GraphWalker: Agentic Knowledge Graph Question Answering via Synthetic Trajectory Curriculum

Shuwen Xu[1*], **Yao Xu**[1,2*], **Jiaxiang Liu**[1,2], **Chenhao Yuan**[1], **Wenshuo Peng**[3], **Jun Zhao**[1,2], **Kang Liu**[1,2†]
[1] School of Artificial Intelligence, University of Chinese Academy of Sciences
[2] The Key Laboratory of Cognition and Decision Intelligence for Complex Systems, Institute of Automation, Chinese Academy of Sciences
[3] Department of Electronic Engineering, Tsinghua University
xushuwen23@mails.ucas.ac.cn, {yao.xu,jzhao,kliu}@nlpr.ia.ac.cn

Abstract

Agentic knowledge graph question answering (KGQA) requires an agent to iteratively interact with knowledge graphs (KGs), posing challenges in both training data scarcity and reasoning generalization. Specifically, existing approaches often restrict agent exploration: prompting-based methods lack autonomous navigation training, while current training pipelines usually confine reasoning to predefined trajectories. To this end, this paper proposes ***GraphWalker***, a novel agentic KGQA framework that addresses these challenges through *Automated Trajectory Synthesis* and *Stage-wise Fine-tuning*. GraphWalker adopts a two-stage SFT training paradigm: First, the agent is trained on structurally diverse trajectories synthesized from constrained random-walk paths, establishing a broad exploration prior over the KG; Second, the agent is further fine-tuned on a small set of expert trajectories to develop reflection and error recovery capabilities. Extensive experiments demonstrate that our stage-wise SFT paradigm unlocks a higher performance ceiling for a lightweight reinforcement learning (RL) stage, enabling GraphWalker to achieve state-of-the-art performance on CWQ and WebQSP. Additional results on GrailQA and our constructed GraphWalkerBench confirm that GraphWalker enhances generalization to out-of-distribution reasoning paths. The code is publicly available at https://github.com/XuShuwenn/GraphWalker

1 Introduction

Knowledge graph question answering (KGQA) aims to answer natural language questions by reasoning over large-scale structured Knowledge Graphs (KGs) (Jang et al., 2017), where world knowledge is encoded as (subject, predicate, object) triples. Answering complex questions over KGs demands multi-hop reasoning, systematic path planning, and structural awareness of the underlying graph topology (Xiong et al., 2026b). The advent of Large Language Models (LLMs) (Brown et al., 2020) has opened new opportunities for KGQA due to their strong language understanding and reasoning capabilities (Petroni et al., 2019).

As illustrated in Figure 1, traditional KGQA methods follow two main paradigms: **Information Retrieval (IR) methods** (Zhang et al., 2022; Dong et al., 2023; Li et al., 2024), which retrieve relevant subgraphs and extract answers from the gathered evidence, and **Semantic Parsing (SP) methods** (Berant et al., 2013; Yih et al., 2015; Li et al., 2023; Luo et al., 2024a), which translate questions into executable logical forms (e.g., SPARQL) to query KGs directly. However, both paradigms confine reasoning within a static, pre-defined scope, which severely limits their *generalization to unseen reasoning structures* in real-world KGs (Auer et al., 2007; Bollacker et al., 2008; Xiong et al., 2025).

[*] Equal contribution. [†]Corresponding author.

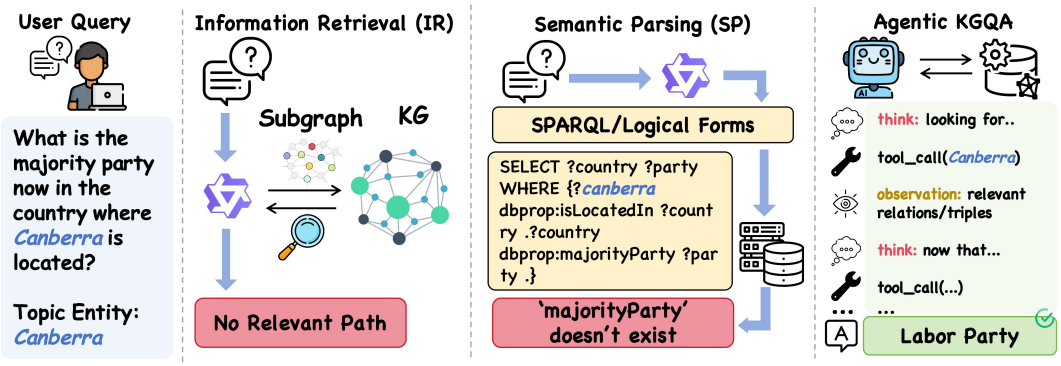


Figure 1: Comparison of three KGQA paradigms.

Type	Method	Global KG	Autonomous Interaction	RL Optimization
Prompting	ToG (Sun et al., 2024)	✓	✓	✗
	GoG (Xu et al., 2024)	✓	✓	✗
Training	RoG (Luo et al., 2024b)	✗	✗	✗
	KBQA-o1 (Luo et al., 2025)	✓	✓	✗
	KG-Agent (Jiang et al., 2025)	✓	✓	✗
	KG-R1 (Song et al., 2025)	✗	✓	✓
Ours	**GraphWalker**	✓	✓	✓

Table 1: Comparison of GraphWalker with existing agentic KGQA methods across three key properties: ***Global KG*** (operating on global KGs rather than pre-extracted subgraphs), ***Autonomous Interaction*** (interacting autonomously rather than via pre-defined workflows), and ***RL Optimization*** (incorporating reinforcement learning to optimize reasoning policies).

To overcome this limitation, recent **Agentic KGQA methods** allow LLMs to interact with KGs via successive tool calls, dynamically refining their reasoning based on real-time feedback (Yao et al., 2023; Sun et al., 2024). However, as shown in Table 1, existing approaches still exhibit notable limitations. Prompting-based and SFT-based methods (Sun et al., 2024; Xu et al., 2024; Luo et al., 2024b; Jiang et al., 2025) often hit performance ceilings due to *insufficient environmental adaptation*. Meanwhile, existing RL-based methods (Song et al., 2025) suffer from a critical exploration bottleneck: they apply policy optimization on pre-extracted subgraphs without a comprehensive SFT prior. By confining their training to simplified environments, they severely restrict the agent's search space, hindering effective navigation of the vast, noisy *global KG* during inference. Most importantly, recent studies (Guo et al., 2025; Zhang et al., 2026; Xiong et al., 2026a) consistently demonstrate that RL requires a structurally diverse SFT foundation; without a broad exploration prior, policy optimization on global KGs proves ineffective. This raises a central question: ***How can an agent build a robust exploration foundation to push the boundaries of its reasoning capacity?***

To address these issues, this paper proposes *GraphWalker*, a framework that autonomously synthesizes interaction trajectories for *Stage-Wise Fine-tuning*, followed by lightweight RL with a sparse exact-match (EM) reward. Specifically, the two SFT stages directly target the aforementioned exploration and robustness bottlenecks. First, we conduct *Constrained Random Walk (CRW)* on the KG and synthesize *GraphSynth*, a corpus of 15k diverse trajectories that establishes a broad exploration prior. Second, we construct *GraphRoll*, a dataset of 6k distilled expert trajectories that equips the agent with reflection and error recovery capabilities. Importantly, we demonstrate that the first stage serves as a *mid-training exploration scaffold*, which unlocks a higher performance ceiling during subsequent RL.

Our main contributions are summarized as follows:

- We identify the lack of an exploration prior as the core bottleneck in agentic KGQA generalization, and propose GraphWalker to address this limitation through two com-

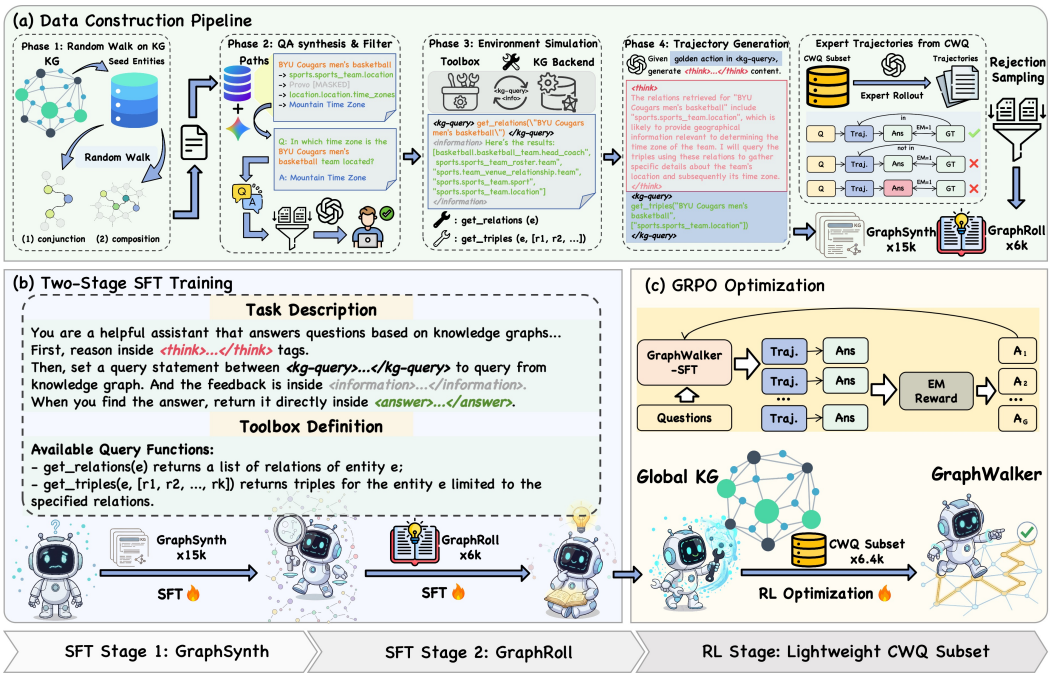


Figure 2: **Overview of our proposed GraphWalker. (a) Data Construction:** a multi-phase pipeline for GraphSynth-15k and outcome-based rejection sampling for GraphRoll-6k. **(b) Two-Stage SFT:** Stage 1 establishes a broad exploration prior, while Stage 2 instills reflection and error recovery. **(c) RL Optimization:** GRPO with a simple exact-match reward unlocks a higher performance ceiling enabled by the two-stage SFT prior.

plementary datasets: *GraphSynth*, 15k autonomously constructed, structurally diverse interaction trajectories, and *GraphRoll*, 6k curated high-quality expert trajectories.

- We introduce a *Stage-Wise SFT curriculum* to build a robust exploration foundation. By sequentially establishing a broad search prior and instilling error recovery, our method enables the agent to navigate vast, noisy KGs, unlocking a higher performance ceiling for subsequent RL optimization.
- Extensive experiments demonstrate that GraphWalker achieves SOTA performance on WebQSP and CWQ. Strong zero-shot results on GrailQA and our GraphWalkerBench confirm its superior robustness and generalization to unseen reasoning structures.

2 GraphWalker Framework

In this section, we present the GraphWalker framework, which consists of three components: the agentic environment, the data construction pipeline, and the training procedure. Figure 2 provides an overview of the full data construction and training pipeline.

2.1 Agentic Initialization

Environment. Following prior agentic KGQA systems (Sun et al., 2024; Jiang et al., 2025), we formalize the KG as an executable environment backed by a *SPARQL* endpoint. Unlike approaches that operate on pre-defined subgraphs, the agent interacts directly with the global KG in real time during both training and inference. This design exposes the agent to the full structural complexity of real-world KGs, including large branching factors and long-tail relations, which encourages more robust reasoning strategies.

Toolbox. To interact with this environment, we equip the agent with two primitive tools: (1) **get_relations**(e) returns all relations incident to entity e, allowing the agent to survey the local neighborhood before selecting a retrieval direction. (2) **get_triples**$(e, \mathcal{R}')$ returns

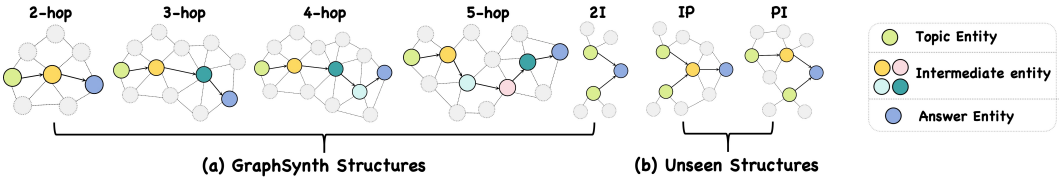


Figure 3: Representative interaction paths generated by constrained random walks.

all triples whose head or tail is e and whose relation belongs to a subset $\mathcal{R}' \subseteq \mathcal{R}_{\{e\}}$ selected from the preceding **get_relations** observation, enabling targeted and efficient KG traversal.

Agent Loop. At each turn $t \leq T$, where T is the maximum number of interaction turns, the agent is conditioned on the current state $s_t = (q, \tau_{t-1})$, where q is the input question and $\tau_{t-1} = \{(c_i, a_i, o_i)\}_{i=1}^{t-1}$ is the interaction history up to the preceding turn. Here, c_i denotes the model's internal reasoning process at turn i, a_i the executed action, and o_i the resulting observation. Given s_t, the agent produces a response consisting of two parts: (1) an internal reasoning process c_t, enclosed in `<think>...</think>`, which reflects on the current state and determines the next action; and (2) an action a_t, which is either a KG retrieval query enclosed in `<kg-query>...</kg-query>`, or a final answer enclosed in `<answer>...</answer>` when sufficient evidence has been gathered. The executor processes a_t on the KG $\mathcal{G}$ and returns the retrieval result as an observation o_t, enclosed in `<information>...</information>` and appended to the trajectory. The full turn record (c_t, a_t, o_t) is then appended to τ_t.

2.2 Data Construction

In this section, we describe the construction of two complementary training corpora, **GraphSynth-15k** and **GraphRoll-6k**. GraphSynth-15k focuses on broad structural coverage for exploration, while GraphRoll-6k emphasizes reflection and error recovery.

2.2.1 GraphSynth: Diverse Trajectory Synthesis

We first construct GraphSynth, a 15k corpus of KG-grounded interaction trajectories, using a four-phase pipeline designed to promote structural diversity, incorporate weak supervision, and simulate realistic interaction.

Phase 1: Constrained Random Walk on KG.

We first extract the predicate set $\mathcal{P}$ and the seed entity set $\mathcal{E}$ from the SPARQL logical forms of the CWQ training split. Despite using seen atomic relations, our CRW systematically recombines them into unobserved topological structures to drive compositional generalization. Starting from a seed entity $e_0 \in \mathcal{E}$, we perform constrained random walks over relations in $\mathcal{P}$ to generate structurally diverse reasoning paths. As shown in Figure 3(a), we introduce two complementary families of reasoning structures. **Composition chains** span **2–5 hops**, capturing sequential multi-hop exploration patterns. **Conjunction graphs** (denoted as **2I** in Figure 3(a)) originate from two distinct topic entities and converge on a shared target entity. Formally, each transition samples a relation–node pair subject to the predicate constraint:

$$r_t \in \mathcal{P}, \qquad e_{t+1} \in \mathcal{N}(e_t, r_t), \qquad d_{\min} \leq |\mathcal{N}(e_t, r_t)| \leq d_{\max}, \tag{1}$$

where $\mathcal{N}(e_t, r_t)$ denotes the set of neighbors reachable from e_t via relation r_t, and the cardinality constraint $d_{\min} \leq |\mathcal{N}(e_t, r_t)| \leq d_{\max}$ ensures that sampled relations are neither trivially unique nor excessively branching, thereby promoting structural diversity while avoiding degenerate or uninformative paths.

Phase 2: Question Synthesis and Semantic Filtering.

As shown in Figure 2 (Phase 2), for each reasoning path, we use `Gemini-2.5-Pro` (Comanici et al., 2025) to generate a natural language question. To prevent information leakage, we **mask intermediate entities** (as $entity_1, entity_2, \ldots$) along reasoning paths. For example, reasoning path "BYU Cougars men's basketball $\xrightarrow{\texttt{team.location}}$ $entity_1$ $\xrightarrow{\texttt{time_zones}}$

Mountain Time Zone" yields: "In which time zone is the BYU Cougars men's basketball team located?" To ensure data quality, we apply **LLM-based semantic filtering** using `GPT-4o-mini` (Hurst et al., 2024), which scores each question along three dimensions: (1) **reasoning path completeness**, (2) **question-path relevance**, and (3) **question semantic coherence**. Questions with an average score below 9 out of 10 are discarded. In case of test-set leakage, we also verify that GraphSynth contains negligible semantic overlap with all the benchmarks (details in Appendix B.2).

To further validate the model's generalization capabilities, we also construct our own evaluation set, *GraphWalkerBench*, which contains unseen reasoning structures in the GraphSynth dataset (shown in Figure 3(b), details in Appendix C).

Phase 3: Environment Feedback Simulation.

To faithfully simulate inference-time interaction, we execute each CRW path as a sequence of SPARQL queries against the full KG. We apply the same retrieval and filtering pipeline used at inference time, including **BM25** (Robertson & Zaragoza, 2009) candidate retrieval and **LLM-based relation reranking** (e.g., `Qwen2.5-7B-Instruct`). At each step t, we prompt an LLM to rerank the top-k relations and up to 5 triples per relation as the observation o_t.

Phase 4: Agent Thought Generation.

As shown in Figure 2 (Phase 4), at each step t, given the current state s_t and the corresponding oracle next action a_t, we use `GPT-4o-mini` as a rationale generator to generate the reasoning process c_t, i.e., chain-of-thought rationales that explain the selection of subsequent actions under the current context. The resulting tuples (c_t, a_t, o_t) are then assembled into complete interaction trajectories τ, forming **GraphSynth-15k**.

2.2.2 *GraphRoll: Expert Trajectory Collection*

We then construct a smaller set of expert trajectories that demonstrate reflection and error recovery. Given the complexity and diversity of CWQ, we sample a representative subset of 6k questions via stratified sampling across different question types to ensure balanced training coverage. For each question, we use `GPT-4o-mini` as the expert model to generate high-quality trajectories that involve reflecting on uninformative observations and backtracking from dead ends. We apply **outcome-based rejection sampling** based on two complementary criteria ensuring factual correctness and KG grounding: **(1) Answer Correctness:** the predicted answer must be correct ($EM = 1$). **(2) Evidence Grounding:** all predicted answers must appear in the observation history $\{o_0, \ldots, o_T\}$, ensuring every answer is grounded in the retrieved KG evidence rather than parametric knowledge. The resulting trajectories form **GraphRoll-6k**.

2.3 GraphWalker Training

We adopt a two-stage SFT cold start followed by RL training. The first stage equips the agent with general exploration skills, the second cultivates reflection and error recovery, and RL subsequently aligns long-horizon decision making. We parameterize the agent policy as an autoregressive language model π_θ. Specifically, in stage 1 and stage 2, the model is optimized using a next-token prediction objective over agentic interaction sequences, while treating `<information>` blocks as environmental context rather than learning targets:

$$\mathcal{L}(\theta) = -\sum_{t \in \mathcal{I}_{\text{agent}}} \log P_\theta(x_t \mid x_{<t}), \tag{2}$$

where $\mathcal{I}_{\text{agent}}$ indexes tokens in the `<think>`, `<kg-query>`, and `<answer>` blocks.

SFT Stage 1: Learning to Reason and Explore on Diverse KGs.

In stage 1, we first train the agent on **GraphSynth-15k**. This stage aims to internalize robust navigation behaviors and reliable tool-invocation patterns under realistic retrieval noise, while treating KG retrieval results as environmental observations.

SFT Stage 2: Learning to Reflect and Recover from Errors.

In stage 2, we fine-tune the model on **GraphRoll-6k**. Unlike stage 1, which emphasizes broad exploration, the expert trajectories in GraphRoll-6k contain explicit instances of error detection and reasoning recovery, guiding the model to develop more deliberate and self-corrective interaction strategies.

RL Stage: Unlocking the Performance Ceiling

Building on the two-stage SFT cold-start model, we apply RL to push the agent's performance beyond the SFT ceiling through autonomous exploration. We train the model on a uniformly sampled CWQ subset using Group Relative Policy Optimization (GRPO) (Guo et al., 2025) with a sparse, trajectory-level exact match reward:

$$R(\tau) = s_{\mathrm{EM}}(\tau) \in \{0, 1\}. \tag{3}$$

Since our two-stage SFT already establishes an exploration and reflection prior, this sparse reward is sufficient to drive further policy improvement without dense intermediate supervision. The GRPO objective function is:

$$\mathcal{J}_{\mathrm{GRPO}}(\theta) = \mathbb{E}_{q,\{\tau^{(i)}\}}\left[\frac{1}{N}\sum_{i=1}^{N}\frac{1}{|\tau^{(i)}|}\sum_{j=1}^{|\tau^{(i)}|}\min\Big(\rho_{i,j}\hat{A}_{i,j},\ \mathrm{clip}(\rho_{i,j}, 1-\epsilon, 1+\epsilon)\,\hat{A}_{i,j}\Big)\right] - \beta\,\mathbb{D}_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}}), \tag{4}$$

where $\rho_{i,j} = \frac{\pi_\theta(a_{i,j}|s_{i,j})}{\pi_{\mathrm{old}}(a_{i,j}|s_{i,j})}$ is the importance ratio, $\hat{A}_{i,j}$ is the group-normalized advantage, and π_{ref} is the two-stage SFT model used as the KL reference policy.

3 Experiments

In this section, we conduct comprehensive experiments and report the experimental setup, main results, and ablation analyses. We answer the following research questions (RQs):

RQ1: Does GraphWalker outperform other LLM-based KGQA methods? **RQ2:** Does GraphSynth broaden GraphWalker's search space and improve generalization? **RQ3:** How does each training stage contribute to GraphWalker's final performance? **RQ4:** How do the main components of our data construction pipeline work?

3.1 Experimental Setup

Datasets and Metrics. Following previous research (Sun et al., 2024; Luo et al., 2024b), we evaluate GraphWalker on two widely used KGQA benchmarks: CWQ (Talmor & Berant, 2018) and WebQSP (Yih et al., 2016). To ensure a fair comparison, we utilize the gold topic entities provided by the benchmarks and report standard Exact Match (EM) and $F1$ scores (Talmor & Berant, 2018). Details of the datasets are described in Appendix C.

Models and Baselines. We compare GraphWalker with seven previous prompting-based and training-based KGQA methods. To better evaluate the effectiveness of various methods on knowledge reasoning tasks, we use the full Freebase as a KG backend (*Global KG* setting). This setting is more challenging than approaches based on predefined subgraphs, as the agent must navigate the full KG. For KG-R1, we evaluate the released checkpoint in the official repository under our Global KG setting. We additionally evaluate strong proprietary LLMs, including `DeepSeek-V3.2` and `GPT-4o-mini`, under the GraphWalker agent framework. Details for all baselines are provided in Appendix D.

Implementation Details. We train and evaluate GraphWalker on three open-source backbone models: **Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct** (Qwen et al., 2024), and **Llama-3.1-8B-Instruct** (Dubey et al., 2024). For SFT data collection, we use `GPT-4o-mini` to generate trajectories for both GraphSynth and GraphRoll. For evaluation, we set the max turn number $T = 10$. Our training implementation details are shown in Appendix G.

Method	Backbone	CWQ		WebQSP	
		EM	F1	EM	F1
Vanilla LLMs					
IO Prompt	Qwen2.5-3B-Instruct	22.0	17.7	44.6	30.3
IO Prompt	Qwen2.5-7B-Instruct	25.7	20.7	50.9	33.2
IO Prompt	GPT-4o-mini	45.5	33.6	47.1	39.3
IO Prompt	DeepSeek-V3.2	50.1	43.5	63.8	55.7
Agentic KGQA Methods					
RoG	LLaMA-2-7B-Instruct	62.6	56.2	85.7	70.8
ToG	GPT-4	69.5	-	81.9	-
ToG-2.0	GPT-3.5	68.9	65.8	77.8	74.5
GoG	GPT-4	75.2	-	84.4	-
KBQA-o1	LLaMA3.1-8B-Instruct	-	-	75.8	82.1
KG-Agent	LLaMA2-7B-Instruct	72.2	69.8	83.3	81.0
†KG-R1	Qwen2.5-3B-Instruct	66.8	61.7	82.1	78.9
GraphWalker (Our Method)					
†Vanilla Agent	Qwen2.5-7B-Instruct	40.7	33.2	68.4	66.1
†Vanilla Agent	GPT-4o-mini	63.4	60.3	79.6	70.6
†Vanilla Agent	DeepSeek-V3.2	69.8	63.5	76.7	71.8
GraphWalker-7B-SFT	Qwen2.5-7B-Instruct	68.3	63.2	82.0	79.1
GraphWalker-3B-SFT-RL	Qwen2.5-3B-Instruct	70.9	65.2	83.5	81.7
GraphWalker-8B-SFT-RL	LLaMA3.1-8B-Instruct	78.5	69.6	88.2	84.5
GraphWalker-7B-SFT-RL	Qwen2.5-7B-Instruct	**79.6**	**74.2**	**91.5**	**88.6**

Table 2: Performance comparison (in percentage) of KGQA methods on CWQ and WebQSP. The best and second-best results are shown in **bold** and underlined, respectively. † denotes models evaluated under the GraphWalker interaction framework with full global KG.

3.2 Main Results (RQ1)

As shown in Table 2, GraphWalker achieves SOTA performance across all benchmarks, demonstrating its effectiveness from two perspectives:

Superiority in Complex Reasoning. GraphWalker consistently outperforms both prompting and training baselines. Notably, it surpasses the GPT-4-powered GoG by 4.4% EM on CWQ, and outperforms training-based models such as KG-Agent (+7.4% EM on CWQ) and KBQA-o1 (+15.7% EM on WebQSP). Under the more challenging Global KG setting, GraphWalker-3B-SFT-RL outperforms KG-R1 (same `Qwen2.5-3B-Instruct` backbone) by a clear margin (e.g., +4.1% EM on CWQ), suggesting that KG-R1's reliance on pre-extracted subgraphs undermines its robustness. Moreover, equipping proprietary LLMs (`GPT-4o-mini` and `DeepSeek-V3.2`) with our interaction framework yields substantial gains over their IO Prompt counterparts (+17.9% and +19.7% EM on CWQ). Impressively, GraphWalker with trained backbones outperforms these proprietary LLMs across all model sizes, demonstrating the effectiveness of our synthetic agentic curriculum and training strategy.

Cross-task Transferability. Despite being trained solely on CWQ, GraphWalker effectively cross-task transfers to WebQSP (91.5% EM), outperforming all baselines trained directly on WebQSP data. This confirms that training on CWQ's structurally diverse, high-complexity paths equips the agent with highly transferable exploration strategies.

3.3 Comparison Analysis (RQ2)

Pass@k Analysis. To validate that GraphSynth effectively broadens the agent's reasoning search space, we sample 500 questions from the CWQ test set and evaluate exploration capacity via **Pass@k** over 32 runs. As shown in Figure 4, within both the SFT-only and SFT-RL settings, removing GraphSynth (**w/o SFT Stage 1**) consistently reduces Pass@k across all k, confirming its critical role in expanding the agent's exploration capacity. In contrast, models trained with GraphSynth exhibit higher Pass@k values as k increases, indicating that our CRW synthesis establishes a broader exploration prior. Furthermore, **GraphWalker-SFT-RL** achieves the highest

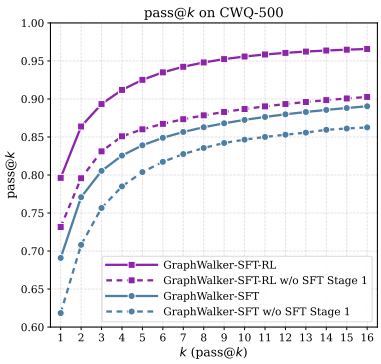


Figure 4: Pass@k comparison.

Method	GrailQA		GraphWalker Bench		$\Delta\ \overline{\text{EM}}$
	EM	F1	EM	F1	
GraphWalker-Full	**86.3**	**84.4**	**63.5**	**60.8**	—
Ablations					
w/o SFT Stage 1	80.9	78.0	55.3	52.4	−6.80
w/o SFT Stage 2	74.3	69.7	53.3	49.5	−11.10
w/o RL	79.4	76.3	49.7	47.0	−10.35

Table 3: Zero-shot performance on *GrailQA* and *GraphWalkerBench*, which involves unseen structures in *GraphSynth*.

Method	CWQ		WebQSP		$\Delta\ \overline{\text{EM}}$
	EM	F1	EM	F1	
GraphWalker-Full	**79.6**	**74.2**	**91.5**	**88.6**	—
Training Stage Ablations					
w/o SFT Stage 1	75.2	70.4	85.7	82.5	−5.10
w/o SFT Stage 2	72.3	66.3	80.8	77.5	−9.00
w/o SFT	70.6	66.0	79.8	75.6	−10.35
w/o RL	68.3	63.2	82.0	79.1	−10.40
Training Order Ablation					
Mixed-SFT+RL	70.2	64.2	80.1	75.8	−10.40

Table 4: Ablation study on *CWQ* and *WebQSP*.

Pass@k across all k, suggesting that a stronger exploration foundation enables RL to further amplify these gains.

Comparison with GPT-4o-mini Best-of-N. We further evaluate `GPT-4o-mini` with Best-of-N sampling in the same GraphWalker scaffold on the full CWQ test set. With an oracle selector, its Best-of-8 reaches 81.70% EM, whereas GraphWalker reaches 93.73%. GraphWalker also consistently achieves higher Pass@k at $k = 2, 4, 8, 16$ (86.38%, 91.19%, 94.80%, and 96.57%) than `GPT-4o-mini` (71.91%, 77.85%, 82.78%, and 86.97%). This result shows that the teacher has non-trivial trajectory diversity, while GraphWalker produces a substantially broader set of successful trajectories. Unlike oracle Best-of-N selection, our RL stage directly increases the likelihood of successful trajectories during standard inference.

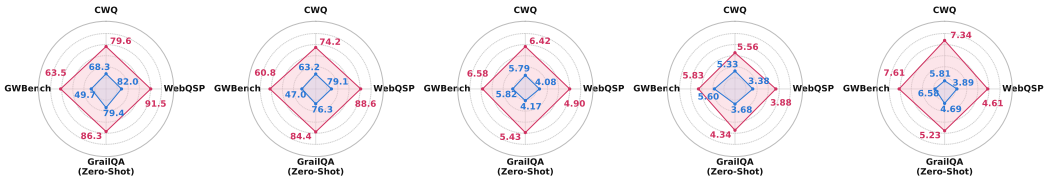
Zero-shot Generalization. To assess whether GraphSynth improves reasoning generalization, we evaluate GraphWalker and its ablated variants on two zero-shot benchmarks: the zero-shot split of *GrailQA* and *GraphWalkerBench*, a structurally diverse evaluation set involving unseen reasoning structures not present in GraphSynth (see Appendix C). As shown in Table 3, both SFT stages are crucial for out-of-distribution (OOD) generalization. Removing GraphSynth (**w/o SFT Stage 1**) significantly impairs generalization (6.80% average EM drop), confirming its essential role in broadening the agent's search space. Similarly, removing GraphRoll (**w/o SFT Stage 2**) leads to substantial performance degradation (11.10% average EM drop), as the model's ability to self-correct is significantly weakened. Furthermore, the consistent improvement of *GraphWalker-Full* over *w/o RL* confirms that RL further enhances the generalization capability established by the SFT stages.

3.4 Analysis on Each Training Stage (RQ3)

Ablation of Each Stage. To understand the contribution of each training stage, we ablate individual components from the training pipeline. As shown in Table 4, removing GraphSynth (**w/o SFT Stage 1**) causes a substantial decline (5.10% average EM drop), indicating its critical role in establishing broad exploration priors for navigating complex KG structures. Meanwhile, removing GraphRoll (**w/o SFT Stage 2**) leads to an even greater drop (9.00% average EM drop), demonstrating that GraphRoll's expert trajectories are essential for instilling reflection and self-correction capabilities. Finally, removing RL results in a severe 10.40% average EM drop, confirming that RL further consolidates the capabilities established by the SFT stages.

Importance of Stage-wise Training. To validate the proposed curriculum, we compare stage-wise training against a **Mixed** baseline combining GraphSynth and GraphRoll into a single SFT pass with identical RL training. As shown in Table 4, mixed training consistently underperforms (dropping from 79.6% to 70.2% EM on CWQ) despite using identical data, confirming that training order is critical: Stage 1 must first establish broad exploration priors before Stage 2 can effectively instill reflection and recovery capabilities.

Behavioral Changes after RL. The RL stage is not designed to learn KG navigation from scratch with sparse rewards alone. Instead, the two-stage SFT model already makes rewardable exploration and recovery trajectories reachable, and RL shifts probability mass toward



(a) EM (b) F1 (c) Avg. Turns (d) Avg. Tool-calls (e) Avg. Reflections

GraphWalker w/o RL GraphWalker

Figure 5: Behavioral comparison between *GraphWalker* and *GraphWalker w/o RL* across all benchmarks. Each axis is independently scaled to highlight per-benchmark differences. Fine-grained per-reasoning-type EM analysis is shown in Appendix E.1.

trajectories with more complete evidence acquisition and verification. To characterize this change, we compare the trajectories produced before and after RL across all evaluation sets.

As shown in Figure 5, RL leads to higher EM and F1 scores across all benchmarks, with longer exploration trajectories (Avg. Turns) and more tool calls, while reflection and recovery behaviors also become more frequent. These changes suggest that the performance gain comes primarily from more complete evidence acquisition and verification rather than merely improved tool syntax or earlier stopping. Fine-grained per-type analysis (Appendix E.1) further shows that the largest gain occurs on IP (+32.05 EM), confirming that RL especially strengthens evidence acquisition on structurally challenging queries.

3.5 Ablation Study (RQ4)

To better understand the internal mechanisms of GraphWalker, we conduct ablation experiments on key components of our pipeline.

Impact of Quality Filtering. As shown in Figure 6, removing the *quality filter* from the GraphWalker-SFT model causes substantial performance drops on both datasets: EM drops from 68.3% to 46.2% on CWQ and from 82.0% to 55.6% on WebQSP, confirming that filtering out low-quality questions is critical to the overall data pipeline.

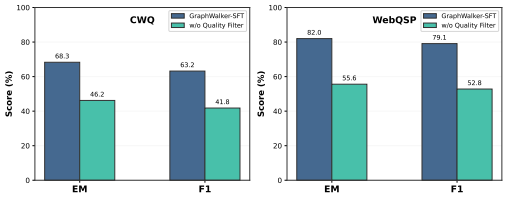


Figure 6: Ablation study on the *quality filter*.

Impact of Data Scale.

We investigate the effect of data scale on each training stage separately. For Stage 1, we vary the size of GraphSynth across $\{5k, 10k, 15k\}$ samples; for Stage 2, we vary GraphRoll across $\{0.5k, 1k, 2k, 3k, 6k\}$ samples. Here, *Retrieval Rate* measures the proportion of test cases where the gold answer appears in the agent's retrieved observations, reflecting the agent's ability to find relevant evidence through KG exploration; *Recovery Rate* measures the proportion of cases where the agent successfully corrects an erroneous `<kg-query>` step and recovers to the correct answer. As shown in Figure 7, both *EM* and the corresponding *Retrieval/Recovery Rates* improve consistently with data scale, confirming the benefit of scaling trajectory supervision. Notably, the performance gap between the full model and the ablated variants (**w/o GraphRoll** and **w/o GraphSynth**) persists across all scales, demonstrating that the two stages provide complementary and irreplaceable contributions.

4 Related work

Knowledge Graph Question Answering (KGQA). To address the limitations of unstructured text retrieval and LLM hallucination, KGs have been widely adopted in question answering tasks for their ability to provide structured world knowledge (Baek et al., 2023; Xu et al., 2023). Traditional KGQA methods fall into two categories: **Information retrieval**

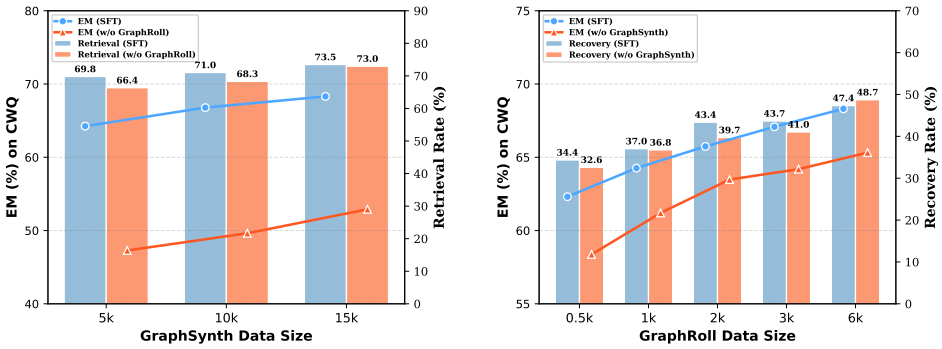


Figure 7: Impact of data scale on performance. *Left*: scaling GraphSynth data improves *EM* and *Retrieval Rate*. *Right*: scaling GraphRoll data improves *EM* and *Recovery Rate*.

(IR) methods (Zhang et al., 2022; Dong et al., 2023; Li et al., 2024) retrieve subgraphs from KGs and extract answers from the gathered evidence, while **Semantic parsing (SP)** methods (Berant et al., 2013; Yih et al., 2015; Li et al., 2023; Luo et al., 2024a) translate questions into logical forms (e.g., SPARQL) for verifiable execution. However, both are brittle under noisy or incomplete KG schemas and struggle to handle long-tail entities, motivating the shift toward agentic paradigms.

Agentic KGQA. Agentic KGQA treats LLMs as autonomous agents that interact with KGs through iterative exploration and retrieval. **Prompting-based** methods such as ToG (Sun et al., 2024) and GoG (Xu et al., 2024) perform iterative graph traversal but rely on predefined workflows or costly closed-source LLMs. **Fine-tuning** methods learn from curated training trajectories: CoT-based approaches (Wu et al., 2024) enhance step-by-step reasoning transparency, RoG (Luo et al., 2024b) grounds plans in KGs, and KG-Agent (Jiang et al., 2025) employs multi-agent reasoning but remains limited to synthesized program data. **Reinforcement learning** methods such as KG-R1 (Song et al., 2025) optimize the retrieval policy end-to-end but still rely on subgraph pre-extraction.

5 Conclusion

In this paper, we propose GraphWalker, a novel agentic KGQA framework that addresses training data scarcity and generalization bottlenecks via automated trajectory synthesis and stage-wise fine-tuning. Through our two-stage SFT curriculum, the agent first establishes a broad exploration prior over diverse KG structures, and subsequently acquires critical reflection and error recovery capabilities. Notably, this robust foundation unlocks a higher performance ceiling for subsequent reinforcement learning (RL) optimization. Extensive experiments demonstrate that GraphWalker achieves state-of-the-art performance and strong zero-shot generalization to unseen reasoning topologies, confirming its effectiveness and scalability for autonomous reasoning over global KGs.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No.U24A20335) and the independent research project of the Key Laboratory of Cognition and Decision Intelligence for Complex Systems.

References

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of ISWC*, 2007.

Jinheon Baek, Soyeong Jeong, Minki Kang, Jong Park, and Sung Hwang. Knowledge-augmented language model verification. In Houda Bouamor, Juan Pino, and Kalika Bali

(eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1720–1736, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.107. URL https://aclanthology.org/2023.emnlp-main.107.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard (eds.), *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1533–1544, Seattle, Washington, USA, 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1160.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of SIGMOD*, 2008.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and Sandhini Agarwal. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the association for computational linguistics: ACL 2024*, pp. 2318–2335, 2024.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL https://arxiv.org/abs/2507.06261.

Guanting Dong, Rumei Li, Sirui Wang, Yupeng Zhang, Yunsen Xian, and Weiran Xu. Bridging the kb-text gap: Leveraging structured knowledge-aware pre-training for kbqa, 2023. URL https://arxiv.org/abs/2308.14436.

Abhimanyu Dubey et al. The llama 3 herd of models. *ArXiv preprint*, abs/2407.21783, 2024. URL https://arxiv.org/abs/2407.21783.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, and Xu. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *ArXiv preprint*, abs/2410.21276, 2024. URL https://arxiv.org/abs/2410.21276.

Heewon Jang, Yeongtaek Oh, Seunghee Jin, Haemin Jung, Hyesoo Kong, Dokyung Lee, Dongkyu Jeon, and Wooju Kim. Kbqa: Constructing structured query graph from keyword query for semantic search. In *Proceedings of the International Conference on Electronic Commerce*, pp. 1–8, 2017.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. StructGPT: A general framework for large language model to reason over structured data. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9237–9251, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.574. URL https://aclanthology.org/2023.emnlp-main.574.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. Kg-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9505–9523, 2025.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. Few-shot in-context learning for knowledge base question answering, 2023. URL https://arxiv.org/abs/2305.01750.

Xingxuan Li, Ruochen Zhao, Yew Ken Chia, Bosheng Ding, Shafiq Joty, Soujanya Poria, and Lidong Bing. Chain-of-knowledge: Grounding large language models via dynamic knowledge adapting over heterogeneous sources. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=cPgh4gWZlz.

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. 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*, pp. 2039–2056. Association for Computational Linguistics, 2024a. doi: 10.18653/v1/2024.findings-acl.122. URL http://dx.doi.org/10.18653/v1/2024.findings-acl.122.

Haoran Luo, Yikai Guo, Qika Lin, Xiaobao Wu, Xinyu Mu, Wenhao Liu, Meina Song, Yifan Zhu, Luu Anh Tuan, et al. Kbqa-o1: Agentic knowledge base question answering with monte carlo tree search. *ArXiv preprint*, abs/2501.18922, 2025. URL https://arxiv.org/abs/2501.18922.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=ZGNWW7xZ6Q.

Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation, 2024. URL https://arxiv.org/abs/2407.10805.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2463–2473, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1250. URL https://aclanthology.org/D19-1250.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, et al. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389, 2009. doi: 10.1561/1500000019.

Jinyeop Song, Song Wang, Julian Shun, and Yada Zhu. Efficient and transferable agentic knowledge graph rag via reinforcement learning. *ArXiv preprint*, abs/2509.26383, 2025. URL https://arxiv.org/abs/2509.26383.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=nnVO1PvbTv.

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

2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1059. URL https://aclanthology.org/N18-1059.

Yike Wu, Yi Huang, Nan Hu, Yuncheng Hua, Guilin Qi, Jiaoyan Chen, and Jeff Z. Pan. Cotkr: Chain-of-thought enhanced knowledge rewriting for complex knowledge graph question answering, 2024. URL https://arxiv.org/abs/2409.19753.

Guanming Xiong, Junwei Bao, and Wen Zhao. 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)*, pp. 10561–10582. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.569. URL http://dx.doi.org/10.18653/v1/2024.acl-long.569.

Siheng Xiong, Ali Payani, Yu'an Yang, and Faramarz Fekri. Deliberate reasoning in language models as structure-aware planning with an accurate world model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31900–31931, 2025.

Siheng Xiong, Oguzhan Gungordu, Blair Johnson, James C Kerce, and Faramarz Fekri. Scaling search-augmented llm reasoning via adaptive information control. *arXiv preprint arXiv:2602.01672*, 2026a.

Siheng Xiong, Ali Payani, and Faramarz Fekri. Enhancing language model reasoning with structured multi-level modeling. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Ran Xu, Hejie Cui, Yue Yu, Xuan Kan, Wenqi Shi, Yuchen Zhuang, Wei Jin, Joyce Ho, and Carl Yang. Knowledge-infused prompting: Assessing and advancing clinical text data generation with large language models, 2023. URL https://arxiv.org/abs/2311.00287.

Yao Xu, Shizhu He, Jiabei Chen, Zihao Wang, Yangqiu Song, Hanghang Tong, Guang Liu, Jun Zhao, and Kang Liu. Generate-on-graph: Treat llm as both agent and kg for incomplete knowledge graph question answering. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 18410–18430, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=WE_vluYUL-X.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation. In *Proceedings of ACL*, 2015.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 201–206, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-2033. URL https://aclanthology.org/P16-2033.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5773–5784, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.396. URL https://aclanthology.org/2022.acl-long.396.

Xuqin Zhang, Quan He, Zhenrui Zheng, Zongzhang Zhang, Xu He, and Dong Li. Aster: Agentic scaling with tool-integrated extended reasoning, 2026. URL https://arxiv.org/abs/2602.01204.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models, 2024. URL https://arxiv.org/abs/2403.13372.

A Preliminaries

Knowledge graph. A knowledge graph (KG) $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ consists of a set of entities $\mathcal{E}$, a set of relations $\mathcal{R}$, and a triple set $\mathcal{T} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where each triple $\langle e, r, e' \rangle \in \mathcal{T}$ encodes a directed relation r from entity e to entity e'. Each entity is assigned a unique entity ID (e.g., `m.02mjmr` for *Barack Obama*) and belongs to one or more entity types such as *Country* or *Person*. Furthermore, we introduce *neighboring relations* to denote both the incoming and outgoing relations for a set of entities $\{e\}$, denoted as $\mathcal{R}_{\{e\}} = \{r \mid \langle e, r, e' \rangle \in \mathcal{G}\} \cup \{r \mid \langle e', r, e \rangle \in \mathcal{G}\}$.

Reasoning Paths. A reasoning path is an instance of a relation path z in the KG: $w_z = e_0 \xrightarrow{r_1} e_1 \xrightarrow{r_2} \cdots \xrightarrow{r_l} e_l$, where $e_i \in \mathcal{E}$ denotes the i-th entity and r_i denotes the i-th relation in z.

Example. Given a relation path $z = \texttt{marry_to} \rightarrow \texttt{father_of}$, a reasoning path instance could be: $w_z = \text{Alice} \xrightarrow{\texttt{marry_to}} \text{Bob} \xrightarrow{\texttt{father_of}} \text{Charlie}$, which denotes "Alice" is married to "Bob" and "Bob" is the father of "Charlie".

Problem Formulation. In this work, we assume that a KG is available and contains the answer entities for the given natural language question. Our objective is to develop an LLM-based agent that can autonomously infer the answer to the question based on the given KG. As it has been shown that an interface is helpful for LLMs to manipulate structured data (Jiang et al., 2023), we further assume that a toolbox can be provided to facilitate access to the information of the KG. Formally, given a natural language question q, topic entities $\mathcal{E}_q \subseteq \mathcal{E}$ obtained via entity linking, a toolbox $\mathcal{T}$, and a KG $\mathcal{G}$, we aim to develop a capable agent that interacts with the KG through iterative tool calls (Yao et al., 2023; Jiang et al., 2025; Xiong et al., 2024) to deduce the final answers $\widehat{\mathcal{A}}_q \subseteq \mathcal{E}$ for the question q.

B Data Construction Details

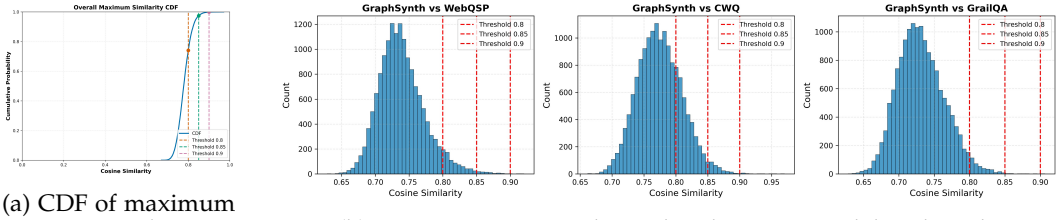
B.1 Constrained Random Walk Algorithm

Algorithm 1 Constrained Random Walk (CRW)

Input: KG executor, seed set $\mathcal{E}$, predicate set $\mathcal{P}$, target size N
Output: trajectory set $\mathcal{D}_{\text{crw}}$
$\mathcal{D}_{\text{crw}} \leftarrow \emptyset$
while $|\mathcal{D}_{\text{crw}}| < N$ **do**
  Sample seed entity $e_0 \sim \mathcal{E}$ and hop length L
  Initialize partial path $\pi \leftarrow [e_0]$
  for $t = 0$ to $L - 1$ **do**
    Retrieve outgoing candidate relations of e_t, then keep $\mathcal{R}_t = \mathcal{R}(e_t) \cap \mathcal{P}$
    if $\mathcal{R}_t = \emptyset$ **then**
      break
    end if
    Sample relation $r_t \sim \mathcal{R}_t$, then sample next node $e_{t+1} \sim \mathcal{N}(e_t, r_t)$
    Append (r_t, e_{t+1}) to π
  end for
  if π is valid **then**
    Add π to $\mathcal{D}_{\text{crw}}$
  end if
end while
return $\mathcal{D}_{\text{crw}}$

B.2 Semantic Contamination Analysis

To preclude knowledge leakage in GraphSynth-15k, we compute the maximum cosine similarity between each synthetic question and all test questions across WebQSP, CWQ, and GrailQA using BGE-M3 (Chen et al., 2024). At a strict threshold of 0.85, only 2.57% of



(a) CDF of maximum semantic similarity.

(b) Semantic cosine similarity distribution on each benchmark.

Figure 8: Data contamination analysis between GraphSynth and downstream benchmarks.

GraphSynth questions exhibit high semantic overlap with any test question, and all flagged samples are excluded from the final training corpus. The elevated overlap rate against CWQ at threshold 0.80 (23.30%) reflects expected domain proximity rather than leakage, as CWQ encompasses the most diverse relations and broadest domain coverage among the three benchmarks. Table 5 and Figure 8a present the full results.

Threshold	Overall	CWQ	WebQSP	GrailQA
0.80	25.97% (3960)	23.30% (3553)	5.27% (804)	2.67% (407)
0.85	2.57% (392)	2.13% (325)	0.56% (85)	0.09% (13)
0.90	0.15% (23)	0.12% (18)	0.03% (4)	0.01% (1)

Table 5: Contamination rates at three similarity thresholds (BGE-M3).

B.3 Statistics of GraphSynth

Structure Type	Composition				Conjunction	Total
	2-hop	3-hop	4-hop	5-hop	2I	
Count	1,982	4,920	1,778	576	5,599	14,855
Ratio (%)	13.34	33.12	11.97	3.88	37.69	100.00

Table 6: Structure distribution of the GraphSynth-15k dataset.

In this section, we characterize GraphSynth-15k along two dimensions to validate its suitability as a training corpus. Structurally, as summarized in Table 6, GraphSynth spans five reasoning structure types, covering multi-hop compositions (2- to 5-hop) and conjunctions (2I), with 3-hop and 2I patterns accounting for over 70% of samples while longer chains ensure exposure to complex reasoning patterns. Semantically, the synthesized questions are grounded in Freebase relations drawn from diverse topical domains, achieving broad coverage of real-world entity types and relation schemas. As visualized in Figure 9, the question embeddings of GraphSynth spread across a wide region of the semantic space, overlapping substantially with both CWQ and WebQSP without concentrating around any single benchmark cluster, confirming that the corpus is neither narrowly specialized nor artificially constrained to a particular domain. Together, these properties ensure that GraphSynth provides structurally and semantically diverse supervision for Stage 1 training.

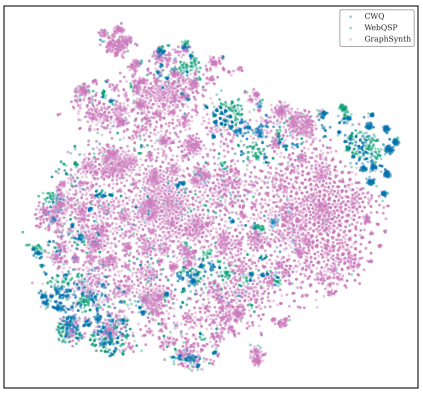


Figure 9: t-SNE visualization of GraphSynth, CWQ, and WebQSP question embeddings, illustrating the broad semantic coverage of the synthesized corpus.

B.4 Structural Contamination Analysis

To complement question-level semantic filtering, we compare GraphSynth relation paths with the ground-truth reasoning paths of all 3,531 CWQ test examples. A test example is considered covered if any of its ground-truth relation paths exactly matches a GraphSynth path. This occurs for only 85 examples (2.41%), indicating low exact structural overlap.

B.5 Data Construction Cost

Table 7 reports the average per-sample API usage and estimated cost for constructing GraphSynth and GraphRoll using `GPT-4o-mini`. All values are averaged across samples in each dataset.

Dataset	API Calls	Input Tokens	Output Tokens	Cost
GraphSynth	6.673	11,533.56	796.53	**$0.0110**
GraphRoll	5.680	9,296.93	553.87	**$0.0086**

Table 7: Average per-sample API usage and cost for GraphSynth and GraphRoll data construction.

C Datasets

In this paper, we conduct experiments on three widely used KGQA datasets, and we also construct our own GraphWalkerBench to further study the agent's capability to generalize to unseen structures:

WebQSP. WebQSP is a foundational KGQA benchmark designed to evaluate a model's ability to answer simple, fact-based questions that typically require retrieving a single fact from the knowledge graph. The dataset provides full SPARQL queries for its questions, which are executed against Freebase.

CWQ. CWQ extends the complexity beyond WebQSP by introducing questions that require compositional reasoning. These questions often involve multiple constraints, conjunctions, and superlatives, necessitating multi-hop or multi-relation inference paths on the Freebase knowledge graph. The dataset is annotated with complex SPARQL queries that reflect these reasoning structures.

GrailQA. GrailQA is a large-scale dataset built on Freebase that is designed to evaluate the generalization capabilities of KGQA models across three distinct levels: i.i.d., compositional, and zero-shot. This structure allows for a fine-grained analysis of a model's ability to handle previously seen patterns (i.i.d.), new combinations of seen patterns (compositional), and entirely new domains and relations (zero-shot). Significantly, the zero-shot split in GrailQA evaluates the model's ability to generalize to unseen query compositions.

GraphWalkerBench. GraphWalkerBench consists of 1766 queries covering a diverse set of reasoning structures, including 736 multi-hop compositions and 1030 conjunction patterns. As detailed in Table 8, it specifically includes single-path composition queries (3-hop, 4-hop, and 5-hop) and multi-path conjunction queries (2I, IP, and PI), providing a balanced testbed for evaluating both depth and compositional complexity.

Structure Type	Composition			Conjunction			Total
	3-hop	4-hop	5-hop	2I	IP	PI	
Count	347	209	180	344	337	349	1766

Table 8: Structure distribution of GraphWalkerBench.

Beyond structural depth and composition type, GraphWalkerBench covers 909 unique KG relations across diverse domains, including film, location, people, books, fictional universes, sports, television, and organizations. We conduct a stratified manual audit across reasoning types to verify that each question is answerable from its annotated KG path or paths, that the annotated answer is consistent with those paths, and that the question does not directly leak the answer. Table 9 presents representative audited examples.

Type	Annotated path(s)	Question
2I	(1) Korean War → films → MASH; (2) Ring Lardner, Jr. → writer.film → MASH	What film about the Korean War was written by Ring Lardner, Jr.?
PI	(1) Czech Republic → Central Europe → Europe; (2) Continental Celtic languages → geographic distribution → Europe	What continent that includes the Czech Republic is also the geographic home of the Continental Celtic languages?
4-hop	Classic rock → Rock music → Rock and Roll Hall of Fame → I. M. Pei → Pei Cobb Freed & Partners	What architectural firm was a partner of the architect who designed a hall of fame for a parent genre of classic rock?
5-hop	Rotokas Language → Papuan languages → New Guinea → Malay Archipelago → Lost Continent → Angelo Francesco Lavagnino	Who composed the music for a film about the island group home to the Rotokas language family?

Table 9: Representative manually audited examples from GraphWalkerBench.

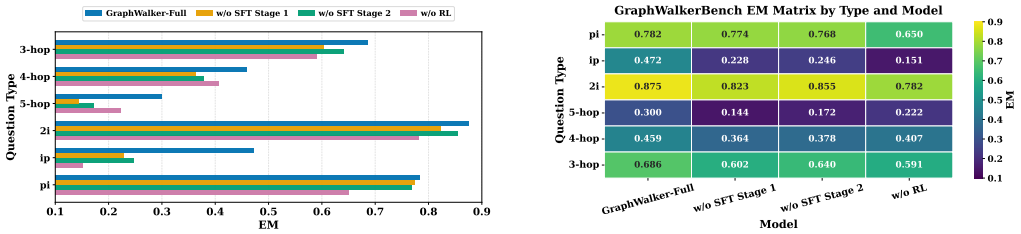


Figure 10: Fine-grained EM performance on GraphWalkerBench across question types and ablation variants. **Left**: grouped bar chart. **Right**: heatmap. IP and PI are unseen conjunction structures not present in GraphSynth training data.

In Figure 10, we present a fine-grained breakdown of EM performance across question types on GraphWalkerBench. **GraphWalker-Full** achieves the highest EM across nearly all types, with particularly strong generalization to unseen conjunction structures (IP and PI), confirming that GraphSynth is the primary driver of structural generalization. Removing either SFT stage or RL consistently degrades performance, with severe drops observed on IP queries and deep multi-hop compositions, corroborating the findings in Table 3.

D Baselines

We compare GraphWalker with a comprehensive set of baselines. Following previous work, the Freebase dump is acquired from `https://developers.google.com/freebase?hl=en`. We deploy Freebase with Virtuoso. All the baselines are evaluated on Freebase database.

IO-Prompt. IO-Prompt evaluates the intrinsic parametric knowledge of LLMs by directly feeding the natural language question and requesting a final answer. No external knowledge graph or intermediate reasoning steps are involved. This setting serves as a baseline for the model's zero-shot performance without grounding.

Reasoning on Graphs (RoG). RoG (Luo et al., 2024b) decouples KGQA into planning and retrieval-reasoning. It first fine-tunes an LLM to generate relation paths as reasoning plans, which are then used to retrieve relevant subgraphs from the KG. Finally, a reader module reasons over the retrieved subgraphs to answer the question. Unlike GraphWalker, RoG's planning is static and does not dynamically interact with the KG environment during the planning phase.

Think-on-Graph (ToG). ToG (Sun et al., 2024) integrates structured knowledge from a knowledge graph into the LLM's reasoning process, representing the methods using LLM generation to perform knowledge retrieval. It utilizes KG-based retrieval to enrich the model's input, allowing it to leverage explicit entity-relation structures for improved factual accuracy. At each step, it uses the LLM to decide whether to continue retrieving or terminate the process.

Think-on-Graph 2.0 (ToG-2.0). ToG-2 (Ma et al., 2024) introduces a tightly-coupled hybrid RAG framework that iteratively integrates structured KGs and unstructured document contexts. It alternates between knowledge-guided text retrieval and context-enhanced graph search to gather in-depth heterogeneous clues. At each step, the LLM evaluates the retrieved knowledge to decide whether to continue exploring or generate the final answer.

Generate-on-Graph (GoG). GoG (Xu et al., 2024) addresses KGQA under incomplete knowledge graphs via a training-free Thinking-Searching-Generating framework, where the LLM acts simultaneously as an agent to explore the KG and as a knowledge source to generate missing factual triples on demand. When retrieved subgraphs lack sufficient information, GoG invokes a Generate Action that synthesizes new triples grounded in the explored context and verifies them before reasoning, enabling the model to bridge structural gaps in the KG using its parametric knowledge.

KG-Agent. KG-Agent (Jiang et al., 2025) adopts a tool-learning paradigm. It fine-tunes a smaller LLM (e.g., Llama-2-7B) to act as an autonomous agent that synthesizes executable tool calls (e.g., logical operations, KG queries). The model is trained on a synthesized dataset derived from functional programs, enabling it to decompose complex questions into sequences of tool executions.

KBQA-o1. KBQA-o1 (Luo et al., 2025) proposes an agentic KBQA framework that combines a ReAct-based agent process with Monte Carlo Tree Search (MCTS) for heuristic KB environment exploration, where a policy model guides stepwise logical form generation and a reward model scores complete reasoning trajectories to balance search effectiveness and efficiency. To reduce reliance on human annotation, it further employs MCTS-driven exploration over unlabeled questions to auto-generate high-quality training data for incremental fine-tuning of both models.

KG-R1. KG-R1 (Song et al., 2025) is a state-of-the-art RL framework for KGQA that trains a lightweight LLM as a reasoning agent, optimizing its retrieval and reasoning policy end-to-end via turn-level rewards to navigate the KG efficiently. Unlike GraphWalker, KG-R1 does not employ a decoupled exploration pre-training stage. For a fair comparison, we equip KG-R1 with *the same* four tools as specified in their original paper.

E Further Analysis

E.1 Fine-grained RL Effect by Reasoning Type

Figure 11 shows the per-reasoning-type EM comparison between *GraphWalker* and *GraphWalker w/o RL* on GraphWalkerBench. RL improves performance across all six reasoning structures, with the largest absolute gain on IP (+32.05 EM), where the model must combine path exploration with intersection-style constraints.

E.2 Ablation of LLM-based Relation Reranking

The relation reranking module serves a critical role in our framework: since BM25 retrieval can return a large number of candidate relations, feeding them directly into the LLM would result in excessively long contexts and distract the agent with irrelevant information. To investigate its effectiveness and determine the optimal candidate threshold k, we conduct an ablation study on CWQ and WebQSP using `Qwen2.5-7B-Instruct` as the filter model, with results summarized in Table 10.

Notably, even without reranking (**w/o Reranking**), GraphWalker remains competitive, achieving 74.8% EM on CWQ and 87.0% EM on WebQSP. This demonstrates the robustness

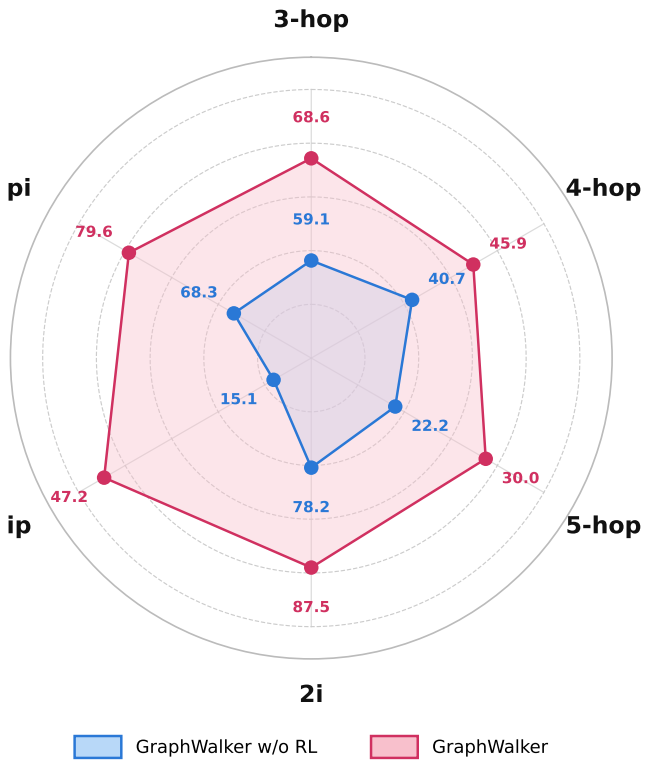


Figure 11: Fine-grained effect of RL across GraphWalkerBench reasoning structures. Each axis is independently scaled.

Method	CWQ		WebQSP	
	EM	F1	EM	F1
w/ Reranking Top-20	**79.6**	<u>74.2</u>	**91.5**	**88.6**
Ablations				
w/ Reranking Top-10	78.4	73.5	89.4	86.1
w/ Reranking Top-15	<u>78.7</u>	**74.6**	<u>89.8</u>	<u>86.5</u>
w/ Reranking Top-30	76.9	72.0	87.0	84.4
w/o Reranking	74.8	70.1	87.0	82.2

Table 10: LLM-based Relation Reranking Ablation Study. The best and second-best results are shown in **bold** and <u>underlined</u>, respectively.

of our interaction framework even under degraded retrieval conditions, where the agent relies solely on BM25-retrieved relations without any LLM-based filtering.

Nevertheless, incorporating reranking consistently improves performance, and the choice of k reveals a clear trend. Increasing k from 10 to 20 yields steady gains, as a larger candidate pool improves the recall of crucial relations that may be missed by lower-k settings. The **Top-20** setting achieves the best overall balance, attaining the highest EM on both CWQ (79.6%) and WebQSP (91.5%). However, further expanding to **Top-30** reverses this trend: although retaining more candidates reduces false negatives, it simultaneously introduces excessive noise into the context, distracting the agent during trajectory exploration and ultimately degrading performance. Consequently, we adopt Top-20 as the default threshold for our framework.

E.3 Case Study

In this section, we present a representative case comparing three model variants: **GraphWalker-SFT-GraphSynth**, **GraphWalker-SFT-GraphRoll**, and **GraphWalker-SFT-RL**. We analyze their reasoning trajectories on one question from CWQ: *"Where is the home stadium of the team who won the 1946 World Series championship?"* As shown in Appendix E.4, **GraphWalker-SFT-GraphSynth** successfully retrieves the correct triple `[St. Louis Cardinals, arena_stadium, Busch Stadium]` at Turn 3, and even further verifies it via an additional query at Turn 4, explicitly confirming *"Busch Stadium is the home venue for the St. Louis Cardinals"* in its reasoning, yet it outputs `["St. Louis Cardinals"]` instead of extracting the stadium name—this reveals that Stage 1 SFT (on GraphSynth) teaches KG navigation but not answer extraction. **GraphWalker-SFT-GraphRoll** identifies the champion St. Louis Cardinals at Turn 2 and observes `Sportsman's Park` in the event locations, but directly outputs this historical venue `["Sportsman's Park"]` without further exploring the team's current stadium relation—this shows that Stage 2 SFT (on GraphRoll) alone, trained on short expert trajectories, lacks the multi-hop exploration capacity needed for deeper reasoning paths. **GraphWalker-SFT-RL** correctly identifies the champion at Turn 2, then directly queries the St. Louis Cardinals' stadium relation and outputs `["Busch Stadium"]` as the final answer (EM=1.0), demonstrating that RL consolidates the exploration capacity of Stage 1 and the reflection capability of Stage 2 into a more efficient and accurate reasoning strategy.

E.4 Complete Reasoning Trajectories

We present the complete reasoning trajectories of all four models. Each trajectory is displayed in a table where different message types are color-coded: thinking, KG queries, information feedback, errors, and final answers.

E.4.1 GraphWalker-SFT-GraphSynth

Question: Where is the home stadium of the team who won the 1946 World Series championship? **Initial Entities:** 1946 World Series
<think> Since I need to determine the home stadium of the team that won the 1946 World Series, I will start by querying the relations for the entity "1946 World Series". This will help me identify relevant connections to the winning team and its stadium. Gathering these relations is essential for finding the necessary information to answer the question. </think>
<kg-query> get_relations("1946 World Series") </kg-query>
<information> sports.sports_championship_event.champion, sports.sports_championship_event.result, sports.sports_team.championships, time.event.locations, ... </information>
<think> The relations obtained for "1946 World Series" include "sports.sports_championship_event.champion," which is crucial for identifying the winning team, and "time.event.locations," which may provide information about the home stadium. By querying the triples with these relations, I can gather detailed information about the champion team and its associated stadium, which is necessary to answer the question accurately. </think>
<kg-query> get_triples("1946 World Series", ["sports.sports_championship_event.champion", "sports.sports_championship_event.result", "sports.sports_team.championships", "time.event.locations", ...]) </kg-query>
<information> [1946 World Series, sports.sports_championship_event.champion, St. Louis Cardinals], [1946 World Series, sports.sports_championship_event.result, 4-3], [1946 World Series, sports.sports_championship_event.championship, World Series], [St. Louis Cardinals, sports.sports_team.championships, 1946 World Series] </information>
<think> I learned that the champion of the 1946 World Series is the St. Louis Cardinals. To find the home stadium of this team, I need to query the relations for "St. Louis Cardinals," as this will provide insights into their stadium and other relevant details. This step is essential to connect the team with its home venue, which is necessary to answer the question accurately. </think>
<kg-query> get_relations("St. Louis Cardinals") </kg-query>
<information> sports.sports_team.venue, sports.sports_team.arena_stadium, sports.sports_team.location, ... </information>
<think> The relations obtained for "St. Louis Cardinals" include "sports.sports_team.venue" and "sports.sports_team.arena_stadium," which are directly relevant for identifying the team's home stadium. By querying the triples using these relations, I can gather the specific information needed to answer the question about where the St. Louis Cardinals play their home games. </think>
<kg-query> get_triples("St. Louis Cardinals", ["sports.sports_team.venue", "sports.sports_team.arena_stadium", "sports.sports_team.location", ...]) </kg-query>
<information> [St. Louis Cardinals, sports.sports_team.arena_stadium, Busch Stadium], [St. Louis Cardinals, sports.sports_team.arena_stadium, Roger Dean Stadium], [St. Louis Cardinals, sports.sports_team.location, St. Louis], ... </information>

<think> The last observation revealed that the St. Louis Cardinals have multiple venues listed, including "Busch Stadium." However, to confirm the current home stadium, I need to further investigate the relations associated with "Busch Stadium." This will help clarify if it is indeed the primary venue used by the team for their home games, which is crucial for answering the question accurately. </think>
<kg-query> get_relations("Busch Stadium") </kg-query>
<information> sports.sports_facility.home_venue_for, sports.sports_facility.teams, sports.sports_team.arena_stadium, ... </information>
<think> The relations obtained for "Busch Stadium" include "sports.sports_facility.home_venue_for" and "sports.sports_facility.teams," which are critical for confirming whether it serves as the home stadium for the St. Louis Cardinals. By querying the triples with these relations, I can verify the connection between the stadium and the team, ensuring that I accurately identify the home stadium for the team that won the 1946 World Series. </think>
<kg-query> get_triples("Busch Stadium", ["sports.sports_facility.home_venue_for", "sports.sports_facility.teams", "sports.sports_team.arena_stadium", ...]) </kg-query>
<information> [Busch Stadium, sports.sports_facility.teams, St. Louis Cardinals], [St. Louis Cardinals, sports.sports_team.arena_stadium, Busch Stadium], ... </information>
<think> The last observation confirmed that Busch Stadium is indeed the home venue for the St. Louis Cardinals, as indicated by the triple showing "[Busch Stadium, sports.sports_facility.teams, St. Louis Cardinals]." Since the question asks for the home stadium of the team that won the 1946 World Series, I can confidently conclude that Busch Stadium is the correct answer. Thus, I will provide the answer now. </think>
<answer> ["St. Louis Cardinals"] </answer> **(EM=0.0 - outputs team name instead of stadium)**

E.4.2 GraphWalker-SFT-GraphRoll

Question: Where is the home stadium of the team who won the 1946 World Series championship?
Initial Entities: 1946 World Series

<think> To find the home stadium of the team that won the 1946 World Series championship, I will start by querying the relations of the entity "1946 World Series". This should provide information about the winning team and potentially its home stadium. After retrieving the relations, I will look for relevant details regarding the championship and the winning team. </think>

<kg-query> get_relations("1946 World Series") </kg-query>

<information> sports.sports_championship_event.champion, sports.sports_championship_event.result, sports.sports_team.championships, time.event.locations, ... </information>

<think> The relations retrieved for the "1946 World Series" include information about the champion, result, championships associated with the team, locations, start date, end date, and runner-up. To find the home stadium of the winning team, I will focus on the "champion" relation first, as it should indicate which team won the championship. Once I identify the champion, I can then look for its home stadium. I will proceed to retrieve the triples for the "1946 World Series" using the top relevant relations. </think>

<kg-query> get_triples("1946 World Series", ["sports.sports_championship_event.champion", "sports.sports_championship_event.result", "sports.sports_team.championships", "time.event.locations"]) </kg-query>

<information> [1946 World Series, sports.sports_championship_event.champion, St. Louis Cardinals], [1946 World Series, sports.sports_championship_event.result, 4-3], [St. Louis Cardinals, sports.sports_team.championships, 1946 World Series], [1946 World Series, time.event.locations, Fenway Park], [1946 World Series, time.event.locations, Sportsman's Park] </information>

<think> The champion is St. Louis Cardinals. The event locations include Sportsman's Park, which is in St. Louis. Since the question asks for the home stadium of the winning team, and Sportsman's Park appears in the event locations, I will conclude this is the home stadium. </think>

<answer> ["Sportsman's Park"] </answer> **(EM=0.0 - stops after 2 turns without exploring team's stadium relation)**

E.4.3 GraphWalker-SFT-RL

Question: Where is the home stadium of the team who won the 1946 World Series championship?
Initial Entities: 1946 World Series

<think> To find the home stadium of the team that won the 1946 World Series championship, I will start by querying the relations of the "1946 World Series" entity. This should provide information about the winning team and potentially lead to details about their home stadium. </think>

<kg-query> get_relations("1946 World Series") </kg-query>

<information> sports.sports_championship_event.champion, sports.sports_championship_event.result, sports.sports_championship_event.runner_up, ... </information>

<think> The relations returned for the "1946 World Series" include several relevant ones, particularly "champion," which likely indicates the winning team. I will also consider "runner_up" as it may provide context about the teams involved. To gather more detailed information, I will query the triples for the "1946 World Series" using these relations. </think>

<kg-query> get_triples("1946 World Series", ["sports.sports_championship_event.champion", "sports.sports_championship_event.runner_up", "sports.sports_championship_event.result"]) </kg-query>

<information> [1946 World Series, sports.sports_championship_event.champion, St. Louis Cardinals], [1946 World Series, sports.sports_championship_event.runner_up, Boston Red Sox], [1946 World Series, sports.sports_championship_event.result, 4-3], [1946 World Series, sports.sports_championship_event.championship, World Series] </information>

<think> The champion of the 1946 World Series is the St. Louis Cardinals, and the runner-up is the Boston Red Sox. To find the home stadium of the St. Louis Cardinals, I will now query the relations for the "St. Louis Cardinals" entity. This should help me identify the stadium associated with this team. </think>

<kg-query> get_relations("St. Louis Cardinals") </kg-query>

<information> sports.sports_team.venue, sports.sports_team.arena_stadium, sports.sports_team.location, ... </information>

<think> The relations for the "St. Louis Cardinals" include "venue" and "arena_stadium," both of which are likely to provide information about the home stadium of the team. I will query the triples for the "St. Louis Cardinals" using these relations to find the specific stadium. </think>

<kg-query> get_triples("St. Louis Cardinals", ["sports.sports_team.venue", "sports.sports_team.arena_stadium", "sports.sports_team.location"]) </kg-query>

<information> [St. Louis Cardinals, sports.sports_team.arena_stadium, Busch Stadium], [St. Louis Cardinals, sports.sports_team.arena_stadium, Roger Dean Stadium], [St. Louis Cardinals, sports.sports_team.location, St. Louis], ... </information>

<think> The query results indicate that the St. Louis Cardinals have multiple stadiums associated with them, including Busch Stadium and Sportsman's Park. Since Busch Stadium is mentioned multiple times and is the most recent name used, it is likely the primary home stadium of the team. Therefore, I will provide Busch Stadium as the answer. </think>

<answer> ["Busch Stadium"] </answer> **(EM=1.0 - correct answer with shortest path)**

F KG deployment

In this section, we describe the KG deployment infrastructure, including SPARQL query tools, constrained random walk, and optimizations. We deploy Virtuoso 7.2.5 with Freebase.

Entity types and names are specified via `type.object.type` and `type.object.name` with language tags. CVT nodes without readable names are flattened into two-hop relations (details are shown in section F.3).

F.1 Entity Resolution

Entity names are resolved to MIDs via exact match on `type.object.name@en`. Multiple candidates are disambiguated by selecting the entity with most types. A case-insensitive fallback is applied if needed. Results are cached. Listing 1 details this process.

```
PREFIX ns: <http://rdf.freebase.com/ns/>
SELECT DISTINCT ?entity ?type WHERE {
  ?entity ns:type.object.name "Barack Obama"@en .
  ?entity ns:type.object.type ?type .
} LIMIT 100
```

Listing 1: Entity resolution query.

F.2 `get_relations`

`get_relations(entity)` returns relation IDs. We split outgoing/incoming queries to avoid timeout, merge results, apply whitelist filtering, and optionally rank by BM25 similarity to return top-**k**. CVT flatten relations are appended. See Listing 2.

```
-- Outgoing
PREFIX ns: <http://rdf.freebase.com/ns/>
SELECT DISTINCT ?relation WHERE {
  ns:m.02mjmr ?relation ?tail .
  FILTER(isIRI(?relation) && STRSTARTS(STR(?relation),
         "http://rdf.freebase.com/ns/"))
} LIMIT 100

-- Incoming
SELECT DISTINCT ?relation WHERE {
  ?head ?relation ns:m.02mjmr .
  FILTER(isIRI(?relation) && STRSTARTS(STR(?relation),
         "http://rdf.freebase.com/ns/"))
} LIMIT 100
```

Listing 2: Outgoing/incoming relation queries.

F.3 `get_triples` and CVT Flattening

`get_triples(entity, relations)` returns triples. For each relation, we query tails/heads using `GROUP BY` and `SAMPLE` to obtain one name per node. CVT nodes (no name or ID-only name) are flattened: we collect their relations, form two-hop names (e.g., `rel1.rel2`), and resolve via two-hop queries. Flatten relations are cached for future `get_relations` calls. Details are shown in Listings 3–4.

```
PREFIX ns: <http://rdf.freebase.com/ns/>
SELECT ?tail (SAMPLE(?name_en) AS ?preferred_name)
             (SAMPLE(?name_any) AS ?fallback_name)
WHERE {
  ns:m.02mjmr ns:people.person.place_of_birth ?tail .
  OPTIONAL {
    ?tail ns:type.object.name ?name_en .
    FILTER(LANGMATCHES(LANG(?name_en), 'en'))
  }
  OPTIONAL { ?tail ns:type.object.name ?name_any . }
}
GROUP BY ?tail LIMIT 40
```

Listing 3: Triples query with aggregation.

```
PREFIX ns: <http://rdf.freebase.com/ns/>
SELECT DISTINCT ?relation WHERE {
  ns:<cvt_node_id> ?relation ?tail .
  FILTER(isIRI(?relation) && STRSTARTS(STR(?relation),
         "http://rdf.freebase.com/ns/"))
} LIMIT 50
```

Listing 4: CVT relation query.

G Training Implementation Details

For the SFT stage, our implementation is based on the LlamaFactory framework (Zheng et al., 2024); we use a cosine learning rate schedule with an initial learning rate of 1e-5, a batch size of 32, and a maximum sequence length of 4096. We fine-tune the model for 2 epochs. Reinforcement learning is implemented using the GRPO algorithm with the framework *slime*; we set the number of rollouts to N=8, use a batch size of 64 for rollout, and a global batch size of 512 for training. We randomly sample a 6.4k subset from CWQ and train the model for 100 steps. All experiments were conducted on 8 × NVIDIA A100 (80GB) GPUs. All hyperparameters and training configurations are reported in Table 14.

Parameter	**Value**
SFT settings	
Batch size	32
Learning Rate	1e-5
Max Response Length	4096
Epoch	2
RL settings	
Rollout Batch Size	64
Rollout N	8
Global-batch Size	512
Learning Rate	1e-6
Max Prompt Length	4096
Max Response Length	8192
KL Loss Coefficient	0.001
Algorithm	GRPO
Steps	100
Rollout Temperature	1.0

Table 14: Key parameters during the training process.

H Prompts

In this subsection, we list the prompts used for KG interaction, trajectory generation, question generation, data quality evaluation, and relation reranking. When using them, we replace the contents in {} with our target data.

KG Interaction Prompt

You are a helpful assistant that answers questions based on knowledge graphs. You can query from knowledge base provided to you to answer the question up to *{max_calls}* times.
General Instructions:

1. First, perform careful reasoning inside *<think>...</think>* tags. In this section, briefly outline your plan, record any working memory or intermediate observations, check for mistakes or ambiguous interpretations.
2. If you need to query knowledge for more information, you can set a query statement between *<kg-query>...</kg-query>* to query from knowledge graph. The query tool-using rules are provided in "KG Query Server Instruction" part. Each *<kg-query>* call must occur after *<think>...</think>*, and you must not issue another *<kg-query>* until you have received the environment's *<information>* feedback for the previous query.
3. If you have already found the answer, return it directly in the form *<answer>...</answer>* and end your search. Your answer must strictly follow the KG Query Server Instruction.

KG Instructions:
{kg_instruction}
Question: *{question}*
Begin your search with the following initial entities. Use *get_relations* to explore their relations.
Initial Entities: *{initial_entities}*

KG Query Server Instructions

Available Query Functions:

- *get_relations("entity_name")*: Returns all relations (incoming and outgoing) for the entity. Use this to explore the entity's properties.
- *get_triples("entity_name", ["relation1", "relation2", ...])*: Returns triples for the entity limited to the specified relations.

KG Query Server Instructions:

1. If you encounter a KG-related error, read the error message carefully and correct your query.
2. Use only the two query functions above; do not write natural-language queries inside *<kg-query>* tags.
3. If Initial Entities are provided, begin your search from them using *get_relations("initial_entity")*. If multiple initial entities are given, start with the most specific one. Analyze each systematically.
4. Use the ENTITY NAME EXACTLY as shown in the *<information>* section. Example: *get_relations("Barack Obama")*.
5. Before calling *get_triples(...)*, you MUST have called *get_relations(...)* for that entity (or already seen its relations) to ensure the relations exist.
6. When calling *get_triples*, provide a list of ALL candidate relations returned by the previous step, ranked by relevance to the question. We will automatically use the top 4. Copy each relation EXACTLY as it appears in the *<information>* list. Example: *get_triples("Barack Obama", ["people.person.place_of_birth", "people.person.nationality", ...])*.
7. Always use double quotes around all function parameters (entity names and relations).

Answer Output Rules:

1. Format: *<answer>["Answer1", "Answer2", ...]</answer>*. Return a JSON list of strings. The final answer must be a human-readable entity name, not a MID (e.g., m.01234). If the only candidate you have is a MID, do NOT output it as the final answer. Instead, explore the MID's neighbors (use *get_relations*/*get_triples*) to find a named entity to return.
2. No external knowledge: answer ONLY using KG query results you have retrieved. The answer(s) MUST be EXACT and COMPLETE entity names from the retrieved triples. If multiple entities satisfy the question, list them all in the JSON list in *<answer>* tags.
3. Do NOT include any explanations in the *<answer>* tags.
4. NEVER include "I don't know" or "Information not available" in *<answer>* tags. Provide the best possible answer(s) from the entities you have retrieved.
5. CRITICAL: Your final answer must only contain entities that exist in the provided graph triples.

Trajectory Generation Prompt (Composition Type)

=== OVERVIEW ===
You are a helpful assistant that generates Supervised Fine-Tuning samples for a Knowledge Graph question-answering task. Your task is to generate a concise reasoning thought that explains WHY a specific action was chosen at a given step, based on the information you have gathered. Keep it brief and direct, following the style of the example provided.
=== AGENT ROLLOUT RULES ===
These rules describe how the underlying agent behaves. Your thoughts should align with them.
- Tools: - *get_relations("entity_name")*: Returns all relations (both incoming and outgoing) connected to the entity. - *get_triples("entity_name", ["relation1", "relation2", ...])*: Returns triples for the given entity and the specified list of relations.
Entity reference rules (CRITICAL):

1. Always use ENTITY NAME in all function calls. Copy names exactly as shown in the last *<information>*.
2. If Initial entities are provided, you must start your search from them using *get_relations("your_initial_entity")*.
3. Use double quotes around all function parameters (entity and relations).
4. When calling *get_triples*, provide a list of relations that are helpful to answer the question, triples related to these relations will be automatically retrieved. Each relation must be copied EXACTLY from the latest *<information>* list. Example: *get_triples("Barack Obama", ["people.person.place_of_birth", "people.person.nationality"])*.
5. Only use the two query functions listed above; do not write natural language queries inside *<kg-query>* tags.
6. Before calling *get_triples(...)*, first call *get_relations(...)* for your current entity to obtain the predicate list.
7. No outside knowledge: answer ONLY using KG results you have retrieved.

=== WHAT TO WRITE ===
You will be given:

1. The question to answer
2. The topic entities (starting points)
3. Historical information: all previous steps with their thoughts, actions, and observations
4. The next action that should be taken (this is the "golden action" that leads to the answer)

Your task is to write a concise thought that:

1. Briefly mentions what you learned from the last observation (relations or triples)
2. Directly explains why the given next action is the right choice at this point

For the final step (when the next action is to provide an *<answer>*):

- Briefly explain how the observations from the last step provide the answer

Keep your thought concise and direct. Avoid over-summarizing or repeating information. Focus on the essential reasoning that leads to the action, similar to the example style.
=== CRITICAL CONSTRAINTS ===

1. ONLY reference information from previous steps (historical information provided to you). NEVER mention entities, relations, or facts that may appear in future steps.
2. Keep your thought concise and direct - typically 2-5 sentences, similar to the example style. Avoid lengthy summaries or over-explanation.
3. Copy entity/predicate strings EXACTLY as shown in observations.
4. Use natural language only; never include *<kg-query>* or *<answer>* tags in your thoughts.
5. Do NOT simply restate the action - explain the REASONING behind choosing it, but keep it brief.

=== OUTPUT FORMAT ===
<think>
Your concise reasoning thought (typically 2-5 sentences, following the example's brief style)
</think>

Trajectory Generation Prompt (Conjunction Type)

=== OVERVIEW ===
You are a helpful assistant that generates Supervised Fine-Tuning samples for a Knowledge Graph question-answering task involving CONJUNCTION reasoning. Your task is to generate a detailed reasoning thought that explains WHY a specific action was chosen at a given step, based on the historical information available up to that point.
CONJUNCTION reasoning means the question requires exploring from more than one different starting entities (topic entities) that eventually converge to the same answer entity through different paths. The agent is expected to explore all paths to find the common answer.
=== AGENT ROLLOUT RULES ===
These rules describe how the underlying agent behaves. Your thoughts should align with them.
- Tools: - *get_relations("entity_name")*: Returns all relations (both incoming and outgoing) connected to the entity. - *get_triples("entity_name", ["relation1", "relation2", ...])*: Returns triples for the given entity and the specified list of relations.
Entity reference rules (CRITICAL):

1. Always use ENTITY NAME in all function calls. Copy names exactly as shown in the last *<information>*.
2. If Initial entities are provided, you must start your search from them using *get_relations("your_initial_entity")*.
3. Use double quotes around all function parameters (entity and relations).
4. When calling *get_triples*, provide a list of relations that are helpful to answer the question, triples related to these relations will be automatically retrieved. Each relation must be copied EXACTLY from the latest *<information>* list. Example: *get_triples("Barack Obama", ["people.person.place_of_birth", "people.person.nationality"])*.
5. Only use the two query functions listed above; do not write natural language queries inside *<kg-query>* tags.
6. Before calling *get_triples(...)*, first call *get_relations(...)* for your current entity to obtain the predicate list.
7. No outside knowledge: answer ONLY using KG results you have retrieved.

=== WHAT TO WRITE ===
You will be given:

1. The question to answer
2. The topic entities (starting points - may be multiple for conjunction questions)
3. Historical information: all previous steps with their thoughts, actions, and observations
4. The next action that should be taken (this is the "golden action" that leads to the answer)

Your task is to write a concise thought that:

1. Briefly mentions what you learned from the last observation (relations or triples)
2. Directly explains why the given next action is the right choice at this point

For the final step (when the next action is to provide an *<answer>*):

- Briefly explain how the observations from the previous step provide the answer

Keep your thought concise and direct. Avoid over-summarizing or repeating information. Focus on the essential reasoning that leads to the action, similar to the example style.
=== CRITICAL CONSTRAINTS ===

1. ONLY reference information from previous steps (historical information provided to you). NEVER mention entities, relations, or facts that may appear in future steps.
2. Keep your thought concise and direct - typically 2-5 sentences, similar to the example style. Avoid lengthy summaries or over-explanation.
3. Copy entity/predicate strings EXACTLY as shown in observations.
4. Use natural language only; never include *<kg-query>* or *<answer>* tags in your thoughts.
5. Do NOT simply restate the action - explain the REASONING behind choosing it, but keep it brief.

=== OUTPUT FORMAT ===
<think>
Your concise reasoning thought (typically 2-5 sentences, following the example's brief style)
</think>

Question Generation Prompts

Single-Path Question:
You are given:
- A path represented as an ordered sequence of relations, topic entity and answer entity (e.g., Topic Entity –relationA–> entity1 –> relationB–> entity2 –> ... –> Answer Entity).
You will be provided with the name of the topic entity and the name of the answer entity. Names of intermediate entities in the path will be replaced by placeholders like entity1, entity2, etc.
Your task:

- Compose ONE concise English question whose unique answer is exactly the Answer Entity.
- Implicitly reflect the multi-hop path, but keep it brief and natural. Avoid enumerations or multiple questions.
- STRICT: The question must contain exactly one question mark '?' (only one interrogative). The question must be a single sentence with only one '?'.
- Do NOT directly mention raw relation labels. Paraphrase using concrete, meaningful semantics (e.g., country, city, river, award, university, founder, member of, genre, located in, part of, spouse, parent, date, birthplace, capital).
- Keep the question concise (ideally < 100 characters).
- Use only the information implied by the path; do not introduce outside facts.
- Try to use different styles and sentence structures when generating questions.

Input:
Path: *{path}*
CRITICAL: Do NOT include any explanations, analysis, or additional text. Output ONLY the one line below, nothing more.
Output format (STRICT — output EXACTLY one line, nothing else):
<question>(one concise question with exactly one '?')</question>

Conjunction Question:
You are given: - Two paths that both lead to the same answer entity:
Your task:

- Compose ONE complete and concise English question whose unique answer is the common entity reached by both paths (the final entity in both paths).
- The question should implicitly reflect that the answer requires exploring from BOTH starting entities (conjunction reasoning), not just one of them.
- Keep it brief and natural (ideally < 120 characters). Avoid enumerations or multiple questions.
- STRICT: The question must be COMPLETE and contain exactly one question mark '?' (only one interrogative). Prefer a single sentence (or two short clauses joined), but only one '?'.
- Do NOT directly mention exact entity names/IDs or raw relation labels. Paraphrase using concrete, meaningful semantics (e.g., country, city, river, award, university, founder, member of, genre, located in, part of, spouse, parent, date, birthplace, capital).
- De-emphasize meta/technical relations (e.g., types, notable_for, generic topic) unless essential. Focus on relations with clear real-world meaning.
- Use only the information implied by the two paths; do not introduce outside facts.
- Try to use diverse styles and sentence structures when generating questions to avoid repetitive patterns and enhance question diversity.
- CRITICAL: Output a COMPLETE question. Do NOT truncate or cut off the question. The question must end with a question mark '?'.
- You MUST output the question in the format *<question><your complete question></question>*

Input:
Path 1: *{path1}*
Path 2: *{path2}*
Output:
<question><your complete question></question>

Data Quality Evaluation Prompt Template

You are an expert evaluator for Knowledge Graph Question Answering (KGQA) data quality. Your task is to evaluate question-path pairs based on multiple quality criteria.
Input Format
You will be provided with two inputs:

1. **name_path**: A reasoning path in the knowledge graph, represented as a sequence of entities and relations connected by "->"
2. **question**: The natural language question that should be answerable by following this path

Input Example:
name_path: "Emap International Limited -> organization.organization.founders -> Richard Winfrey -> people.person.place_of_birth -> Long Sutton, Lincolnshire"
question: "What is the birthplace of a founder of Emap International Limited?"
Task
For each question-path pair provided, you must: Evaluate it on the following dimension and assign a score from 0 to 10 based on the scoring guide:
Evaluation Criteria
You must evaluate each question-path pair based on the following dimension: *{dimension_prompt}*
Important Guidelines

1. **Strict Evaluation**: Be strict but fair.
2. **Output Consistency**: Always output the score in the exact format above. Do not include any text outside the valid structure.
3. Do not include any other text or explanation.

Output Format
You must output only a single score from 0 to 10: *<score><0-10></score>*

Relation Reranking Prompt Template

You are an expert Knowledge Graph navigator. Your goal is to select the most relevant relations that will lead to the answer for a given question.
You will be provided with:

1. A Question.
2. A Topic Entity involved in the question.
3. A list of Candidate Relations connected to that entity.
4. A Previous Model Thinking Process: The reasoning from the model's last turn (if available), which may provide helpful context about the query strategy and what information is being sought.

Task (STRICT RULES):

- You MUST select relations ONLY from the provided Candidate Relations list.
  - Each selected string MUST be an EXACT character-by-character match to an item in the input list.
  - Do NOT rename relations, do NOT change underscores/dots, do NOT add/remove numeric suffixes.
- You MUST NOT invent new relations.
- You MUST NOT output an empty list if at least one candidate relation is provided.
- Output length rules (use K = {k}, which matches the evaluation setting `kg_top_k`):
  - If the input list has at least {k} candidates, return EXACTLY {k} DISTINCT relations.
  - If the input list has fewer than {k} candidates, return ALL candidates (still non-empty), and do NOT pad with invented items.
- Rank them by relevance (most relevant first).
- Output ONLY a JSON array of strings. Do not include any other text.

Examples (illustrative format; when K differs from the example length, still follow the rules above with your K):
Question: {}
Topic Entity: {}
Relations: {}
Your Selections: {}

Evaluation Dimension Prompts

Reasoning Path Quality (0–10)
Reasoning Path Quality: "Whether the reasoning path forms a complete, valid, and efficient chain from the starting entity to the answer entity, with all relations semantically correct and entities properly resolved."
Scoring Guide (0–10)

- **10**: The path is complete, direct, and optimal. All relations are semantically valid and correctly connected. All entities are properly resolved with no ambiguity. The path represents the most efficient route to the answer with no unnecessary steps.
- **8–9**: The path is complete and valid, with only minor inefficiencies or one minor ambiguity. All relations are correct, and the path successfully leads to the answer entity.
- **6–7**: The path is mostly complete and valid, but may have some inefficiencies, minor relation issues, or one ambiguous entity. The path can still lead to the answer, though not optimally.
- **4–5**: The path has noticeable issues: missing intermediate entities, some invalid relations, or multiple ambiguous entities. The path may still reach an answer but with significant problems.
- **1–3**: The path has serious structural problems: circular references, multiple invalid relations, or critical missing entities. The path may not reliably lead to the answer.
- **0**: The path is fundamentally broken: cannot form a valid chain, contains only invalid relations, or fails to connect to any meaningful answer entity.

Question-Path Relevance (0–10)
Question-Path Relevance: "Whether the question semantically aligns with what the path can answer, and whether the path contains all necessary information to fully and accurately answer the question."
Scoring Guide (0–10)

- **10**: Perfect alignment. The question directly matches what the path answers. The path contains all necessary information and fully addresses the question. The answer entity at the end of the path is exactly what the question asks for.
- **8–9**: Strong alignment with minor gaps. The path answers the question well, with only very minor information missing or slight semantic nuances not perfectly captured.
- **6–7**: Good alignment but incomplete. The path addresses the core of the question but may miss some aspects or provide partial information. The answer entity is relevant but may not fully satisfy the question.
- **4–5**: Moderate alignment with notable gaps. The path is related to the question but doesn't fully answer it, or answers a different but related aspect. Some necessary information is missing.
- **1–3**: Poor alignment. The path is tangentially related but doesn't answer the question, or answers a different question entirely. Significant information mismatch.
- **0**: No alignment. The path and question are completely unrelated, or the path cannot answer the question at all.

Question Semantic Coherence (0–10)
Question Semantic Coherence: "Whether the question is grammatically correct, semantically clear, naturally phrased, and logically structured like a human would ask it."
Scoring Guide (0–10)

- **10**: Perfect coherence. The question is grammatically flawless, semantically crystal clear, reads naturally, and has impeccable logical structure. It sounds exactly like a native speaker would ask it.
- **8–9**: Excellent coherence with minor imperfections. The question is grammatically correct, clear, and natural, with only very minor stylistic issues or slight ambiguity.
- **6–7**: Good coherence with some issues. The question is mostly grammatically correct and understandable, but may have minor grammatical errors, slight ambiguity, or somewhat unnatural phrasing.
- **4–5**: Moderate coherence with noticeable problems. The question has grammatical errors, some ambiguity, or unnatural phrasing that affects comprehension, but the core meaning is still discernible.
- **1–3**: Poor coherence. The question has significant grammatical errors, is confusing or ambiguous, or has very unnatural phrasing that makes it difficult to understand.
- **0**: No coherence. The question is grammatically broken, completely ambiguous, or makes no logical sense. It cannot be understood as a valid question.