) \mid u, v \in \mathcal{E},\ r \in \mathcal{R}\}, \qquad (1)$$

where $\mathcal{E}$ and $\mathcal{R}$ denote the sets of entities and relations, respectively. Each triple $\langle u, r, v\rangle$ expresses that a relation r holds between the head entity u and the tail entity v. Every entity u and relation r are typically associated with textual descriptions, denoted as TEXT_u and TEXT_r, which encode human-readable semantics and can be leveraged by language models or retrieval systems.

Subgraph Partition (Partition). Given a full KG $\mathcal{G}$, in our work, we need to divide it into smaller, semantically coherent subgraphs for efficient storage, retrieval, and reasoning. We refer to each such unit as a *subgraph partition*, or simply a *partition*. Formally, a partition is defined as:

$$\mathcal{SG} = \{(u, r, v) \mid u, v \in \mathcal{E}_{\mathcal{SG}},\ r \in \mathcal{R}_{\mathcal{SG}}\}, \qquad (2)$$

where $\mathcal{E}_{\mathcal{SG}} \subseteq \mathcal{E}$ and $\mathcal{R}_{\mathcal{SG}} \subseteq \mathcal{R}$ represent the entity and relation sets included in the partition.

A subgraph partition can be viewed as a local region within the original KG that preserves the majority of intra-partition relational edges while minimizing cross-partition links. In practice, we adopt an *edge-balanced* graph partitioning strategy that enforces balanced partition sizes while preserving coherent semantic neighborhoods and minimizing inter-partition edges.

Note that partitions are not necessarily disjoint—entities or relations can appear in multiple partitions when such overlap helps preserve semantic completeness or local connectivity.

By partitioning $\mathcal{G}$ into $\mathcal{SG}_1, \ldots, \mathcal{SG}_K$, we obtain structured subgraphs that are later serialized into subgraph chunks. These chunks support efficient batch retrieval and enable coarse-grained reasoning while preserving key structural information from the original graph.

4 Methodology

We present **TiC**, a framework for Knowledge Graph Question Answering (KGQA). Instead of fine-grained traversal over triples, TiC converts a KG into *subgraph chunks* and reasons over these chunks with guided planning and self-verification. As shown in Figure 2, TiC is composed of two main modules.

(1) Knowledge-to-Chunk Transformation (KCT). This module converts the original, link-rich KG into a set of edge-balanced, semantically coherent, and size-controlled *subgraph chunks*. Each chunk preserves all triples and allows limited, controlled overlap to keep the graph connected.

(2) Planning-Guided Chunk Reasoning (PCR). Instead of performing slow, triple-by-triple traversal, this module follows a simple plan–retrieve–verify process. A global plan guides the retrieval of related chunks in one or very few batches, after which a lightweight verification step checks whether the gathered information is sufficient or needs refinement.

Together, these modules reduce LLM calls and latency while preserving factual accuracy and source traceability.

4.1 Module 1: Knowledge-to-Chunk Transformation

Existing approaches, relying on KG traversal via strict relational links, are inflexible for this task(Sun et al., 2024; Jiang et al., 2023). We transform the native graph structure into discrete *subgraph chunks*. Each chunk is designed to be semantically coherent, size-controlled for stable retrieval, and lossless with respect to the original KG.

This transformation consists of two steps: (1) Edge-Balanced Subgraph Partitioning (ESP); and (2) Subgraph Chunk Serialization.

4.1.1 Edge-Balanced Subgraph Partitioning (ESP)

Conventional graph partitioning methods struggle to simultaneously guarantee uniform subgraph sizes and lossless edge preservation in large-scale knowledge graphs.

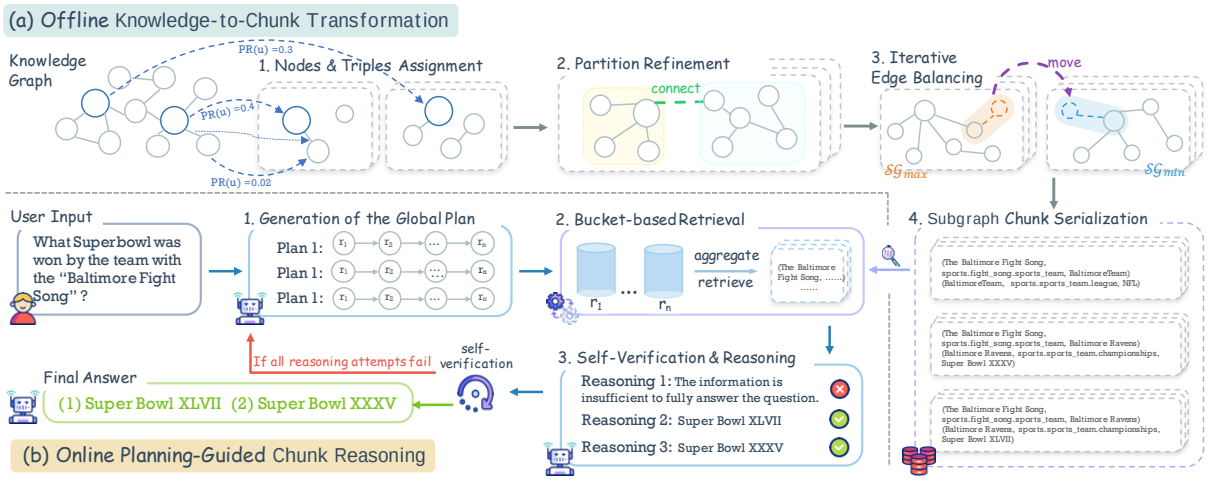

Figure 2: The architecture of TiC framework. **(Top) Offline Knowledge-to-Chunk Transformation:** Our edge-balanced partitioning algorithm converts the source KG into a corpus of semantically coherent, uniformly sized subgraph chunks. **(Bottom) Online Planning-Guided Chunk Reasoning:** Given a query, an LLM generates a global plan to guide single-batch retrieval from the chunk corpus. A self-verifying reasoning cycle then processes the retrieved context to produce the final answer.

To address this gap, we propose an edge-balanced heuristic algorithm that partitions the graph using PageRank-weighted greedy allocation (Page et al., 1999). The resulting subgraphs are edge-balanced, fully connected, and lossless, making them well suited for subsequent subgraph chunk construction.

The process involves four essential steps: (1) Node Pre-allocation; (2) Triple Assignment; (3) Subgraph Connectivity Restoration and Node Overlap Optimization; and (4) Iterative Edge Balancing Refinement.

Prior to graph partitioning, to ensure the balanced size of the resulting partitions, we define an expected partition size, denoted as $N_{\mathcal{SG}}$. This is a critical hyperparameter, and its impact on downstream task performance is systematically analyzed in our experiments (see Figure 5(a)).

Step 1: Node Pre-allocation based on Importance. We aim to evenly distribute structurally important nodes (*hubs*) across partitions by computing PageRank scores for each node $u \in \mathcal{E}$:

$$PR(u) = \frac{1-d}{N} + d \sum_{v \in \mathrm{In}(u)} \frac{PR(v)}{L(v)}, \quad (3)$$

where $PR(u)$ denotes the PageRank score of node u, d is the damping factor (set to 0.85), $N = |\mathcal{E}|$ is the number of nodes, $In(u)$ is the set of in-neighbors that link to u, and $L(v)$ is the out-degree of node v. After computing $PR(\cdot)$, we sort nodes in descending order and assign them to partitions in a round-robin manner. Given the expected subgraph size, we preallocate a sufficient number of partitions to cover the entire graph.

Step 2: Greedy Triple Assignment. Triples are assigned based on the importance of their endpoints: we sort all triples by the sum of the PageRank scores of their endpoints, and process each triple $t = (u, r, v)$ in order—if u and v belong to the same partition, we assign t to that partition; otherwise, we assign t to the least-loaded partition.

Step 3: Connectivity Restoration and Node Overlap. After the initial assignment, subgraphs may contain disconnected components. To ensure semantic integrity and facilitate cross-partition reasoning, we perform two operations.

Connectivity Restoration. For each subgraph $\mathcal{SG}$, we use the Weakly Connected Components (WCC) algorithm to identify its components. The largest component is designated as the main component:

$$\mathcal{C}_{main} = \{(u, r, v) \mid u, v \in \mathcal{E}_{\mathcal{C}_{main}}, r \in \mathcal{R}_{\mathcal{C}_{main}}\}, \quad (4)$$

where $\mathcal{E}_{\mathcal{C}_{main}} \subseteq \mathcal{E}_{\mathcal{SG}}$ and $\mathcal{R}_{\mathcal{C}_{main}} \subseteq \mathcal{R}_{\mathcal{SG}}$. For each smaller sub-component $\mathcal{C}_{sub}$, we search the global graph $\mathcal{G}$ for a "bridge edge" $e_{bridge} = (u, v)$ where $u \in \mathcal{E}_{\mathcal{C}_{main}}$ and $v \in \mathcal{E}_{\mathcal{C}_{sub}}$. To select the optimal bridge, we define a bridging value function:

$$W_{bridge}(e_{uv}) = PR(u) + PR(v), \quad (5)$$

The intuition is that an edge connecting two high-importance nodes is more likely to represent a se-

mantically significant link, making it the best candidate to merge disconnected components meaningfully. The edge maximizing W_{bridge} is added to $\mathcal{SG}$. This process iterates until $\mathcal{SG}$ becomes a single connected component or a maximum iteration count is reached.

Controlled Node Overlap. If the node overlap ratio between partitions $(\mathcal{SG}_i, \mathcal{SG}_j)$ falls below a threshold O_r, high-PageRank nodes are replicated to enforce the required overlap, subject to a maximum attempt limit.

Step 4: Iterative Edge Balancing Refinement. To further balance edge loads, we iteratively migrate a *movable* edge from the most overloaded partition $\mathcal{SG}_{max}$ to the most underloaded partition $\mathcal{SG}_{min}$. An edge is movable if (i) its removal does not increase the number of weakly connected components in $\mathcal{SG}_{max}$, and (ii) both endpoints already exist in $\mathcal{SG}_{min}$. The process repeats for a fixed number of iterations or until balance is achieved. The resulting subgraph partitions are both edge-balanced and fully connected, with all original graph information preserved.

4.1.2 Subgraph Chunk Serialization

The final step serializes each subgraph partition into text suitable for LLM processing. We convert edges into textual triples using a $(\text{TEXT}_u, \text{TEXT}_r, \text{TEXT}_v)$ template, ensuring factual fidelity and structural clarity. This completes the transformation from the original KG to subgraph chunks.

4.2 Module 2: Planning-Guided Chunk Reasoning

Having transformed the KG into a corpus of subgraph chunks, we now introduce a reasoning framework tailored for this new knowledge form. Our approach departs significantly from two existing paradigms. First, unlike traditional KGQA methods that perform triple-by-triple iterative expansion, our framework operates on coarse-grained subgraph chunks, enabling retrieval of all necessary context in a single batch. Second, while inspired by general-purpose LLM agent frameworks like ReAct(Yao et al., 2023), our model operates under a different principle. Instead of multiple, sequential interactions with a dynamic environment, TiC interacts with a static, pre-processed corpus of subgraph chunks. This fundamental difference allows us to replace costly, multi-turn LLM deliberations with a highly efficient, plan-then-batch-retrieve cycle.

This cycle, which we term Planning-Guided Chunk Reasoning, consists of three main stages: generating a global plan, retrieving subgraph chunks in a single batch, and performing self-verification to decide whether to refine the plan or generate the final answer.

4.2.1 Generation of the Global Plan

We prompt the LLM with the input question to generate a problem-solving plan. Specifically, the LLM predicts a planning path, denoted as $L = (r_1, r_2, \ldots, r_n)$ with $r_1, \ldots, r_n \in \mathcal{R}$, which allows traversal from a topic entity along this path to find relevant entities that help answer the question.

4.2.2 Bucket-based Retrieval

The quality of retrieved subgraph chunks is crucial for subsequent question answering. To maximize the recall of relevant subgraph chunks, especially for multi-hop questions, we introduce a two-stage retrieval heuristic we call "bucket-based retrieval". This strategy is designed to gather both broadly relevant context and path-specific evidence.

Based on the global plans generated in the previous step, the retrieval process unfolds in two stages. First, for each relation r in a planning path L, we treat it as an independent "bucket" and retrieve the top-K_r subgraph chunks most relevant to that relation. Second, after aggregating all chunks from all buckets, we use the complete planning paths as queries to re-rank and select the final top-K_p chunks from this aggregated pool.

4.2.3 Self-Verification and Reasoning

In this stage, we leverage the subgraph chunks retrieved in the previous step to prompt the LLM to determine whether the existing information is sufficient to answer the question. Inspired by Self-Refine (Madaan et al., 2023) and ReAct (Yao et al., 2023), we adopt multi-round iteration with self-feedback for progressive improvement. If the LLM deems the information insufficient, it is re-prompted with the current plan and reasons for failure to encourage a revised global plan and initiate new retrieval. Otherwise, it proceeds to infer an answer. We also incorporate multi-path reasoning: when multiple planning paths are predicted, a new cycle is started only if all paths lack information; otherwise, all answers are aggregated. This iterative process repeats until an answer is predicted or the iteration limit is reached. If no definitive answer emerges, the LLM is directed to infer a final answer based on its internal knowledge.

	WebQSP		CWQ		GrailQA	
	Macro-F1	Hit	Macro-F1	Hit	Macro-F1	Hit
UniKGQA (Jiang et al., 2022)	72.2	-	49.0	-	-	-
KD-CoT (Wang et al., 2023)	52.5	68.6	-	55.7	-	-
ToG(ChatGPT) (Sun et al., 2024)	-	76.2	-	58.9	-	68.7
ToG(GPT-4) (Sun et al., 2024)	-	82.6	-	67.6	64.2*	80.6*
G-Retriever (He et al., 2024)	53.4	73.4	-	-	-	-
EtD (Liu et al., 2024)	-	82.5	-	62.0	-	-
GNN-RAG (Mavromatis and Karypis, 2024)	71.3	85.7	59.4	66.8	-	-
RoG-Joint (Luo et al., 2023)	70.2	86.6	54.6	61.9	-	-
RoG-Sep (Luo et al., 2023)	66.4	82.1	53.8	60.5	-	-
KB-BINDER (Li et al., 2023)	74.4	-	-	-	58.5	53.2
SubgraphRAG (GPT4o) (Li et al., 2025)	78.2	90.9	**59.4**	67.4	-	-
SubgraphRAG (GPT4o-mini) (Li et al., 2025)	77.8	90.5	54.6	63.4	-	-
TiC (GPT4o-mini)	72.8	91.2	55.6	67.4	66.9	80.4
TiC (GPT4.1-mini)	**78.7**	**92.3**	56.8	**68.5**	**69.6**	**82.6**

Table 1: Question-answering performance on WebQSP, CWQ and GrailQA. Best results are in **bold** and second-best are underlined. Results marked with * are reproduced by us.

5 Experiments

This section evaluates our proposed TiC framework through a series of experiments. Our evaluation will be guided by the following four research questions (RQs): **RQ1**: How effective and efficient is TiC for KGQA? **RQ2**: How effective is the semantic coherence introduced by the design of subgraph chunks? **RQ3**: How effective is our proposed edge-balanced subgraph partitioning strategy? **RQ4**: How do the key hyperparameters within the TiC framework affect its reasoning performance?

5.1 Experimental Setup

Datasets and Metrics. We evaluate TiC on three public KGQA datasets: WebQuestionsSP (WebQSP) (Yih et al., 2016), ComplexWebQuestions (CWQ) (Talmor and Berant, 2018), and Strongly Generalizable Question Answering (GrailQA) (Gu et al., 2021). For GrailQA, we follow ToG (Sun et al., 2024) and test on a random subset of 1,000 instances to reduce computational cost. Following prior work, we use Hits@1 and Macro F1 as our evaluation metrics.

To validate the effectiveness of the subgraph chunks, we utilize SKA-Bench (Liu et al., 2025), a heterogeneous QA dataset containing KG, text, and tables. We exclusively use the KG-text subset, and employ accuracy as the metric.

Baselines. We compare TiC with nine KGQA baselines: UniKGQA (Jiang et al., 2022), KD-CoT (Wang et al., 2023), ToG (Sun et al., 2024), G-Retriever (He et al., 2024), EtD (Liu et al., 2024), GNN-RAG (Mavromatis and Karypis, 2024), RoG (Luo et al., 2023), SubgraphRAG (Li et al., 2025), and KB-BINDER (Li et al., 2023). We propose two implementations for RoG: RoG-Joint and RoG-Sep. The original RoG-Joint is fine-tuned on the combined training sets of WebQSP and CWQ, causing severe label leakage. As noted in SubgraphRAG, a high degree of overlap (over 50%) exists between the training and test sets of the two datasets. Following SubgraphRAG, we develop the RoG-Sep version by training the model on each dataset individually for a more rigorous evaluation.

For SKA-Bench, we implement a minimalist retrieval-based baseline using LangChain. The system follows a two-stage retrieval-generation pipeline, where each triple is treated as an independent chunk and serialized following the $(\text{TEXT}_u, \text{TEXT}_r, \text{TEXT}_v)$ template. We employ `gte-Qwen2-1.5B` as the retriever and use `GPT-4o-mini` as the generator.

Implementation. TiC is a training-free, pluggable framework, where components can be upgraded as needed. In the main experiment, we employ gte-Qwen2-1.5B as the retriever and conduct explorations on both GPT-4o-mini and GPT-4.1-mini. We set the subgraph partition size to 7, the multi-path reasoning width to 5, and the temperature to 0.7. Under this configuration, the offline subgraph chunk indexing process requires approxi-

mately 1.5 hours and yields an index file of 12GB.

	WebQSP	CWQ	GrailQA
ToG	11.2	14.3	10.4
TiC	**8.4**	**8.1**	**7.5**
Δ Impr.	↓25.0%	↓43.4%	↓27.9%

Table 2: Average number of LLM calls per question.

5.2 Performance and Efficiency on KGQA (RQ1)

5.2.1 Main Performance Comparison

Table 1 presents TiC's performance on three public datasets. To evaluate adaptability across LLM backbones, we test TiC with GPT-4o-mini and GPT-4.1-mini. TiC achieves SOTA performance on four of six metrics and ranks second on the remaining two. Compared with SubgraphRAG using the same GPT-4o-mini backbone, TiC consistently achieves superior results. Upgrading to GPT-4.1-mini further improves performance, showing that TiC does not constrain the LLM with rigid templates and can scale with stronger backbones.

5.2.2 Efficiency Comparison

In contrast to hop-by-hop methods like ToG, TiC's plan-and-retrieve architecture drastically reduces LLM calls (Table 2). A basic reasoning cycle requires just 2 calls, while our full multi-path reasoning configuration for maximum accuracy uses only 6. This represents a significant improvement in efficiency and cost over iterative approaches.

5.3 Advantages of Semantic Coherence Brought by the Chunk-Based Design (RQ2)

The generation strategy designed for subgraph chunks introduces local semantic coupling constraints, ensuring that each subgraph chunk is semantically coherent. To verify the advantage of semantic coherence within chunks, we conduct a series of experiments on the SKA-Bench dataset. For a fair comparison, we control both the input text chunks and the number of factual statements provided to the LLMs. Given that each subgraph chunk contains 7 facts, we compare the performance differences between subgraph chunks and discrete triples when the number of facts is set to 7, 14, and 21, respectively.

As shown in Figure 4, subgraph chunks with higher semantic coherence consistently outperform discrete triples. This suggests that LLMs reason more effectively over tightly connected and semantically coherent text segments, while disordered inputs may cause contextual confusion. These results further demonstrate the advantage of the chunk-based design in promoting coherent reasoning.

	Edge Pres.	Size CV	Recall	F1	Hit
METIS	9.9%	**0.25**	85.4	47.9	60.7
Leiden	32.2%	1.34	90.1	48.5	62.7
Our	**99.9%**	<u>0.38</u>	**90.7**	**55.6**	**67.4**

Table 3: Performance of graph partitioning methods, evaluating both partitioning quality and KGQA performance on the CWQ dataset.

5.4 Effectiveness of the Edge-Balanced Subgraph Partitioning Strategy (RQ3)

We compare the Edge-Balanced Subgraph Partitioning (ESP) strategy with METIS (Karypis and Kumar, 1998) and Leiden (Traag et al., 2019) in terms of knowledge preservation (retained-edge ratio), partition balance (CV of subgraph sizes), and downstream KGQA performance on CWQ.

As shown in Table 3, ESP retains 99.9% of edges, while METIS and Leiden incur substantial edge loss. METIS achieves the lowest CV (0.25), and ESP remains competitive (0.38), providing a better preservation-balance trade-off than the baselines.

ESP also yields the best downstream KGQA results on CWQ, indicating that improved preservation with competitive balance translates to stronger question answering performance.

5.5 Hyperparameter Sensitivity (RQ4)

In this section, we evaluate the impact of subgraph chunk size on KGQA performance, investigating whether coarse-grained subgraphs are more effectively utilized by LLMs for reasoning than fine-grained triples. All reported results are averaged over multiple runs to ensure statistical reliability.

As shown in Figure 5(a), partition size has a significant impact on performance. KGQA performance improves with larger partitions but begins to saturate when the size reaches 7. This suggests an upper bound on the benefit of coarse-grained chunks, likely due to LLM limitations in long-context reasoning. Furthermore, all coarse-grained results outperform the fine-grained, triple-level baseline. This indicates that subgraphs provide richer context, thus facilitating more effective reasoning and question answering by the LLM.

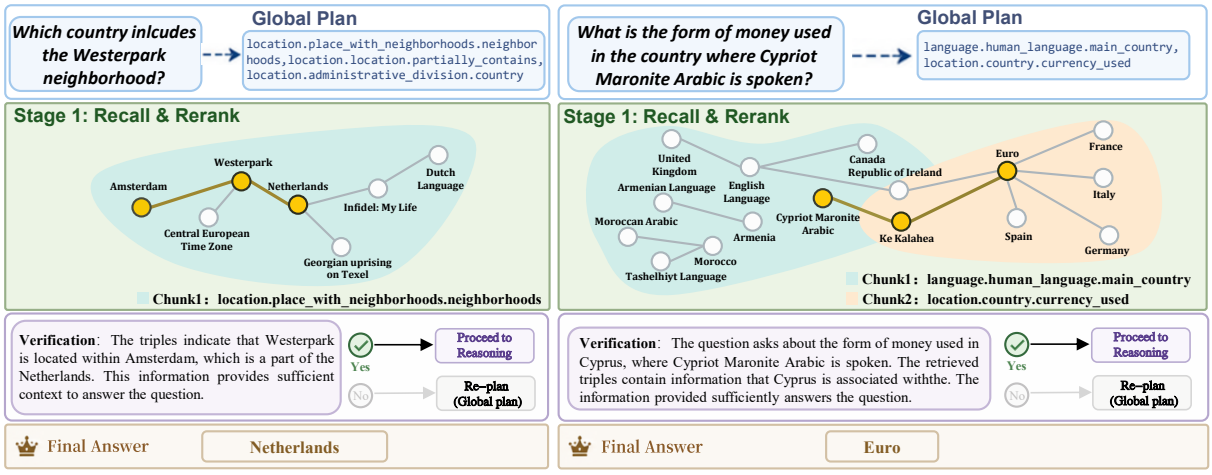

Figure 3: Complete TiC reasoning pipeline on CWQ: generating global plans, executing bucket-based retrieval and path-based reranking, followed by self-verifying reasoning to deduce the final answer.

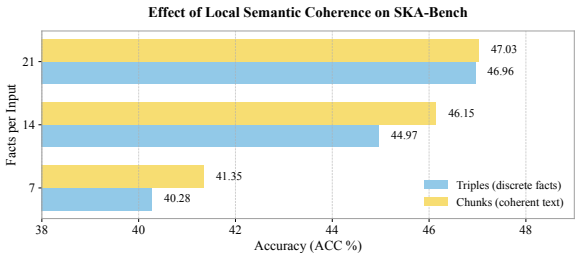

Figure 4: Accuracy comparison between chunk-based and triple-based representations on SKA-Bench.

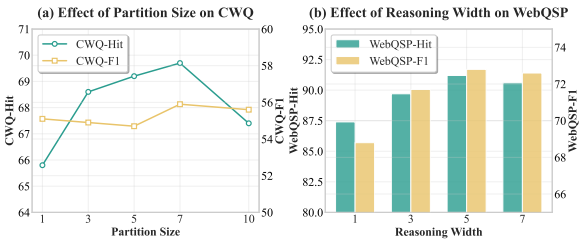

Figure 5: Effects of partition size on CWQ (a) and multi-path reasoning width on WebQSP (b).

In Figure 5(b), we investigate the effect of multi-path reasoning width on performance. Wider paths improve results, suggesting that the diversity of reasoning paths benefits LLM question answering. However, the gain exhibits diminishing returns and performance saturates beyond a certain width.

5.6 Further Analysis

Case Study. Figure 3 presents the representative reasoning pipeline. For a multi-hop question from the CWQ, the LLM first predicts the global plan. Instead of iterative, step-by-step searches, TiC retrieves the corresponding subgraph chunks and gathers multi-hop evidence in a single step. The retrieved subgraph chunks enable self-verification-based reasoning, illustrating the efficiency and effectiveness of the framework.

Comparison of Serialization Methods. In Appendix A, we examine different serialization methods. The results show that the LLM maintains strong comprehension across various expressions, and our simplest $(\text{TEXT}_u, \text{TEXT}_r, \text{TEXT}_v)$ template exhibits considerable competitiveness.

Adaptability to Heterogeneous Knowledge. In form, subgraph chunks are similar to document chunks, allowing them to be indexed and retrieved by standard dense retrievers and to serve as a unified evidence interface for hybrid question answering with KG and text. The quantitative results in Appendix B indicate that the chunk-based approach improves KGQA performance by integrating heterogeneous knowledge in a unified manner.

Complexity Analysis. The time complexity of offline Edge-Balanced Subgraph Partitioning is analyzed in Appendix C.

6 Conclusion

This paper introduces TiC, a novel *chunk-based KGQA* framework. By enabling LLMs to reason "chunk by chunk" over a knowledge graph, TiC achieves state-of-the-art performance and improves inference efficiency. Future work includes developing dynamic chunking mechanisms for evolving KGs, extending the framework to other structured data sources, and designing more expressive chunk-to-text serialization techniques.

Limitations

Our current work primarily focuses on English knowledge graphs. Extending TiC to cross-lingual and multimodal knowledge graphs is an interesting direction for future work, and would require revisiting the design of serialization, retrieval, and verification mechanisms under heterogeneous settings.

References

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond iid: three levels of generalization for question answering on knowledge bases. In *Proceedings of the web conference 2021*, pages 3477–3488.

Tiezheng Guo, Qingwen Yang, Chen Wang, Yanyi Liu, Pan Li, Jiawei Tang, Dapeng Li, and Yingyou Wen. 2024. Knowledgenavigator: Leveraging large language models for enhanced reasoning over knowledge graph. *Complex & Intelligent Systems*, 10(5):7063–7076.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems*, 37:132876–132907.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251, Singapore. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. *arXiv preprint arXiv:2212.00959*.

George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392.

Jiho Kim, Yeonsu Kwon, Yohan Jo, and Edward Choi. 2023. KG-GPT: A general framework for reasoning on knowledge graphs using large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9410–9421, Singapore. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Mufei Li, Siqi Miao, and Pan Li. 2025. Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation. In *The Thirteenth International Conference on Learning Representations*.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6966–6980, Toronto, Canada. Association for Computational Linguistics.

Ke Liang, Lingyuan Meng, Meng Liu, Yue Liu, Wenxuan Tu, Siwei Wang, Sihang Zhou, Xinwang Liu, Fuchun Sun, and Kunlun He. 2024. A survey of knowledge graph reasoning on graph types: Static, dynamic, and multi-modal. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9456–9478.

Guangyi Liu, Yongqi Zhang, Yong Li, and Quanming Yao. 2024. Dual reasoning: A gnn-llm collaborative framework for knowledge graph question answering. *arXiv preprint arXiv:2406.01145*.

Zhiqiang Liu, Enpei Niu, Yin Hua, Mengshu Sun, Lei Liang, Huajun Chen, and Wen Zhang. 2025. Ska-bench: A fine-grained benchmark for evaluating structured knowledge understanding of llms. *Preprint*, arXiv:2507.17178.

Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, Yifan Zhu, and Anh Tuan Luu. 2024. ChatKBQA: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2039–2056, Bangkok, Thailand. Association for Computational Linguistics.

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*.

Jie Ma, Zhitao Gao, Qi Chai, Wangchun Sun, Pinghui Wang, Hongbin Pei, Jing Tao, Lingyun Song, Jun Liu, Chen Zhang, and 1 others. 2025. Debate on graph: a

flexible and reliable reasoning framework for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24768–24776.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

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

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab. Previous number = SIDL-WP-1999-0120.

Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval augmentation reduces hallucination in conversation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3784–3803, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.

Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From louvain to leiden: guaranteeing well-connected communities. *Scientific reports*, 9(1):1–12.

Keheng Wang, Feiyu Duan, Sirui Wang, Peiguang Li, Yunsen Xian, Chuantao Yin, Wenge Rong, and Zhang Xiong. 2023. Knowledge-driven cot: Exploring faithful reasoning in llms for knowledge-intensive question answering. *arXiv preprint arXiv:2308.13259*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206, Berlin, Germany. Association for Computational Linguistics.

A Comparison of Serialization Methods

Previous studies categorize graph-to-text serialization methods into three main types: (1) Template-based, which fills each triple into a semantically coherent template; (2) Path serialization, which randomly samples paths from the graph and inserts them into templates, though this approach may lead to factual omissions or redundancies. (3) Adjacency-list-style code structures, which explicitly preserve the topological structure of the graph, as illustrated in table 4.

We evaluate the performance differences among these three serialization methods and our $(\text{TEXT}_u, \text{TEXT}_r, \text{TEXT}_v)$ template on the SKA-Bench dataset. Consistent with the baseline setup, we construct a simple RAG workflow using LangChain. In addition, we design two baseline variants to examine different retrieval strategies: (1) RAGjoint, which retrieves the top-K chunks from a combined pool of textual and KG data; and (2) RAGsep, which independently retrieves top-$K/2$ chunks from the text and KG corpora, respectively, before merging them for LLM inference.

As shown in Table 5, these serialization methods exhibit noticeable performance variations. Nevertheless, our simplest template achieves a comparably competitive level of performance. This observation indicates that, although LLMs vary in their ability to interpret different representations, their overall understanding remains largely consistent.

B Adaptability to Heterogeneous Knowledge

We continue to use the SKA-Bench dataset, which integrates both textual and knowledge graph (KG) data, and employ a RAG workflow built upon LangChain to quantify the performance of subgraph chunks.

In practice, each triple in the KG is typically serialized into an individual text chunk, and we adopt this procedure as a comparative baseline. To ensure a fair validation of the subgraph chunks generated by TiC, we exclusively replace the baseline's triple-based chunks with those produced by Section 4.1, while all other settings are kept identical to the

Type	Example
Template-based	`Entity_A is RELATION of Entity_B`
Path-based	`Entity_A is RELATION_1 of Entity_B, and Entity_B is RELATION_2 of Entity_C`

Adjacency-list	`Entity(A):`
	`  - RELATION_1 -> Entity_B`
	`Entity(B):`
	`  - RELATION_2 -> Entity_C`

Table 4: Examples of the three serialization template types.

Serialization Method	$\text{RAG}_{\text{joint}}$ (ACC%)	RAG_{sep} (ACC%)
Template-based	42.19	<u>46.26</u>
Path-based	**44.52**	**47.05**
Adjacency-list	43.23	45.77
Our	<u>43.59</u>	46.15

Table 5: Performance comparison of different serialization methods on SKA-Bench.

baseline. As shown in Table 6, our method consistently outperforms both RAG baseline variants across all evaluation metrics. Notably, in the sep variant, it demonstrates an improvement of over 30% compared to the baseline. These results provide compelling evidence that our approach can be effectively and seamlessly integrated into mainstream RAG systems.

	$\text{RAG}_{\text{joint}}$	$\text{Our}_{\text{joint}}$	RAG_{sep}	Our_{sep}
ACC (%)	39.45	**43.59**	36.41	**46.15**

Table 6: Accuracy (ACC) on a subset of SKA-Bench, a hybrid KG-Text QA benchmark.

C Detailed Analysis of Time Complexity

Although the Edge-Balanced Subgraph Partitioning (ESP) module operates as an offline component within the TiC framework and is therefore not subject to stringent real-time constraints, it is nonetheless necessary to analyze its time complexity. This allows us to evaluate the potential challenges it may encounter when applied to ultra-large-scale graphs. We denote the number of nodes as V, edges as E, partitions as K, PageRank iterations as I_{PR}, and balancing iterations as I_{balance}. The analysis proceeds through the four sequential steps of the algorithm.

C.1 Step 1: Node Pre-allocation based on Importance

The complexity of this step is dominated by two primary operations: the PageRank computation, which requires $O(I_{\text{PR}} \cdot E)$, and the subsequent sorting of all V nodes based on their scores, which costs $O(V \log V)$. The total complexity is therefore $O(I_{\text{PR}} \cdot E + V \log V)$.

C.2 Step 2: Greedy Triple Assignment

This step begins with sorting all E edges, incurring a cost of $O(E \log E)$. To assign each edge to the least-loaded partition, we employ a min-heap data structure to track partition sizes. This standard optimization allows for querying the minimum-load partition in $O(1)$ and updating it in $O(\log K)$, leading to a total assignment cost of $O(E \log K)$. This is a significant improvement over a naive linear scan, which would cost $O(E \cdot K)$. Thus, the step's overall complexity is $O(E(\log E + \log K))$.

C.3 Step 3: Connectivity Restoration and Node Overlap

This phase can present a significant computational bottleneck. Since the node overlap rate is set to 0 in our main experiments, the analysis for this stage focuses exclusively on the time complexity of the connectivity restoration. Initially, verifying the connectivity of all K partitions requires a total of $O(V + E)$ time using BFS or DFS. If a partition is found to be disconnected, the algorithm must introduce "bridge edges" to merge its disparate components. The complexity arises from the search for these edges.

The **worst-case scenario** occurs when a single partition, containing V_p nodes, is maximally fragmented into V_p single-node components. To merge these components into a single connected graph, the algorithm needs to find and add $V_p - 1$ suitable bridge edges. Our current greedy strategy involves iteratively finding one bridge edge at a time. For each of these $V_p - 1$ iterations, the algorithm may need to scan the entire global edge set E to find an edge that connects two previously disconnected components within the partition. This results in a cost of $O(V_p \cdot E)$ for restoring a single partition. In the most extreme case where one partition encompasses nearly all nodes of the graph ($V_p \approx V$), the complexity for this step becomes $O(V \cdot E)$. While searching the global edge set repeatedly is computationally expensive, this design

choice is a trade-off for implementation simplicity in our greedy, iterative restoration approach. More sophisticated methods, such as pre-indexing potential bridge edges, could reduce this complexity but would introduce significant overhead in memory and pre-computation. Thus, this analysis highlights a critical trade-off between runtime performance and algorithmic simplicity.

C.4 Step 4: Iterative Edge Balancing Refinement

The balancing phase is the second major computational bottleneck. For each of the I_{balance} rounds, the algorithm identifies the largest partition, $\mathcal{SG}_{max}$, and searches for a non-bridge edge that can be moved to a smaller partition. Verifying if a single edge is a bridge in $\mathcal{SG}_{max}$ requires a graph traversal (e.g., BFS/DFS), which costs $O(V_{max} + E_{max})$.

The worst case per round arises when the algorithm must test a large number of edges—or potentially all E_{max} edges in the partition—before finding a movable, non-bridge edge (or determining that none exist). This leads to a per-round complexity of $O(E_{max} \cdot (V_{max} + E_{max}))$. Assuming relatively balanced partitions ($E_{max} \approx E/K$) and dense graphs where $E \gg V$, the total complexity for this step simplifies to $O(I_{\text{balance}} \cdot (E/K)^2)$. However, it is important to note that this represents a pessimistic upper bound; in practice, a movable edge is often found after checking only a small fraction of candidates, making the average-case performance significantly better.

This analysis reveals a clear trade-off. While the initial allocation steps are highly efficient, the sophisticated mechanisms for ensuring connectivity and balance introduce significant computational costs, particularly in their worst-case scenarios.

D Prompt Examples in TiC Framework

This section provides prompt examples used in TiC during runtime, including global planning and self-verification with answer reasoning.

D.1 Prompt for Global Planning

```
You are the reasoning module for a KBQA system. Your task is to analyze the following question and its query entities to develop an initial reasoning plan. Your plan should outline your approach to answer the question, specifying what entities, relations, and paths need to be retrieved.

Question: "{question}"
Query Entities: {q_entities}

Please output a JSON object with the following fields:
{
  "plan": "A detailed explanation of your approach.",
  "entities": ["List of entities to retrieve"],
  "relations": ["List of relations to retrieve"],
  "paths": ["List of retrieval paths"],
  "notes": "Any additional clarifications or assumptions."
}
If any field is not applicable, return an empty list for that field.
```

Listing 1: Global planning prompt used in TiC.

Retrieval with relation-path guidance. TiC further leverages the relation paths generated by the LLM to guide knowledge retrieval in a two-stage, bucket-aware manner. In the first stage, each relation path is decomposed into individual relations or sub-path segments. Each segment is used to query the knowledge base independently, retrieving a set of candidate subgraph chunks associated with the corresponding relation bucket based on semantic similarity. This step ensures that relevant evidence linked to different relations along the reasoning chain can be broadly covered.

In the second stage, TiC re-evaluates the retrieved candidates by computing semantic similarity between the *complete* relation path and the subgraph chunks aggregated from all relation buckets. By jointly considering the global relational semantics of the entire path, TiC ranks and selects the most relevant subgraph chunks as final evidence. This bucket-based two-stage retrieval mechanism enables TiC to retrieve more comprehensive and coherent subgraph chunks, which are then fed into the subsequent verification and answer reasoning stage.

D.2 Prompt for Self-Verification and Answer Reasoning

```
# Input Context
**Question**: "{question}"
**Retrieved Triples**: {retrieved_triples}

# Reasoning
Based on the question, the provided list of possibly relevant triples, and all your internal knowledge, analyze step by step how to infer the answer. Please follow the steps below:

```

```
1. **Think Phase**
In this phase, analyze step by step how to deduce the answer from all the provided relevant triples. Write your detailed reasoning process in the code block below, ensuring that each step of your thought process is clearly expressed:
```Think
# Write your detailed step-by-step reasoning here, outlining how you deduced the answer
```

2. **Determine if There is Enough Information**
After the Think Phase, evaluate whether the provided triples contain sufficient and relevant information to answer the question. If they do, ensure that your final answer strictly reflects the entities from the triples without introducing new entities or modifying the given information.

3. **Output if You Can Answer**
If you determine that you can answer the question, output "Yes" in the code block below; otherwise, output "No":
```CanAnswer
Yes/No
```

4. **Output the Final Answer**
If you can answer the question, output all possible brief answers in the next code block (the answers must be strictly faithful to the reasoning path, and reflect only the entities found in the retrieved triples if possible, or reasonable extensions from your knowledge if necessary, to avoid final matching failure). If you cannot answer, output "None":
```FinalAnswer(s)
All possible brief answers/None
```
```

Listing 2: Self-verification and answer reasoning prompt used in TiC.