RouterKGQA: Specialized–General Model Routing for Constraint-Aware Knowledge Graph Question Answering

Bo Yuan[1*] **Hexuan Deng**[1,3*] **Bang Long**[2] **Xuebo Liu**[1†]
Shuai Zheng[2] **Mingzhen Zhou**[2] **Min Zhang**[1]
[1]Institute of Computing and Intelligence, Harbin Institute of Technology, Shenzhen, China
[2]Shenzhen Farben Information Technology Co., Ltd., Shenzhen, China
[3]Zhongguancun Academy, Beijing, China
yuanbo@stu.hit.edu.cn, hxuandeng@gmail.com
{liuxuebo,zhangmin2021}@hit.edu.cn
{Bang.long,climber.zheng,James.zhou}@farben.com.cn

Abstract

Knowledge graph question answering (KGQA) is a promising approach for mitigating LLM hallucination by grounding reasoning in structured and verifiable knowledge graphs. Existing approaches fall into two paradigms: *retrieval-based* methods utilize small specialized models, which are efficient but often produce unreachable paths and miss implicit constraints, while *agent-based* methods utilize large general models, which achieve stronger structural grounding at substantially higher cost. We propose RouterKGQA, a framework for specialized–general model collaboration, in which a specialized model generates reasoning paths and a general model performs KG-guided repair only when needed, improving performance at minimal cost. We further equip the specialized model with constraint-aware answer filtering, which reduces redundant answers. In addition, we design a more efficient general agent workflow, further lowering inference cost. Experimental results show that RouterKGQA outperforms the previous best by 3.57 points in F1 and 0.49 points in Hits@1 on average across benchmarks, while requiring only 1.15 LLM calls per question on WebQSP (1.23 on CWQ). Code and models are available at https://github.com/Oldcircle/RouterKGQA.

1 Introduction

Recent advances in large language models (LLMs) have enabled remarkable progress across natural language processing tasks (OpenAI, 2023; Llama Team, 2024). However, LLMs remain prone to **factual hallucination**—generating plausible but factually incorrect content (Wagner et al., 2025). A promising direction for mitigating hallucination is to ground LLM reasoning in knowledge graphs (KGs), which store large-scale factual knowledge in a structured, interpretable form (Bollacker et al., 2008; Ji et al., 2024; Sui et al., 2025).

Current approaches to integrating LLMs with KGs fall into two paradigms. **Agent-based methods** utilize general LLMs to form an agent workflow that explores the KG through multi-round interactions to accumulate evidence (Sun et al., 2024; Chen et al., 2024b; Dong et al., 2025). While effective, this incurs *high computational overhead*, leading to significant inference latency and cost.

Retrieval-based methods train specialized LLMs to generate structured retrieval plans for KG execution (Luo et al., 2024a,b, 2025). While more efficient, they may generate *hallucinated relations* absent from the target KG (Tian et al., 2025b), especially under domain shift. Existing specialized LLMs also often overlook *implicit constraints* in the question. For example, "Who is the U.S. president?" may correspond to multiple tail entities unless a temporal constraint is specified.

To address these issues, we propose RouterKGQA, a framework of specialized–general model collaboration, which achieves better performance with lower cost. Specifically, RouterKGQA answers each question with a specialized model, which is efficient and performs well in-domain, and invokes a general model, which offers stronger generalization, only when a deterministic check finds that the specialized model's predicted path is unreachable on the KG. This reachability-gated repair requires neither an additional LLM call nor router training. We further improve both components. For the specialized model, we train it to capture implicit constraint semantics in questions, enabling more precise answer filtering. For the general agent workflow, we replace costly multi-round exploration with a cheaper exploration only on relations, which

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

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 1903–1924
November 6-10, 2026 ©2026 Association for Computational Linguistics

reduces cost. As a result, RouterKGQA achieves strong accuracy while substantially lowering cost compared with conventional agent workflows. Our contributions are:

- We propose RouterKGQA, a specialized–general routing framework gated by deterministic structural verification against the KG, i.e., reachability-gated repair rather than a learned router, that combines efficient specialized generation with general repair for KGQA.
- We improve both systems by introducing Constraint-aware Reasoning Paths (CRPs) for specialized models, which filter answer entities with explicit constraints that can be applied independently of the main path, and by designing a more efficient agent workflow for general models whose LLM cost does not scale with the beam width, requiring at most $d + 1$ calls regardless of the beam width w.
- Experiments on WebQSP and CWQ show that RouterKGQA achieves the highest F1 on both benchmarks with only 1.15 LLM calls per question on WebQSP (1.23 on CWQ). CRPs with constraint-aware execution account for the average accuracy gain, while reachability-gated repair provides targeted recovery (+21.78 / +17.42 F1 on the 7.81% / 9.57% of routed questions) and keeps the deployed cost at the level of retrieval-based methods.

2 Related Work

Factual Hallucination of LLMs. Despite language capabilities, LLMs remain prone to factual hallucination, producing fluent yet incorrect text (Ji et al., 2023; Deng et al., 2024, 2026b; Huang et al., 2025). Prompting-based methods such as chain-of-thought (Wei et al., 2022) and self-consistency (Wang et al., 2023) improve coherence but cannot guarantee factual correctness (Ye and Durrett, 2022). Retrieval-augmented generation (Lewis et al., 2020; Gao et al., 2023) grounds outputs in external evidence, but standard RAG retrieves unstructured text and lacks the relational structure needed for complex reasoning. KG-augmented approaches offer a more structured alternative by constraining reasoning to verifiable triples (Pan et al., 2024; Agrawal et al., 2024).

Agent-based KGQA. Agent-based methods typically utilize general LLMs, treating the LLM as an interactive agent that explores the KG through multi-round tool calls (Tian et al., 2025a; Wang et al., 2026). ToG (Sun et al., 2024) performs iterative beam-search expansion over entities and relations. StructGPT (Jiang et al., 2023) provides specialized interfaces for structured data interaction. DoG (Ma et al., 2025) introduces multi-role debate for question decomposition. PoG (Chen et al., 2024b) decomposes questions into sub-objectives with backtracking and self-correction. While offering strong structural grounding, these methods require complex search over entities and relations and are often time-consuming. In contrast, we use a more efficient search scheme that operates only over relations, significantly reducing cost. We further route simple questions to a small specialized model, reducing cost even further.

Retrieval-based KGQA. These methods typically train specialized models to generate retrieval plans in a single pass, substantially reducing cost. *Reasoning-path methods* generate relation paths for KG execution. RoG (Luo et al., 2024b) predicts relation sequences with a fine-tuned LLM. GCR (Luo et al., 2025) further constrains decoding with a KG-Trie to ensure path validity. GNN-RAG (Mavromatis and Karypis, 2025) scores candidate entities with GNNs and extracts shortest paths as reasoning evidence. *Logical-form methods* instead generate executable queries, such as S-expressions or SPARQL. ChatKBQA (Luo et al., 2024a) generates S-expressions that are converted to SPARQL, while MemQ (Xu et al., 2025) reconstructs reasoning steps into query statements through tool calling. However, both paradigms struggle with implicit constraints within the question: path-based methods cannot explicitly encode them in specialized models, while logical-form methods handle them in nested formal languages that are harder to learn (Tian et al., 2025b).

A central challenge of KGQA is *relation hallucination*, i.e., generating relations absent from the target KG (Tian et al., 2025b). Existing methods mitigate this mainly by restricting the search space, such as KG-Trie-constrained decoding (Luo et al., 2025), subgraph-based path extraction (Mavromatis and Karypis, 2025), and training-set relation libraries (Xu et al., 2025). However, these strategies do not address the weak out-of-domain generalization of specialized models, which limits their performance upper bound. ChatKBQA (Luo et al., 2024a) instead repairs erroneous relations via exhaustive post-hoc search, which is prohibitively

slow for complex multi-hop queries.

Model Cascades and Routing. Another line of work reduces inference cost by combining models of different sizes. Routing methods train a router to predict question difficulty or model preference before generation (Ding et al., 2024; Ong et al., 2025; Rao et al., 2025; Deng et al., 2025, 2026a). Cascade methods first query a small model and invoke a larger model when its output fails a check, such as a learned scoring function (Chen et al., 2024a), answer consistency (Yue et al., 2024), or LLM self-verification (Aggarwal et al., 2024). Speculative decoding similarly uses a large model to verify the drafts of a small model (Leviathan et al., 2023). However, these signals are estimated by the models themselves. In contrast, we verify the output of the specialized model directly against the KG, and the general model only repairs the unreachable main path while keeping the constraints, rather than answering the question from scratch.

Our method builds on reasoning-path methods with explicit constraint generation, improving accuracy. To further address OOD-induced hallucination, we route to a general model when specialized models fail to handle the question.

3 Method

3.1 Problem Formulation

A knowledge graph is defined as

$$\mathbb{G} = \{(s, r, o) \mid s \in \mathbb{E},\ r \in \mathbb{R},\ o \in \mathbb{E} \cup \mathbb{L}\}, \quad (1)$$

where $\mathbb{E}$, $\mathbb{R}$, and $\mathbb{L}$ denote the sets of entities, relations, and literal values, respectively. Given an N-hop question q over $\mathbb{G}$, the goal is to recover the reasoning path

$$\mathbf{P} = \langle (s_1, r_1, o_1), \ldots, (s_N, r_N, o_N) \rangle, \quad (2)$$

where o_N is the final answer to q, and adjacent triples satisfy $s_{i+1} = o_i$ for $i = 1, \ldots, N-1$.

To solve this problem, we propose `RouterKGQA`, as illustrated in Figure 1. The framework consists of three stages. First, we train a specialized model, which maps a natural-language question to a *Constraint-aware Reasoning Path* (CRP), which contains a main path and its associated constraints (§3.2). Second, if the main path is unreachable, an agent-based workflow repairs it via beam search, which requires fewer LLM calls than other methods (§3.3). Finally, the corrected CRP is converted into a SPARQL query to retrieve the final answer (§3.4). The prompts we used are in Appendix A.

3.2 Constraint-Aware Path Generation with Specialized LLMs

We describe how the specialized model generates a CRP from a natural-language question, and reachability verification that determines whether repair is needed (Stage 1).

Initial entity. A CRP starts from a topic entity name s_1. We let the specialized model first generate the initial entity, then ground this predicted name to its corresponding entity using ELQ (Li et al., 2020), yielding the grounded topic entity $s_1 \in \mathbb{E}$.

Main path. Following Luo et al. (2024b), we represent the reasoning path as a flat sequence of relations rather than as a nested logical form. This format is easier for small LLMs to learn because it avoids recursive syntax and function tokens, and it keeps the main path separate from constraints, enabling independent reachability verification and progressive constraint relaxation. Specifically, from $\mathbf{P}$ we extract the relation sequence $\mathbf{R} = \langle r_1, r_2, \ldots, r_N \rangle$ as the main path. Starting from s_1, we follow the relations in $\mathbf{R}$ hop by hop to reach the answer o_N.

Constraint set. However, a relation path alone is often insufficient because, in a KG, a pair (s, r) may correspond to multiple objects, yielding multiple candidate answers. To resolve this ambiguity, building on prior work, we equip the specialized model with the ability to generate constraints. We generate constraints over intermediate entities, which is defined as $C = (i, r_i^c, o_i^c)$, requiring o_i to satisfy relation r_i^c with value or condition o_i^c. Formally, this restricts o_i to

$$\mathbb{E}_i^c = \{e \in \mathbb{E} \mid (e, r_i^c, o_i^c) \in \mathbb{G}\}. \quad (3)$$

For example, for the question "Which U.S. president graduated from Harvard and assumed office after 2000?", the main path $\mathbf{R}$ = ⟨country.presidents, president.office_holder⟩ retrieves candidate presidents. We then generate C = (2, education.institution, Harvard) to require the candidate to be linked to Harvard, and C = (2, position.from, ≥"2000") to retain only those whose term started in or after 2000.

Specialized Model Training. To construct instruction fine-tuning data, we convert the SPARQL annotations in the training set into gold CRPs using the deterministic procedure described in Appendix B. Each training instance pairs a natural-language question as input with the processed gold

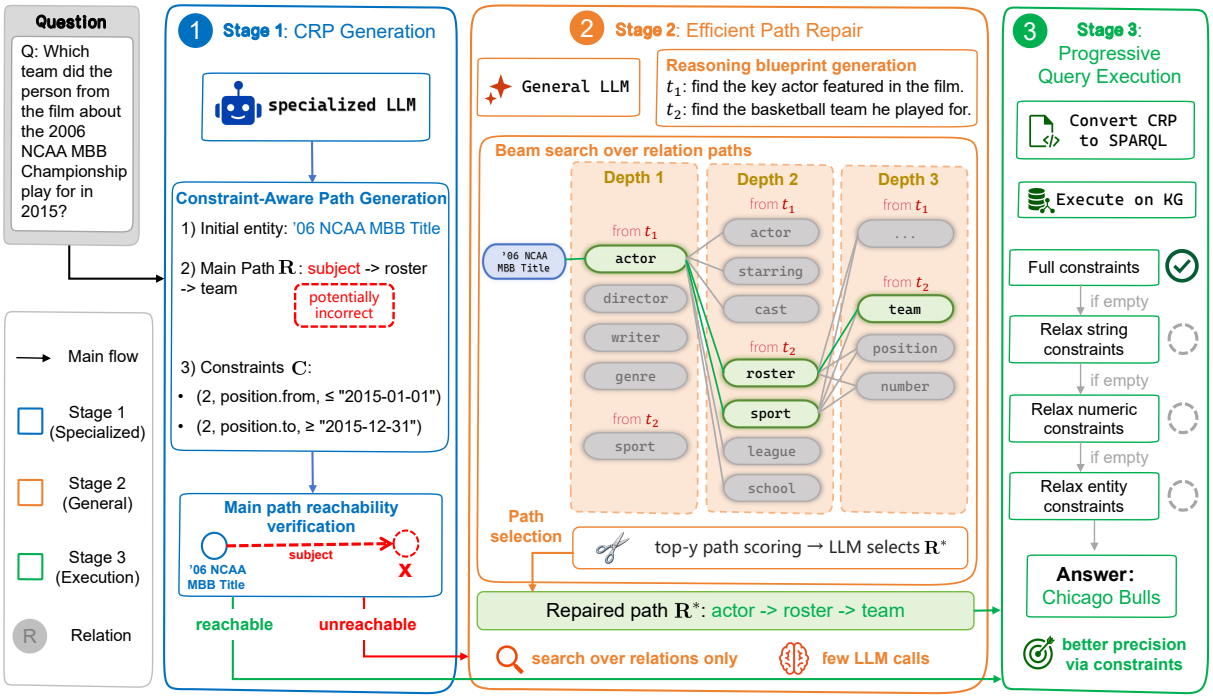


Figure 1: Overview of the RouterKGQA framework.

CRP as output, following the instruction template in Appendix A.1. We fine-tune the specialized model with LoRA (Hu et al., 2022); implementation details are provided in Section 4.1.

Main path reachability verification. The generated main path may still be inconsistent with the KG because of relation errors, direction mismatches, or structural mistakes. We therefore execute the skeleton query induced by $\mathbf{R}$ on $\mathbb{G}$. If it returns a non-empty result set, the main path is reachable. Otherwise, we assume that the specialized model has failed on this case and invoke the general model.

3.3 Efficient Path Repair with General LLMs

The specialized model is accurate and cost-effective on in-domain questions, but often fails under domain shift. When reachability verification fails, we treat the case as beyond the capability of the specialized model and route the question to a stronger agent workflow (Stage 2). To keep this stage efficient, we avoid iterative search over entities and relations. Instead, we first ask the general model to generate a sequence of high-level subtasks in one shot, and then recover an executable relation path via KG-guided beam search.

Reasoning blueprint generation. Given a question q, the general model decomposes it into a sequence of textual subtasks

$$\mathbf{T} = \langle t_1, t_2, \ldots, t_M \rangle, \tag{4}$$

where each t_k describes one semantic step in the reasoning process. These blueprints provide semantic guidance for search, but are not assumed to align one-to-one with KG relations.

Beam search over relation paths. To map the above subtasks to a relation path, we search over relation sequences rather than joint entity–relation states. We describe one beam-search step; the depth index and beam-member position are kept implicit (their iteration is described at the end of the paragraph). Let

$$\mathcal{B} = \{\mathbf{R}_1, \ldots, \mathbf{R}_w\} \tag{5}$$

denote the current beam, where w is the beam width and each

$$\mathbf{R} = \langle r_1, r_2, \ldots, r_\ell \rangle \tag{6}$$

is a partial relation path whose length ℓ equals the current search depth.

For each partial path $\mathbf{R} \in \mathcal{B}$, we first compute the set of entities reachable from the topic entity s_1:

$$\mathcal{E} = \mathrm{Reach}(s_1, \mathbf{R}), \tag{7}$$

where $\mathrm{Reach}(s_1, \mathbf{R})$ denotes the set of entities reached by following the relation sequence $\mathbf{R}$

from s_1 on $\mathbb{G}$. For initialization, $\mathcal{B} = \{\langle\rangle\}$ and $\mathcal{E} = \{s_1\}$. We then enumerate all outgoing relations from these reachable entities:

$$\mathcal{R}^{\text{next}} = \{\, r \mid \exists e \in \mathcal{E},\ \exists o,\ (e, r, o) \in \mathbb{G} \,\}. \quad (8)$$

Relation selection. We first perform coarse filtering to retain relations that are most relevant to the blueprint. Because blueprint steps and KG relations are not always perfectly aligned, we score each candidate relation against every blueprint t_k using sentence-BERT (SBERT; Reimers and Gurevych, 2019) and keep the top-x relations:

$$\widetilde{\mathcal{R}}_k = \text{Top}_x\big(\mathcal{R}^{\text{next}}; \text{sim}(t_k, r)\big), \quad (9)$$

where $\text{Top}_x(\mathcal{S}; f(r))$ denotes the subset of $\mathcal{S}$ containing the x elements with the largest values of $f(r)$, where $r \in \mathcal{S}$. $\text{sim}(t_k, r)$ is the cosine similarity between the SBERT embeddings of t_k and r.

We then expand each partial path in the beam with the filtered relations and obtain the candidate set

$$\widetilde{\mathcal{P}} = \bigcup_{\mathbf{R} \in \mathcal{B}} \bigcup_{k=1}^{M} \{\, \mathbf{R} \oplus r \mid r \in \widetilde{\mathcal{R}}_k \,\}, \quad (10)$$

where $\oplus$ appends a relation to a path.

Path selection. To further reduce the search space, we score each expanded path against the full question q and retain the top-y candidates:

$$\widehat{\mathcal{P}} = \text{Top}_y(\widetilde{\mathcal{P}}; \text{sim}(q, \mathbf{R})), \quad (11)$$

where each path $\mathbf{R}$ is linearized by concatenating its relation names.

Finally, the general model further reduces the candidate set by selecting the best w paths from $\widehat{\mathcal{P}}$, which form the next beam (replacing $\mathcal{B}$ in the next iteration). Each search step therefore requires only one LLM call, while the remaining filtering is handled by small SBERT at negligible cost. Including the initial blueprint generation, the total number of LLM calls is at most $d+1$, where d is the maximum search depth. Moreover, because every expansion is derived from KG-reachable entities, all candidate paths in the beam remain executable.

Stopping criterion. To determine when to stop the search, we use the hop count predicted by the specialized model as the maximum search depth d. Even when the specialized model predicts an incorrect relation path, its depth prediction remains highly reliable: with Llama-2-7B, the predicted depth matches the ground-truth depth in 97.20% of cases on CWQ and 98.70% on WebQSP. After depth d, we reduce the beam size to 1 and return the remaining path as the corrected main path $\mathbf{R}^*$.

3.4 Progressive Query Execution

In this section, we explain how the corrected CRP is converted into an executable SPARQL query and retrieve answers from the KG. We also describe the progressive constraint relaxation strategy for improving robustness when full execution fails.

Execution with full constraints. After correction, we deterministically convert the CRP into an executable SPARQL query. The main path $\mathbf{R} = \langle r_1, \ldots, r_N \rangle$ is mapped to a chain of triple patterns rooted at the topic entity s_1, with each r_i corresponding to one hop. Each constraint $C = (i, r_i^c, o_i^c)$ is then translated into an additional condition: entity constraints are converted into triple patterns requiring the intermediate entity o_i to connect to o_i^c via r_i^c, numeric constraints are converted into `FILTER` clauses imposing inequality conditions on numeric or temporal attributes, and string constraints, when present, are converted into literal-matching `FILTER` clauses. Executing the resulting query yields the answer set

$$\mathcal{A} \leftarrow \text{Execute}(\text{ConvertCRP}(\text{CRP}), \mathbb{G}).$$

If execution succeeds, we return $\mathcal{A}$ directly. Details on constructing gold CRPs from SPARQL annotations for training are provided in Appendix B.

Constraint relaxation. If execution with all constraints fails, e.g., due to noisy constraint predictions or KG incompleteness, we progressively relax lower-priority constraints while keeping the corrected main path fixed, and re-execute the simplified query. This strategy improves robustness without discarding the core reasoning chain.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate on two widely used KBQA benchmarks: **WebQuestionsSP (WebQSP)** (Yih et al., 2016) and **Complex WebQuestions (CWQ)** (Talmor and Berant, 2018), using Freebase (Bollacker et al., 2008) as the underlying KG. Compared with WebQSP, CWQ generally contains questions with greater hop depth and more complex constraints, and is therefore substantially

Category	Method	WebQSP		CWQ	
		Hits@1	F1	Hits@1	F1
LLM-only	Llama-2-7B (Touvron et al., 2023)	53.87	35.67	22.74	18.11
	Llama-3.1-8B (Llama Team, 2024)	55.96	31.61	30.81	21.83
	GPT-4o-mini	64.86	48.59	40.89	40.89
IR-based	KV-Mem (Miller et al., 2016)	46.70	34.50	21.10	15.70
	PullNet (Sun et al., 2019)	68.10	–	47.20	–
	EmbedKGQA (Saxena et al., 2020)	66.60	–	44.70	–
	NSM+h (He et al., 2021)	74.30	67.40	48.80	44.00
	TransferNet (Shi et al., 2021)	71.40	–	48.60	–
	Subgraph Retrieval (Zhang et al., 2022)	68.90	64.10	50.20	47.10
LLM-based	ToG (GPT-4o-mini) (Sun et al., 2024)	77.20	50.04	60.67	41.90
	PoG (GPT-4o-mini) (Chen et al., 2024b)	83.13	61.15	64.67	55.56
	InteractiveKBQA (GPT-4) (Xiong et al., 2024)	72.47	–	59.17	–
	EffiQA (Dong et al., 2025)	82.90	–	69.50	–
	RoG (Llama-2-7B) (Luo et al., 2024b)	86.18	70.24	61.71	54.98
	ChatKBQA (Llama-2-7B)† (Luo et al., 2024a)	86.33	82.13	85.16	80.94
	MemQ (Xu et al., 2025)	88.74	77.90	88.16	75.56
	GNN-RAG (Mavromatis and Karypis, 2025)	90.73	73.49	68.65	60.45
	GCR (Llama-3.1-8B + GPT-4o-mini) (Luo et al., 2025)	**91.68**	75.27	74.64	62.94
Ours	RouterKGQA (Llama-2-7B + GPT-4o-mini)	90.73	**86.43**	88.96	83.72
	RouterKGQA (Llama-3.1-8B + GPT-4o-mini)	91.15	86.38	**89.66**	**83.78**

Table 1: Performance comparison on WebQSP and CWQ (%). Results for methods without public code are taken from the original papers; all other results are reproduced in our environment using the same Freebase snapshot. †The original ChatKBQA paper reports 86.0 Hits@1 / 81.3 F1 on CWQ with Llama-2-13B under its golden-entity setting; we report our reproduction under the unified protocol.

Category	Method	Hits@1	F1	Avg. # LLM Calls	Avg. # Tokens	Avg. Cost (USD) / 10K Qs
Retrieval-based	GNN-RAG	90.73	73.49	2.00	778.17	0.56
	RoG	86.18	70.24	2.00	592.55	0.45
	ChatKBQA	86.33	82.13	1.00	208.08	0.46
	GCR	91.68	75.27	2.00	725.57	0.89
	MemQ	88.74	77.90	1.00	175.62	0.21
Agent-based	ToG	77.20	50.04	10.91	8,426.59	19.35
	PoG	83.13	61.15	7.64	4,704.47	8.07
Ours	RouterKGQA (Stage 2+3)	86.07	70.68	3.31	1,353.78	2.57
	RouterKGQA (Full)	91.15	**86.38**	1.15	612.68	0.57

Table 2: Efficiency and performance comparison on WebQSP (%). Cost estimation details are in Appendix D.

more challenging. More detailed dataset statistics and descriptions are provided in Appendix C.1.

Metrics and baselines. Following prior work, we report Hits@1 and F1. We compare against three categories: (1) LLM-only methods; (2) IR-based methods using subgraph retrieval; and (3) LLM-based methods, including agent-based and retrieval-based approaches. Detailed metric definitions are provided in Appendix C.2.

Configuration of RouterKGQA. RouterKGQA uses a LoRA-tuned Llama-2-7B (Touvron et al., 2023) or Llama-3.1-8B (Llama Team, 2024) specialized model for CRP generation, and GPT-4o-mini (T=0) for path repair. The specialized model is trained on (question, gold CRP) pairs derived from SPARQL annotations. For Stage 2, we set the relation filter size to x=4, the path filter size to y=10, and the beam width to w=3, and use SBERT (nomic-embed-text-v1; Nussbaum et al., 2025) for semantic matching. All experiments use seed 42. Implementation and reproduction details are in Appendix D. Gains are stable: the mean−std lower bound over three training seeds (each evaluated three times) still exceeds the strongest prior baseline on both benchmarks (Appendix E.1).

4.2 Main Results

RouterKGQA improves F1 while preserving strong Hits@1. Table 1 shows that RouterKGQA consistently improves F1 without sacrificing Hits@1, indicating that constraint-aware execution

effectively filters spurious candidates. Compared with the strongest prior F1 baseline, RouterKGQA improves F1 by 4.30 points on WebQSP and 2.84 points on CWQ. At the same time, its Hits@1 remains comparable to the strongest prior baseline on WebQSP and surpasses it on CWQ, showing that the F1 gains come from better candidate filtering while preserving the correct answer in most cases. These results are obtained while the general model is invoked on only 7.81% / 9.57% of questions on WebQSP / CWQ, where repair adds +21.78 / +17.42 F1 over Stage 1+3 on the same subset (§4.5).

Agent-based methods are often more effective, but substantially more expensive. Table 2 shows that representative agent-based methods, such as ToG and PoG, outperform simple LLM-only baselines on average, reflecting the benefit of explicit multi-step KG exploration. However, this gain comes at a substantial efficiency cost: compared with retrieval-based methods, agent-based methods require 5.80× more LLM calls and incur 26.67× higher cost on average.

Our agent-based path repair is much cheaper, and model routing reduces the cost further. RouterKGQA (Stage 2+3), i.e., the agent-based-only configuration, already outperforms representative agent-based baselines while reducing cost by 3.14× relative to PoG and 7.53× relative to ToG. Adding model routing further improves the trade-off: compared with the agent-based-only setting, RouterKGQA (Full) reduces cost by 4.51× and LLM calls by 2.88×, while improving F1 by 15.70 points. This shows that routing between retrieval-based generation and agent-based repair is more effective than relying on either strategy alone. The same efficiency–performance advantage carries over to CWQ (Appendix F) and transfers under schema shift to GrailQA's I.I.D., Compositional, and Zero-shot splits without GrailQA-specific fine-tuning (Appendix G). Routing also fires on only a small fraction of questions but concentrates large repair gains where it does fire (Appendix H.1).

4.3 Ablation Study

We examine model routing, stopping criteria, constraint modeling, and representation design.

Agent-based repair brings marginal gains. Table 3 shows that Stage 1 is the primary source of performance, while Stage 2 provides reliable additional improvements. Removing Stage 2 leads to smaller drops than Stage 1 on both benchmarks, but the drop is consistent. This shows that agent-based path repair delivers meaningful marginal gains by correcting unreachable relation paths that the specialized model fails to recover.

Variants	WebQSP		CWQ	
	Hits@1	F1	Hits@1	F1
RouterKGQA	**91.15**	**86.38**	**89.66**	**83.78**
Stage-level ablations				
w/o Stage 1 (Stage 2+3)	86.07	70.68	70.67	59.30
w/o Stage 2 (Stage 1+3)	89.02	84.68	88.05	82.12
Stopping criterion variants				
w/ blueprint step count	88.47	83.96	88.59	82.81
w/ LLM self-judge	90.76	86.11	89.44	83.40
Constraint ablations (on Stage 1+3)				
w/o Entity Constraint	88.65	78.21	87.20	58.32
w/o Numeric Constraint	89.62	81.66	88.67	71.15
w/o String Constraint	89.02	84.58	87.96	80.72
w/o All Constraints	89.45	75.56	88.28	46.60

Table 3: Ablation study of RouterKGQA (%). Constraint ablations are conducted on Stage 1+3.

Depth prediction is robust. In the main experiments, we use the hop count predicted by the specialized model as the stopping criterion. We compare it with two alternative variants as stopping criterion. Using blueprint step count consistently underperforms, indicating that semantic subtasks do not align perfectly with KG hop structure. We further let the general LLM self-judge when to stop. This variant performs closer to the default setting, but is still slightly worse and requires an additional LLM call. These results support using the specialized model to control the depth of agent-based search, achieving the best performance with minimal cost.

Constraints mainly improve precision. Within Stage 1+3, removing all constraints leaves Hits@1 nearly unchanged but substantially reduces F1. This shows that constraints primarily improve precision by filtering spurious candidates rather than changing top-1 reachability. Among the constraint types, entity constraints contribute the most, followed by numeric constraints. Stage 3 constraint relaxation in turn mainly benefits the more constraint-heavy CWQ. Per-module ablation of Stage 2 (LLM path selection, SBERT relation filtering, blueprint generation) and sensitivity to the filter sizes x and y are reported in Appendices E.2 and E.3.

To further verify the effectiveness of our con-

Variants	WebQSP		CWQ	
	Hits@1	F1	Hits@1	F1
GCR (w/o LLM Filter)	92.13	57.00	74.99	49.65
GCR	91.68	75.27	74.64	62.94
GCR (w/ RouterKGQA Filter)	87.13	83.95	80.46	68.05
RouterKGQA	**91.15**	**86.38**	**89.66**	**83.78**

Table 4: Reference comparison with GCR variants (%).

Dataset	Metric	ChatKBQA	RouterKGQA
WebQSP	Skeleton Accuracy	82.43	**85.78**
	Exact Match	62.42	**69.68**
CWQ	Skeleton Accuracy	74.00	**75.62**
	Exact Match	54.23	**54.63**

Table 5: First-prediction generation accuracy (%) of ChatKBQA (S-expression) and RouterKGQA (CRP).

straints, we apply them to the baseline GCR, with results shown in Table 4. The results show that removing GCR filtering yields relatively high Hits@1 but much lower F1, while adding our generated constraint filtering further strengthens this effect, especially on the harder CWQ benchmark. This suggests that its filtering mechanism remains insufficient. Besides, even the best GCR variant remains clearly below RouterKGQA, suggesting that our explicit CRP formulation is more effective than post-hoc filtering over monolithic outputs.

CRPs are easier to generate. We further compare whether CRP generation offers intrinsic advantages over the S-expression format, e.g., ChatKBQA (Luo et al., 2024a). Using the same backbone (Llama-2-7B), we measure: (1) **skeleton accuracy**—whether the predicted structure matches the gold, ignoring entity/literal values; and (2) **exact match**—whether the prediction is fully correct.

As shown in Table 5, CRP achieves higher skeleton accuracy on both benchmarks (+3.35% on WebQSP, +1.62% on CWQ), indicating that the flat triple-sequence format is structurally easier for a 7B model to learn than nested S-expressions with function keywords. The advantage is more pronounced on exact match (+7.26% on WebQSP), confirming that CRP's explicit constraint separation also improves value-level prediction.

4.4 Analysis by Query Complexity

Table 6 jointly analyzes RouterKGQA's performance across two complexity dimensions on CWQ.

Advantage over GCR across composition types. RouterKGQA achieves consistent and large improvements over GCR across all four CWQ composition types. The advantage is most pronounced on **comparative** (+32.40 F1) and **superlative** (+34.65 F1) queries—precisely the types that demand explicit numeric and ordering constraints, where GCR's constraint-free path generation is most inadequate. Even on simpler conjunction and composition types, RouterKGQA maintains substantial margins (+18.43 and +19.95 F1), demonstrating broad improvements across all query structures.

Dimension	Category	N	Full	S1+S3	ΔF1	GCR
Composition Type	Conjunction	1575	**85.95**	83.18	+2.77	67.52
	Composition	1546	**82.34**	81.91	+0.43	62.39
	Comparative	213	**82.02**	80.88	+1.14	49.62
	Superlative	197	**79.70**	76.53	+3.17	45.05
Constraint Complexity	0	1176	**85.26**	84.70	+0.56	65.33
	1	1511	**84.90**	82.94	+1.96	61.03
	2	633	**78.72**	75.76	+2.96	63.73
	3+	211	**82.76**	80.88	+1.88	61.04

Table 6: CWQ performance by query complexity (F1, %). ΔF1 = Full − Stage 1+3. GCR results are included for cross-method comparison.

Role	Model	WebQSP		CWQ	
		Hits@1	F1	Hits@1	F1
Specialized (Stage 1)	Qwen2.5-7B	88.04	84.67	88.19	81.32
	Qwen3-8B	88.04	84.80	89.32	83.32
	Llama-2-7B	90.73	**86.43**	88.96	83.72
	Llama-3.1-8B	**91.15**	86.38	**89.66**	**83.78**
General (Stage 2)	Qwen2.5-7B	90.79	86.25	89.49	83.63
	Llama-2-7B	89.63	85.30	88.25	82.67
	Llama-3.1-8B	89.57	**86.90**	89.47	83.35
	ChatGPT	89.38	85.20	89.21	83.49
	GPT-4o-mini	**91.15**	86.38	**89.66**	**83.78**

Table 7: Impact of model choice in RouterKGQA's specialized and general roles.

Stage 2 contribution amplified by constraints. Stage 2's correction contributes most on **superlative** queries (ΔF1 = +3.17), where precise paths are essential for executing ordering constraints, and on **conjunction** queries (ΔF1 = +2.77), which involve multiple joined conditions. The breakdown by constraint complexity reveals a clear trend: the gain is largest on questions of moderate difficulty, i.e., those with 2 constraints. For overly simple questions, the specialized model already handles them well, while for overly difficult questions, the repair success rate also decreases.

4.5 Specialized–General Model Analysis

Table 7 examines the impact of model choice in each role. When varying the specialized model (fix-

Metric	WebQSP	CWQ
Routed fraction	7.81	9.57
Within-trigger ΔHits@1	+27.34	+16.86
Within-trigger ΔF1	+21.78	+17.42
Routing precision	65.62	63.61
Missed (reachable, Hits@1=0)	6.62	7.55

Table 8: Reachability routing analysis (%). Within-trigger Δ: RouterKGQA minus Stage 1+3 on the routed subset. Routing precision: routed questions ending at Hits@1=1. Missed: share of non-routed questions. Details in Appendix H.

ing GPT-4o-mini as the general model), stronger models produce substantially better results: Llama-3.1-8B outperforms Qwen2.5-7B by +3.11 Hits@1 on WebQSP and +2.46 F1 on CWQ. This confirms that CRP generation quality—which benefits from fine-tuning on KG-structured data—is the primary driver of end-to-end performance.

When varying the general model (fixing Llama-3.1-8B as specialized), the gap between models is notably smaller. Even Llama-2-7B as the general achieves 89.63% Hits@1 on WebQSP, a gap of only 1.52 points with GPT-4o-mini. Among stronger generals, performance differences are within 1.8 points. This is because RouterKGQA's correction task is relatively simple—selecting the best path from a small candidate set—rather than generating answers from scratch, making RouterKGQA deployable with fully open-source models at minimal performance loss. Appendix I further shows that RouterKGQA yields reasoning paths that are more semantically consistent with the original question. On a representative question, RouterKGQA produces a more faithful reasoning path than GCR and ChatKBQA (Appendix J).

Routing is triggered rarely but brings large gains. Table 8 summarizes the routing analysis, with full results in Appendix H. Stage 2 is triggered on only 7.81% / 9.57% of questions on WebQSP / CWQ, but improves Hits@1 by +27.34 / +16.86 and F1 by +21.78 / +17.42 over Stage 1+3 on these questions. This shows that reachability verification selects a small set of questions on which general-model repair is most useful. However, a reachable path may still be semantically incorrect, which happens on 6.62% / 7.55% of the questions that are not routed. We further compare reachability with generation confidence as the routing signal, and find that confidence cannot replace reachability (Appendix H.2).

5 Conclusion

We proposed RouterKGQA, a framework for knowledge graph question answering based on specialized–general model routing, which combines efficient specialized generation with general model repair. We further improve both components: for the specialized model, we introduce Constraint-aware Reasoning Paths for deterministic answer filtering. For the general workflow, we design an efficient KG-guided repair procedure that reduces the cost of multi-round exploration. Experiments on WebQSP and CWQ show that RouterKGQA achieves strong performance gains, while requiring only 1.15 LLM calls per question on WebQSP (1.23 on CWQ). Further analysis shows that the repair stage provides consistent gains at low cost, and that the overall framework remains effective across different query types and model choices.

Limitations

RouterKGQA has several limitations. First, the current CRP representation supports only flat entity, numeric, and string constraints. This keeps the representation simple and easier for specialized models to learn, but does not explicitly capture more nested or compositional constraint structures. Second, the current routing decision uses main-path reachability as a simple and efficient signal. However, a reachable path may still be semantically incorrect, and such cases are not repaired. Our analysis shows that generation confidence cannot replace reachability (Appendix H.2). Incorporating semantic signals, such as consistency between the expected answer type and the retrieved answers, could further improve robustness. Third, like other KGQA pipelines, our framework benefits from accurate entity linking and a sufficiently complete underlying KG; errors in topic-entity disambiguation or missing relations or literals may still affect both correction and final answer filtering. Fourth, training the specialized model requires gold CRPs converted from SPARQL annotations, which are also required by fine-tuned baselines such as ChatKBQA and MemQ. For a new KG without such annotations, only the agent-based configuration (Stage 2+3) can be applied directly. Finally, our experiments are limited to English benchmarks on Freebase (WebQSP and CWQ). Although the framework is general, its transferability to other knowledge graphs, domains, or multilingual settings remains to be validated.

Ethical considerations

Our work adheres to the ACL Ethics Policy and uses publicly available datasets for reproducibility. LLMs may exhibit racial and gender biases, so we strongly recommend users assess potential biases before applying the models in specific contexts. Additionally, due to the difficulty of controlling LLM outputs, users should be cautious of issues arising from hallucinations.

Acknowledgments

We would like to thank the anonymous reviewers and the area chair for their insightful comments and suggestions.

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. 2024. AutoMix: Automatically mixing language models. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, volume 37, pages 131000–131034. Curran Associates, Inc.

Garima Agrawal, Tharindu Kumarage, Zeyad Alghamdi, and Huan Liu. 2024. Can knowledge graphs reduce hallucinations in LLMs? : A survey. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 3947–3960, Mexico City, Mexico. Association for Computational Linguistics.

Kurt Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250. ACM.

Lingjiao Chen, Matei Zaharia, and James Zou. 2024a. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*. Featured Certification.

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024b. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, pages 37665–37691.

Hexuan Deng, Wenxiang Jiao, Xuebo Liu, Jing Li, Min Zhang, and Zhaopeng Tu. 2025. DRPruning: Efficient large language model pruning through distributionally robust optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 29152–29173, Vienna, Austria. Association for Computational Linguistics.

Hexuan Deng, Wenxiang Jiao, Xuebo Liu, Jun Rao, and Min Zhang. 2026a. Rea-rl: Reflection-aware online reinforcement learning for efficient reasoning. *Preprint*, arXiv:2505.19862.

Hexuan Deng, Wenxiang Jiao, Xuebo Liu, Min Zhang, and Zhaopeng Tu. 2024. Newterm: Benchmarking real-time new terms for large language models with annual updates. In *Advances in Neural Information Processing Systems*, volume 37, pages 35760–35795. Curran Associates, Inc.

Hexuan Deng, Xiaopeng Ke, Yichen Li, Ruina Hu, Dehao Huang, Derek F. Wong, Yue Wang, Xuebo Liu, and Min Zhang. 2026b. Cocoreviewbench: A completeness- and correctness-oriented benchmark for ai reviewers. *Preprint*, arXiv:2605.07905.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. 2024. Hybrid LLM: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Zixuan Dong, Baoyun Peng, Yufei Wang, Jia Fu, Xiaodong Wang, Xin Zhou, Yongxue Shan, Kangchen Zhu, and Weiguo Chen. 2025. EffiQA: Efficient question-answering with strategic multi-model collaboration on knowledge graphs. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 7180–7194, Abu Dhabi, UAE. Association for Computational Linguistics.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, et al. 2023. Retrieval-augmented generation for large language models: A survey. *CoRR*, abs/2312.10997.

Yu Gu, Sue Kase, Michelle Vanni, Brian M. Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond I.I.D.: three levels of generalization for question answering on knowledge bases. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 3477–3488. ACM / IW3C2.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8-12, 2021*, pages 553–561. ACM.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2):42:1–42:55.

Yixin Ji, Kaixin Wu, Juntao Li, Wei Chen, Mingjie Zhong, Xu Jia, and Min Zhang. 2024. Retrieval and reasoning on KGs: Integrate knowledge graphs into large language models for complex question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, volume EMNLP 2024 of *Findings of ACL*, pages 7598–7610, Miami, Florida, USA. Association for Computational Linguistics.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12):248:1–248:38.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 9237–9251, Singapore. Association for Computational Linguistics.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, pages 9459–9474.

Belinda Z. Li, Sewon Min, Srinivasan Iyer, Yashar Mehdad, and Wen-tau Yih. 2020. Efficient one-pass end-to-end entity linking for questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6433–6441, Online. Association for Computational Linguistics.

Llama Team. 2024. The Llama 3 herd of models. *CoRR*, abs/2407.21783.

Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, et al. 2024a. ChatKBQA: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, volume ACL 2024 of *Findings of ACL*, pages 2039–2056, Bangkok, Thailand. Association for Computational Linguistics.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024b. 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.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Yuan-Fang Li, Chen Gong, and Shirui Pan. 2025. Graph-constrained reasoning: Faithful reasoning on knowledge graphs with large language models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*, pages 41540–41565. PMLR / OpenReview.net.

Jie Ma, Zhitao Gao, Qi Chai, Wangchun Sun, Pinghui Wang, Hongbin Pei, Jing Tao, Lingyun Song, Jun Liu, Chen Zhang, et al. 2025. Debate on graph: A flexible and reliable reasoning framework for large language models. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 24768–24776. AAAI Press.

Costas Mavromatis and George Karypis. 2025. GNN-RAG: graph neural retrieval for efficient large language model reasoning on knowledge graphs. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pages 16682–16699, Vienna, Austria. Association for Computational Linguistics.

Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1400–1409, Austin, Texas. The Association for Computational Linguistics.

Zach Nussbaum, John Xavier Morris, Andriy Mulyar, and Brandon Duderstadt. 2025. Nomic embed: Training a reproducible long context text embedder. *Trans. Mach. Learn. Res.*, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed

Kadous, and Ion Stoica. 2025. RouteLLM: Learning to route LLMs from preference data. In *The Thirteenth International Conference on Learning Representations (ICLR)*.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Trans. Knowl. Data Eng.*, 36(7):3580–3599.

Jun Rao, Xuebo Liu, Hexuan Deng, Zepeng Lin, Zixiong Yu, Jiansheng Wei, Xiaojun Meng, and Min Zhang. 2025. Dynamic Sampling that Adapts: Iterative DPO for Self-Aware Mathematical Reasoning. *arXiv preprint arXiv:2505.16176*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Apoorv Saxena, Aditay Tripathi, and Partha P. Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4498–4507, Online. Association for Computational Linguistics.

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. 2021. TransferNet: An effective and transparent framework for multi-hop question answering over relation graph. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 4149–4158, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yuan Sui, Yufei He, Nian Liu, Xiaoxin He, Kun Wang, and Bryan Hooi. 2025. Fidelis: Faithful reasoning in large language models for knowledge graph question answering. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pages 8315–8330, Vienna, Austria. Association for Computational Linguistics.

Haitian Sun, Tania Bedrax-Weiss, and William W. Cohen. 2019. PullNet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2380–2390, Hong Kong, China. Association for Computational Linguistics.

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

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.

Chunhao Tian, Yutong Wang, Xuebo Liu, Zhexuan Wang, Liang Ding, Miao Zhang, and Min Zhang. 2025a. AgentInit: Initializing LLM-based multi-agent systems via diversity and expertise orchestration for effective and efficient collaboration. In *Findings of the Association for Computational Linguistics: EMNLP 2025*.

Yuhang Tian, Dandan Song, Zhijing Wu, Pan Yang, Changzhi Zhou, Jun Yang, Hao Wang, Huipeng Ma, Chenhao Li, and Luan Zhang. 2025b. CompKBQA: Component-wise task decomposition for knowledge base question answering. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 293–309, Suzhou, China. Association for Computational Linguistics.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Robin Wagner, Emanuel Kitzelmann, and Ingo Boersch. 2025. Mitigating hallucination by integrating knowledge graphs into LLM inference – a systematic literature review. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 795–805, Vienna, Austria. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Zhexuan Wang, Xuebo Liu, Li Wang, Zifei Shan, Yutong Wang, Zhenxi Song, and Min Zhang. 2026.

MASPO: Joint prompt optimization for LLM-based multi-agent systems. In *Proceedings of the 43rd International Conference on Machine Learning*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, pages 24824–24837.

Guanming Xiong, Junwei Bao, and Wen Zhao. 2024. Interactive-KBQA: Multi-turn interactions for knowledge base question answering with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 10561–10582, Bangkok, Thailand. Association for Computational Linguistics.

Mufan Xu, Gewen Liang, Kehai Chen, Wei Wang, Xun Zhou, Muyun Yang, Tiejun Zhao, and Min Zhang. 2025. Memory-augmented query reconstruction for LLM-based knowledge graph reasoning. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pages 24068–24084, Vienna, Austria. Association for Computational Linguistics.

Xi Ye and Greg Durrett. 2022. The unreliability of explanations in few-shot prompting for textual reasoning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, pages 30378–30392.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, pages 201–206, Berlin, Germany. The Association for Computer Linguistics.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. 2024. Large language model cascades with mixture of thought representations for cost-efficient reasoning. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. 2022. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5773–5784, Dublin, Ireland. Association for Computational Linguistics.

A Prompt Templates

This appendix provides the full prompt templates used in each stage of RouterKGQA. Placeholders are shown in `<angle brackets>`.

A.1 Stage 1: CRP Generation (specialized)

The specialized model is fine-tuned on (question, gold CRP) pairs using the following instruction format:

CRP Generation Prompt

Instruction: Generate a reasoning path that retrieves the information corresponding to the given question.
Input: Question: { `<question>` }
Output: `<structured CRP>`

A.2 Stage 2: Reasoning Blueprint Generation (general)

When the specialized-generated main path is unreachable on the KG, the general LLM decomposes the question into a sequence of single-hop reasoning steps via the following few-shot prompt:

Reasoning Blueprint Prompt

Given a question, generate the reasoning steps without providing the final answer or specific entities.
Q: What countries are located in Eastern Europe and have the country calling code +373?
A: #1 Identify the countries located in Eastern Europe.
#2 Determine which of these countries has the country calling code +373.
Q: What country that trades with Turkey has an ISO numeric code lower than 012?
A: #1 Identify the countries that are trade partners of Turkey.
#2 Determine which of these countries has an ISO numeric code lower than 012.
Q: Who were the inspirations for the author of This Side of Paradise?
A: #1 Identify the author of This Side of Paradise.
#2 Determine the individuals or works that influenced the author.
Q: What countries border the location where the film Amen is set?
A: #1 Identify the location where the film Amen is set.
#2 Determine the countries that border this location.
Q: Lou Seal is the mascot for the team that last won the World Series when?
A: #1 Identify the team for which Lou Seal is the mascot.
#2 Determine the year this team last won the World Series.
Here is the question.
Q: `<question>`
A:

A.3 Stage 2: Path Selection

At each beam search step, after SBERT-based coarse filtering, the general LLM selects the most promising paths from the candidate set. The same prompt template is used for both intermediate steps (selecting up to w paths) and the final step (selecting 1 path), with only the value of n varying:

Path Selection Prompt

Q: '<question>'
Paths from '<start_entity_names>':
Path 1: <entity> -> <relation_1> -> ... -> <end_entity>...
Path 2: <entity> -> <relation_1> -> ... -> <end_entity>...
...
Select up to <n> paths most likely to reach the FINAL answer (not intermediate entities). Reply with path numbers only, e.g.: Path 1, Path 2

During intermediate beam search steps, n is set to the beam width w; at the final step (maximum depth reached), n=1 to select a single best path.

B SPARQL-to-CRP Conversion

Gold CRPs used for specialized training are deterministically derived from SPARQL annotations via the following rule-based procedure.

Step 1: Subgraph construction and main path extraction. We construct a directed graph from all triple patterns in the `WHERE` clause, where each variable or named entity is a node and each triple pattern (s, r, o) forms a directed edge. The topic entity s_1 is identified as the Freebase MID that lies on the path to the `SELECT` target variable o_N. We then extract the path from s_1 to o_N in this graph; the relation sequence along this path forms the main path $\mathbf{R} = \langle r_1, \ldots, r_N \rangle$. Since constraint branches attach to intermediate variables but do not connect to o_N, this path is unique in the constructed graph.

Step 2: Constraint classification. All triple patterns and `FILTER` clauses not on the main path are classified into three constraint types, each represented as $C = (i, r_i^c, o_i^c)$:

- **Entity restrictions.** Starting from each intermediate entity o_i on the main path, we search over the remaining subgraph. Any branch that terminates at a named KG entity (i.e., a Freebase MID not on the main path) yields an entity restriction, where o_i^c is the target entity.
- **Numeric restrictions.** Triples whose object is a numeric or temporal literal (e.g., `xsd:dateTime`, `xsd:float`) yield a numeric restriction, where o_i^c encodes both the comparison operator ($=, \geq, \leq, >$) and the threshold value, extracted from the associated `FILTER` clause. `ORDER BY` clauses with `LIMIT 1` are also converted to numeric restrictions with o_i^c set to `argmax` or `argmin`.
- **String restrictions.** Triples whose object is a string literal (optionally with a language tag) yield a string restriction, where o_i^c is the literal string value.

Round-trip verification. We verify the fidelity of this conversion by round-trip testing (SPARQL $\rightarrow$ CRP $\rightarrow$ SPARQL): both directions use deterministic rules.

C Experimental Setup Details

C.1 Dataset Details

We evaluate RouterKGQA on two standard Freebase-based KBQA benchmarks: WebQuestionsSP (WebQSP) (Yih et al., 2016) and Complex WebQuestions (CWQ) (Talmor and Berant, 2018). WebQSP contains 4,737 questions and is commonly used to evaluate KBQA systems on relatively simpler semantic parsing and multi-hop reasoning cases. CWQ is substantially larger, with 34,689 questions, and is designed to emphasize more complex compositional reasoning. Compared with WebQSP, CWQ contains a much larger proportion of deeper multi-hop questions and questions with richer constraints, making it a more challenging benchmark for constraint-aware KGQA. Table 9 reports the distributions of question depth and constraint complexity on the test sets of the two benchmarks.

Aspect	Category	CWQ		WebQSP	
		#	%	#	%
Depth	1-hop	854	24.19	1,042	63.58
	2-hop	1,859	52.65	592	36.12
	3-hop+	818	23.17	5	0.31
Constraints	0	1,176	33.31	1,169	71.32
	1	1,511	42.79	332	20.26
	2	633	17.92	92	5.61
	3+	211	5.98	46	2.81

Table 9: Test-set distributions of question depth and constraint complexity on WebQSP and CWQ.

C.2 Metric Definitions

Prior KBQA methods adopt inconsistent definitions of Hits@1, which complicates fair comparison across approaches. We identify two answer-retrieval paradigms and describe how each has computed this metric.

Query-based methods. These methods retrieve answers by executing a structured query (e.g., SPARQL) against the KG. The returned answer set is inherently *unordered*. ChatKBQA (Luo et al., 2024a) counts Hits@1=1 if the answer set of the first executable predicted query intersects with the gold answer set. In contrast, MemQ (Xu et al., 2025) considers only the *first* returned entity—whose position is determined by the query engine and does not carry ranking semantics—and checks whether it belongs to the gold answer set.

LLM-filtering methods. These methods use an LLM to select or generate answers from candidate entities. Both RoG (Luo et al., 2024b) and GCR (Luo et al., 2025) compute the metric identically: checking whether the LLM-filtered answer set has a non-empty intersection with the gold answer set. However, RoG labels this metric Hits@1, while GCR refers to it as Hit, despite the underlying computation being the same.

Unified evaluation protocol. To ensure a fair comparison across methods, we standardize the computation as follows:

- For **query-based methods**, Hits@1=1 if the answer set returned by the *first executable query* has a non-empty intersection with the gold answer set.
- For **LLM-filtering methods**, Hits@1=1 if the answer set generated by the *first LLM reasoning step* has a non-empty intersection with the gold answer set.

F1 is computed using standard set-level precision and recall between the predicted and gold answer sets. This unified protocol ensures that Hits@1 reflects answer correctness rather than implementation-specific ordering effects.

D Implementation Details

D.1 Training Details

The specialized CRP generation model is fine-tuned using LLaMA-Factory[1] with LoRA (Hu et al., 2022). We train dataset-specific specialized models for evaluation on WebQSP and CWQ. We apply the same training recipe to both Llama-2-7B and Llama-3.1-8B backbones unless otherwise specified. The main optimization settings are shared across datasets, while the number of training epochs differs between WebQSP and CWQ. Table 10 summarizes the main training hyperparameters.

Hyperparameter	WebQSP	CWQ
Stage 1: CRP Generation (Specialized Model)		
Fine-tuning Type	LoRA	LoRA
Learning Rate	5e-5	5e-5
LR Scheduler	Cosine	Cosine
Train Batch Size	4	4
Gradient Accum. Steps	4	4
Effective Batch Size	16	16
Train Epochs	50	10
Precision	bf16	bf16
Stage 2: Path Repair (General Model)		
General Model	GPT-4o-mini	GPT-4o-mini
Temperature T	0	0
Beam Width w	3	3
Relation Filter Size x	4	4
Path Filter Size y	10	10
Semantic Matching	nomic-embed-text-v1	
Stage 3: Progressive Query Execution		
Relaxation Order	string → numeric → entity	

Table 10: Implementation details of RouterKGQA. Stage 1 settings are shared across Llama-2-7B and Llama-3.1-8B unless otherwise noted.

D.2 Cost Estimation

Costs reported in Tables 2 and 16 are estimated using Llama-2-7B pricing ($0.05/1M input, $0.25/1M output) and Llama-3.1-8B pricing ($0.10/1M input, $0.10/1M output) from Artificial Analysis, and official GPT-4o-mini pricing ($0.15/1M input, $0.60/1M output) from OpenAI, all accessed 2026-02-14.

D.3 Baseline Reproduction

Table 11 summarizes the reproduction status of all compared methods. For methods with publicly available code, we reproduce results in our environment using the same Freebase snapshot to ensure fair comparison. All reproduced methods follow the original paper's settings unless otherwise noted. For methods without public code, we report results directly from the original papers.

[1] https://github.com/hiyouga/LLaMA-Factory

Category	Method	Reprod.	Notes
LLM-only	Llama-2-7B	✓	Direct inference
	Llama-3.1-8B	✓	Direct inference
	GPT-4o-mini	✓	Direct inference
IR-based	KV-Mem	✗	Original paper
	PullNet	✗	Original paper
	EmbedKGQA	✗	Original paper
	NSM+h	✗	Original paper
	TransferNet	✗	Original paper
	Subgraph Retrieval	✗	Original paper
LLM-based	ToG	✓	GPT-4o-mini
	PoG	✓	GPT-4o-mini
	InteractiveKBQA	✗	Original paper
	EffiQA	✗	Original paper
	RoG	✓	Official code
	ChatKBQA	✓	Modified†
	MemQ	✓	Official code
	GNN-RAG	✓	Official code
	GCR	✓	Official code

Table 11: Baseline reproduction status. ✓: reproduced in our environment; ✗: results from the original paper. †The original ChatKBQA uses Llama-2-13B on CWQ and Llama-2-7B on WebQSP; we unify both to Llama-2-7B for fair comparison under the same model scale.

Metric	Paper (seed 42)	Mean ± std (n=9)
WebQSP Hits@1	91.15	91.25 ± 0.12
WebQSP F1	86.38	86.73 ± 0.27
CWQ Hits@1	89.66	89.59 ± 0.40
CWQ F1	83.78	84.08 ± 0.45

Table 12: Multi-seed variance of RouterKGQA across n=9 evaluations (3 training seeds × 3 evaluation runs).

E Additional Ablations and Sensitivity

E.1 Multi-Seed Variance and Bootstrap CI

The paper's main results use seed 42. We additionally re-train the Llama-3.1-8B specialized model with three seeds $\{42, 123, 2024\}$, keeping the training configuration unchanged. Each trained model is evaluated three times through the full pipeline, giving $3 \times 3 = 9$ evaluations per dataset. Stage 2/3 uses GPT-4o-mini at T=0, so the small run-to-run variance mainly comes from API non-determinism.

Tables 12 and 13 show that the variance is small: the total standard deviation is at most 0.45 on all four metrics, and the paper's seed-42 single-run results are close to the multi-seed mean. The mean−std lower bound is 86.46 F1 on WebQSP and 83.63 F1 on CWQ, both above the strongest prior F1 baseline ChatKBQA (82.13 / 80.94) by +4.33 / +2.69 F1. Within-seed std (0.04 to 0.17) is much smaller than the total std (0.12 to 0.45), indicating that most variance comes from training-time stochasticity rather than evaluation-time stochasticity.

Dataset	Seed	Hits@1 mean±std	F1 mean±std
WebQSP	42	91.11 ± 0.07	86.39 ± 0.07
WebQSP	123	91.30 ± 0.07	86.91 ± 0.17
WebQSP	2024	91.34 ± 0.06	86.89 ± 0.04
CWQ	42	89.59 ± 0.08	83.64 ± 0.08
CWQ	123	90.04 ± 0.11	84.64 ± 0.12
CWQ	2024	89.15 ± 0.14	83.98 ± 0.13

Table 13: Within-seed variance across three evaluation runs at T=0 for each training seed.

As an additional descriptive uncertainty estimate, we apply a non-parametric bootstrap with 10,000 resamples over the 9 evaluation results. The resulting 95% confidence intervals are $[86.55, 86.89]$ for WebQSP F1 and $[83.82, 84.37]$ for CWQ F1, both above the strongest prior F1 baselines. Thus, the main F1 gains remain robust across training seeds and evaluation runs.

E.2 Module-Isolation Ablation

Table 3 ablates RouterKGQA at the stage level. Here we further isolate the internal modules of Stage 2 and Stage 3. All ablations modify exactly one component while keeping the main configuration unchanged, and are run on the full WebQSP and CWQ test sets.

Variant definitions (Stage 2). *w/o Blueprint (question-only SBERT)* skips the blueprint generation and directly uses the original question as the single SBERT query over candidate relations, with the relation filter size doubled to $2x$=8 to keep the candidate-pool size comparable. *w/o SBERT Filtering (BM25)* replaces both the SBERT relation filter and the SBERT path-level filter with BM25Okapi. *w/o LLM Path Selection (random)* replaces the per-step LLM beam selector with a uniform random selection from the top-y candidate paths. *w/o LLM Path Selection (SBERT top-1)* replaces the LLM beam selector with the top-1 path ranked by SBERT.

Variant definition (Stage 3). *w/o Progressive Constraint Relaxation (drop-all)* drops all constraints at once after the first execution failure, instead of progressively relaxing constraints by priority.

Table 14 shows that LLM Path Selection is the most critical Stage 2 module: replacing it with random selection costs −2.28 / −2.25 F1, and replacing it with the SBERT top-1 path still costs −1.59 / −1.61 F1, indicating that the LLM se-

Variant	WebQSP H@1/F1	CWQ H@1/F1
RouterKGQA	**91.15 / 86.38**	**89.66 / 83.78**
Stage 2 ablations		
w/o Blueprint	90.67 / 86.02	89.20 / 83.49
w/o SBERT (BM25)	89.75 / 85.34	88.16 / 82.58
w/o LLM Select (random)	88.35 / 84.10	87.06 / 81.53
w/o LLM Select (SBERT-top1)	89.26 / 84.79	87.71 / 82.17
Stage 3 ablation		
w/o Progressive Relax (drop-all)	91.34 / 86.35	89.75 / 83.09

Table 14: Module-isolation ablation of Stage 2 and Stage 3 components (%).

$x \backslash y$	5	10	15	20	30
2	69.41/58.19	70.25/59.19	70.25/59.61	71.39/60.18	71.39/61.83
4	68.56/58.40	**72.24/61.23**	71.67/61.26	71.39/60.77	72.80/62.27
6	67.99/56.86	73.65/63.51	74.50/64.38	75.35/63.83	74.50/64.30
8	67.42/57.04	71.67/60.94	74.50/63.48	75.35/63.09	72.52/62.01
12	68.84/56.55	70.82/59.84	75.07/63.47	72.80/63.33	74.50/64.05

Table 15: CWQ Hits@1 / F1 (%) on the 353-question sample under varying relation filter size x and path filter size y. The default (x=4, y=10) is bolded.

lector provides semantic discrimination beyond SBERT similarity. SBERT filtering is also important: replacing SBERT with BM25 reduces F1 by 1.04 / 1.20, likely because Freebase relation names such as `country.presidents` share few surface tokens with natural-language questions. Blueprint generation provides a smaller but consistent gain (+0.36 / +0.29 F1). Progressive constraint relaxation in Stage 3 is flat on WebQSP but improves CWQ by +0.69 F1, consistent with CWQ's higher average number of constraints per question.

E.3 Filtering-Size Sensitivity

In the main configuration, Stage 2 uses relation filter size x=4 and path filter size y=10. We sweep $x \in \{2, 4, 6, 8, 12\}$ and $y \in \{5, 10, 15, 20, 30\}$, keeping the other settings unchanged. The sweep is run on the Stage 2+3 configuration to isolate Stage 2 hyperparameters. Each cell is evaluated on a 1/10 sample of the source test set (WebQSP 164 Q / CWQ 353 Q). We report CWQ here because it is more constraint-heavy and exposes the trade-off more clearly; the WebQSP sweep follows the same trend and is omitted for space.

Table 15 shows three trends. First, x and y need to scale together: x=12, y=5 still performs poorly because the path filter is too tight, while x=6, y=15 reaches the best balanced point. Second, once $x \geq 6$ and $y \geq 15$, F1 reaches a plateau around 63–64 on this sample, suggesting that the correct path is usually already retained and the final LLM selection becomes the bottleneck. Third, very small filtering sizes such as x=2, y=5 over-prune the search space and drop F1 by 3–5 points. The default x=4, y=10 is a cost-effective operating point: larger x/y improve sample F1 by 2–3 points but mainly increase the prompt length for LLM path selection (1,881 to 3,192 average tokens per question), while the number of LLM calls remains nearly unchanged because x/y do not affect the Stage 2 call structure, whose upper bound is determined by the search depth d.

We do not sweep the beam width w separately because w is a soft upper bound on the candidate pool that x/y already control, and the total LLM-call count (at most d+1) is independent of w; we fix w=3 in all experiments.

F Efficiency Comparison on CWQ

Table 2 in the main text reports efficiency on WebQSP. Here we mirror the same comparison on CWQ, using identical baselines and metric definitions. Cost estimates follow the same pricing assumptions stated in Table 2.

Table 16 preserves the same trends observed on WebQSP. RouterKGQA (Full) achieves the best F1 (83.78) at 1.23 average LLM calls per question, 11.55× cheaper than PoG and 19.48× cheaper than ToG while improving Hits@1 by +24.99 and +28.99 respectively. The agent-based-only configuration RouterKGQA (Stage 2+3) uses only 4.58 LLM calls per question, already much lower than ToG (13.27) and PoG (13.40), showing that the efficiency gain comes not only from reachability routing but also from the relation-only Stage 2 search design. Adding routing further reduces calls by skipping Stage 2 when the Stage 1 main path is already reachable.

Note on wall-clock latency. We report #LLM Calls, #Tokens, and estimated cost as the main efficiency metrics rather than wall-clock latency. The latter is useful as a system-level reference, but it depends heavily on API concurrency, queuing, batching, network conditions, and hardware/runtime implementation, which makes it less reproducible across environments. The metrics we report are derived directly from model behavior and the pricing assumptions stated in Appendix D, so any reader can re-derive the same numbers.

G Cross-Dataset Generalization

To strengthen the generalization evidence beyond WebQSP and CWQ, we evaluate RouterKGQA on

Category	Method	Hits@1	F1	Avg. # LLM Calls	Avg. # Tokens	Avg. Cost (USD) / 10K Qs
Retrieval-based	GNN-RAG	68.65	60.45	2.00	1,146	0.72
	RoG	61.71	54.98	2.00	838	0.56
	ChatKBQA	85.16	80.94	1.00	249	0.55
	GCR	74.64	62.94	2.00	950	1.17
	MemQ	88.16	75.56	1.00	213	0.29
Agent-based	ToG	60.67	41.90	13.27	10,185	23.96
	PoG	64.67	55.56	13.40	8,381	14.21
Ours	RouterKGQA (Stage 2+3)	70.67	59.30	4.58	2,327	4.20
	RouterKGQA (Full)	**89.66**	**83.78**	1.23	1,015	1.23

Table 16: Efficiency and performance comparison on CWQ (%), mirroring Table 2.

GrailQA (Gu et al., 2021) under a zero-shot setting: the specialized model is trained on WebQSP only and evaluated on GrailQA without any GrailQA-specific fine-tuning. The general model is GPT-4o-mini, and Stage 2/3 use the same main configuration as in the paper. For consistency, the reproduced specialized-model baselines (MemQ and GCR) also use WebQSP-trained models and are not retrained on GrailQA. We use the 1,000-question GrailQA validation subset released by PoG (Chen et al., 2024b), covering the official I.I.D., Compositional, and Zero-shot splits. For PoG we report the corresponding released GrailQA results under its original GPT-3.5-turbo setting as a reference comparison.

Table 17 shows that RouterKGQA achieves the best results among the compared methods on Overall and all three splits. LLM-only methods (IO-Prompt, CoT, SC) all score below 20 Overall F1, confirming that KG grounding is still necessary under schema shift. Among retrieval-based methods, GCR generalizes much better than MemQ because GCR dynamically queries the topic entity's local KG subgraph at test time, while MemQ relies more heavily on reusing SPARQL fragments from the training distribution. Compared with the released PoG results, RouterKGQA improves Overall Hits@1 / F1 by +7.70 / +6.78. The largest gain appears on the Compositional split (+12.63 Hits@1 / +9.40 F1) and RouterKGQA also improves on the Zero-shot split (+5.51 Hits@1 / +6.22 F1). These results provide additional evidence that the CRP-based design transfers beyond WebQSP/CWQ: CRP can be directly executed as a structured query, and when the specialized model fails under schema shift, Stage 2 can still repair the main path by exploring KG structure. Multilingual and non-Freebase KG generalization remain future work.

H Routing Analysis Details

This appendix provides detailed analyses of the reachability-based routing signal: quantitative quality metrics (§H.1), a confidence-based reverse test (§H.2), and a diagnostic breakdown of remaining errors (§H.3).

H.1 Routing Quality Metrics

Table 18 reports diagnostic quantities for the reachability-based routing signal, computed from the per-question logs of the main runs (WebQSP n=1,639; CWQ n=3,531).

Reachability is selective but effective. Stage 2 is triggered on only 7.81% / 9.57% of questions on WebQSP / CWQ, but yields large within-trigger gains of +27.34 / +16.86 Hits@1 and +21.78 / +17.42 F1. This confirms that reachability identifies a small subset of questions on which general-model repair brings concentrated benefits, rather than routing indiscriminately.

Routing precision and recall. The routed subset contains the questions whose Stage 1 main path is unreachable (128 / 338); the remaining 1,511 / 3,193 questions are reachable. Routing precision is the fraction of routed questions whose final answer is correct (Hits@1=1): 84/128 on WebQSP and 215/338 on CWQ. It is not identical to the repair rate of Stage 2: execution tries Stage 1's beam-decoded candidate CRPs in order (Appendix C.2), so on 49 / 158 routed questions a lower-ranked candidate already executes correctly without Stage 2, and Stage 2 repairs 35 / 57 of the remaining 79 / 180. Routing recall requires a positive class; since whether Stage 2 would have helped an untriggered question is not observable, we conservatively count every question that is either routed or reachable with Hits@1=0 as positive (228 / 579), so every reachable-but-wrong case

Category	Method	Overall	I.I.D.	Compositional	Zero-shot
LLM-only	IOPrompt	34.10 / 19.38	30.83 / 15.38	22.73 / 11.74	39.50 / 23.79
	CoT	35.10 / 16.95	30.42 / 17.37	23.74 / 10.36	41.10 / 19.09
	SC	35.60 / 17.13	32.92 / 17.84	24.24 / 11.40	40.75 / 18.84
Retrieval-based	MemQ	3.70 / 2.04	—	—	—
	GCR	66.00 / 48.12	58.75 / 42.56	53.54 / 33.69	73.49 / 55.58
Agent-based	PoG	77.90 / 63.44	76.67 / 62.15	62.12 / 44.21	83.99 / 70.77
Ours	RouterKGQA	**85.60 / 70.22**	**85.42 / 68.06**	**74.75 / 53.61**	**89.50 / 76.99**

Table 17: Cross-dataset generalization on GrailQA, reported as Hits@1 / F1 (%). PoG numbers are taken from its released GrailQA evaluation under the original GPT-3.5-turbo setting.

Metric	WebQSP	CWQ
Trigger rate	7.81%	9.57%
Within-trigger ΔHits@1	+27.34	+16.86
Within-trigger ΔF1	+21.78	+17.42
Routing precision	65.62%	63.61%
Routing recall	56.1%	58.4%
Reach FN (Hits@1=0)	6.62%	7.55%
Reach FN (F1<1)	13.91%	17.32%
Oracle UB ΔHits@1	+1.83	+2.07
Oracle UB ΔF1	+2.24	+2.02

Table 18: Routing quality analysis on WebQSP and CWQ (%). Reach FN rates are computed over the reachable (non-routed) questions.

counts as missed.

Reachable but semantically wrong paths are the main blind spot. Among the reachable questions, 6.62% / 7.55% end at Hits@1=0 (6.10% / 6.83% of all questions), and 13.91% / 17.32% end at F1<1; these cases bypass Stage 2 by design and form the dominant false-negative mode of reachability-based routing.

Stronger routing signals still have headroom. An Oracle Routing UB that, for each question, takes the larger metric value between Stage 1+3 (no routing) and RouterKGQA (with routing) leaves +1.83 / +2.07 Hits@1 and +2.24 / +2.02 F1 over RouterKGQA, suggesting that routing signals beyond reachability remain a meaningful future direction. The confidence-based reverse test in §H.2 further shows that average log-probability cannot replace reachability, and a diagnostic breakdown of the remaining errors is provided in §H.3.

H.2 Reachability vs. Model Confidence

We test whether intrinsic model confidence can replace reachability as a routing signal. On the WebQSP reachable subset, we split questions into five quintiles by the specialized model's average log-probability over its CRP generation, and compare Stage 1+3 with Stage 2+3. If low confidence were a good signal for invoking the general model, Stage 2+3 should outperform Stage 1+3 in the low-confidence quintiles.

Quintile	S1+3 H@1/F1	S2+3 H@1/F1	#C	Depth
Q1 (lowest 20%)	83.95 / 74.17	76.42 / 61.60	0.60	1.44
Q2	90.73 / 85.64	81.58 / 66.89	0.53	1.42
Q3	95.07 / 91.62	81.65 / 67.15	0.35	1.32
Q4	97.68 / 95.70	88.82 / 74.68	0.37	1.40
Q5 (highest 20%)	99.01 / 98.16	94.43 / 82.60	0.11	1.25

Table 19: Confidence-based reverse test on WebQSP. #C: average number of constraints per question. Depth: average gold hop count.

Table 19 shows that Stage 1+3 outperforms Stage 2+3 in all five quintiles. Both configurations also improve from Q1 to Q5, while the average number of constraints decreases from 0.60 to 0.11. This indicates that average log-probability mainly tracks intrinsic question difficulty—especially constraint complexity—rather than identifying cases where the specialized model's path should be repaired by the general model. Reachability is therefore not replaceable by confidence: it is a structured signal grounded in verifying the structured prediction against the KG, whereas confidence is an intrinsic property of the model output.

H.3 Diagnostic Error Analysis

Stage 2 in our pipeline only repairs the main path R; the constraints C come unchanged from Stage 1 in both routed and not-routed cases. Based on this pipeline structure, we classify non-perfect cases into three observable error types:

- **E1: Reachable but semantically wrong path** (¬routed ∧ Hits@1=0). The predicted main path is reachable, so routing does not fire, but the predicted answer set is disjoint from gold.

Type	Count	/ total	/ errors
WebQSP (n=1,639)			
Correct	1,362	83.10%	—
E2	132	8.05%	47.83%
E1	100	6.10%	36.23%
E3	44	2.68%	15.94%
CWQ (n=3,531)			
Correct	2,800	79.30%	—
E2	366	10.37%	50.14%
E1	241	6.83%	33.01%
E3	123	3.48%	16.85%

Table 20: Diagnostic error breakdown on the full WebQSP and CWQ test sets.

- **E2: Answer-set narrowing / constraint error** (Hits@1=1 $\wedge$ F1<1). The predicted answer set intersects gold, so the main path reaches at least one correct answer, but the constraints C fail to narrow the answer set exactly to gold.
- **E3: Triggered but unrepaired path error** (routed $\wedge$ Hits@1=0). The original $\mathbf{R}$ is unreachable, Stage 2 is triggered, but the repaired path $\mathbf{R}^*$ still produces an answer set disjoint from gold.

Table 20 shows that answer-set narrowing / constraint errors dominate, accounting for about half of all errors on both datasets (47.83% / 50.14%). Reachable but semantically wrong paths account for about one third of errors (36.23% / 33.01%). Triggered but unrepaired path errors account for about one sixth of errors (15.94% / 16.85%), making Stage 2 repair the smallest remaining error source under this diagnostic. This suggests that future improvements should prioritize constraint generation and verification, followed by stronger routing signals to reduce reachable-but-wrong paths, before further optimizing Stage 2 repair.

I Sub-Question Consistency Analysis

To assess the semantic alignment between predicted reasoning paths and the original question, we reconstruct each predicted path into a natural-language question q_r using an LLM and decompose both q_r and the original question q into sub-question sets, denoted by A and B, respectively, following the decomposition style of CWQ (Talmor and Berant, 2018). Based on the set relationship between A and B, we classify each example into four categories: EXACT_EQUAL ($A = B$), RECON_SUBSET ($A \subset B$), PARTIAL_OVERLAP ($A \cap B \neq \emptyset$, with neither set containing the other), and DISJOINT ($A \cap B = \emptyset$).

Figure 2(b) confirms that semantic alignment directly determines downstream performance: EXACT_EQUAL yields the best results across all methods, while DISJOINT yields the worst. Figure 2(a) shows that RouterKGQA substantially increases the proportion of EXACT_EQUAL predictions (34.7% vs. 10.8% for GCR and 5.9% for GNN-RAG) while reducing DISJOINT to 1.4% (vs. 24.9% and 46.0%). This demonstrates that RouterKGQA generates reasoning paths that are semantically equivalent or highly consistent with the original question intent.

J Case Study

J.1 Three-Way Comparison

We present a three-way comparison on the CWQ question *"What city contains Ulsan's capital City?"*, whose gold answer is Seoul (`m.0hsqf`). The intended reasoning is to find the country containing Ulsan, then return that country's capital. This case illustrates how each method handles a 2-hop question with a topic-entity type mismatch.

GCR. Output: Ulsan. GCR's constrained decoding generates reasoning paths, but the generated paths do not guarantee that the correct semantic path is retained. In this case, its paths either loop back to Ulsan or move from South Korea to sibling administrative divisions. A representative wrong path is

```
Ulsan
  → location.administrative_
division.country
  → location.country.
administrative_divisions
```

The correct continuation should be `location.country.capital`, but it is absent from the generated path candidates, so Seoul never enters the reader's candidate set.

ChatKBQA. Output: `m.02qkt`, `m.0j0k`, `m.02qnk`, `m.01z5y` (four wrong cities). The decoded logical form is

```
(JOIN (R [location, location,
containedby])
  (JOIN [location, country, capital]
    [Ulsan]))
```

This treats Ulsan as a country and applies a direction-flipped `containedby` relation. The logical form is semantically wrong but happens to be executable on Freebase, returning non-empty

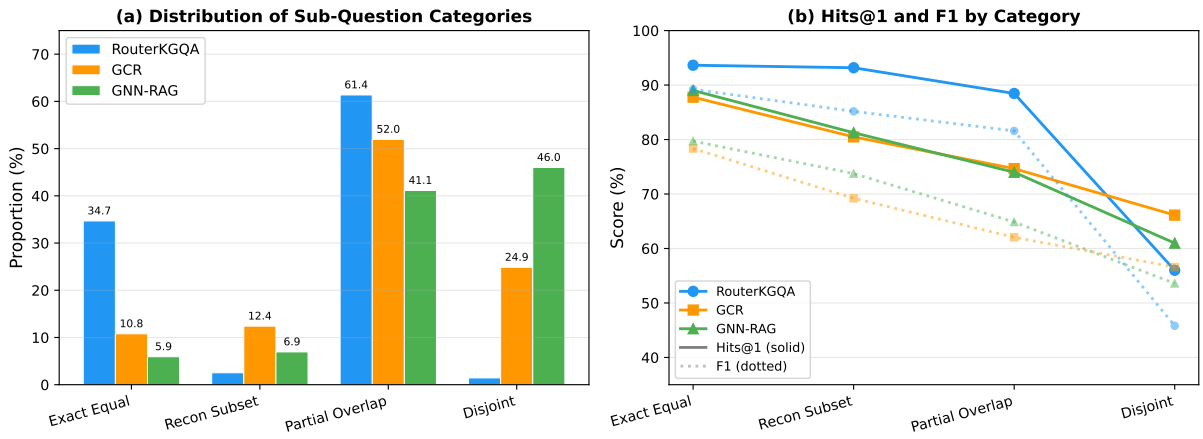


Figure 2: Sub-question consistency analysis on CWQ. (a) Distribution of sub-question set relationship categories across methods. (b) Hits@1 (solid) and F1 (dotted) by sub-question relationship category.

results. ChatKBQA's unexecutable-LF fallback is therefore not triggered, and the executable-but-wrong logical form is retained.

RouterKGQA. Output: Seoul (`m.0hsqf`). Stage 1 predicts the main path

```
Ulsan
  → R[location.country.capital]
  → location.country.capital → ?x
```

which recursively misuses `country.capital`. The skeleton SPARQL returns empty, so Stage 2 is triggered.

Blueprint generation. The general model decomposes the question into the blueprint $\mathbf{T} = \langle t_1, t_2 \rangle$: t_1: *identify the capital city of Ulsan*; t_2: *determine which city contains this capital city*. The depth d=2 is taken from the hop count predicted by the specialized model in Stage 1.

Depth ℓ=1. The beam is initialized as $\mathcal{B} = \{\langle\rangle\}$ with $\mathcal{E} = \{s_1\} = \{\text{Ulsan}\}$ (Eq. 7). Enumerating outgoing relations from Ulsan via Eq. (8) yields $|\mathcal{R}^{\text{next}}|$=21 relations. Applying the SBERT relation filter (Eq. 9) with x=4 against each t_k gives $\widetilde{\mathcal{R}}_{t_1}$:

```
1. aviation.airport.serves
2. location.administrative_
division.country
3. location.administrative_
division.iso_3166_2_code
4. location.location.gns_ufi
```

$\widetilde{\mathcal{R}}_{t_2}$:

```
1. location.location.contains
2. location.location.containedby
3. transportation.road.major_cities
4. location.location.area
```

Expanding the empty beam with these relations (Eq. 10) yields 8 candidate paths in $\widetilde{\mathcal{P}}$. After path-level filtering (Eq. 11) with y=10, all 8 are retained in $\widehat{\mathcal{P}}$. The general LLM is asked to select at most w=3 paths; in this case it returns $|\mathcal{B}|$=2 partial paths:

```
R1: Ulsan → location.administrative_
division.country
R2: Ulsan →
location.location.containedby
```

Depth ℓ=2. For each $\mathbf{R} \in \mathcal{B}$, Eq. (7) gives the reachable entity sets $\mathcal{E}_{\mathbf{R}_1}$={South Korea} and $\mathcal{E}_{\mathbf{R}_2}$={South Korea}. Enumerating outgoing relations from these entities and applying Eq. (9) with x=4 gives $\widetilde{\mathcal{R}}_{t_1}$:

```
1. location.country.capital
2. travel.travel_destination.
tour_operators
3. location.country.iso3166_1_alpha2
4. geography.river.basin_countries
```

$\widetilde{\mathcal{R}}_{t_2}$:

```
1. location.location.contains
2. location.location.partially_contains
3. location.location.containedby
4. location.location.partiallycontains
```

The expansion (Eq. 10) appends these relations to $\mathbf{R}_1$ and $\mathbf{R}_2$ to form $|\widetilde{\mathcal{P}}|$=16 candidate 2-hop paths. Path filtering (Eq. 11) reduces them to $\widehat{\mathcal{P}}$ (y=10); representative members include

```
1. location.administrative_
division.country
  → location.location.contains
2. location.location.containedby
  → location.country.capital
3. location.administrative_
division.country
```

```
  → location.country.capital
4. location.location.containedby
  → location.location.contains
```

Because ℓ has reached d, the beam size is reduced to 1, and the general LLM selects the final corrected main path

```
R*: Ulsan
  → location.location.containedby
  → location.country.capital
```

Stage 3 composes $\mathbf{R}^*$ with the original constraints $\mathbf{C}$, converts the result to SPARQL, and returns Seoul (`m.0hsqf`) on the first try. Notably, `location.location.containedby` is absent from all Stage 1 beams, illustrating that Stage 2 can discover new KG-grounded relations through relation-level search rather than perturbing Stage 1 outputs.

J.2 Constraint Generation and Execution

Constraint type	WebQSP	CWQ
Entity	22.76 (21.97)	46.76 (47.80)
Numeric	8.91 (9.64)	23.25 (24.70)
String	0.31 (0.67)	3.26 (3.45)
$\geq$1 constraint	28.68 (28.62)	65.45 (66.27)

Table 21: Percentage of test questions whose Stage 1 CRP contains each constraint type; gold frequencies in parentheses. Temporal conditions are expressed as numeric restrictions.

Table 21 counts, over the Stage 1 predictions of the main runs, how often each constraint type is generated. The generated distribution closely tracks the gold distribution, so no constraint type is systematically over- or under-produced. Progressive relaxation fires on only 1.28% / 4.81% of questions on WebQSP / CWQ. On these questions the full CRP returns an empty answer set, so any recovered answer is a net gain: relaxation restores Hits@1=1 on 18/21 (85.7%) and 69/170 (40.6%) of them, worth +0.76 / +0.75 overall F1. The following WebQSP examples illustrate the three typical patterns.

The constraint carries the selection logic. For "who did jackie robinson first play for", Stage 1 predicts

```
Main path: Jackie Robinson
  → sports.pro_athlete.teams → ?y
  → sports.sports_team_
roster.team → ?x
Numeric: ?y →
sports.sports_team_roster.from → ?sk0
  ORDER BY ASC(?sk0) LIMIT 1
```

The main path alone retrieves every team Robinson played for, because it cannot express "first". The numeric restriction sorts stints by start date and keeps the earliest, returning exactly the gold answer (F1=1).

A mislinked constraint costs precision. For "who are the senators of hawaii 2012", Stage 1 predicts

```
Main path: Hawaii
  → government.political_
district.representatives → ?y
  → government.government_
position_held.office_holder → ?x
Entity: ?y → government.government_
position_held.office_position_or_title
  → United States Senator
Numeric: ?y → government.government_
position_held.from → ?sk0
  ORDER BY DESC(?sk0) LIMIT 1
```

The object of the entity restriction is resolved to a wrong entity ID, so the full query returns an empty set. Relaxation re-executes the bare main path, which over-generates across offices and years (Hits@1=1, F1=0.25). This is the typical answer-set narrowing error (E2 in Appendix H.3), the largest residual error class.

Relaxation saves a correct main path. For "what kind of cancer did carl wilson have", Stage 1 predicts

```
Main path: Carl Wilson
  → people.deceased_person.
cause_of_death → ?x
Entity: ?x → people.cause_of_death.
parent_cause_of_death → Cancer
```

The constraint object "Cancer" is mislinked to the topic entity's own ID, so the full query returns an empty set. Relaxation removes the constraint, and the main path alone returns exactly the gold answer (F1=1). Without relaxation, a noisy constraint would zero out a correct main path; this is why relaxation drops constraints in priority order but never modifies the main path (§3.4).