SAFE: Schema-Driven Approximate Distance Join for Efficient Knowledge Graph Querying

Sangoh Lee **Sungho Park** **Wook-Shin Han***
Graduate School of Artificial Intelligence (GSAI)
Pohang University of Science and Technology (POSTECH)
Pohang, South Korea
{solee, shpark, wshan}@dblab.postech.ac.kr

Abstract

To reduce hallucinations in large language models (LLMs), researchers are increasingly investigating reasoning methods that integrate LLMs with external knowledge graphs (KGs). Existing approaches either map an LLM-generated query graph onto the KG or let the LLM traverse the entire graph; the former is fragile because noisy query graphs derail retrieval, whereas the latter is inefficient due to entity-level reasoning over large graphs. In order to tackle these problems, we propose **SAFE** (*Schema-Driven Approximate Distance Join For Efficient Knowledge Graph Querying*), a framework that leverages schema graphs for robust query graph generation and efficient KG retrieval. SAFE introduces two key ideas: (1) an *Approximate Distance Join (ADJ)* algorithm that refines LLM-generated pseudo query graphs by flexibly aligning them with the KG's structure; and (2) exploiting a compact schema graph to perform ADJ efficiently, reducing overhead and improving retrieval accuracy. Extensive experiments on WebQSP, CWQ and GrailQA demonstrate that SAFE outperforms state-of-the-art methods in both accuracy and efficiency, providing a robust and scalable solution to overcome the inherent limitations of LLM-based knowledge retrieval.

1 Introduction

Recent large language models (LLMs) have achieved state-of-the-art results in a wide range of NLP and data-science tasks (Wang et al., 2024; Li et al., 2024a; Zhao et al., 2024; Zhang et al., 2023). Since they are pre-trained on massive corpora, LLMs can produce fluent and context-aware responses. Nevertheless, their hallucinations, stale world knowledge, and opaque decision processes continue to erode user trust.

Recent efforts mitigate hallucinations, stale knowledge, and the opaque reasoning of LLMs by enriching them with external knowledge graphs (KGs) (Chen et al., 2024b; Sun et al., 2024; Xu et al., 2024). Because a KG explicitly encodes entities and relations, supplying an LLM with a small, query-relevant subgraph can ground its answers in verifiable facts—but uncovering that subgraph is non-trivial. The pipeline must (i) synthesize a query graph that precisely captures the user's intent and (ii) fetch matching subgraphs from a billion-scale KG quickly enough to keep latency low. We cast this retrieval step as ranked subgraph search: the engine maintains running top-k subgraphs. It prunes any (partial or complete) subgraph whose score cannot beat the current frontier, thus skipping the enumeration of less-promising subgraphs. Robust query-graph generation paired with such aggressively pruned top-k search is therefore critical to unlock the full potential of KG-augmented LLMs, yet existing solutions attempting to meet this goal typically fall into three paradigms—semantic parsing, LLM agents, and retrieval-augmented generation—each with significant limitations.

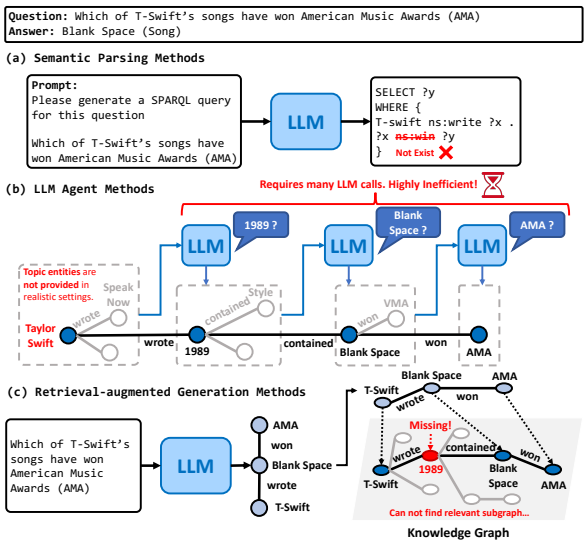


Figure 1: Limitations of Previous Methods

Semantic Parsing Methods. Figure 1(a) illustrates key limitations of semantic parsing meth-

*Corresponding author

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 17464–17489
November 4-9, 2025

ods. These methods convert natural language queries into structured query languages such as Cypher (Francis et al., 2018) or SPARQL (Harris and Seaborne, 2013), primarily using LLMs as translators (Yu et al., 2023; Luo et al., 2024a,b). However, these methods often require specialized training on specific workloads, resulting in poor generalization to unseen query templates (Gu et al., 2021) and execution failures when the generated queries are incorrect (Sun et al., 2024).

LLM Agent Methods. Figure 1(b) highlights key limitations of another line of work, which leverage LLMs to traverse KGs based on natural language queries. While intuitive, these methods unrealistically assume that the correct topic entity is known beforehand—a condition rarely satisfied in large-scale KGs where many entities may seem relevant (Cai et al., 2024). Moreover, traversal-based querying requires frequent LLM calls, making it highly inefficient. For example, the state-of-the-art PoG (Chen et al., 2024b) method calls 13.3 closed-source LLM on average and takes over 34 seconds per query on the CWQ (Talmor and Berant, 2018).

Retrieval-augmented Generation Methods. Figure 1(c) illustrates the core limitations of SimGRAG (Cai et al., 2024), a recent method in retrieval-augmented generation (RAG) (Cai et al., 2024; He et al., 2024; Baek et al., 2023; Kim et al., 2023a; Liu et al., 2024). SimGRAG lowers LLM overhead by first letting the LLM synthesize a query graph, then semantically mapping each query node to candidate KG nodes, and finally running an exact subgraph-isomorphism search over those candidates. While conceptually efficient, it relies on the unrealistic assumption that the LLM-generated query graph will structurally align with an actual subgraph in the KG. In practice, an LLM has no explicit, up-to-date knowledge of the KG's schema, so the query graphs it invents frequently include nonexistent relations or miswired node types; these schema mismatches lead to large retrieval errors—particularly in large, heterogeneous KGs—and sharply reduce end-to-end accuracy.

To address these challenges, we propose SAFE, a schema-aware pipeline for KGQA that leverages schema graphs for robust query graph generation and effective ranked semantic subgraph matching. SAFE introduces two key ideas.

(1) Approximate Distance Join (ADJ) We introduce an *Approximate Distance Join (ADJ)* algorithm that relaxes edge matching by allowing any pair of candidate KG vertices to stand in for a query edge (u, v) as long as the shortest path between them is no longer than a user-specified threshold δ. Because an edge in the query graph can thus be realized by a shortest path in the KG, ADJ still succeeds even when the query graph is missing—or has inserted—intermediate nodes, *i.e.*, when its topology is partially wrong. After this distance-aware alignment, *ranked subgraph matching* based on ADJ is executed: it scores partial matches incrementally, maintains a running top-k list, and prunes any subgraph whose score cannot enter that list, yielding fast, high-quality evidence for KG-augmented LLMs.

(2) Exploiting Schema Graph Executing an ADJ seemingly demands either (i) storing all–pairs shortest-path (APSP) distances for the entire KG or (ii) issuing expensive per-query BFSs to find vertices within a distance threshold. Both options are prohibitive in memory and latency. Our key insight is that ADJ only needs distance information *between vertex* types, not between every individual entity. Hence, we precompute APSP *only once on the schema graph*—a compact abstraction whose nodes are types (classes) and whose edges summarise admissible relations. Because the schema graph is several orders of magnitude smaller than the KG, its APSP fits easily in memory and can be consulted at query time with negligible cost. Mapping candidate entities to their types and using this lightweight distance oracle allows ADJ to enforce the path-length threshold without traversing the full KG, yielding dramatic savings in both storage and online latency while tightening type constraints for subsequent ranked subgraph search. In summary, our contributions are as follows:

- We propose the ADJ that converts an LLM-generated (and possibly noisy) query graph into an efficient, ranked semantic subgraph matching task: each query edge is satisfied by any pair of candidate KG vertices whose shortest-path length is no greater than a threshold δ, and a *ranked subgraph search* prunes partial matches that cannot break into the current top-k list.
- We are the first to exploit a *schema graph*—a compact, type-level summary of the KG—as the distance oracle for ADJ. Precomputing APSP on relatively small graph is enough to enforce all distance constraints, slashing memory use and online latency while refining entity-type filters for downstream retrieval.
- Extensive experiments on three KGQA bench-

marks, CWQ, WebQSP, and GrailQA, demonstrate that our schema-aware ADJ yields both higher answer accuracy and orders-of-magnitude speed-ups compared with state-of-the-art semantic-parsing, agent-based, and RAG baselines.

2 Related Work

Large Language Models (LLMs). Recent advances in Large Language Models (LLMs) have shown notable progress in complex reasoning tasks through explicit reasoning methods like Chain-of-Thought (CoT)(Wei et al., 2022; Kojima et al., 2022; Zhou et al., 2023) and its variants such as Tree-of-Thought(Yao et al., 2024), Graph-of-Thought (Besta et al., 2024), and Skeleton-of-Thought (Ning et al., 2024). Efforts have also enhanced LLM capabilities in interpreting graph structures (Tang et al., 2023; Tan et al., 2024; Chen et al., 2024a, 2022). However, hallucinations, outdated knowledge, and opaque reasoning still limit their reliability, motivating integration with structured external knowledge such as KGs.

Knowledge Graph-Augmented LLMs. Integrating KGs with LLMs has been widely explored to address LLM limitations through approaches such as semantic parsing, LLM-agent traversal, and retrieval-augmented generation.

Semantic parsing methods convert natural language questions into structured queries executed on KGs (Yu et al., 2023; Li et al., 2023; Luo et al., 2024a). Despite simplicity, these methods often require task-specific training, leading to poor generalization and query execution failures.

LLM agent methods utilize LLMs to directly navigate KGs based on natural language queries (Sun et al., 2024; Chen et al., 2024b; Xu et al., 2024). ToG (Sun et al., 2024) proposed LLM-based beam search traversal, while PoG (Chen et al., 2024b) enhanced it with memory modules for effective traversal. GoG (Xu et al., 2024) integrated LLM parametric knowledge with KG traversal. However, these methods assume accurate initial entities and involve extensive LLM calls, reducing efficiency.

Retrieval-augmented generation methods enhance LLM responses by retrieving KG information. Embedding-based methods (He et al., 2024; Baek et al., 2023) struggle with complex reasoning tasks (Cai et al., 2024). Methods such as KELP (Liu et al., 2024) and KG-GPT (Kim et al., 2023a) generate relevant KG paths but face scalability issues (Cai et al., 2024). SimGRAG (Cai et al., 2024) retrieves structurally similar subgraphs but is limited by its rigid structural alignment assumptions.

SAFE addresses these challenges through schema graph based query graph generation and type-level semantic matching, significantly improving retrieval accuracy and robustness.

3 Preliminary

Knowledge Graph (KG) represents factual knowledge as a structured collection of triples: $\mathcal{G} = \{(e, r, e') \mid e, e' \in E, r \in R\}$, where E is the set of entities and R is the set of relations.

Schema Graph (SG) defines the structural schema of a KG, specifying node types and their relational connections. Formally, it is represented as: $\mathcal{S} = \{(c, r, c') \mid c, c' \in C, r \in R\}$, where C denotes the set of entity types derived from KG G, and R is the same set of relations as in G.

Query Graph represents the structural form of a query using a set of triple patterns: $Q = \{(e_q, r_q, e'_q) \mid e_q, e'_q \in E \cup V_E, r_q \in R \cup V_R\}$, where V_E and V_R are sets of variables disjoint from E and R. We assume the query graph is connected; otherwise, we apply our algorithm independently per connected component.

Knowledge Graph Question Answering addresses answering natural language questions using factual information from a KG. Given a question q and a KG $\mathcal{G}$, the goal is to retrieve the correct answer set A_q.

Node distance. Let $G = (V, E)$ denote a (directed) graph. The node distance $d_G(u, v)$ is the length (number of edges) of a shortest path from u to v; if no path exists, $d_G(u, v) = \infty$ (Zou et al., 2009). We use $d_{\mathcal{S}}$ for the schema graph and $d_{\mathcal{G}}$ for the KG.

All-pairs shortest paths (APSP). For $G = (V, E)$ with non-negative edge weights, APSP computes a matrix $D \in \mathbb{R}^{|V| \times |V|}$ where $D[u, v]$ equals the length of a shortest $u \to v$ path. We precompute APSP once on the compact schema graph via Floyd–Warshall (Cormen et al., 2022).

4 Design of SAFE

In this section, we introduce the technical details of the SAFE, a novel schema-aware query graph generation and semantically enhanced subgraph matching framework. As shown in Figure 2, SAFE consists of four distinct modules: pseudo query graph generation, approximate distance join (ADJ), ranked semantic subgraph matching and answer generation with retrieved subgraphs.

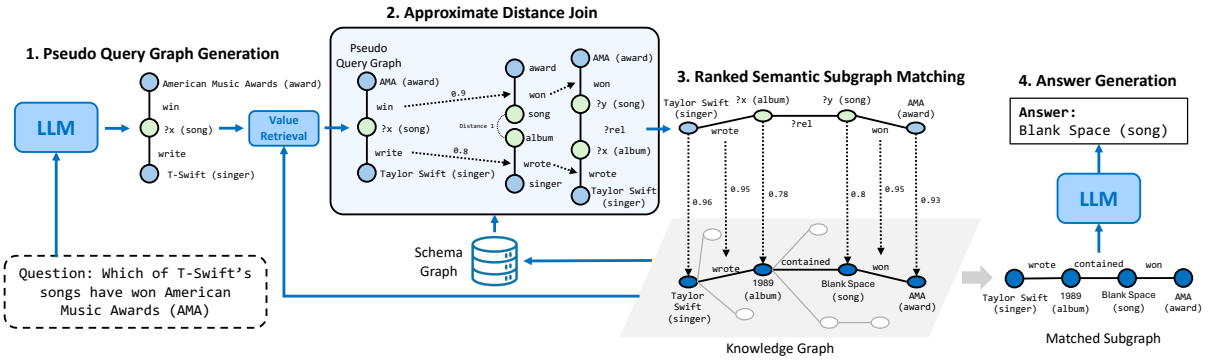


Figure 2: The framework overview of SAFE.

4.1 Pseudo Query Graph Generation

Given a natural-language question q, we first utilize a large language model (LLM)'s reasoning ability to generate a *pseudo query graph*. Formally, the pseudo query graph is defined as $Q_{\text{pseudo}} = \{(u_1, r_1, u_1'), \ldots, (u_m, r_m, u_m')\}$, where each node u_i either represents a variable in V_E or an explicitly mentioned entity that may not exist in the entity set E. For each node u in the pseudo query graph, the LLM additionally predicts its expected type (class), denoted by $\texttt{type}(u)$, which is later used to identify similar schema edges. Since we do not assume prior knowledge of the correct topic entity, the LLM might reference entities that do not exist in the KG. In such cases, we perform a value retrieval step to find the most similar matching entity within the KG based on embedding similarity. This retrieved entity serves as a known constant in subsequent steps, while unresolved entities remain as variables.

4.2 Approximate Distance Join (ADJ)

Approximate distance join (ADJ) refines the pseudo query graph Q_{pseudo} by aligning it with the schema of the KG in two steps: (1) *Candidate Schema Edge Selection* selects candidate schema edges semantically similar to the pseudo edges from Q_{pseudo}, and (2) *Distance Edge Join* connects these selected candidate schema edges under distance constraints to ensure graph connectivity and structural coherence.

Candidate Schema Edge Selection. The pseudo query graph Q_{pseudo} represents high-level relationships inferred from the natural language question, but may not precisely align with the underlying KG schema. To bridge this gap, we perform a schema-level alignment as follows. For each pseudo edge $\ell = (u_i, r_i, u_i') \in Q_{\text{pseudo}}$, we first construct its corresponding type-level representation $\texttt{type}(\ell) = (\texttt{type}(u_i), r_i, \texttt{type}(u_i'))$, which has been predicted during the query graph generation process. We then embed $\texttt{type}(\ell)$ into a high-dimensional vector space and measure its similarity to pre-embedded schema edges $(c, r, c') \in \mathcal{S}$ using the L2-distance metric. If ℓ explicitly references a specific entity, we fix the entity's actual type and associated relation, improving the quality of candidate retrieval. Finally, for each pseudo edge ℓ, we select the top-k_{schema} schema edges $(\alpha_1, \ldots, \alpha_{k_{\text{schema}}})$, ranked by their respective L2-distances $(d_1, \ldots, d_{k_{\text{schema}}})$, as the candidate set $\text{Cands}(\ell)$. This procedure is repeated for every edge in Q_{pseudo}.

Distance Edge Join. To maintain connectivity among candidate schema edges, we connect edges within a specified distance threshold. By allowing flexible distances when joining edges, we mitigate disconnections and effectively preserve the logical structure of the original pseudo query graph generated by the LLM.

While the original distance join (Zou et al., 2009) primarily considers node distances, we extend this notion to measure edge distances. Specifically, for two schema graph edges $\alpha = (c_1, r_1, c_1')$ and $\beta = (c_2, r_2, c_2')$, we define their distance as follows:

$$\mathcal{D}_{\text{edge}}(\alpha, \beta) := \min_{x_1 \in \{c_1, c_1'\}, x_2 \in \{c_2, c_2'\}} \mathcal{D}_{\text{node}}(x_1, x_2),$$

where $\mathcal{D}_{\text{node}}(x_1, x_2)$ denotes the node distance in the schema graph $\mathcal{S}$. Under this definition, two edges directly connected through a common node have distance 0, whereas disconnected edges have distance of at least 1.

Leveraging this edge distance, Algorithm 1 finds the top-k_{qg} schema-aligned query graphs by assigning pseudo edges to candidate schema edges, ensuring structural coherence under the distance threshold δ and minimizing the cumulative L2-distance. initialize a priority queue $\mathcal{L}$ for final assignments and an adjacency mapping $\mathcal{A}$ indicating shared nodes among pseudo edges. The $\texttt{MatchSG}$ recursively assigns each pseudo edge ℓ_i to schema-edge candidates. handle termination when

Algorithm 1: Distance Edge Join

Input: A pseudo query graph $Q_{\text{pseudo}} = \{\ell_1, \ldots, \ell_n\}$, $\text{Cands}(\ell_i) = \{\alpha_j\}$, an edge distance function $\mathcal{D}_{\text{edge}}(\cdot, \cdot)$, an edge distance threshold $\delta \geq 0$, and the parameter $k_{qg} > 0$.
Output: Top-k_{qg} best valid query graphs minimizing total L2 distance.

$\mathcal{L} \leftarrow \varnothing$ (priority queue of final schema edge assignments by L2 distance)
$\mathcal{A} \leftarrow$ adjacency among edges, where $\mathcal{A}(i)$ collects indices j such that ℓ_i, ℓ_j share a pseudo node
Function `MatchSG`(M, D, i)
  if $i > n$ **then**
    $\mathcal{L}$.push$((M,\ D))$
    return
  $\ell_i \leftarrow i$-th pseudo edge.
  foreach $\alpha \in \text{Cands}(\ell_i)$ **do**
    feasible $\leftarrow$ true
    foreach $j \in \{j' \in \mathcal{A}(i) \mid \ell_{j'} \in M\}$ **do**
      $\beta \leftarrow M[j]$ (chosen edge for ℓ_j)
      if $\mathcal{D}_{edge}(\alpha, \beta) > \delta$ **then**
        feasible $\leftarrow$ false; **break**
    if feasible **then**
      $M[i] \leftarrow \alpha$; $d_i \leftarrow$ `L2dist`(α, ℓ_i)
      `MatchSG`$(M, D + d_i, i + 1)$
      $M[i] \leftarrow \varnothing$
`MatchSG`$(\{\}, 0, 1)$
return top-k_{qg} elements from $\mathcal{L}$

all edges are assigned, pushing the current assignment and total L2-distance into $\mathcal{L}$. iterate candidate edges $\alpha \in \text{Cands}(\ell_i)$, verifying feasibility by ensuring adjacent edges' distance $\mathcal{D}_{\text{edge}}(\alpha, \beta)$ is within δ. Feasible assignments update the partial match M and total L2-distance D. Finally, the top-k_{qg} assignments are returned from $\mathcal{L}$, providing schema-aligned query graphs balancing structural constraints with semantic relevance.

After aligning edges, we introduce intermediate variable nodes or relations to bridge schema edges connected at nonzero distances, and explicitly instantiate each node position either as a known entity or as a typed variable node. This yields fully connected and semantically precise query graphs. Overall, ADJ produces structurally coherent, semantically aligned query graphs, suitable for effective knowledge retrieval.

4.3 Ranked Semantic Subgraph Matching

Following ADJ, we employ a single fixed text-embedding function for semantic matching. We encode entity labels, relation names, and type names with the same $\text{emb}(\cdot)$ and represent a schema edge $e = (c, r, c')$ by a composite vector $h(e) = \text{emb}([\text{text}(c); \text{text}(r); \text{text}(c')])$; similarity is measured by $\|h(e) - h(e')\|_2$. We use precomputed nearest-neighbor candidate sets `SimEnt`$(\cdot)$, `SimRel`$(\cdot)$, and `SimTyp`$(\cdot)$ to bound the search; implementation details are provided in Section 5.

Algorithm 2: Semantic Subgraph Retrieval

Input: Query graph Q, KG $\mathcal{G}$, and the parameter $k_{\text{retrieval}}$.
Output: Top-$k_{\text{retrieval}}$ matches from Q in $\mathcal{G}$.

$\mathcal{L} \leftarrow \varnothing$ (priority queue by L2 distance)
$\psi \leftarrow$ DFS order of edges in Q (start from entity u^*)
Function `MatchKG`(M, D, i)
  if $i > |\psi|$ **then**
    $\mathcal{L}$.push$((M, D))$; **return**
  $(u, r, u') \leftarrow \psi[i]$ (an edge where u is matched)
  $C_r \leftarrow$ `AdjRel`$(M[u]) \cap$ `SimRel`(r)
  foreach $r_{\mathcal{G}} \in C_r$ **do**
    $M[r] \leftarrow r_{\mathcal{G}}$;
    $d_r \leftarrow$ `SemDist`$(r,\ r_{\mathcal{G}})$
    $C_{u'} \leftarrow$ `GetCandNode`$(M[u],\ r_{\mathcal{G}},\ u')$
    foreach $u'_{\mathcal{G}} \in C_{u'}$ **do**
      if `consistent`$(u'_{\mathcal{G}},\ M)$ **then**
        $M[u'] \leftarrow u'_{\mathcal{G}}$;
        $d_{u'} \leftarrow$ `SemDist`$(u',\ u'_{\mathcal{G}})$
        `MatchKG`$(M, D + d_r + d_{u'}, i + 1)$
        $M[u'] \leftarrow \varnothing$
    $M[r] \leftarrow \varnothing$
foreach $u^*_{\mathcal{G}} \in$ `SimEnt`(u^*) **do**
  $d \leftarrow$ `L2dist`$(u^*,\ x)$;
  `MatchKG`$(\{u^* \mapsto u^*_{\mathcal{G}}\}, d, 1)$
return top-$k_{\text{retrieval}}$ elements from $\mathcal{L}$

Given generated query graphs, our goal is to identify relevant subgraphs from the knowledge graph $\mathcal{G}$ for each query graph. We formalize this task as *ranked semantic subgraph matching*, allowing for minor semantic discrepancies between the query graphs and candidate subgraphs from $\mathcal{G}$.

Definition 1 (Ranked Semantic Subgraph Matching)**.** *Given a query graph Q, a knowledge graph $\mathcal{G}$ and the threshold τ, we say a subgraph $H \subseteq \mathcal{G}$ is a semantic match of Q if there exists a mapping $\phi : (E \cup V_E) \cup (R \cup V_R) \rightarrow E \cup R$ such that $\forall (e_q, r_q, e'_q) \in Q, (\phi(e_q), \phi(r_q), \phi(e'_q)) \in H$ and the semantic distance over all distinct nodes and relations in Q remains below τ:*

$$\sum_{e_q \in nodes(Q)} \mathcal{D}_{\text{sem}}(e_q, \phi(e_q)) + \sum_{(\cdot, r_q, \cdot) \in Q} \mathcal{D}_{\text{sem}}(r_q, \phi(r_q)) \ \leq \ \tau,$$

where $\mathcal{D}_{\text{sem}}(\cdot, \cdot)$ is a semantic distance function, and $nodes(Q)$ collects the distinct nodes in Q.

To perform ranked semantic subgraph matching, we propose the *Semantic Subgraph Retrieval* algorithm (Algorithm 2). It extends conventional subgraph isomorphism (Han et al., 2013) by incorporating semantic similarity as SimGRAG (Cai et al., 2024), along with explicit type-level dynamic pruning and scoring.

We leverage precomputed similarity-based candidate sets to prune the matching space: `SimEnt`(u) for known entities, `SimTyp`(u') for variable nodes, and `SimRel`(r) for relations. Algorithm 2 maintains

a priority queue $\mathcal{L}$ of partial/final matches and follows a DFS-based edge ordering ψ. The search is initialized by assigning the anchor node u^* to KG candidates from $\texttt{SimEnt}(u^*)$.

Given a partial match M, `MatchKG` processes each query edge (u, r, u') in the order ψ. It intersects the relations incident to $M[u]$ with $\texttt{SimRel}(r)$ to obtain feasible relation candidates, adds the relation-level contribution $d_r = \texttt{L2dist}(r, r_{\mathcal{G}})$ to the running score, and then determines endpoint candidates through the procedure below. Specifically, `GetCandNode` returns only KG nodes that are reachable from $M[u]$ via the chosen relation $r_{\mathcal{G}}$; for variable nodes, it also enforces type consistency via $\texttt{SimTyp}(u')$, while for known entities it intersects neighbors with $\texttt{SimEnt}(u')$:

$$\texttt{SemDist}(x, x_{\mathcal{G}}) = \begin{cases} \min\limits_{t \in \texttt{type}(x_{\mathcal{G}})} \texttt{L2dist}(\texttt{type}(x), t), & \text{If } \bigwedge\limits^{x \in V_E \cup V_R}_{\texttt{type}(x) \text{ known}} \\ 0, & \text{If } \bigwedge\limits^{x \in V_E \cup V_R}_{\texttt{type}(x) \text{ unknown}} \\ \texttt{L2dist}(x, x_{\mathcal{G}}), & \text{If } x \in E \cup R \end{cases}$$

For each feasible neighbor $u'_{\mathcal{G}}$ in $C_{u'}$, the check $\texttt{consistent}(u'_{\mathcal{G}}, M)$ prevents mapping a query node to multiple entities. If consistent, we tentatively extend M with $u' \mapsto u'_{\mathcal{G}}$ and compute the node-level contribution $d_{u'}$ using `SemDist`, which incorporates type-level distance when types are available:

$$\texttt{SemDist}(x, x_{\mathcal{G}}) = \begin{cases} \min\limits_{t \in \texttt{type}(x_{\mathcal{G}})} \texttt{L2dist}(\texttt{type}(x), t), & \text{If } \bigwedge\limits^{x \in V_E \cup V_R}_{\texttt{type}(x) \text{ known}} \\ 0, & \text{If } \bigwedge\limits^{x \in V_E \cup V_R}_{\texttt{type}(x) \text{ unknown}} \\ \texttt{L2dist}(x, x_{\mathcal{G}}), & \text{If } x \in E \cup R \end{cases}$$

The procedure then recurses on the next edge in ψ with the updated score $D + d_r + d_{u'}$. After exploring all neighbors, $M[u']$ is reset for backtracking. The search continues until every edge in ψ is realized; the top-$k_{\text{retrieval}}$ subgraphs, ranked by total distance, are returned from $\mathcal{L}$, ensuring that retrieved subgraphs respect both the structural layout of Q and the semantic constraints. Following SimGRAG, we also prune any partial match whose current lower bound already exceeds the worst score in the top-$k_{\text{retrieval}}$ list; see Appendix C for implementation details.

5 Experiments

In this section, we describe the experimental setup and evaluation results. Additional details regarding hyperparameters, configurations and implementations are provided in Appendix C.

Preprocessing (offline). We precompute all-pairs shortest paths (APSP) on the schema graph to enable constant-time node-distance lookups used by ADJ. On our schema with $|\mathcal{S}| = 12{,}797$ nodes, Floyd–Warshall takes $\sim$27.4 seconds once offline; this one-time cost is amortized across queries and is *not* included in end-to-end latency.

Embedding model and candidate sets. We use `snowflake-arctic-embed-l` uniformly for entities, relations, types, and schema-edge composites across all methods to isolate the effect of our schema-aware refinement. Nearest neighbors are retrieved with FAISS-HNSW (L2; M=64, *efConstruction*=512), with fixed candidate set sizes $|\texttt{SimEnt}|$=3, $|\texttt{SimRel}|$=10, and $|\texttt{SimTyp}|$=8.

Datasets & Evaluation Metrics. We assess the performance of SAFE on three widely-used multi-hop KGQA benchmarks: CWQ (Talmor and Berant, 2018), WebQSP (Yih et al., 2016), and GrailQA (Gu et al., 2021), all based on the Freebase knowledge graph (Bollacker et al., 2008). For a fair and efficient comparison, we follow prior works (Sun et al., 2024; Chen et al., 2024b) by using their GrailQA test subsets. In line with common practice (Sun et al., 2024; Chen et al., 2024b; Luo et al., 2024a; Cai et al., 2024), exact match accuracy (Hits@1) serves as our evaluation metric.

Compared Methods. Considering the distinct characteristics and performance variations across different benchmarks, we select state-of-the-art (SOTA) baselines tailored for each dataset. These include (1) LLM-only methods—IO prompting (Brown et al., 2020), Chain-of-Thought (Wei et al., 2022), and Self-Consistency (Wang et al., 2023)—and (2) KG-enhanced methods that incorporate knowledge graphs via prompting or fine-tuning. For CWQ and WebQSP, we include fine-tuned models RoG (Luo et al., 2024b) and ChatKBQA (Luo et al., 2024a), and prompting methods ToG (Sun et al., 2024), PoG (Chen et al., 2024b), and SimGRAG (Cai et al., 2024). For GrailQA, we evaluate Pangu (Gu et al., 2023), FlexKBQA (Li et al., 2024b), GAIN (Shu and Yu, 2024), ChatKBQA, and the same prompting methods. We also test ToG, PoG, and ChatKBQA without topic entities using GPT-3.5 to assess performance without topic entity.

5.1 Performance Comparison

We compare SAFE with state-of-the-art KG-augmented LLM baselines to demonstrate its effectiveness in multi-hop KGQA. Tables 1 and 2 sum-

Method	CWQ	WebQSP
LLM-Only		
IO Prompt (Brown et al., 2020)	37.6	63.3
CoT (Wei et al., 2022)	38.8	62.2
SC (Wang et al., 2023)	45.4	61.1
Fine-Tuned KG-Augmented LLM		
RoG (Luo et al., 2024b)	62.6	85.7
ChatKBQA (Luo et al., 2024a)	86.0	86.4
ChatKBQA w/o topic entity	82.7	83.2
Prompting KG-Augmented LLM w/GPT-3.5 or others		
ToG (Sun et al., 2024)	57.1	76.2
ToG w/o topic entity	51.2	68.3
PoG (Chen et al., 2024b)	63.2	82.0
PoG w/o topic entity	57.8	73.2
SimGRAG (Cai et al., 2024)	50.2	69.1
SAFE	**66.8**	**84.7**
Prompting KG-Augmented LLM w/GPT-4		
InteractiveKBQA (Xiong et al., 2024)	59.2	72.5
ToG (Sun et al., 2024)	67.6	82.6
PoG (Chen et al., 2024b)	75.0	87.3
SimGRAG (Cai et al., 2024)	70.5	86.7
SAFE	**79.4**	**90.6**

Table 1: Performance comparison across different methods on CWQ and WebQSP. (The best-performing result is shown in bold)

marize the experimental results on CWQ, WebQSP, and GrailQA. Overall, SAFE consistently surpasses prompting-based methods (e.g., PoG, ToG, SimGRAG), achieving competitive or superior performance relative to fine-tuned approaches.

Compared to existing prompting-based methods, SAFE improves accuracy while eliminating the restrictive assumption of known topic entities. For instance, on CWQ (GPT-4), SAFE achieves a Hits@1 of 79.4%, outperforming PoG (75.0%) and SimGRAG (70.5%) by margins. This performance advantage is further emphasized by results in Table 1, which show significant accuracy drops for PoG (75.0% to 57.8%) and ToG (67.6% to 51.2%) when topic entities are not provided. In contrast, SAFE inherently avoids this reliance, flexibly resolving mismatches through schema-guided query correction and type-based filtering.

Although SAFE relies purely on prompting without any training or provided topic entities, it consistently outperforms fine-tuned methods on most datasets. For example, on WebQSP (GPT-4), SAFE achieves 90.6%, surpassing ChatKBQA's 86.4%. On CWQ, SAFE attains 79.4%, slightly lower than ChatKBQA (86.0%), primarily because CWQ's questions are derived through a template-based construction, inherently favoring fine-tuned models (see Appendix B).

On GrailQA (GPT-4), SAFE achieves an overall accuracy of 85.5%, significantly improving performance on zero-shot (89.5%) and compositional queries (77.3%), surpassing PoG by 7.6% on compositional cases. Notably, ChatKBQA achieves the highest I.I.D. accuracy (96.7%), but its accuracy drastically drops to only 4.98% on zero-shot queries (Table 2), illustrating severe generalization limitations when encountering unseen patterns. In contrast, SAFE consistently demonstrates strong generalization across all subsets, effectively addressing challenging queries through its schema-aware correction and type-level semantic matching, thereby significantly reducing errors.

5.2 Ablation Study

We conduct an ablation study to assess the contributions of individual components within SAFE by independently removing the Approximate Distance Join (ADJ), ranked semantic subgraph matching, and type-level constraints. Table 3 summarizes the performance on CWQ, WebQSP, and GrailQA datasets. Specifically, **w/o ADJ** excludes the schema-alignment step, directly using the LLM-generated pseudo query graph for subgraph matching. **w/o Ranked semantic subgraph matching** strictly matches query graphs to the KG without retrieving semantically similar entities, relations, or types. **w/o Type-level constraints** omits dynamic type-based pruning and type-level semantic scoring based on L2 distance.

As shown in Table 3, removing ADJ significantly decreases Hits@1 accuracy by 14.6%, 12.6%, and 12.7% on CWQ, WebQSP, and GrailQA, respectively. This demonstrates that aligning pseudo query graphs with the KG schema is essential to correct structural mismatches and ensure accurate retrieval. Similarly, removing ranked semantic subgraph matching reduces accuracy by 7.6% on CWQ, 6.0% on WebQSP, and 8.2% on GrailQA. While ADJ alone provides substantial performance by aligning query graphs structurally, ranked semantic subgraph matching further enhances accuracy by considering additional candidate relations and types beyond the top-k_{qg} schema-aligned query graphs selected by ADJ. Additionally, eliminating type-level constraints leads to accuracy reductions of 4.7% on CWQ, 3.9% on WebQSP, and 3.8% on GrailQA, confirming the effectiveness of type-level pruning and scoring in filtering out irrelevant candidate subgraphs. We also considered performing ADJ directly on the KG. However, this requires computing APSP distances over the entire KG, consuming over 6 petabytes (4×10^7 entities^2 $\times$ 4 bytes). This computational infeasibility highlights

Method	Overall	I.I.D.	Compositional	Zero-shot
w.o. Knowledge Graph				
IO Prompt (Brown et al., 2020)	29.4	-	-	-
CoT (Wei et al., 2022)	28.1	-	-	-
SC (Wang et al., 2023)	29.6	-	-	-
w.t. Knowledge Graph / Fine-tuned				
Pangu (Gu et al., 2023)	75.4	84.4	74.6	71.6
FlexKBQA (Li et al., 2024b)	62.8	71.3	59.1	60.6
GAIN (Shu and Yu, 2024)	76.3	88.5	73.7	71.8
ChatKBQA (Luo et al., 2024a)	38.7	96.7	64.1	4.98
ChatKBQA w/o topic entity	36.2	91.7	58.1	4.80
w.t. Knowledge Graph / Prompting (GPT-3.5 or others)				
ToG (Sun et al., 2024)	68.7	70.1	56.1	72.7
ToG w/o topic entity	58.6	59.2	51.5	60.9
PoG (Chen et al., 2024b)	76.5	76.3	62.1	81.7
PoG w/o topic entity	66.3	65.4	57.6	69.8
SimGRAG (Cai et al., 2024)	62.7	58.8	50.5	68.7
SAFE	**78.5**	**74.2**	**66.7**	**84.5**
w.t. Knowledge Graph / Prompting (GPT-4)				
ToG (Sun et al., 2024)	81.4	79.4	67.3	86.5
PoG (Chen et al., 2024b)	84.7	87.9	69.7	88.6
SimGRAG (Cai et al., 2024)	77.2	72.5	68.7	82.2
SAFE	**85.5**	**82.9**	**77.3**	**89.5**

Table 2: Performance comparison on GrailQA under various generalization types: IID (i-distribution), compositional, and zero-shot. "-" indicates the result was not reported. (The best-performing result is shown in bold)

Method	CWQ	WebQSP	GrailQA
SAFE	**66.8**	**84.7**	**78.5**
w/o ADJ	52.2	72.1	65.8
w/o Ranked semantic subgraph matching	59.2	78.7	70.3
w/o Type-level constraints	62.1	80.8	74.7

Table 3: Impact of each module on Hits@1

the efficiency of our schema-based approach.

Overall, these results underline the critical roles of ADJ, ranked semantic subgraph matching, and type-level constraints in achieving the superior performance of SAFE on complex multi-hop queries.

5.3 Efficiency Study

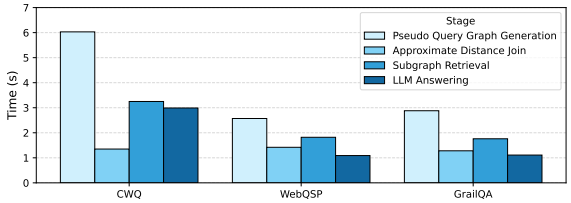
To highlight the efficiency of SAFE, we compare its inference latency, number of LLM calls, and token usage with the SOTA KG-augmented method PoG. Additionally, we provide a detailed breakdown of inference time across individual stages to better understand their relative contributions.

Metric	CWQ		WebQSP		GrailQA	
	PoG	SAFE	PoG	SAFE	PoG	SAFE
LLM Call	13.8	**2**	9.0	**2**	6.5	**2**
Input Token	8,068.6	**3,165.6**	5,234.8	**2,341.4**	3,392.3	**2,553.2**
Output Token	372.7	**470.1**	282.9	**272.0**	203.1	**327.7**
Total Token	8,441.3	**3,635.7**	5,517.7	**2,613.4**	3,595.6	**2,880.9**
Time (s)	34.3	**13.6**	16.8	**7.08**	11.6	**7.18**

Table 4: Efficiency comparison across three datasets.

Overall Efficiency Comparison. Table 4 compares the inference efficiency of SOTA KG-augmented method PoG and SAFE on CWQ, WebQSP, and GrailQA. PoG requires multiple iterative interactions with the LLM, averaging between 6.5 and 13.8 calls per query. In contrast, SAFE consistently performs only two LLM calls (one for pseudo query graph generation and another for final answer generation) regardless of the dataset.

Figure 3: Inference time breakdown of SAFE across the CWQ, WebQSP, and GrailQA datasets.



This substantially reduces both the total LLM token usage and the inference latency.

Specifically, on CWQ, PoG processes an average of 8,441.3 tokens and requires 34.3 seconds per query, while SAFE reduces token usage by 56.9% to 3,635.7 tokens and inference time by 60.3% to 13.6 seconds. A similar efficiency improvement appears on WebQSP, where SAFE uses only 2,613.4 tokens (a 52.6% reduction from PoG) and takes 7.08 seconds (57.9% faster than PoG). On GrailQA, token usage decreases by 19.9% (2,880.9 vs. 3,595.6 tokens) and runtime decreases by 38.1% (7.18 vs. 11.6 seconds).

These results clearly illustrate the superior efficiency of SAFE. Rather than relying on repeated reasoning steps and multiple LLM interactions, SAFE combines a schema-aligned retrieval step with minimal LLM interactions, significantly lowering token consumption and latency. This efficient design makes SAFE more scalable and practical for deployment in real-world scenarios.

Inference Time Breakdown. Figure 3 shows the inference time breakdown of SAFE across CWQ, WebQSP, and GrailQA for four pipeline stages: (i) Pseudo Query Graph Generation, (ii) Approximate Distance Join (ADJ), (iii) Subgraph Retrieval, and (iv) LLM Answering. WebQSP (6.91s) and

Dataset	Method	1-shot	2-shot	3-shot
CWQ	SimGRAG	44.9	48.3	50.2
	SAFE	**64.5**	**65.7**	**66.8**
WebQSP	SimGRAG	64.2	66.9	69.1
	SAFE	**83.5**	**84.2**	**84.7**
GrailQA	SimGRAG	57.9	59.7	62.7
	SAFE	**76.9**	**77.4**	**78.5**

Table 5: Robustness to prompt variations (Hits@1, %). Lower prompt shot counts reduce context. SAFE exhibits smaller degradation than SimGRAG.

# Max Types	0–7	7–16	16–38	38–62	62–156
Hits@1 (%)	66.9	66.6	66.5	66.8	67.2
Latency (s)	13.54	13.39	13.48	13.88	13.83

Table 6: SAFE accuracy and latency across type-multiplicity bins on CWQ.

GrailQA (7.00s) show similar patterns, with pseudo query graph generation dominating the runtime. CWQ (13.63s) takes nearly twice as long due to longer prompts and increased retrieval time. Notably, ADJ consistently maintains low latency (1.28–1.42s), and Subgraph Retrieval is efficiently conducted within approximately 1.7–3.2s across datasets. As pseudo query graph generation time correlates strongly with input prompt length, further optimization in prompt design and LLM inference could substantially enhance SAFE 's scalability and overall inference efficiency.

5.4 Robustness Analysis

Robustness to prompt variations. We vary the number of in-context examples (1/2/3-shot) when generating pseudo graphs with GPT-3.5 and evaluate Hits@1 on CWQ, WebQSP, and GrailQA. As shown in Table 5, SAFE shows only a small average drop when reducing from 3-shot to 1-shot (about 1.7% absolute) whereas SimGRAG decreases by about 5.0%, indicating that ADJ mitigates pseudo-graph variations.

Effect of type multiplicity. To quantify whether entities with many types degrade performance, we bucket CWQ test queries by the maximum number of types assigned to any entity in the gold graph (equi-height bins). Table 6 shows that SAFE maintains stable accuracy and latency across bins: Hits@1 varies within $\pm 1\%$ and latency within ± 0.5s, indicating negligible impact on our type-aware refinement.

6 Conclusion

In this paper, we introduce SAFE, a novel framework addressing limitations of existing KG-augmented language model systems. SAFE leverages a compact schema graph to generate semantically coherent query graphs via an Approximate Distance Join (ADJ) method, effectively managing misalignment of pseudo query graph and missing entities. Extensive experiments on CWQ, WebQSP, and GrailQA benchmarks demonstrated SAFE's superior performance over SOTA methods, underscoring its effectiveness and scalability for reliable knowledge retrieval.

7 Limitation

Our approach currently supports basic graph query operations but does not handle advanced constructs such as `ORDER BY`, `LIMIT`, `UNION`, or complex `FILTER` clauses. Incorporating these may be possible by enabling the LLM to generate additional query operators and appropriately applying them during the retrieval process.

Acknowledgement

This work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2025-00517736, 50%), Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. RS-2024-00509258, Global AI Frontier Lab, 30%) (No. RS-2018-II181398, Development of a Conversational, Self-tuning DBMS, 10%) (No. RS-2024-00454666, Developing a Vector DB for Long-Term Memory Storage of Hyperscale AI Models, 5%), and Basic Science Research Program through the National Research Foundation of Korea Ministry of Education(No. RS-2024-00415602, 5%)

References

Jinheon Baek, Alham Fikri Aji, and Amir Saffari. 2023. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. *arXiv preprint arXiv:2306.04136*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yuzheng Cai, Zhenyue Guo, Yiwen Pei, Wanrui Bian, and Weiguo Zheng. 2024. Simgrag: Leveraging similar subgraphs for knowledge graphs driven retrieval-augmented generation. *arXiv preprint arXiv:2412.15272*.

Liyi Chen, Zhi Li, Tong Xu, Han Wu, Zhefeng Wang, Nicholas Jing Yuan, and Enhong Chen. 2022. Multi-modal siamese network for entity alignment. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 118–126.

Liyi Chen, Chuan Qin, Ying Sun, Xin Song, Tong Xu, Hengshu Zhu, and Hui Xiong. 2024a. Collaboration-aware hybrid learning for knowledge development prediction. In *Proceedings of the ACM on Web Conference 2024*, pages 3976–3985.

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024b. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. *arXiv preprint arXiv:2410.23875*.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2022. *Introduction to Algorithms*, 4 edition. MIT Press.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281*.

Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 international conference on management of data*, pages 1433–1445.

Yu Gu, Xiang Deng, and Yu Su. 2023. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4928–4949.

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.

Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. 2013. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD international conference on management of data*, pages 337–348.

Steve Harris and Andy Seaborne. 2013. Sparql 1.1 query language. https://www.w3.org/TR/sparql11-query/. W3C Recommendation.

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.

Jiho Kim, Yeonsu Kwon, Yohan Jo, and Edward Choi. 2023a. Kg-gpt: A general framework for reasoning on knowledge graphs using large language models. *arXiv preprint arXiv:2310.11220*.

Jiho Kim, Sungjin Park, Yeonsu Kwon, Yohan Jo, James Thorne, and Edward Choi. 2023b. Factkg: Fact verification via reasoning on knowledge graphs. *arXiv preprint arXiv:2305.06590*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

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.

Xiaoxi Li, Yujia Zhou, and Zhicheng Dou. 2024a. Unigen: A unified generative framework for retrieval and question answering with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 8688–8696.

Zhenyu Li, Sunqi Fan, Yu Gu, Xiuxing Li, Zhichao Duan, Bowen Dong, Ning Liu, and Jianyong Wang. 2024b. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18608–18616.

Haochen Liu, Song Wang, Yaochen Zhu, Yushun Dong, and Jundong Li. 2024. Knowledge graph-enhanced large language models via path selection. *arXiv preprint arXiv:2406.13862*.

Haoran Luo, E Haihong, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, and 1 others. 2024a. 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.

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

Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. 2024. Skeleton-of-thought: Large language models can do parallel decoding. In *The Twelfth International Conference on Learning Representations*.

OpenAI. Chatgpt via chat completions api.

Kashif Rabbani, Matteo Lissandrini, and Katja Hose. 2023. Extraction of validating shapes from very large knowledge graphs. *Proceedings of the VLDB Endowment*, 16(5):1023–1032.

Yiheng Shu and Zhiwei Yu. 2024. Distribution shifts are bottlenecks: Extensive evaluation for grounding language models to knowledge bases. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 71–88.

Snowflake. 2025. snowflake-arctic-embed-l. https://huggingface.co/Snowflake/snowflake-arctic-embed-l.

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.

Yanchao Tan, Hang Lv, Xinyi Huang, Jiawei Zhang, Shiping Wang, and Carl Yang. 2024. Musegraph: Graph-oriented instruction tuning of large language models for generic graph mining. *arXiv preprint arXiv:2403.04780*.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2023. Graphgpt: Graph instruction tuning for large language models. *arXiv preprint arXiv:2310.13023*.

Lei Wang, Yi Hu, Jiabang He, Xing Xu, Ning Liu, Hui Liu, and Heng Tao Shen. 2024. T-sciq: Teaching multimodal chain-of-thought reasoning via large language model signals for science question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19162–19170.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Guanming Xiong, Junwei Bao, and Wen Zhao. 2024. Interactive-kbqa: Multi-turn interactions for knowledge base question answering with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers*, pages 10561–10582.

Yao Xu, Shizhu He, Jiabei Chen, Zihao Wang, Yangqiu Song, Hanghang Tong, Guang Liu, Kang Liu, and Jun Zhao. 2024. Generate-on-graph: Treat llm as both agent and kg in incomplete knowledge graph question answering. *arXiv preprint arXiv:2404.14741*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Wen-tau Yih, Matthew Richardson, Christopher 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.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Yang Wang, Zhiguo Wang, and Bing Xiang. 2023. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases. In *The International Conference on Learning Representations*.

Yuting Zhang, Ying Sun, Fuzhen Zhuang, Yongchun Zhu, Zhulin An, and Yongjun Xu. 2023. Triple dual learning for opinion-based explainable recommendation. *ACM Transactions on Information Systems*, 42(3):1–27.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Lili Zhao, Qi Liu, Linan Yue, Wei Chen, Liyi Chen, Ruijun Sun, and Chao Song. 2024. Comi: Correct and mitigate shortcut learning behavior in deep neural networks. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 218–228.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and 1 others. 2023. Least-to-most prompting enables complex reasoning in large language models. *The International Conference on Learning Representations*.

Lei Zou, Lei Chen, and M Tamer Özsu. 2009. Distancejoin: Pattern match query in a large graph database. *Proceedings of the VLDB Endowment*, 2(1):886–897.

Appendix

A Overview of Comparison Methods

We organize the baseline methods into two main groups: (1) Methods relying solely on LLM reasoning, and (2) Methods integrating external KGs with LLMs, further subdivided into fine-tuning and prompt-based approaches.

LLM-Only Approaches

- IO Prompt (Brown et al., 2020) evaluates standard LLM capabilities in few-shot, task-agnostic scenarios, comparing their efficacy to traditional language models.
- Chain-of-Thought (CoT)(Wei et al., 2022) provides intermediate reasoning steps explicitly in the prompt to enhance LLM performance across various NLP tasks.
- Self-Consistency (SC)(Wang et al., 2023) extends CoT by generating multiple reasoning chains through diverse sampling and then selecting the most consistently derived answer.

KG-Enhanced Methods with Fine-Tuning

- RoG (Luo et al., 2024b) integrates LLMs with KG structural knowledge to enhance the reliability and interpretability of reasoning processes.
- Pangu (Gu et al., 2023) employs a symbolic reasoning agent in collaboration with neural LLM discriminators to effectively leverage knowledge graph information.
- FlexKBQA (Li et al., 2024b) proposes a versatile framework capable of quickly adapting LLMs for KGQA tasks across different knowledge graphs and query languages with minimal labeled examples.
- GAIN (Shu and Yu, 2024) focuses on improving robustness against distributional shifts in KGQA through data augmentation and comprehensive robustness evaluations across various dimensions.
- ChatKBQA (Luo et al., 2024a) converts natural language queries into structured SPARQL queries using fine-tuned LLMs, subsequently refining the generated logical forms using retrieved knowledge to reduce hallucinations.

KG-Enhanced Methods with Prompting

- Interactive KBQA (Xiong et al., 2024) generates logical queries by direct interactions with knowledge graphs, facilitated by a set of predefined KG interfaces.
- ToG (Sun et al., 2024) employs iterative KG retrieval combined with beam-search-based reasoning verification through LLMs, dynamically deciding whether additional retrieval steps are required.
- PoG (Chen et al., 2024b) extends ToG by introducing memory-enhanced reasoning, backtracking capabilities, and adaptive beam search breadth.
- GoG (Xu et al., 2024) integrates LLM-generated implicit knowledge with explicit KG traversal to address question answering in scenarios involving incomplete knowledge graphs.

B Datasets

	CWQ	WebQSP	GrailQA
Answer Format	Entity	Entity/Number	Entity/Number
Train	27,734	3,098	44,337
Test	3,531	1,639	1,000

Table 7: Summary of KGQA Benchmark Datasets.

In our experiments, we evaluate our method using three challenging multi-hop KGQA benchmarks: ComplexWebQuestions (CWQ)(Talmor and Berant, 2018), WebQSP(Yih et al., 2016), and GrailQA (Gu et al., 2021). Key statistics for each dataset are summarized in Table 7. WebQSP comprises naturally occurring questions derived from WebQuestions and primarily tests models under the standard independent and identically distributed (i.i.d.) assumption, meaning the training and testing distributions are identical. CWQ, also created under the i.i.d. assumption, is constructed by combining multiple SPARQL queries associated with WebQSP questions. These combined queries are initially converted into natural language questions via rule-based generation, followed by manual refinement for readability. Consequently, CWQ questions often exhibit repetitive or structured patterns, resulting in a more predictable, template-like style.

Lastly, GrailQA, another Freebase-based benchmark, explicitly evaluates model generalization across three distinct dimensions: i.i.d., compositional, and zero-shot scenarios. For computational efficiency and consistency in evaluation, we use the same set of 1,000 test samples from GrailQA as previously utilized by ToG (Sun et al., 2024) and PoG (Chen et al., 2024b).

C Implementation Details

C.1 Hardware and Software Settings

Experiments were conducted on a server equipped with an Intel Xeon Gold 6230 CPU @ 2.10GHz, 1TB of RAM, and four NVIDIA RTX A6000 GPUs, running Ubuntu 22.04.3 LTS.

C.2 Schema Graph Generation

The goal of a *schema graph* is to summarize "which type can be linked to which other type **in practice**", discarding those links that appear only as rare accidents in the raw knowledge graph (KG). A tempting baseline is to look at every triple $\langle s, p, o \rangle$ in the KG, collect the `type.object.type` sets of its subject s and object o, and then output *all* pairs (t_d, p, t_r) with $t_d \in \texttt{type}(s)$, $t_r \in \texttt{type}(o)$. Unfortunately, on a Freebase dump, such a cartesian expansion explodes into *tens of billions* of candidate links; most are backed by a single noisy triple and therefore useless for reasoning while still occupying prohibitive memory.

Counting phase. We therefore adopt the *Quality Shapes Extraction* technique of Rabbani et al. (2023). While streaming the KG once we *(i)* assign dense integer IDs to each predicate p and type string, *(ii)* count how many distinct entities carry each individual type—this will be the *support of the type itself*—and *(iii)* for every triple increment

$$cnt_p(t_d, t_r) \quad \forall t_d \in \texttt{type}(s),\ t_r \in \texttt{type}(o).$$

Here $cnt_p(t_d, t_r)$ is the raw frequency of observing predicate p *between* a subject of type t_d and an object of type t_r.

Support and confidence. After the scan we derive two quality metrics:

$$supp(t_d, p, t_r) = cnt_p(t_d, t_r)$$

$$conf(t_d, p, t_r) = \frac{cnt_p(t_d, t_r)}{supp(t_d)}.$$

The *support* (*supp*) is the absolute number of triples backing a candidate edge; the *confidence* (*conf*) is the relative fraction of all t_d–entities that actually use predicate p towards some t_r. Intuitively, high support guards against random noise, while high confidence rules out idiosyncratic links attached to only a handful of subjects.

Pruning. We keep candidate edges only if their support (*supp*) and confidence (*conf*) satisfy $supp \geq \theta_{supp}$ and $conf \geq \theta_{conf}$. Using conservative thresholds ($\theta_{supp} = 1$, $\theta_{conf} = 0.0001$), our original Freebase slice containing 1B triples shrinks to a schema graph with approximately 40M edges. To further simplify the schema graph, higher confidence and support thresholds can be applied.

C.3 Embedding Generation

We adopt the snowflake-arctic-embed-l embedding model (Snowflake, 2025) to generate semantic embeddings of dimension 1024 for values, relations, and types. To efficiently retrieve similar embeddings, we utilize the HNSW algorithm (Malkov and Yashunin, 2018) as implemented in the Faiss library (Douze et al., 2024), configured with a maximum degree of $M = 64$ and $efConstruction = 512$.

C.4 Retrieval Optimization

Following SimGRAG (Cai et al., 2024), we optimized retrieval using early pruning based on semantic distance. We prune partial mappings whose semantic lower bounds exceed the largest semantic distance among current top-$k_{\text{retrieval}}$ subgraphs, and prioritize expansion of nodes and relations with smaller semantic distances to enhance pruning effectiveness. This optimization accelerates retrieval while preserving the exact quality of top-$k_{\text{retrieval}}$ results.

C.5 Parameters Setting

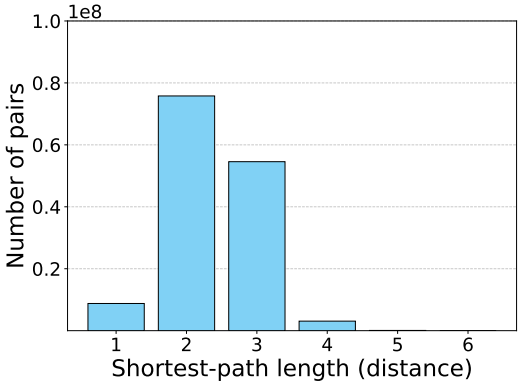
We set $\delta = 1$, $k_{\text{schema}} = 3$, $k_{\text{qg}} = 5$, and $k_{\text{retrieval}} = 10$ to effectively balance query-graph diversity, schema-level relevance, and retrieval comprehensiveness. Additionally, we configure $|\texttt{SimTyp}(\cdot)| = 8$, $|\texttt{SimEnt}(\cdot)| = 3$, and $|\texttt{SimRel}(\cdot)| = 10$ to achieve sufficient semantic coverage while limiting irrelevant candidate matches. We configure the large language models using a temperature of 0.3, with both frequency and presence penalties set to 0, and limit the maximum generated tokens to 2000. Further rationale and detailed analysis of these parameters are provided in Appendix D.

D Searching Parameters

D.1 Determining Distance Threshold δ

The schema graph is highly dense, containing approximately 13,000 nodes (types) and 40 million edges. Thus, selecting a large distance threshold δ could unintentionally allow nearly all schema edges to become interconnected. To carefully choose δ, we analyze the shortest-path distances between all node pairs in the schema graph.

Figure 4: Distribution of shortest-path lengths between schema graph nodes.



As shown in Figure 4, the shortest-path lengths for most node pairs are concentrated at length 2. Therefore, setting a distance threshold $\delta \geq 2$ would lead to excessive and noisy schema edge combinations. Hence, we set $\delta = 1$ to avoid this issue.

D.2 Determining Other Parameters

Given our choice of $\delta = 1$, we set $k_{\text{schema}} = 3$ to avoid unnecessary computational overhead, as higher values can exponentially increase the maximum number of query graphs, potentially up to $k_{\text{schema}}^{|Q_{\text{pseudo}}|}$ in the worst case. In fact, setting $k_{\text{schema}} = 3$ is already sufficient to generate more schema-aligned query graphs than required by k_{qg}. Additionally, unlike SimGRAG, which utilizes very large similarity parameters ($\texttt{SimEnt} = 16384$, $\texttt{SimRel} = 512$), we obtain strong performance with significantly smaller values: $\texttt{SimEnt} = 3$, $\texttt{SimRel} = 10$, and $\texttt{SimTyp} = 8$. These optimal parameter settings were determined through grid search over the ranges $|\texttt{SimEnt}(\cdot)| \in \{1, 3, 5, 7\}$, $|\texttt{SimRel}(\cdot)| \in \{5, 10, 15\}$, and $|\texttt{SimTyp}(\cdot)| \in \{2, 4, 8, 16\}$, using 100 random samples from the CWQ dataset, which is the most complex dataset among the three datasets used in our experiments.

E Parameter Sensitivity

The retrieval and reasoning performance is significantly influenced by two parameters: (i) k_{qg}, the number of query graphs generated by the LLM, and (ii) $k_{\text{retrieval}}$, the number of retrieved subgraphs per query graph. Both parameters directly affect the quantity and diversity of information provided to the LLM during answer generation. To systematically analyze their impact, we conducted two controlled experiments: (1) varying $k_{qg} \in 1, 3, 5, 7$ while fixing $k_{\text{retrieval}} = 10$, and (2) varying $k_{\text{retrieval}} \in 5, 10, 15, 20$ while keeping $k_{qg} = 5$. We measured Hits@1 accuracy on 100 randomly selected questions from the CWQ dataset to observe performance trends clearly.

Figure 5 illustrates that as we increase k_{qg} from 1 to 5, Hits@1 accuracy steadily rises, indicating that multiple query graphs help the model access more relevant subgraphs. However, accuracy plateaus beyond $k_{qg} = 5$, suggesting that additional query graphs often lead to redundant retrievals, introducing noise rather than meaningful information. Thus, $k_{qg} = 5$ strikes the best balance between retrieval comprehensiveness and redundancy.

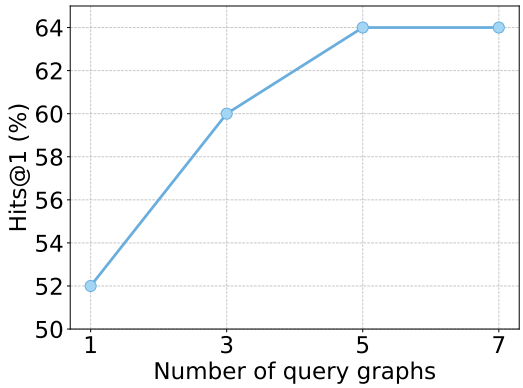


Figure 5: Hits@1 according to the number of query graphs (k_{qg}).

Similarly, Figure 6 shows a pronounced performance increase when raising $k_{\text{retrieval}}$ from 5 to 10. This highlights that retrieving up to 10 subgraphs per query graph is essential for capturing sufficiently diverse evidence to generate accurate answers. However, further increases in $k_{\text{retrieval}}$ beyond 10 produce marginal returns and can introduce irrelevant or redundant details, hindering the LLM's reasoning effectiveness. Consequently, we select $k_{\text{retrieval}} = 10$ as it provides optimal diversity and accuracy without excessive information overload.

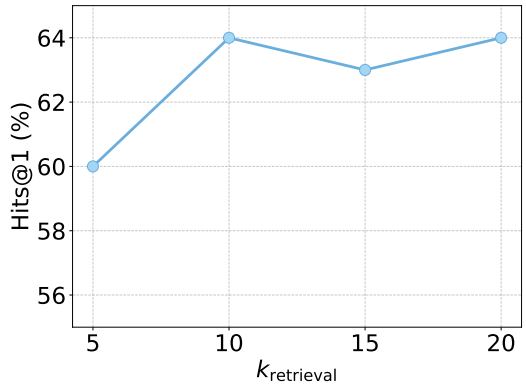


Figure 6: Hits@1 according to the number of retrieved subgraphs ($k_{\text{retrieval}}$).

F Software and Data Licenses

The software and datasets utilized in this paper have the following licenses:

- snowflake-arctic-embed-l: Apache 2.0 License
- CWQ: Creative Commons (CC) License
- WebQSP: Creative Commons (CC) License
- GrailQA: No explicit license provided
- Freebase: Creative Commons (CC) License

All resources were strictly employed for research purposes and not for any non-research or commercial applications. The Freebase knowledge graph used in this study primarily contains data sourced from Wikipedia.

G AI Assistants

We utilized ChatGPT-4o, 4.5 and o1-pro (OpenAI) to efficiently debug our code, rapidly pinpointing and correcting implementation errors. Moreover, we leveraged it to refine sentence structures, enhancing the clarity and readability of the manuscript.

H Prompt Templates

For pseudo query graph generation, we selected three training examples from each dataset and provided explicit verbalized generation steps. For answer generation, we adapted the prompt from SimGRAG (Cai et al., 2024), which was initially designed for the FactKG (Kim et al., 2023b) and MetaQA (Zhang et al., 2018) datasets.

Prompt Used to Generate Pseudo Query Graphs on CWQ

You are an expert Freebase question planner. Return a STRICT JSON with these keys in EXACT order:

1. `generation_plan`: step-by-step plan to generate the graph based on the divisions.
2. `nodes`: dictionary of {node: expected type of node}.
3. `edges`: list of [[source, predicate, target], description of what you have completed].
4. `filters`: at most one filter (dict with 'node', 'op', and 'value' if needed), else None.
5. `sort`: dict with 'key' and 'direction' if needed, else None.
6. `limit`: integer if needed, else None.

RULES (MUST FOLLOW ALL):

1. **Single Connected Component:**
   - All nodes must form one connected graph (no disconnected parts).
   - E.g., ["John", "likes", "unk1"], ["unk2", "likes", "unk3"] is INVALID (disconnected).
   - E.g., ["John", "likes", "unk1"], ["unk1", "likes", "unk2"] is VALID (connected).
2. **Known Entities Not Directly Connected:**
   - Two known entities cannot appear together in one edge. Use an 'unk' node between them.
3. **Node-Edge Consistency:**
   - Every node in 'edges' must be defined in 'nodes'.
   - Every node in 'nodes' must appear in at least one edge.
   - After enumerating edges, ensure all nodes are used.
4. **Unknown Nodes:**
   - Must be named 'unk1', 'unk2', etc.
   - Known entities stay exactly as in the query (no renaming).
5. **Filters and Sort:**
   - Use filter condition if comparative (e.g., >, <, >=, <=).
   - Use sort and limit if superlative (e.g., max, min).
   - Only one filter or one sort allowed (not both simultaneously).
   - Filter target node type must be 'date', 'gYear', 'gYearMonth', 'integer', 'dateTime', or 'float'.
6. **Output only the final JSON (no extra text).**

Example Queries:

1. query: "What country trades with Portugal and have an iSO alpha 3 of ARE?"

{"generation_plan": ("Step 0: Identify the known entity 'Portugal' from the query. Step 1. Find the country that trades with Portugal", "Step 2. Find the country that has an iSO alpha 3 of ARE", "Step 3: Introduce an unknown variable 'unk1' as the country trading with 'Portugal' to avoid direct connection. Link 'Portugal' to 'unk1' using 'location.statistical_region.places_exported_to||location.imports_and_exports.exported_to'. Step 4: Specify that 'unk1' has the property 'location.country.iso_alpha_3' = 'ARE'. All unknown nodes appear in both edges and nodes sections, and the graph remains connected."), "nodes": {"unk1": "location.country", "Portugal": "location.country", "ARE": "iso_alpha_3"}, "edges": [["Portugal", "location.statistical_region.places_exported_to||location.imports_and_exports.exported_to", "unk1"], "Completed steps 0 to 3, not step 4. Node 'ARE' unused."], [["unk1", "location.country.iso_alpha_3", "ARE"], "Completed step 4. All nodes used."]], "filters": [], "sort": None, "limit": None}

2. query: “What state is home to the university that is represented in sports by George Washington Colonials men’s basketball?"

{“generation_plan”: (“Step 0: Identify known entity ‘George Washington Colonials men’s basketball’. Step 1. Find the sports team. Step 2. Find the university represented by this team. Step 3. Find the state of the university. Step 4: Introduce ‘unk1’ as the university associated with the sports team, ‘unk2’ as the university’s mailing address, and ‘unk3’ as the state linked via ‘location.mailing_address.state_province_region’. Ensure connectivity and no direct known entity links.”), “nodes”: {“George Washington Colonials men’s basketball”: “sports.sports_team”, “unk1”: “education.educational_institution”, “unk2”: “location.mailing_address”, “unk3”: “location.administrative_division”}, “edges”: [[[”unk1”, “education.educational_institution.sports_teams”, “George Washington Colonials men’s basketball”], “Completed steps 0-3, nodes ‘unk2’, ‘unk3’ unused.”], [[“unk1”, “organization.organization.headquarters”, “unk2”], “Completed step 4, node ‘unk3’ unused.”], [[“unk2”, “location.mailing_address.state_province_region”, “unk3”], “Completed step 5, all nodes used.”]], “filters”: [], “sort”: None, “limit”: None}

3. query: “What year did the team with Baltimore Fight Song win the Super Bowl?"

{“generation_plan”: (“Step 0: Identify entities ‘The Baltimore Fight Song’ and ‘Super Bowl’. Step 1. Find the sports team with the fight song. Step 2. Find the championship event won. Step 3. Link event to ‘Super Bowl’ via ‘sports.sports_championship_event.championship’. Introduce ‘unk1’ (team) and ‘unk2’ (championship event) ensuring no direct known entity connection, full node usage, and connected graph.”), “nodes”: {“The Baltimore Fight Song”: “music.composition”, “Super Bowl”: “sports.sports_championship”, “unk1”: “sports.sports_team”, “unk2”: “sports.sports_championship_event”}, “edges”: [[[”unk1”, “sports.sports_team.fight_song”, “The Baltimore Fight Song”], “Completed steps 0-3, ‘unk2’, ‘Super Bowl’ unused.”], [[“unk1”, “sports.sports_team.championships”, “unk2”], “Completed step 4, ‘Super Bowl’ unused.”], [[“unk2”, “sports.sports_championship_event.championship”, “Super Bowl”], “Completed step 5, all nodes used.”]], “filters”: [], “sort”: None, “limit”: None}

YOUR TASK:

query: {}

Before finalizing the JSON, revise the above rules and examples.

Prompt Used to Answer on CWQ

TASK
Produce every answer that can be supported by at least one evidence graph.

Guidelines

1. **Exact quoting** — copy entity / literal text exactly as it appears in the triples, if the evidence is useful.
2. **Graph independence** — treat each evidence graph separately; if different graphs justify different answers, list them all.
3. **Brief rationale** — begin each answer with a short reason such as "from graph [1]".
4. **Fallback knowledge** — if a graph is incomplete or off-topic, you may rely on your own knowledge while still obeying Rule 1.
5. **No abstention** — even with zero evidence you must think step-by-step and state a concrete answer (never respond "I don't know" or "There is no ...").

Examples

1. query: 'Which country that trades with Portugal carries the ISO alpha-3 code "ARE"?'
evidences:
`graph [1]: [('Portugal', 'location.statistical_region.places_exported_to', 'trade_rel_por_are'), ('trade_rel_por_are', 'location.imports_and_exports.exported_to', 'United Arab Emirates'), ('United Arab Emirates', 'location.country.iso_alpha_3', 'ARE')]`
`graph [2]: [('Portugal', 'location.country.currency_used', 'Euro'), ('Portugal', 'location.country.capital', 'Lisbon'), ('United Arab Emirates', 'location.country.capital', 'Abu Dhabi')]`
`graph [3]: [('Portugal', 'location.country.official_language', 'Portuguese'), ('United Arab Emirates', 'location.country.official_language', 'Arabic'), ('Portugal', 'location.country.iso_alpha_3', 'PRT')]`
answer: 'According to graph [1], the country is United Arab Emirates. Graphs [2] and [3] are not useful.'

2. query: 'The universities that have Nobel-Prize-winning physics faculty are located in which cities?'
evidences:
`graph [1]: [('University of Cambridge', 'education.educational_institution.notable_faculty', 'Peter Higgs'), ('Peter Higgs', 'awards.award_winner.awards_won', 'Nobel Prize in Physics'), ('University of Cambridge', 'location.location.containedby', 'Cambridge')]`
`graph [2]: [('Massachusetts Institute of Technology', 'education.educational_institution.notable_faculty', 'Wolfgang Ketterle'), ('Wolfgang Ketterle', 'awards.award_winner.awards_won', 'Nobel Prize in Physics'), ('Massachusetts Institute of Technology', 'location.location.containedby', 'Cambridge (MA)')]`
`graph [3]: [('University of Oxford', 'education.educational_institution.notable_faculty', 'Roger Penrose'), ('Roger Penrose', 'awards.award_winner.awards_won', 'Nobel Prize in Physics'), ('University of Oxford', 'location.location.containedby', 'Oxford')]`
answer: 'Graphs [1][2][3] show the cities Cambridge, Cambridge (MA), and Oxford.'

3. query: 'Which mountains higher than 8000 m lie in Nepal and also sit on the border with China?'
evidences:
`graph [1]: [('Mount Everest', 'location.location.country', 'Nepal'), ('Mount Everest', 'location.mountain.elevation', '8848'), ('Mount Everest', 'location.location.adjoin_s', 'China')]`

graph [2]: [('Lhotse', 'location.location.country', 'Nepal'), ('Lhotse', 'location.mountain.elevation', '8516'), ('Lhotse', 'location.location.adjoin_s', 'China')]
graph [3]: [('Kangchenjunga', 'location.location.country', 'Nepal'), ('Kangchenjunga', 'location.mountain.elevation', '8586'), ('Kangchenjunga', 'location.location.adjoin_s', 'India')]
answer: 'According to graphs [1][2], the mountains are Mount Everest and Lhotse. Graph [3] is not useful.'

Your turn
query: 'What is the true name of Vito Corleone and also played Deism?'
evidences:

Prompt Used to Generate Pseudo Query Graphs on WebQSP

You are an expert Freebase question planner.
Return a STRICT JSON with these keys in EXACT order:

1. generation_plan (step-by-step plan to generate the graph based on the divisions)
2. nodes (dict of {{node: expected type of node}})
3. edges (list of [[source, predicate, target], description what you have completed])

RULES (MUST FOLLOW ALL):

1. **Single Connected Component:**
   - All nodes must form one connected graph. No disconnected parts.
   - E.g., ["John", "likes", "unk1"], ["unk2", "likes", "unk3"] is INVALID since they are disconnected.
   - E.g., ["John", "likes", "unk1"], ["unk1", "likes", "unk2"] is VALID since they are connected.
2. **Known Entities Not Directly Connected:**
   - Two known entities cannot appear together in one edge. Use an 'unk' node in between.
3. **Node-Edge Consistency:**
   - Every node mentioned in 'edges' must be defined in 'nodes',
   - and every node in 'nodes' must appear in at least one edge.
   - After enumerating all edges, ensure all nodes are used.
4. **Unknown Nodes:**
   - Must be named 'unk1', 'unk2', etc.
   - Known entities stay exactly as in the query (no renaming).
5. **Output only the final JSON (no extra text).**

1. query: "who played michael myers in halloween 4?"

{'generation_plan': ("Step 0: Identify the known entity 'Halloween 4: The Return of Michael Myers' from the query. Step 1. Find the starring actor of 'Halloween 4: The Return of Michael Myers'", "Step 2. Introduce 'unk1' as the actor who starred in 'Halloween 4: The Return of Michael Myers'. Step 3: We link the known entity 'Halloween 4: The Return of Michael Myers' to 'unk1' using the edge film.film.starring||film.performance.actor. The graph is connected and all unknown nodes (unk1) and known entities (Halloween 4: The Return of Michael Myers) must be appeared in the edges ensuring NO isolated nodes."), 'nodes': {'unk1': 'people.person', 'Halloween 4: The Return of Michael Myers': 'film.film'}, 'edges': [['Halloween 4: The Return of Michael Myers', 'film.film.starring||film.performance.actor', 'unk1'], 'I have completed step 0 to 3. Among all nodes (unk1, Halloween 4: The Return of Michael Myers), I have used all nodes (unk1, Halloween 4: The Return of Michael Myers) in the edges.']]}

2. query: "in which country is mount everest found?"

{'generation_plan': ("Step 0: Identify the known entity 'Mount Everest' from the query. Step 1. Find the country where 'Mount Everest' is located", "Step 2. Introduce 'unk1' as the country where 'Mount Everest' is located. Step 3: We link the known entity 'Mount Everest' to 'unk1' using the edge location.location.partially_containedby. The graph is connected and all unknown nodes (unk1) and known entities (Mount Everest) must be appeared in the edges ensuring NO isolated nodes."), 'nodes': {'unk1': 'location.country', 'Mount Everest': 'location.location'}, 'edges': [['Mount Everest', 'location.location.partially_containedby', 'unk1'], 'I have completed step 0 to 3. Among all nodes (unk1, Mount Everest), I have used all nodes (unk1, Mount Everest) in the edges.']]}

3. query: "who is marilyn monroe and why is she famous?"

{'generation_plan': ("Step 0: Identify the known entity 'Marilyn Monroe' from the query. Step 1. To find who she is, find profession of 'Marilyn Monroe'", "Step 2. Introduce 'unk1' as the profession of 'Marilyn Monroe'. Step 3: We link the known entity 'Marilyn Monroe' to 'unk1' using the edge people.person.profession. The graph is connected and all unknown nodes (unk1) and known entities (Marilyn Monroe) must be appeared in the edges ensuring NO isolated nodes."), 'nodes': {'unk1': 'people.profession', 'Marilyn Monroe': 'people.person'}, 'edges': [['Marilyn Monroe', 'people.person.profession', 'unk1'], 'I have completed step 0 to 3. Among all nodes (unk1, Marilyn Monroe), I have used all nodes (unk1, Marilyn Monroe) in the edges.']]}

YOUR TASK:
query: "what are the mountains in northern italy called?"
Before finalizing the JSON, revise above rules and examples.

Prompt Used to Answer on WebQSP

TASK
Produce every answer that can be supported by at least one evidence graph.

Guidelines

1. **Exact quoting** — copy entity / literal text exactly as it appears in the triples, if the evidence is useful.
2. **Graph independence** — treat each evidence graph separately; if different graphs justify different answers, list them all.
3. **Brief rationale** — begin each answer with a short reason such as "from graph [1]".
4. **Fallback knowledge** — if a graph is incomplete or off-topic, you may rely on your own knowledge while still obeying Rule 1.
5. **No abstention** — even with zero evidence you must think step-by-step and state a concrete answer (never respond "I don't know" or "There is no ...").

Examples

1. query: 'Who voiced the character Woody in the film Toy Story?'
evidences: `graph [1]: [('Toy Story', 'film.film.starring.film.performance.character', 'Woody'), ('Toy Story', 'film.film.starring.film.performance.actor', 'Tom Hanks'), ('Toy Story', 'film.film.initial_release_date', '1995')]`
`graph [2]: [('Toy Story', 'film.film.starring.film.performance.character', 'Buzz Lightyear'), ('Toy Story', 'film.film.starring.film.performance.actor', 'Tim Allen'), ('Toy Story', 'film.film.initial_release_date', '1995')]`
`graph [3]: [('Toy Story', 'film.film.subjects', 'Friendship'), ('Friendship', 'common.topic.alias', 'friendship'), ('Toy Story', 'film.film.initial_release_date', '1995')]`
answer: 'According to graph [1], the actor is Tom Hanks. Graphs [2] and [3] are not useful.'

2. query: 'In which country is the city of Kyoto located?'
evidences: `graph [1]: [('Kyoto', 'location.location.country', 'Japan'), ('Japan', 'location.country.capital', 'Tokyo'), ('Kyoto', 'location.location.population', '1475000')]`
`graph [2]: [('Kyoto', 'location.location.containedby', 'Kyoto Prefecture'), ('Kyoto Prefecture', 'location.location.country', 'Japan'), ('Kyoto', 'location.location.area', '827')]`
`graph [3]: [('Kyoto', 'location.location.time_zone', 'Asia/Tokyo'), ('Asia/Tokyo', 'location.time_zone.offset', 'UTC+09'), ('Kyoto', 'location.geocode.latitude', '35.0116')]`
answer: 'Graphs [1] and [2] confirm the country is Japan. Graph [3] is not useful.'

3. query: 'What is the birthplace of Albert Einstein?'
evidences: `graph [1]: [('Albert Einstein', 'people.person.place_of_birth', 'Ulm'), ('Ulm', 'location.location.country', 'Germany'), ('Albert Einstein', 'people.person.date_of_birth', '1879-03-14')]`
`graph [2]: [('Albert Einstein', 'people.person.place_of_death', 'Princeton'), ('Princeton', 'location.location.country', 'United States'), ('Albert Einstein', 'people.person.date_of_death', '1955-04-18')]`
`graph [3]: [('Albert Einstein', 'people.person.profession', 'Physicist'), ('Physicist', 'common.topic.notable_types', 'Profession'), ('Albert Einstein', 'awards.award_winner.awards_won', 'Nobel Prize in Physics')]`
answer: 'According to graph [1], his birthplace is Ulm. Graphs [2] and [3] are not useful.'

Your turn
query: {}
evidences:

Prompt Used to Generate Pseudo Query Graphs on GrailQA

You are an expert Freebase question planner.
Return a STRICT JSON with these keys in EXACT order:

1. generation_plan (step-by-step plan to generate the graph based on the divisions)
2. nodes (dict of {{node: expected type of node}})
3. edges (list of [[source, predicate, target], description what you have completed])

RULES (MUST FOLLOW ALL):

1. **Single Connected Component**:
   - All nodes must form one connected graph. No disconnected parts.
   - E.g., ["John", "likes", "unk1"], ["unk2", "likes", "unk3"] is INVALID since they are disconnected.
   - E.g., ["John", "likes", "unk1"], ["unk1", "likes", "unk2"] is VALID since they are connected.
2. **Known Entities Not Directly Connected**:
   - Two known entities cannot appear together in one edge. Use an 'unk' node in between.
3. **Node-Edge Consistency**:
   - Every node mentioned in 'edges' must be defined in 'nodes',
   - and every node in 'nodes' must appear in at least one edge.
   - After enumerating all edges, ensure all nodes are used.
4. **Unknown Nodes**:
   - Must be named 'unk1', 'unk2', etc.
   - Known entities stay exactly as in the query (no renaming).
5. **Output only the final JSON (no extra text).**

1. query: "the character power for knuckles the echidna and lindsey mcdonald is what?"

{'generation_plan': ("Step 0: Identify the known entity 'Knuckles the Echidna' from the query. Step 1. Find the character power for 'Knuckles the Echidna'", "Step 2. Introduce 'unk1' as the character power for 'Knuckles the Echidna'. Step 3: We link 'Knuckles the Echidna' to 'unk1' using the edge fictional_universe.fictional_character.powers_or_abilities. The graph is connected and all unknown nodes (unk1) and known entities (Knuckles the Echidna) must be appeared in the edges ensuring NO isolated nodes."), 'nodes': {'unk1': 'fictional_universe.fictional_character.powers_or_abilities', 'Knuckles the Echidna': 'fictional_universe.fictional_character'}, 'edges': [['Knuckles the Echidna', 'fictional_universe.fictional_character.powers_or_abilities', 'unk1'], 'I have completed step 0 to 3. Among all nodes (unk1, Knuckles the Echidna), I have used all nodes (unk1, Knuckles the Echidna) in the edges.']]}

2. query: "spongebob squarepants: lights, camera, pants! was published by who?"

{'generation_plan': ("Step 0: Identify the known entity 'SpongeBob SquarePants: Lights, Camera, Pants!' from the query. Step 1. Find the publisher of 'SpongeBob SquarePants: Lights, Camera, Pants!'", "Step 2. Introduce 'unk1' as the publisher of 'SpongeBob SquarePants: Lights, Camera, Pants!'. Step 3: We link 'unk1' to the known entity 'SpongeBob SquarePants: Lights, Camera, Pants!' using the edge cvg.cvg_publisher.games_published. The graph is connected and all unknown nodes (unk1) and known entities (SpongeBob SquarePants: Lights, Camera, Pants!) must be appeared in the edges ensuring NO isolated nodes."), 'nodes': {'unk1': 'cvp.cvg_publisher', 'SpongeBob SquarePants: Lights, Camera, Pants!': 'cvg.cvg_video_game'}, 'edges': [['Mount Everest', 'location.location.partially_containedby', 'unk1'], 'I have completed step 0 to 3.

Among all nodes (unk1, Mount Everest), I have used all nodes (unk1, Mount Everest) in the edges.']]}

3. query: "what unit of mass is associated with the measurement system of picofarad?"
{'generation_plan': ("Step 0: Identify the known entity 'Farad' from the query. Step 1: Find the unit of mass associated with the measurement system of 'Farad'", "Step 2: Introduce 'unk1' as the unit of mass associated with the measurement system of 'Farad'. Step 3: We link 'unk1' to the known entity 'Farad' using the edge measurement_unit.capacitance_unit.measurement_system. The graph is connected and all unknown nodes (unk1) and known entities (Farad) must be appeared in the edges ensuring NO isolated nodes."), 'nodes': {'unk1': 'measurement_unit.capacitance_unit.measurement_system', 'Farad': 'measurement_unit.capacitance_unit'}, 'edges': [['Farad', 'measurement_unit.capacitance_unit.measurement_system', 'unk1'], 'I have completed step 0 to 3. Among all nodes (unk1, Farad), I have used all nodes (unk1, Farad) in the edges.']]}

YOUR TASK:
query: {}
Before finalizing the JSON, revise above rules and examples.

Prompt Used to Answer on GrailQA

TASK
Produce every answer that can be supported by at least one evidence graph.

Guidelines

1. **Exact quoting** — copy entity / literal text exactly as it appears in the triples, if the evidence is useful.
2. **Graph independence** — treat each evidence graph separately; if different graphs justify different answers, list them all.
3. **Brief rationale** — begin each answer with a short reason such as "from graph [1]".
4. **Fallback knowledge** — if a graph is incomplete or off-topic, you may rely on your own knowledge while still obeying Rule 1.
5. **No abstention** — even with zero evidence you must think step-by-step and state a concrete answer (never respond "I don't know" or "There is no ...").

Examples

1. query: 'Which character power is shared by Spider-Man and Batman?'
evidences:
graph [1]: [('Spider-Man', 'fictional_universe.fictional_character.powers_or_abilities', 'Superhuman strength'), ('Superhuman strength', 'fictional_universe.character_powers.characters_with_this_ability', 'Batman'), ('Superhuman strength', 'type.object.type', 'fictional_universe.character_powers')]
graph [2]: [('Spider-Man', 'fictional_universe.fictional_character.powers_or_abilities', 'Wall-crawling'), ('Wall-crawling', 'fictional_universe.character_powers.characters_with_this_ability', 'Spider-Man'), ('Wall-crawling', 'type.object.type', 'fictional_universe.character_powers')]
graph [3]: [('Batman', 'fictional_universe.fictional_character.powers_or_abilities', 'Martial arts expert'), ('Martial arts expert', 'fictional_universe.character_powers.characters_with_this_ability', 'Batman'), ('Martial arts expert', 'type.object.type', 'fictional_universe.character_powers')]
answer: 'Graph [1] shows the shared power is Superhuman strength. Graphs [2] and [3] are not useful.'

2. query: 'What mass units belong to the same measurement system as the tesla?'
evidences:
graph [1]: [('tesla', 'measurement_unit.magnetic_flux_density_unit.measurement_system', 'SI'), ('kilogram', 'measurement_unit.mass_unit.measurement_system', 'SI'), ('kilogram', 'type.object.type', 'measurement_unit.mass_unit')]
graph [2]: [('tesla', 'measurement_unit.magnetic_flux_density_unit.measurement_system', 'SI'), ('gram', 'measurement_unit.mass_unit.measurement_system', 'SI'), ('gram', 'type.object.type', 'measurement_unit.mass_unit')]
graph [3]: [('tesla', 'measurement_unit.magnetic_flux_density_unit.measurement_system', 'SI'), ('metre', 'measurement_unit.length_unit.measurement_system', 'SI'), ('metre', 'type.object.type', 'measurement_unit.length_unit')]
answer: 'Graphs [1] and [2] show the units kilogram and gram. Graph [3] is not useful.'

3. query: 'Which video-game publisher released both Halo: Combat Evolved and Forza Motorsport?'
evidences:
graph [1]: [('Microsoft Game Studios', 'cvg.cvg_publisher.games_published', 'Halo: Combat Evolved'), ('Microsoft Game Studios', 'cvg.cvg_publisher.games_published', 'Forza Motorsport'),

('Microsoft Game Studios', 'type.object.type', 'cvg.cvg_publisher')]
graph [2]: [('Halo: Combat Evolved', 'cvg.cvg_version.publisher', 'Microsoft Game Studios'), ('Forza Motorsport', 'cvg.cvg_version.publisher', 'Microsoft Game Studios'), ('Microsoft Game Studios', 'type.object.type', 'cvg.cvg_publisher')]
graph [3]: [('Super Mario Bros', 'cvg.cvg_version.publisher', 'Nintendo'), ('Forza Motorsport', 'cvg.cvg_version.publisher', 'Microsoft Game Studios'), ('Microsoft Game Studios', 'type.object.type', 'cvg.cvg_publisher')]
answer: 'Graphs [1][2][3] agree the publisher is Microsoft Game Studios.'

Your turn
query: 'what content type is on the album the garage tape dayz 78–81?'
evidences: