GraphRAG-Router: Learning Cost-Efficient Routing over GraphRAGs and LLMs with Reinforcement Learning

Dongzhe Fan[1,2], Chuanhao Ji[1,3], Zimu Wang[4], Tong Chen[4], Qiaoyu Tan[1*]

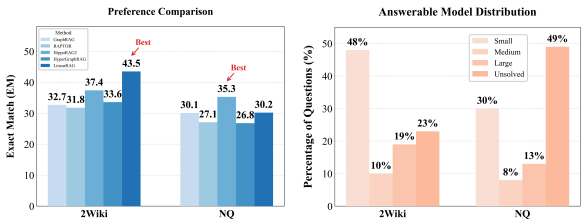
[1]Department of Computer Science, New York University (Shanghai),
[2] Department of Computer Science, New York University,
[3] School of Data Science and Engineering, East China Normal University,
[4]Department of Computer Science, University of Liverpool
{df2362, cj2851, qiaoyu.tan}@nyu.edu, {zimu.wang, tong.chen}@liverpool.ac.uk

Abstract

Graph-based retrieval-augmented generation (GraphRAG) has recently emerged as a powerful paradigm for knowledge-intensive question answering, especially for tasks that require structured evidence organization and multi-hop reasoning. However, existing GraphRAG systems are typically built in a one-size-fits-all manner, relying on a fixed retrieval framework and a single, often large and costly, generator LLM for all queries. This static design limits their ability to adapt to the complexity of varying questions and often incurs unnecessary computational cost. To fill in the gap, we propose GraphRAG-Router, a cost-efficient framework that adopts a hierarchical routing strategy to coordinate heterogeneous GraphRAGs and generator LLMs. Specifically, GraphRAG-Router is first warmed up through supervised fine-tuning and then optimized with a two-stage reinforcement learning procedure, whose second stage introduces a curriculum cost-aware reward to encourage difficulty-aware and economical generator allocation. Extensive experiments on six general-domain and multi-hop QA benchmarks show that GraphRAG-Router consistently outperforms state-of-the-art baselines, reducing the overuse of large LLMs by nearly 30% while maintaining strong generalization capability. Our code is publicly shared at https://github.com/FanDongzhe123/GraphRAG-Router

1 Introduction

Large language models (LLMs) have achieved strong performance across a wide range of tasks (Fan et al., 2026; Sun et al., 2026a; Chen et al., 2025; Liu et al., 2026b; Sun et al., 2025). However, complex knowledge-intensive questions often require reasoning over external, dispersed, and interconnected evidence (Sun et al., 2026b). To address this challenge, graph-based retrieval-augmented generation (GraphRAG) has emerged as a powerful paradigm for knowledge-intensive question answering (QA) (He et al., 2024; Sarthi et al., 2024; Edge et al., 2025; Gutiérrez et al., 2025a), especially for multi-hop questions that require structured evidence organization and compositional reasoning. By organizing knowledge into entities, relations, and higher-level graph structures, GraphRAG enables more explicit evidence aggregation than standard text-based retrieval-augmented generation, making it particularly well suited for complex reasoning over dispersed evidence (Han et al., 2024; Peng et al., 2025).



(a) Dataset preference of GraphRAGs.

(b) Distribution of answerable model scales.

Figure 1: Query-dependent heterogeneity in GraphRAGs and generator LLMs.

Despite these advances, current GraphRAG systems are still largely built in a *one-size-fits-all manner*: they typically rely on a single retrieval framework and a single generator LLM for all queries. Such a static design is inherently suboptimal, because questions vary substantially in both the form of evidence they require and the level of reasoning capacity needed to answer them. On the retrieval side, **different QA benchmarks exhibit markedly different preferences over GraphRAG frameworks** (Figure 1a), suggesting that different questions demand evidence at different levels of granularity. For instance, entity-centric lookup questions are better served by local subgraph retrieval, whereas questions requiring thematic synthesis benefit more from community-level summarization. On the generation side, **existing GraphRAG systems often rely on a large and costly LLM to**

*Corresponding author

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 50460–50491
October 24-29, 2026 ©2026 Association for Computational Linguistics

ensure strong overall performance (Edge et al., 2025; Gutiérrez et al., 2025b) (e.g., GPT-4o (OpenAI et al., 2024)), yet Figure 1b shows that many questions can already be solved correctly by much smaller models. Uniformly invoking a large model therefore wastes computation, while relying only on a smaller model may fail on more challenging queries. These observations indicate that both retrieval infrastructure and generator allocation should be adaptive, so that each query can be matched with the most appropriate evidence source and model capacity. This raises a central question: *How can we adaptively coordinate multiple GraphRAGs and generator LLMs for each query, while maintaining strong QA performance without incurring unnecessary generator cost?*

Solving this problem is non-trivial due to two major challenges. (1) **Combinatorial action space in joint GraphRAG–LLM selection.** While prior studies have explored model-wise routing among candidate LLMs (Zhang et al., 2025; Li, 2025), our setting requires the routing agent to jointly select both the GraphRAG and the generator LLM at each reasoning step. This joint decision substantially expands the decision space compared to model-only routing, and the exponential growth of the action space makes effective optimization particularly challenging. (2) **Cost-efficient routing.** An effective routing strategy must balance answer quality against the computational overhead of generator LLM invocation. Since queries differ widely in their reasoning complexity, the optimal model scale, defined as the minimum capacity required to produce a correct response, is inherently context-dependent. Therefore, the routing policy must calibrate its allocations precisely, as over-provisioning leads to unnecessary computational cost, while under-provisioning risks degrading answer accuracy.

To this end, we introduce **GraphRAG-Router**, a cost-efficient RL framework that enables multi-round joint routing and aggregation across heterogeneous GraphRAG frameworks and generator LLMs. Instead of making a single monolithic dispatch decision over an LLM–GraphRAG pair, we formulate the coordination process as a *hierarchical routing path* that allows the model to select the GraphRAG framework and generator LLM step by step. Specifically, we first warm up the routing policy via supervised fine-tuning (SFT) using curated routing trajectories. We then optimize the policy with a two-stage RL procedure that incorporates three complementary reward signals: a *format reward* encouraging well-structured outputs, an *outcome reward* based on final task correctness, and a *curriculum cost-aware reward* that promotes difficulty-aware model utilization by penalizing unnecessarily expensive routing decisions. Together, these components make GraphRAG-Router a cost-efficient and generalizable solution for multi-round coordination across heterogeneous GraphRAG frameworks and LLMs, achieving state-of-the-art performance on diverse knowledge-intensive QA benchmarks.

The key contributions of this work can be summarized as follows:

- ⋆ We introduce **GraphRAG-Router**, an RL–based framework for multi-round routing and aggregation across heterogeneous GraphRAGs and generator LLMs, enabling question-adaptive routing across retrieval and generation modules.
- ⋆ We propose a hierarchical routing strategy and optimize GraphRAG-Router with a two-stage RL procedure with a curriculum cost-aware reward, allowing the model to route queries to appropriate GraphRAG–LLM pairs while better balancing performance and LLM cost.
- ⋆ Experimental results on six QA benchmarks demonstrate that GraphRAG-Router consistently outperforms strong baselines, achieving superior cost-efficiency, robust generalization, and state-of-the-art overall performance.

2 Related Work

GraphRAG. GraphRAG extends the conventional RAG paradigm by structuring retrieved knowledge as entities and relations, thereby providing stronger support for multi-hop reasoning and knowledge-intensive QA. Early GraphRAG methods (Edge et al., 2025; Sarthi et al., 2024) typically construct knowledge graphs with entities as nodes and relations as edges. Recent works (Guo et al., 2025; Zhuang et al., 2025; Gutiérrez et al., 2025a,b) further enhance the representation power of GraphRAG by incorporating richer graph structures (e.g., HyperGraph), thereby improving evidence organization and multi-hop reasoning over complex knowledge, typically with the support of large-scale LLMs. Meanwhile, another line of work explores GraphRAG systems (Luo et al., 2025a; Yang et al., 2025b; Yu et al., 2026) that rely on smaller models, whose reasoning capabilities are enhanced through RL. In this study, we focus

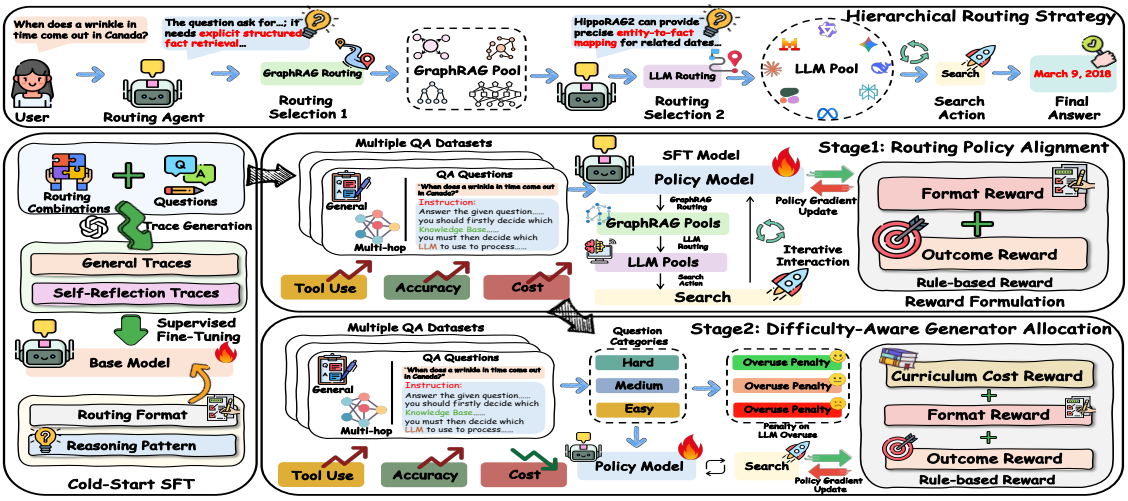


Figure 2: **Overview of the GraphRAG-Router framework. GraphRAG-Router utilizes a hierarchical routing strategy. Initialized with cold-start SFT, it adopts a two-stage RL training paradigm. Stage 1 helps the model learn effective routing behavior, while Stage 2 further optimizes the trade-off between performance and cost.**

on the setting where GraphRAG frameworks are paired with large and costly LLMs, which often incurs substantial computational cost and still relies on a fixed retrieval infrastructure.

LLM Routing. Current LLM routing frameworks aim to select appropriate generator models from a candidate pool according to query complexity. Existing studies have explored both predictive model routing (Li, 2025; Feng et al., 2025) and multi-round routing strategies over candidate LLMs, including RL-based approaches such as Router-R1 (Zhang et al., 2025) that interleave internal reasoning with dynamic model selection and response aggregation. However, these methods focus solely on routing among generator LLMs. In contrast, our setting requires joint routing over both GraphRAG frameworks and generator LLMs, which introduces a substantially larger and more complex action space.

3 Problem Definition

We formulate our task as follows: Given a query q, a candidate pool of GraphRAGs $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$, and a candidate pool of generator LLMs $\mathcal{L} = \{l_1, l_2, \ldots, l_m\}$, the goal is to learn a routing policy π_θ that selects GraphRAGs and generator LLMs over multiple rounds to produce the final answer a under the instruction $\mathcal{T}$. At each round t, the router first performs internal reasoning $\mathcal{R}_t$ to select a GraphRAG $g_t \in \mathcal{G}$ and a generator LLM $l_t \in \mathcal{L}$ based on the current reasoning state s_t. The selected GraphRAG–LLM pair then returns information $I_t = l_m\big(q, g_t(q)\big)$. The information is incorporated into the state for subsequent states, progressively constructing the final answer:

$$P(\mathcal{R}, a \mid q, \mathcal{T}, I) = \underbrace{\prod_{t=1}^{T_r} P(\mathcal{R}_t \mid \mathcal{R}_{<t}, q, \mathcal{T}, I_{<t})}_{\text{Reasoning Process}} \cdot \underbrace{\prod_{t=1}^{T_a} P(a_t \mid a_{<t}, \mathcal{R}, q, \mathcal{T})}_{\text{Answer Generation}},$$

where T_r and T_a are the number of tokens in the reasoning sequence and answer, respectively. $I_{<t}$ denotes all the returned information up to token t.

4 Methods

In this section, we detail GraphRAG-Router into four components: Section 4.1 describes the hierarchical routing strategy over candidate GraphRAGs and generator LLMs. Section 4.2 introduces the SFT paradigm, which equips the model with well-structured reasoning and routing formats. Sections 4.3 and 4.4 describe two complementary RL-based training stages: Stage 1 enables the model to acquire core routing behavior, while Stage 2 further regulates the invocation of generator LLMs to achieve cost-efficient routing.

4.1 Hierarchical Routing Strategy

To make GraphRAG–LLM coordination tractable, we formulate GraphRAG-Router as a hierarchical routing strategy that progressively selects retrieval infrastructures and generators through reasoning. Upon receiving a question, the model first performs internal analysis within <think> and </think> to assess the query's evidence requirement, and then selects a GraphRAG within <graphrag> and

</graphrag> based on the predefined GraphRAG pool and its descriptions, such as evidence granularity and retrieval specialization. The model then performs further reasoning based on the selected GraphRAG, the query, and the expected type and granularity of the retrieved evidence, and selects the most suitable generation LLM within <llm> and </llm> accordingly. Subsequently, it queries the specialist generation LLM with retrieved evidence via <search>*Query:LLM;GraphRAG* </search>. The resulting information is returned within <information> and </information> tags. The model may iteratively route across multiple GraphRAG–LLM pairs to gather complementary insights, which are then integrated to produce the final answer within <answer> and </answer>. This hierarchical routing strategy reduces the per-step action space from $\mathcal{O}(|\mathcal{G}| \cdot |\mathcal{L}|)$ joint choices to $\mathcal{O}(|\mathcal{G}| + |\mathcal{L}|)$ staged choices, thereby facilitating more effective optimization.

4.2 Cold Start Supervised Fine-Tuning

To equip the model with a proper routing skeleton, long-horizon reasoning ability, and initial routing capability, we first conduct supervised fine-tuning (SFT) on curated routing trajectories. To this end, we construct two types of supervised data, namely general routing traces and self-reflection routing traces, and then train the model on these trajectories to learn well-structured reasoning and routing behaviors.

General Routing Trace Generation. We generate high-quality single-turn trajectories that correctly solve the given question while adhering to the desired reasoning, routing, and tool-use manner. Specifically, we begin by collecting GraphRAG–LLM pairs (g_t, l_t) that can correctly solve the question via direct inference, which serve as the basis for constructing general routing trajectories. We then leverage a strong reasoning model (e.g., GPT-5.2 (Singh et al., 2025)) to generate complete trajectories, including the rationale for each routing decision, such as GraphRAG and generation LLM selection. More details are provided in Appendix D.2

Self-Reflection Routing Trace Generation. Since a single routing decision may fail to select the optimal GraphRAG–LLM pair, especially for challenging questions, we further introduce multi-round self-reflective routing traces. To equip the model with self-reflection abilities, we construct self-reflective trajectories on top of single-turn traces. For each question, we first collect multiple GraphRAG–LLM pairs via direct inference. We then use a strong reasoning model to synthesize multi-turn trajectories in which the model reflects on why the previous route failed to solve the question, identifies whether the failure stems from inadequate retrieval or insufficient reasoning ability, and accordingly decides whether to switch to a different GraphRAG or a stronger LLM. In this way, the model learns to diagnose failure, adjust routing decisions, and progressively reach the final answer. More details are provided in Appendix D.2

Training Paradigm. By constructing both general and self-reflection routing trajectories, we collect a high-quality hierarchical routing dataset, denote as $\mathcal{D}_{\text{SFT}}$. We then apply SFT to train the model to conduct expect routing strategy:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}} \left[\sum_{t=1}^{|y|} \log P_\theta(y_t \mid x, y_{<t}) \right],$$

where y is the synthetic trajectory.

4.3 Routing Policy Alignment

Following the cold-start phase, which helps the model acquire an initial structured routing pattern and basic tool-use capability, we leverage RL-based training to further enhance the model's tool-use ability and optimize routing strategies. We extend the general RL optimization objective with external routing pools:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x;\mathcal{G},\mathcal{L})} \Big[r_\phi(x,y) - \beta\, \mathbb{D}_{\text{KL}}[\pi_\theta(y \mid x;\mathcal{G},\mathcal{L}) \,\|\, \pi_{\text{ref}}(y \mid x;\mathcal{G},\mathcal{L})] \Big],$$

where π_θ and π_{ref} represent the policy and reference model, respectively, both of which are initialized from the SFT model, r_ϕ is the reward function and $\mathbb{D}_{\text{KL}}$ is the KL-divergence. To optimize the hierarchical routing policy, we adopt a rule-based reward function including fine-grained format rewards and final outcome rewards:

$$r_{\phi(x,y)} = r_{\text{format}(y)} + r_{\text{outcome}}.$$

Fine-grained Format Reward. Since our framework produces structured hierarchical routing strategy, format correctness is crucial for ensuring valid reasoning traces and executable routing actions. We therefore design a fine-grained format reward

that assigns progressively larger penalties to structural errors of different severity, rather than using a binary format signal:

$$r_{\text{format}}(y) = -\min\left(1, \sum_{k=1}^{K} \lambda_k \, \mathbb{I}_k(y)\right),$$

where $\mathbb{I}_k(y)$ indicates whether the k-th format rule is violated. The detailed rules are provided in Appendix B.4. This formulation provides more informative supervision for learning stable multi-round hierarchical routing behavior.

Final Outcome Reward. In GraphRAG-Router, we leverage Exact Match (EM) to assess the correctness of the final answer generated by the routing agent with respect to the ground truth, and use it as the sole outcome reward to guide optimization:

$$r_{\text{outcome}} = \text{EM}(a, a_{gt}).$$

4.4 Difficulty-Aware Generator Allocation

Although the Routing Policy Alignment stage equips the model with foundational routing ability, the learned policy is often feasible but not optimal, since it remains biased toward large-scale generator LLMs. In practice, a substantial fraction of questions can be handled by smaller models, and thus the unnecessary invocation of large-scale models leads to avoidable computational cost. To bridge this gap, we introduce Difficulty-Aware Generator Allocation, which adaptively penalizes the overuse of large-scale generator models according to question difficulty, encouraging cost-efficient routing without compromising answer correctness. Specifically, we first categorize questions by difficulty level, and then design a curriculum cost-aware reward that penalizes unnecessarily expensive model usage accordingly.

Question Difficulty Categorization. Inspired by Parashar et al. (2025), we extend the definition of question difficulty to the setting of multi-scale LLM routing. Specifically, we define it as the minimum generator model scale required to answer a question correctly and reliably. Specifically, for each question q, we perform direct inference using every generator LLM in the routing pool for N independent trials, and compute the success rate of each model l_m as

$$\text{SR}(l_m, q) = \frac{c(l_m, q)}{N},$$

where $c(l_m, q)$ denotes the number of successful trials of model l_m on question q. We then regard a model $\mathcal{L}_m$ as being able to reliably solve question q if

$$\text{SR}(l_m, q) \geq \tau,$$

where τ is a predefined success-rate threshold. Further, we define the difficulty of question q by the minimum model scale that satisfies this condition:

$$m(q) = \min\Big\{ s \in \{\text{small}, \text{medium}, \text{large}\} \mid \text{SR}(s, q) \geq \tau \Big\}.$$

Accordingly, the difficulty level of question q is defined as

$$\text{D}(q) = \begin{cases} \text{Easy}, & m(q) = \text{small} \\ \text{Medium}, & m(q) = \text{medium} \\ \text{Hard}, & m(q) = \text{large} \end{cases}.$$

Curriculum Cost-aware Reward. To facilitate cost-efficient routing, we first assign each generator LLM a predefined cost $C(l_m)$ according to its model scale, so that model usage becomes explicitly measurable. Combined with the above difficulty categorization, this allows us to define, for each question q, its minimum required cost $C_{\min}(q)$ as the cost of the least expensive generator LLM that can reliably answer it correctly:

$$C_{\min}(q) = \min\left\{ C(l_m) \mid \text{SR}(l_m, q) \geq \tau \right\}.$$

Based on the minimum required cost $C_{\min}(q)$, we can quantify whether the selected generator LLM is unnecessarily expensive for question q. However, such overuse should not be penalized uniformly, since harder questions require greater flexibility to explore stronger models. We therefore adopt a curriculum design that adjusts the penalty strength according to question difficulty:

$$r_{\text{cost}}(y) = \beta \, w_{\text{D}(q)} \, \max(0, \, C(l_m) - C_{\min}(q))$$

where $C(l_m)$ denotes the cost of the generator LLM selected for question q, β is a scaling coefficient, and $w_{\text{D}(q)}$ is the penalty weight associated with the difficulty level of q, with larger weights assigned to easier questions and smaller weights assigned to harder ones.

Reward Shaping. To sum up, the overall reward in the Difficulty-Aware Cost Optimization stage is formulated as:

$$r_{\phi(x,y)} = r_{\text{format}(y)} + r_{\text{outcome}} - \mathbb{1}\{\text{Correct}\} \, r_{\text{cost}(y)}$$

, where $\mathbb{1}\{\text{Correct}\}$ indicates whether the trajectory yields the correct answer. This design encourages the model to improve cost efficiency under successful task completion, naturally converging to the minimal sufficient routing trajectory.

5 Experiments

5.1 Experimental Setup

In this section, we conduct extensive experiments to answer the following key research questions (RQs): **RQ1:** How does GraphRAG-Router perform compared to state-of-the-art baseline models on QA tasks? **RQ2:** Can GraphRAG-Router generalize to unseen GraphRAGs and generator LLMs? **RQ3:** Does GraphRAG-Router yield a better performance-cost trade-off? **RQ4:** How do different components of the GraphRAG-Router contribute to performance and cost efficiency?

Datasets. We evaluate both GraphRAG-Router and baselines on six QA benchmarks: (1) **General QA**: Natural Questions (NQ) (Kwiatkowski et al., 2019), PopQA (Mallen et al., 2023), and TriviaQA (Joshi et al., 2017); (2) **Multi-Hop QA**: HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (2Wiki) (Ho et al., 2020), and Musique (Trivedi et al., 2022). More details are in Appendix A.

Baselines. We compare GraphRAG-Router with four distinct sets of up-to-date, strong baseline models: (1) **Basic LLMs**: Direct Inference, Chain-of-Thought (CoT) Prompting (Wei et al., 2023); (2) **RAG-based Methods**: Vanilla RAG (Lewis et al., 2021), GraphRAG (Edge et al., 2025), RAPTOR (Sarthi et al., 2024), HippoRAG2 (Gutiérrez et al., 2025b), HyperGraphRAG (Luo et al., 2025b), and LinearRAG (Zhuang et al., 2025); (3) **Training-free Agentic Search Systems**: Search-o1 (Li et al., 2025) and GraphSearch (Yang et al., 2025b); (4) **RL-based Search Agent**: Search-R1 (Jin et al., 2025a), Graph-R1 (Luo et al., 2025a), and Router-R1 (Zhang et al., 2025).

Implementation Details. We pre-train both GraphRAG-Router and baselines on the NQ and HotpotQA datasets. We construct a joint train set of 5K samples from HotpotQA and NQ. For the test, we sample 1000 data points from the original test set or development set. For a fair comparison, we use Qwen2.5-3B-Instruct (Qwen et al., 2025) as the backbone LLM for all training-based methods. For the SFT stage of our method, we leverage GPT-5.2 (Singh et al., 2025) to generate 450 general traces and 50 self-reflection traces, respectively. For both RL training stages, we use GRPO (Shao et al., 2024) as the default algorithm. For GraphRAGs, we select five representative frameworks: GraphRAG, RAPTOR, HippoRAG2, HyperGraphRAG, and LinearRAG. For generator LLMs, we select five cutting-edge LLMs that cover three scale ranges: (1) **Small**: Qwen2.5-7B-Instruct (Qwen et al., 2025), LLaMA3.1-8B-Instruct (Grattafiori et al., 2024), Ministral-8B-2512 (Liu et al., 2026a). (2) **Medium**: Mixtral-8×22B-Instruct (Jiang et al., 2024) and (3) **Large**: LLaMA3.3-70B-Instruct (Grattafiori et al., 2024). We evaluate with Exact Match (EM) and F-1. More details are in Appendix B.

5.2 Overall Performance (RQ1)

In this section, we evaluate the overall performance of GraphRAG-Router under both in-domain and cross-domain settings. The results are shown in Table 1. Based on the results, we observe that:

***Observation 1*: GraphRAG-Router consistently outperforms all baselines on QA tasks.** As shown in Table 1, GraphRAG-Router outperforms competing methods, including basic LLM, RAG-based inference, training-free agent, and RL-based agent, across all six QA benchmarks. While the RL-based agent, such as Router-R1 (Zhang et al., 2025) improves upon other baselines by interleaved multi-turn search and reasoning, GraphRAG-Router achieves even stronger results, especially on multi-hop QA datasets. Notably, GraphRAG-Router exceeds the performance of leading RL-based baselines by an average margin of $+4.71\%$ on general QA and $+38.28\%$ on multi-hop QA, respectively. This highlights the effectiveness of our proposed approach.

***Observation 2*: GraphRAG-Router generalizes well to unseen datasets.** Despite being pre-trained on NQ and HotpotQA, it maintains strong performance on cross-domain QA datasets, with an average improvement of $+18.55\%$, compared to leading baselines. This indicates that, even pretrained on limited data, GraphRAG-Router still demonstrates strong generalization capabilities with a transferable routing policy.

5.3 Can GraphRAG-Router Generalize to Unseen GraphRAGs and LLMs? (RQ2)

To evaluate GraphRAG-Router's generalization to unseen LLMs and GraphRAG systems, we expand both the routing LLM pool and the GraphRAG pool. Specifically, we introduce two additional state-of-

System	Methods	General QA			Multi-Hop QA			
		NQ†	PopQA⋆	TriviaQA⋆	HotpotQA†	2Wiki⋆	Musique⋆	Avg.
Basic LLM	Direct Infer♠	0.107	0.117	0.308	0.269	0.295	0.118	0.202
	CoT♠	0.286	0.327	0.552	0.313	0.293	0.165	0.323
RAG-Based	Vanilla RAG	0.294	0.242	0.557	0.326	0.348	0.186	0.326
	HippoRAG2	0.353	0.266	0.601	0.395	0.374	0.263	0.375
	LinearRAG	0.302	0.269	0.608	0.385	0.435	0.238	0.373
	HyperGraphRAG	0.268	0.272	0.585	0.253	0.336	0.209	0.321
	RAPTOR	0.271	0.258	0.620	0.307	0.318	0.231	0.334
	GraphRAG	0.301	0.291	0.604	0.292	0.327	0.108	0.321
Training-free Agent	Search-o1	0.348	0.302	0.618	0.328	0.230	0.149	0.329
	GraphSearch	0.368	0.324	0.623	0.317	0.427	0.133	0.365
RL-based Agent	Search-R1	0.403	0.256	0.568	0.247	0.269	0.100	0.307
	Graph-R1	0.267	0.274	0.464	0.298	0.407	0.289	0.333
	Router-R1	0.386	0.351	0.663	0.368	0.456	0.140	0.394
Our Method	GraphRAG-Router	**0.426**	**0.368**	**0.672**	**0.461**	**0.523**	**0.443**	**0.482**

Table 1: Overall Exact Match (EM) on six QA datasets. ♠ denotes the best result among different scales of LLMs. † denotes the in-domain dataset, and ⋆ denotes the cross-domain dataset. Bold and underline indicate the best and best baseline results, respectively.

Model	Dataset		
	NQ	HotpotQA	2Wiki
Router-R1	0.386	0.368	0.456
Router-R1*	0.390	0.377	0.458
GraphRAG-Router	0.426	**0.461**	0.523
GraphRAG-Router*	**0.439**	0.458	**0.550**

Table 2: Results of the generalization capability of GraphRAG-Router.

the-art LLMs, Qwen3-8B (Yang et al., 2025a) and gpt-oss-120b (OpenAI et al., 2025), as well as one additional GraphRAG method, LightRAG (Guo et al., 2025). We then incorporate the descriptions of these newly added LLMs and GraphRAG systems into the routing template. Without any further training, we directly perform inference using the pre-trained GraphRAG-Router. The results are presented in Table 2.

Observation 3: GraphRAG-Router generalizes well to previously unseen LLMs and GraphRAG systems, allowing new candidates to be incorporated without additional training. The results, as shown in Table 2, indicate that GraphRAG-Router achieves comparable or even slightly improved performance across multiple QA datasets. Notably, it establishes new best EM scores on several benchmarks, including NQ and 2Wiki. This suggests that GraphRAG-Router does not merely overfit to the candidate set seen during training, but instead learns a transferable routing strategy that remains effective when previously unseen LLMs and GraphRAG systems are introduced.

5.4 Cost-efficiency Analysis (RQ3)

In this section, we analyze the performance-cost trade-off of GraphRAG-Router. Specifically, we present the routing statistics for Stages 1 and 2 across small-, medium-, and large-scale LLMs. The results are shown in Figure 3a and 3b.

Observation 4: GraphRAG-Router achieves a more favorable performance-cost trade-off by reducing reliance on large-scale LLMs. In contrast to Stage 1 and Router-R1, which predominantly route queries to large-scale models, GraphRAG-Router reduces large-model usage by nearly 30%, reallocating a considerable fraction of queries to medium- and small-scale models. This result highlights the effectiveness of the proposed curriculum cost-aware reward in enabling GraphRAG-Router to adaptively select models that are better aligned with query difficulty while remaining cost-efficient. Moreover, compared with the Stage 1-only setting, GraphRAG-Router still achieves consistent improvement, underscoring its ability to optimize performance while maintaining cost efficiency.

5.5 Ablation Study (RQ4)

To answer RQ4, we conduct a comprehensive analysis of each component of GraphRAG-Router.

(a) Exact match performance on three datasets.

(b) Distribution of routing calls across model scales.

Figure 3: Comparison of routing behavior and downstream performance.

Strategy	Dataset		
	NQ	HotpotQA	2Wiki
One-time	0.398	0.433	0.479
LLM-First	0.415	0.446	0.518
GraphRAG-First	**0.426**	**0.461**	**0.523**

Table 3: EM of different routing strategies.

5.5.1 Impact of Hierarchical Routing Process

To further evaluate the effectiveness of the hierarchical routing process, we implement two additional variants of GraphRAG-Router with alternative routing strategies, namely one-time routing and LLM-first routing. For one-time routing, the model directly selects a GraphRAG–LLM pair in a single reasoning process. For LLM-first routing, we reverse the routing order and let the model choose the LLM before selecting the GraphRAG. The results are presented in Table 3, from which we draw the following conclusion:

Observation 5: The hierarchical routing strategy is consistently more effective than flat routing alternatives. Specifically, compared with one-time routing, hierarchical routing consistently achieves stronger results, showing the advantage of decomposing the routing decision into multiple stages. Moreover, among the hierarchical variants, the GraphRAG-first design outperforms the LLM-first design across all datasets, indicating that deciding the retrieval framework before the generator better aligns with the dependency between evidence acquisition and answer generation.

5.5.2 Impact of Cold Start SFT

To assess the effectiveness of SFT, we isolated this stage by training the model from the base model. We then compared their EM and the average number of valid tool calls. From the results in Table 4, we draw the following conclusion:

Observation 6: SFT serves to regularize the output format, thereby providing a better foundation for subsequent optimization. Removing SFT leads to a significant drop in both EM and the average number of valid tool calls. This indicates that SFT plays a critical role in standardizing the model's routing behavior.

Method	NQ	HotpotQA	Valid call
w/o SFT	0.185	0.207	0.35
w/ SFT	**0.426**	**0.461**	**1.29**

Table 4: Ablation study of SFT. Valid call indicates the average valid tool call for both datasets.

SFT	Stage1	Stage2	Dataset	
			NQ	HotpotQA
✓	✗	✗	0.237	0.245
✓	✓	✗	0.419	0.448
✓	✓	✓	**0.426**	**0.461**

Table 5: Ablation study on training strategies of GraphRAG-Router.

5.5.3 Ablation on RL Training Strategies

To study the impact of our proposed two-stage RL training: Routing Policy Alignment and Difficulty-Aware Cost Optimization, we conduct a stage-wise ablation study. In detail, we ablate each stage and compare the EM of these variants. ***Observation 7: The proposed two-stage RL training significantly enhances the performance of GraphRAG-Router.*** Compared with other variants, GraphRAG-Router with two-stage RL training achieves superior performance, with an average improvement of $+2.31\%$. The overall results confirm that both designed training strategies contribute positively to the performance of GraphRAG-Router.

6 Router Overhead Analysis

In this section, we conduct an end-to-end efficiency analysis to further evaluate our method. We first compare the hierarchical routing strategy with its non-hierarchical one-time routing variant on NQ.

Dataset	Method	Latency	Token Usage	EM
NQ	One-time	5.43s	318	0.398
	Hierarchical	5.51s	329	0.426
HotpotQA	One-time	6.69s	505	0.433
	Hierarchical	6.82s	527	0.461

Table 6: End-to-end efficiency analysis of hierarchical routing strategy and one-time routing.

Dataset	Method	Router Tokens	LLM Tokens	Total Tokens
NQ	HippoRAG2+Qwen2.5-72B	0	386	386
	GraphRAG-Router	74	255	329
HotpotQA	HippoRAG2+Qwen2.5-72B	0	548	548
	GraphRAG-Router	96	431	527

Table 7: Token usage of GraphRAG-Router and HippoRAG2 with Qwen-72B.

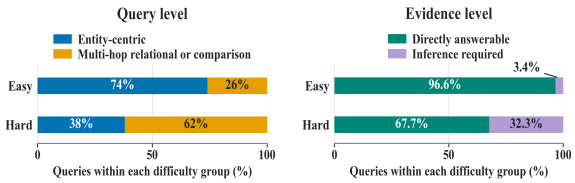
As shown in Table 6, hierarchical routing introduces only a 2.8% average increase in latency and token usage, while improving accuracy by 6.8%. This indicates that its additional overhead is modest relative to the performance gain.

We then compare GraphRAG-Router with the best single GraphRAG paired with a large LLM, namely HippoRAG2 + Qwen2.5-72B-Instruct. As shown in Table 7, the router consumes only 74 tokens per query on NQ and 96 on HotpotQA. Even including router tokens, GraphRAG-Router reduces total per-query token consumption by 14.8% on NQ and 3.8% on HotpotQA, showing that routing overhead does not offset the overall token savings.

Finally, we examine deployment overhead. The five GraphRAG retriever servers together consume 1.8 GB of GPU memory, compared with 10.4 GB for the locally deployed 3B router. Moreover, GraphRAG-Router invokes only the selected retriever backend(s) for each query rather than all five in parallel. Thus, the main additional cost comes from fixed serving and index-maintenance overhead, which can be amortized across queries.

7 Analysis of Difficulty-Aware Routing

We further analyze whether the difficulty labels reflect interpretable reasoning requirements. As shown in Figure 4, easy queries are mostly entity-centric (74%), while hard queries more often involve multi-hop relational or comparison reasoning (62%). Similarly, 96.6% of easy queries are directly answerable from retrieved evidence, compared with 67.7% of hard queries. These results indicate that the difficulty labels correspond to meaningful differences in both query complexity and evidence requirements, rather than merely reflecting model-dependent success rates.



(a) Query level (b) Evidence level

Figure 4: Difficulty profiles at the query and evidence levels.

Having established this correspondence, we further examine whether GraphRAG-Router exhibits routing behavior consistent with these underlying reasoning requirements. The case studies in Figures 14 & 15 show that the router can assess whether the retrieved evidence is sufficient and trigger additional routing when further evidence is needed. This further suggests that the routing behavior adapts to evidence requirements associated with query difficulty.

8 Conclusion

In this paper, we introduce GraphRAG-Router, a cost-efficient framework that coordinates GraphRAGs and generator LLMs through agentic routing. GraphRAG-Router leverages a hierarchical routing strategy and a two-stage RL training paradigm to balance performance and inference cost. Across six complex QA benchmarks, it outperforms all state-of-the-art baselines, achieving an average improvement of 22.3% while reducing the overuse of large LLMs by nearly 30%.

Limitation

While GraphRAG-Router demonstrates strong performance on six QA benchmarks, we acknowledge several limitations that motivate future work. First, the current study primarily focuses on Wikipedia-based QA settings. It remains to be explored whether GraphRAG-Router generalizes equally well to more complex retrieval-intensive tasks, such as multi-hop reasoning over long enterprise or scientific documents. Second, our generalization analysis is conducted with a relatively limited pool of generator LLMs and GraphRAG systems. As future work, we plan to evaluate GraphRAG-Router on larger and more diverse candidate pools to more comprehensively assess its generalization and robustness across heterogeneous retrieval and generation components. Third, our evaluation mainly considers an offline retrieval setting, where the corpora, graph indices, and candidate GraphRAG systems are pre-built and fixed. Evaluating GraphRAG-Router in dynamic online environments with continuously updated knowledge sources and evolving retrieval indices is another important direction for future work.

Acknowledgments

We sincerely thank the anonymous reviewers, AC and SAC for their valuable feedback. This research is partially supported by National Natural Science Foundation of China (62402320), Shanghai Science and Technology Program (24YF2731000), and the NYU Shanghai Boost Fund.

References

Tong Chen, Zimu Wang, Yiyi Miao, Haoran Luo, Yuanfei Sun, Wei Wang, Zhengyong Jiang, Procheta Sen, and Jionglong Su. 2025. MedFact: A large-scale Chinese dataset for evidence-based medical fact-checking of LLM responses. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 32340–32353, Suzhou, China. Association for Computational Linguistics.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2025. From local to global: A graph rag approach to query-focused summarization. *Preprint*, arXiv:2404.16130.

Dongzhe Fan, Jiajin Liu, Yi Fang, Djellel Difallah, and Qiaoyu Tan. 2026. Mlaga: Multimodal large language and graph assistant. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '26, page 1027–1038, New York, NY, USA. Association for Computing Machinery.

Tao Feng, Yanzhen Shen, and Jiaxuan You. 2025. Graphrouter: A graph-based router for llm selections. *Preprint*, arXiv:2410.03834.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, and et al Ahmad Al-Dahle. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2025. Lightrag: Simple and fast retrieval-augmented generation. *Preprint*, arXiv:2410.05779.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2025a. Hipporag: Neurobiologically inspired long-term memory for large language models. *Preprint*, arXiv:2405.14831.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025b. From rag to memory: Non-parametric continual learning for large language models. *Preprint*, arXiv:2502.14802.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, and 1 others. 2024. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309*.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Preprint*, arXiv:2402.07630.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *Preprint*, arXiv:2011.01060.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, and et al Guillaume Lample. 2024. Mixtral of experts. *Preprint*, arXiv:2401.04088.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025a. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *Preprint*, arXiv:2503.09516.

Jiajie Jin, Yutao Zhu, Zhicheng Dou, Guanting Dong, Xinyu Yang, Chenghao Zhang, Tong Zhao, Zhao Yang, and Ji-Rong Wen. 2025b. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. In *Companion Proceedings of the ACM on Web Conference 2025*, WWW '25, page 737–740, New York, NY, USA. Association for Computing Machinery.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *Preprint*, arXiv:1705.03551.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Preprint*, arXiv:2005.11401.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. Search-o1: Agentic search-enhanced large reasoning models. *Preprint*, arXiv:2501.05366.

Yang Li. 2025. Rethinking predictive modeling for llm routing: When simple knn beats complex learned routers. *Preprint*, arXiv:2505.12601.

Alexander H. Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, and et al Alexandre Cahill. 2026a. Ministral 3. *Preprint*, arXiv:2601.08584.

Jiajin Liu, Dongzhe Fan, Chuanhao Ji, Daochen Zha, and Qiaoyu Tan. 2026b. Graphvlm: Benchmarking vision language models for multimodal graph learning. *Preprint*, arXiv:2603.13370.

Haoran Luo, Haihong E, Guanting Chen, Qika Lin, Yikai Guo, Fangzhi Xu, Zemin Kuang, Meina Song, Xiaobao Wu, Yifan Zhu, and Luu Anh Tuan. 2025a. Graph-r1: Towards agentic graphrag framework via end-to-end reinforcement learning. *Preprint*, arXiv:2507.21892.

Haoran Luo, Haihong E, Guanting Chen, Yandan Zheng, Xiaobao Wu, Yikai Guo, Qika Lin, Yu Feng, Zemin Kuang, Meina Song, Yifan Zhu, and Luu Anh Tuan. 2025b. Hypergraphrag: Retrieval-augmented generation via hypergraph-structured knowledge representation. *Preprint*, arXiv:2503.21322.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. *Preprint*, arXiv:2212.10511.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, and et al Florencia Leoni Aleman. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

OpenAI, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, and et al Haiming Bao. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Shubham Parashar, Shurui Gui, Xiner Li, Hongyi Ling, Sushil Vemuri, Blake Olson, Eric Li, Yu Zhang, James Caverlee, Dileep Kalathil, and Shuiwang Ji. 2025. Curriculum reinforcement learning from easy to hard tasks improves llm reasoning. *Preprint*, arXiv:2506.06632.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2025. Graph retrieval-augmented generation: A survey. *ACM Transactions on Information Systems*, 44(2):1–52.

Hongjin Qian, Zheng Liu, Peitian Zhang, Kelong Mao, Defu Lian, Zhicheng Dou, and Tiejun Huang. 2025. Memorag: Boosting long context processing with global memory-enhanced retrieval augmentation. *Preprint*, arXiv:2409.05591.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, and et al Binyuan Hui. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. *Preprint*, arXiv:2401.18059.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, and et al Aidan McLaughlin. 2025. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Yuanfu Sun, Kang Li, Dongzhe Fan, Jiajin Liu, and Qiaoyu Tan. 2026a. Agentgl: Towards agentic graph learning with llms via reinforcement learning. *Preprint*, arXiv:2604.05846.

Yuanfu Sun, Kang Li, Pengkang Guo, Jiajin Liu, and Qiaoyu Tan. 2026b. Mario: Multimodal graph reasoning with large language models. *Preprint*, arXiv:2603.05181.

Yuanfu Sun, Zhengnan Ma, Yi Fang, Jing Ma, and Qiaoyu Tan. 2025. Graphicl: Unlocking graph learning potential in llms through structured prompt design. *Preprint*, arXiv:2501.15755.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Preprint*, arXiv:2108.00573.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, and et al Binyuan Hui. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Cehao Yang, Xiaojun Wu, Xueyuan Lin, Chengjin Xu, Xuhui Jiang, Yuanliang Sun, Jia Li, Hui Xiong, and Jian Guo. 2025b. Graphsearch: An agentic deep searching workflow for graph retrieval-augmented generation. *Preprint*, arXiv:2509.22009.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *Preprint*, arXiv:1809.09600.

Chuanyue Yu, Kuo Zhao, Yuhan Li, Heng Chang, Mingjian Feng, Xiangzhe Jiang, Yufei Sun, Jia Li, Yuzhi Zhang, Jianxin Li, and Ziwei Zhang. 2026. Graphrag-r1: Graph retrieval-augmented generation with process-constrained reinforcement learning. *Preprint*, arXiv:2507.23581.

Haozhen Zhang, Tao Feng, and Jiaxuan You. 2025. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. *Preprint*, arXiv:2506.09033.

Luyao Zhuang, Shengyuan Chen, Yilin Xiao, Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, and Xiao Huang. 2025. Linearrag: Linear graph retrieval augmented generation on large-scale corpora. *Preprint*, arXiv:2510.10114.

A Dataset Details

We conduct evaluations on six widely used RAG benchmarks from the FlashRAG toolkit (Jin et al., 2025b), covering both single-hop and multi-hop question answering tasks:

- **Natural Questions (NQ) (Kwiatkowski et al., 2019).** Real user questions from Google Search paired with Wikipedia passages/answers; commonly used for open-domain, mostly single-hop QA.
- **PopQA (Mallen et al., 2023).** Popular-knowledge question set designed for retrieval-based QA, emphasizing factual queries where the answer must be grounded in retrieved evidence.
- **TriviaQA (Joshi et al., 2017).** Trivia-style questions with evidence documents (often web/Wikipedia); used for open-domain factual QA and long-context evidence matching.
- **HotpotQA (Yang et al., 2018).** Multi-hop QA requiring reasoning over multiple supporting Wikipedia passages; includes labeled supporting facts.
- **Musique (Trivedi et al., 2022)** Multi-hop QA benchmark built to test compositional reasoning across several pieces of evidence, often with more challenging, structured multi-step requirements.
- **2WikiMultiHopQA (2Wiki) (Ho et al., 2020)** Multi-hop QA constructed from Wikipedia that typically requires linking two (or more) pages to reach the answer, focusing on cross-article reasoning.

B Implementation Details

B.1 Baselines

Training-free Search Agents: These approaches do not train an explicit control policy; rather, they use structured prompts and heuristic rules to steer multi-step retrieval and reasoning at inference time. Specifically, we evaluate:

- Search-o1 (Li et al., 2025): Augments large reasoning models with an agentic RAG workflow and a Reason-in-Documents module that refines retrieved evidence before integration, enabling dynamic, noise-reduced knowledge retrieval to improve reliability on complex reasoning tasks and open-domain QA. Our implementation is based on https://github.com/RUC-NLPIR/Search-o1
- GraphSearch (Yang et al., 2025b): An agentic deep-search workflow for GraphRAG that performs multi-turn, modular retrieval with dual-channel querying over both text chunks (semantic) and structural graphs (relational), consistently improving multi-hop RAG accuracy and generation quality over traditional GraphRAG retrieval. Our implementation is based on https://github.com/DataArcTech/GraphSearch

RL-based Search Agent: These approaches use an RL policy to optimize the agent's search and reasoning behavior, typically adopting GRPO as the reinforcement learning algorithm. Specifically, we choose:

- Search-R1 (Jin et al., 2025a): A RL-based retrieval-augmented reasoning framework that trains LLMs to autonomously generate multi-turn search queries during step-by-step reasoning, using retrieved-token masking and an outcome-based reward to improve QA performance over standard RAG baselines. Our implementation is based on https://github.com/PeterGriffinJin/Search-R1
- Graph-R1 (Luo et al., 2025a): An agentic GraphRAG framework trained end-to-end with reinforcement learning that builds lightweight knowledge hypergraphs and performs multi-turn retrieval as an agent–environment interaction. Our implementation is based on https://github.com/LHRLAB/Graph-R1
- Router-R1 (Zhang et al., 2025): An RL-based multi-LLM routing framework that formulates model selection and response aggregation as a sequential decision-making process. By interleaving internal reasoning with dynamic routing actions, Router-R1 can invoke multiple LLMs adaptively and optimize the trade-off between task performance and inference cost. Our implementation is based on https://github.com/ulab-uiuc/Router-R1

For GraphRAGs in main experiments, we adopt 5 representative GraphRAGs:

Dataset	Task	Knowledge Source	#Train	#Dev	#Test
NQ	General QA	Wiki	79,168	8,757	3,610
PopQA	General QA	Wiki	-	-	14,267
TriviaQA	General QA	Wiki & Web	78,785	8,837	11,313
HotpotQA	Multi-hop QA	Wiki	90,447	7,405	-
Musique	Multi-hop QA	Wiki	19,938	2,417	-
2WikiMultiHopQA	Multi-hop QA	Wiki	15,000	12,576	-

Table 8: Dataset Statistics

- **Hypergraph-based**: HypergraphRAG (Luo et al., 2025b): A hypergraph-based RAG framework that represents real-world n-ary facts using hyperedges and integrates hypergraph construction, retrieval, and generation. Our implementation is based on https://github.com/LHRLAB/Graph-R1

- **Entity Graph based**: HippoRAG2 (Gutiérrez et al., 2025b): A memory-inspired RAG framework that extends HippoRAG's Personalized PageRank retrieval with deeper passage integration and stronger online LLM usage, improving factual, sense-making, and associative memory. Our implementation is based on https://github.com/OSU-NLP-Group/HippoRAG

- **Tri-Graph based:** LinearRAG (Zhuang et al., 2025): An efficient GraphRAG framework that avoids noisy, costly relation extraction by building a lightweight relation-free hierarchical "Tri-Graph" (via entity extraction + semantic linking) and retrieving evidence with a two-stage process—local entity activation followed by global importance aggregation—yielding stronger and more reliable passage retrieval on multi-hop QA benchmarks. Our implementation is based on https://github.com/DEEP-PolyU/LinearRAG

- **Tree based**: RAPTOR (Sarthi et al., 2024): A retrieval-augmented approach that builds a hierarchical tree of recursive embeddings, clusters, and bottom-up summaries, enabling inference-time retrieval across long documents at multiple abstraction levels and delivering strong gains. Our implementation is based on https://github.com/parthsarthi03/raptor

- **Tree based**: GraphRAG (Edge et al., 2025): A graph-based QA framework for private corpora that tackles global, corpus-level questions by (1) building an entity knowledge graph and precomputing community summaries, then (2) answering queries via summary-to-partial-response generation followed by a final aggregation, improving comprehensiveness and diversity over standard RAG at million-token scale. Our implementation is based on https://microsoft.github.io/graphrag/

For the generalization experiment, we add an additional GraphRAG:

- LightRAG (Guo et al., 2025): A graph-enhanced retrieval-augmented generation framework that integrates graph structures into indexing and retrieval to better capture entity relationships and complex contextual dependencies. By combining graph-based retrieval with vector representations and a dual-level retrieval mechanism, it improves both retrieval accuracy and efficiency. Our implementation is based on https://github.com/hkuds/lightrag

For all the GraphRAGs, we utilize the context of each question as the document and organize the corpus by the official settings. For retrieval, we set the top-k as top-5. For the description of each GraphRAG, we adopt GPT-5.2 (Singh et al., 2025) to summarize the key method from the original paper. The detailed description is provided in D.

B.2 Generator LLMs

All the generator LLMs are accessed via OpenRouter APIs[1]. We employ the model card as the description of each LLM.

B.3 GraphRAG-Router

We use verl[2] as our reinforcement learning training framework. To conveniently define the minimum

[1] https://openrouter.ai/
[2] https://github.com/verl-project/verl

Learning Rate	Batch Size	Epochs	Weight Decay	Optimizer	Lr Scheduler	BF16
2e-5	4	2	0.01	Adam	Cosine	True

Table 9: Hyperparameter setting for SFT

Hyperparameter	Value	Hyperparameter	Value
Learning Rate	1e-6	Mini-batch Size	32
Train Batch Size	64	Micro-batch Size	8
Rollout Group size	5	Max Training Steps	80
KL coefficient	0.001	Warm Up Ratio	0
Max turns	4	Max Sequence Length	4096
Max Response Length	1024	Max LLM Response Length	600
Tensor Parallel Size	1	GPU Utilization Ratio	0.45
Rollout Temperature (Train)	1.0	Rollout Temperature	1.0

Table 10: Hyperparameter settings for stage 1

Hyperparameter	Value	Hyperparameter	Value
Learning Rate	1e-6	Mini-batch Size	32
Train Batch Size	64	Micro-batch Size	8
Rollout Group size	5	Max Training Steps	40
KL coefficient	0.001	Warm Up Ratio	0
Max turns	4	Max Sequence Length	4096
Max Response Length	1024	Max LLM Response Length	600
Tensor Parallel Size	1	GPU Utilization Ratio	0.45
Rollout Temperature (Train)	1.2	Rollout Temperature (Eval)	1.0

Table 11: Hyperparameter settings for stage 2

cost for each query, we approximate the cost of each LLM scale using the OpenRouter output price per 1M tokens. Specifically, for each model l_m in an LLM pool, we conduct direct inference on 30 randomly sampled queries and compute its empirical inference cost using the OpenRouter price $price_{l_m}$ and token consumption. We then average the costs of all models within the same scale group to obtain the group-level average cost. Based on these average costs, we discretize the three LLM scale groups into relative cost levels and set the cost values to 1, 2, and 4 for the small-, medium-, and large-scale groups, respectively. This discretization provides a consistent and stable approximation of relative inference costs and makes the cost-aware reward more stable during RL training. The penalty weight of each difficulty level is: 1 (easy), 0.6 (medium), 0.2 (hard). The threshold of $SR(q)$ is set to 0.8. In Stage 1, the objective is to train the router to correctly follow the hierarchical routing procedure and acquire the required tool-use capability. Accordingly, we do not assign separate weights to the format reward and the outcome reward in this stage; the two components are directly added together. This design reflects the fact that both objectives are equally important at this stage: the router is expected to follow the hierarchical routing format while acquiring the ability to use the corresponding tools. In Stage 2, the goal is to improve cost efficiency without sacrificing correctness. Thus, we only apply separate scaling to the cost-aware reward, using the coefficient β together with difficulty-dependent weights. This scaling prevents the cost penalty from dominating the out-

come reward and ensures that answer correctness remains the primary optimization objective. Here, the scaling parameter β is set to 0.05. The detailed hyper-parameters of each stage are provided in Table 9, 10 and 11.

B.4 Details of fine-grained format reward

To ensure that the router follows the hierarchical routing protocol, our format reward is designed to cover the complete routing procedure, from reasoning, GraphRAG selection, LLM selection, search execution, to final answer generation. Specifically, the reward penalizes both global structural errors, such as missing, mismatched, or nested tags, and stage-specific violations, such as selecting an LLM before choosing a GraphRAG backend, issuing a search without a valid GraphRAG or LLM selection, or producing an invalid search command. It also checks the validity of selected backend names, the presence of required reasoning stages, the correctness of the search format, and the cardinality of the final answer tag. In this way, the reward provides fine-grained step-wise supervision aligned with the hierarchical routing trace, encouraging the model to learn not only the final output format but also the intended order and dependencies among routing decisions.

- **Fatal format violation** $(+1.0)$: If the output contains no valid tags, includes unclosed or mismatched tags, or contains nested tags, the format penalty is directly set to 1.0.
- **Missing reasoning tag** $(+0.4)$: A penalty of 0.4 is applied if no `<think>` tag is present.
- **LLM selection before GraphRAG selection** $(+0.8)$: If a `<llm>` tag appears before any valid `<graphrag>` tag, a penalty of 0.8 is applied, as this violates the intended hierarchical routing order.
- **Missing GraphRAG selection** $(+0.4$ / $+0.6)$: If no valid `<graphrag>` tag is identified, a penalty of 0.4 is applied. If the model proceeds to `<search>` without a valid GraphRAG selection, the penalty is increased to 0.6.
- **Invalid GraphRAG name** $(+0.2)$: If a `<graphrag>` tag is present but its content does not correspond to a valid GraphRAG option, a penalty of 0.2 is applied.
- **Missing second-stage reasoning** $(+0.3)$: If a `<search>` action is produced with fewer than two `<think>` tags, a penalty of 0.3 is applied, encouraging an explicit two-stage reasoning process before execution.
- **Missing LLM selection before search** $(+0.3)$: If the model issues a `<search>` action without first specifying a valid `<llm>` tag, a penalty of 0.3 is applied.
- **Invalid LLM name** $(+0.1)$: If a `<llm>` tag is present but its content is not a valid LLM name, a penalty of 0.1 is applied.
- **Missing search action** $(+0.4)$: If no `<search>` tag is present, a penalty of 0.4 is applied.
- **Invalid search format** $(+0.3)$: If the content of `<search>` does not follow the required format, i.e., it does not contain exactly one ":" and one ";", a penalty of 0.3 is applied.
- **Invalid answer cardinality** $(+0.3)$: If the output contains zero or more than one `<answer>` tag, a penalty of 0.3 is applied.
- **Empty reasoning content** $(+0.2)$: If any `<think>` tag is empty or contains only a placeholder such as "...", a penalty of 0.2 is applied.

The final format penalty is clipped to a maximum value of 1.0.

B.5 Construction of $\mathcal{D}_{\text{SFT}}$

For the SFT dataset, $\mathcal{D}_{\text{SFT}}$, we adopt GPT-5.2 to generate both the general and self-reflection traces. In detail, for each query, we enumerate all GraphRAG–LLM combinations and use direct inference to identify which pairs can correctly answer the query. For general traces, we randomly select one correct pair and instantiate the template in Figure 8. For self-reflection traces, we select one incorrect pair and one correct pair, and instantiate the template in Figure 9 to teach the router how to diagnose a failed route and switch to a better one. In total, we synthesize 500 training instances, consisting of 450 general traces and 50 self-reflection traces. This is only a small fraction of the 5,000 training instances used during the RL stage. In addition, we also adopt an open-source strong LLM, LLaMA-3.1-70B-Instruct, to generate routing traces and train GraphRAG-Router following the same procedure. The results in Table 13 show that GraphRAG-Router trained with open-source traces achieves competitive performance, indicating that our method does not critically depend on proprietary trace generation.

System	Methods	General QA			Multi-Hop QA			
		NQ†	PopQA⋆	TriviaQA⋆	HotpotQA†	2Wiki⋆	Musique⋆	Avg.
Basic LLM	Direct Infer♠	0.165	0.162	0.347	0.361	0.347	0.135	0.253
	CoT♠	0.399	0.316	0.618	0.436	0.344	0.243	0.393
RAG-Based	Vanilla RAG	0.401	0.275	0.623	0.445	0.376	0.257	0.396
	HippoRAG2	0.465	0.329	0.668	0.501	0.428	0.416	0.468
	LinearRAG	0.436	0.334	0.661	0.487	0.469	0.403	0.465
	HyperGraphRAG	0.375	0.307	0.656	0.376	0.419	0.346	0.413
	RAPTOR	0.382	0.282	0.684	0.409	0.398	0.395	0.425
	GraphRAG	0.423	0.348	0.705	0.388	0.401	0.248	0.419
Training-free Agent	Search-o1	0.476	0.354	0.672	0.439	0.289	0.237	0.411
	GraphSearch	0.483	0.371	0.683	0.425	0.458	0.190	0.435
RL-based Agent	Search-R1	0.473	0.357	0.617	0.371	0.451	0.160	0.405
	Graph-R1	0.415	0.294	0.597	0.468	0.468	0.357	0.433
	Router-R1	0.495	**0.418**	0.723	0.467	0.502	0.224	0.472
Our Method	GraphRAG-Router	**0.525**	0.403	**0.735**	**0.589**	**0.591**	**0.563**	**0.568**

Table 12: Overall F-1 on six QA datasets. ♠ denotes the best result among different scale of LLMs. † denotes the in-domain dataset, and ⋆ denotes the cross-domain dataset. Bold and underline indicate the best and best baseline results, respectively.

Method	NQ	HotpotQA
SFT-LLaMA	0.418	0.463
SFT-GPT	0.426	0.461

Table 13: Impact of trace-generation LLMs on GraphRAG-Router performance.

C Additional Results.

C.1 Overall F-1

The overall F-1 score is shown in Table 12. The results further indicate the effectiveness of our proposed framework.

C.2 Additional Experiment Results for SFT

We provide the reward curves for both variants. As shown in Figure 5, the variant with SFT exhibits a noticeably more stable training process, demonstrating the necessity of SFT for effective optimization.

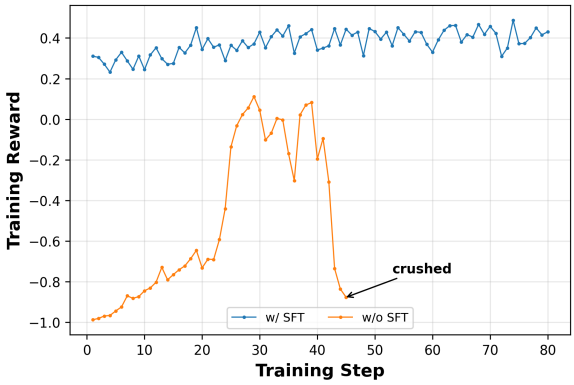


Figure 5: Reward curve

C.3 Ablation on Curriculum Cost-aware Reward

To evaluate the effectiveness of our proposed curriculum cost-aware reward (CCR), we implement a variant that applies the same penalty to all questions, i.e., setting $W_{\text{Difficulty}(q)} = 1$ for every query. The EM results are reported in Table 14, and the routing distribution is shown in Figure 6. From the results, we observe that although applying a uniform penalty to all questions can substantially reduce the routing cost, this reduction comes at the expense of answer performance. This suggests that simply imposing a global cost penalty encourages overly conservative routing, whereas our difficulty-aware CCR achieves a better balance between cost efficiency and performance.

Method	NQ	HotpotQA
w/o curriculum	0.391	0.406
w/ curriculum	**0.426**	**0.461**

Table 14: Ablation study of curriculum cost-aware reward.

C.4 Ablation on GraphRAG Routing Pool

In this section, we conduct an ablation study on the GraphRAG routing pool. Specifically, we consider

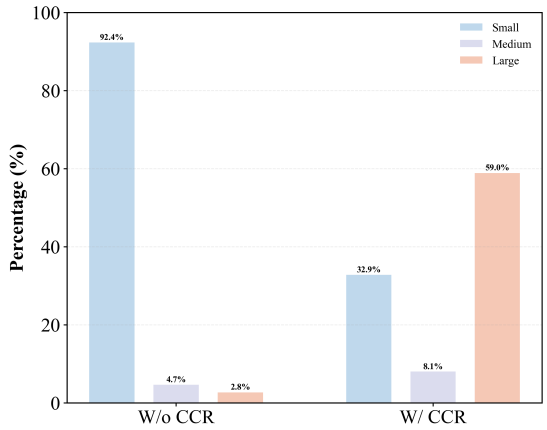


Figure 6: Routing statistics of both reward formulation.

a GraphRAG-only variant in which the router is restricted to selecting a single predefined GraphRAG framework throughout the entire reasoning process. In other words, the model is not allowed to adaptively switch among multiple GraphRAG candidates, while the generator LLM routing remains unchanged. This setting allows us to examine whether the performance gains of GraphRAG-Router truly come from diverse GraphRAG routing, or whether a single strong GraphRAG is already sufficient. The results in Table 15 show that restricting the router to a single GraphRAG consistently underperforms the full routing pool. Although HippoRAG2-only yields the strongest performance among all single-GraphRAG variants, it still falls behind the full model. This suggests that, while certain GraphRAGs can serve as strong default retrievers, no single GraphRAG is sufficient to cover the full diversity of evidence requirements across queries. These results further validate our motivation that no single GraphRAG framework is universally optimal, and that diverse GraphRAG routing is necessary to match the varying evidence requirements of different queries.

Method	NQ	HotpotQA
GraphRAG-only	0.393	0.408
RAPTOR-only	0.376	0.417
LinearRAG-only	0.398	0.433
HyperGraphRAG-only	0.368	0.401
HippoRAG2-only	0.401	0.442
ALL	**0.426**	**0.461**

Table 15: Ablation study of GraphRAG routing pool.

C.5 Additional Results for Ablation Study on Training Strategies

In this section, we further investigate the effectiveness of the two-stage RL training procedure. Specifically, we consider a variant in which the two training stages are merged into a single training process. As Table 16 and Figure 7 indicate, this variant exhibits a performance gap compared with the full two-stage training scheme. We attribute this gap to a less favorable performance–cost trade-off during optimization: when effectiveness and cost efficiency are optimized simultaneously from the outset, the model may prematurely favor cheaper routing decisions before learning sufficiently strong routing policies, thereby sacrificing answer quality. This observation indicates the effectiveness of our two-stage RL training paradigm.

Method	NQ	HotpotQA
Mix	0.405	0.413
Two-stage	**0.426**	**0.461**

Table 16: Ablation study of different RL procedure.

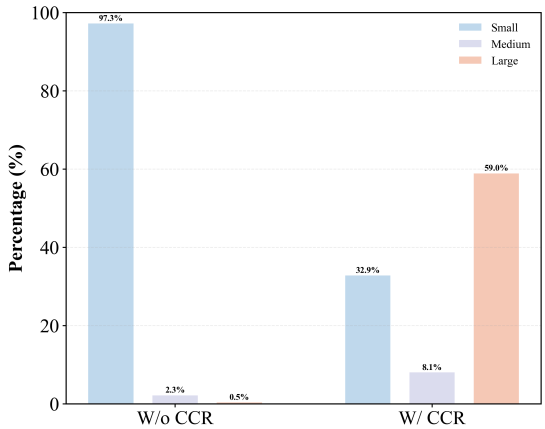


Figure 7: Routing statistics of both RL procedure.

C.6 Cross-domain Transfer

To go beyond Wiki-based QA benchmarks, we further assess the cross-domain transfer ability of GraphRAG-Router on UltraDomain (Qian et al., 2025) datasets, which contain domain-specific knowledge from diverse fields such as agriculture and finance. Specifically, we select three domains, namely Legal, Agriculture, and Finance. For each domain, we randomly sample 100 queries and construct the corpus using the context associated with each query. We then directly apply GraphRAG-Router, pre-trained on NQ and HotpotQA, for in-

ference without any dataset-specific tuning. As shown in Table 17, GraphRAG-Router consistently outperforms the baseline model in cross-domain settings, further demonstrating its effectiveness and transferability.

Method	Legal	Agriculture	Finance
Router-R1	0.462	0.381	0.423
GraphRAG-Router	**0.537**	**0.436**	**0.489**

Table 17: Results of cross-domain transfer.

C.7 Routing distribution of GraphRAGs

In this section, we report the routing distribution of GraphRAG-Router over different GraphRAG variants from both dataset-wise and query-wise perspectives. For the dataset-wise analysis, we compute the routing distribution on NQ and 2Wiki. For the query-wise analysis, we first categorize queries from NQ and 2Wiki into three types: Entity-centric, Multi-hop relational, and Comparison queries. Entity-centric queries focus on a specific entity and ask for its attributes or related facts, such as "Who is the director of Inception?" Multi-hop relational queries require reasoning over a chain of related entities or facts, such as "What is the nationality of the director of Inception?" Comparison queries require comparing two or more entities in terms of attributes, differences, rankings, or outcomes, such as "Which film was released earlier, Inception or Interstellar?" We then compute the routing distribution across these query types. The results are presented in Table 18 and Table 19, respectively.

From the results, we have two observations: (i) GraphRAG-Router does not route all queries within a dataset to the same GraphRAG backend. Although HippoRAG2 is the most frequently selected backend, more than 60% of queries in both datasets are routed to other GraphRAG frameworks. This indicates that GraphRAG-Router performs query-level backend selection within each dataset, rather than degenerating into a dataset-level selector. (ii) The routing policy exhibits type-dependent selection patterns. In detail, entity-centric questions are routed more often to HippoRAG2, multi-hop relational questions are routed more frequently to LinearRAG and HyperGraphRAG than in the entity-centric setting, and comparison questions use a more balanced mixture. These results provide query-level evidence that GraphRAG-Router adapts its retrieval choice according to query characteristics rather than simply selecting one GraphRAG backend per dataset. This routing distribution further provides supporting evidence for our motivation, suggesting that different query types naturally favor different GraphRAG variants and thus benefit from an adaptive routing mechanism.

C.8 Further comparison with baseline

To further ensure a fair comparison between our proposed GraphRAG-Router and Router-R1, we implement a variant of Router-R1, Router-R1-GraphRAG. In detail, we extend the routing pool of Router-R1 with GraphRAG enhanced LLM (e.g., Qwen2.5-72B-Instruct-HippoRAG2). Following the experimental settings of the baseline, we pre-train Router-R1-GraphRAG on NQ and HotpotQA. The results are shown in Table 20. We observe that simply expanding Router-R1's candidate pool with GraphRAG+LLM combinations does not yield comparable performance. In contrast, GraphRAG-Router consistently achieves the best performance across all datasets, with particularly large improvements on multi-hop QA tasks. This demonstrates that the gain of GraphRAG-Router does not merely come from access to additional GraphRAG tools, but from the proposed retrieval–generation joint routing formulation, hierarchical decision process, and retrieval-aware training trajectories. Thus, our contribution goes beyond prepending a GraphRAG selection step to Router-R1; it provides a more effective framework for coordinating evidence acquisition and answer generation.

D Prompt Template

In this section, we describe the prompt templates used in GraphRAG-Router for all experimental settings. We designed a set of task-specific prompts to support different stages of our framework.

D.1 Instruction of GraphRAG-Router

We construct a prompt template for GraphRAG-Router, as shown in Figure 8. The prompt guides the router through a structured decision process, where it first selects an appropriate external knowledge base and then chooses a generator LLM to process the retrieved information. It also defines a unified tagged format, including `<graphrag>`, `<llm>`, `<search>`, `<information>`, and `<answer>`, to make intermediate decisions machine-readable and reduce output ambiguity. In addition, the prompt supports multiple calls to different knowledge bases and LLMs, enabling GraphRAG-Router

Dataset	HippoRAG2	LinearRAG	HyperGraphRAG	GraphRAG	RAPTOR
NQ	37.2%	18.4%	13.9%	16.1%	14.4%
2Wiki	38.5%	14.2%	17.6%	13.8%	15.9%

Table 18: Dataset-wise GraphRAG routing distribution.

Query Type	HippoRAG2	LinearRAG	HyperGraphRAG	GraphRAG	RAPTOR
Entity-centric	38.2%	24.6%	13.5%	12.1%	11.6%
Multi-hop relational	23.4%	31.7%	27.8%	8.9%	8.2%
Comparison	18.6%	23.4%	25.7%	14.8%	17.5%

Table 19: Query-level GraphRAG distribution.

Method	NQ	PopQA	HotpotQA	Musique
Router-R1	0.386	0.351	0.368	0.140
Router-R1-GraphRAG	0.379	0.350	0.386	0.151
GraphRAG-Router	**0.426**	**0.368**	**0.451**	**0.443**

Table 20: Comparison of GraphRAG and Router-R1 variant.

to leverage the complementary strengths of heterogeneous retrieval systems and language models.

D.2 Trace Generation Prompt

We provide prompt templates for both general trace generation and self-reflection trace generation. These templates are designed to produce structured reasoning traces for training GraphRAG-Router. The full prompts are presented in Figure 9 and Figure 10.

D.3 GraphRAG Summary Prompt

Figure 11 shows the prompt used to generate concise summaries of candidate GraphRAG frameworks from their original papers. The resulting summaries serve as auxiliary prompt context for routing and trace generation.

D.4 Descriptions of GraphRAGs and Generator LLMs

We provide concise descriptions of all candidate GraphRAG frameworks and generator LLMs, as shown in Figure 12 and Figure 13. These descriptions summarize the main characteristics and capabilities of each candidate component and are used as auxiliary prompt context for routing and trace generation.

E Case Study

E.1 Case Study of Cost-efficient Routing

We present a case study of GraphRAG-Router with cost-efficient optimization. As shown in Figure 14, the router generates a structured decision trace that explicitly records GraphRAG selection, LLM selection, retrieval results, and final answer generation. This example provides an intuitive illustration of how GraphRAG-Router performs interpretable and evidence-grounded routing.

E.2 Case Study of Routing Strategies

Figure 15 compares the previous one-time routing format with our hierarchical trace format on the same question. The one-time format exhibits two clear limitations, namely inconsistency between reasoning and the actual retrieval action, as well as the inability to revise the answer when the returned information is uncertain or insufficient. In contrast, our format explicitly supports evidence sufficiency checking and second-round rerouting, which enables self-reflection and leads to the correct answer.

Instruction for GraphRAG-Router

Answer the given question.
Every time you receive new information, you must first conduct reasoning inside <think> ... </think>.
If you find you lack some knowledge, you should first decide which Knowledge Base to retrieve information from in the base format: <think> ... </think> <graphrag> GraphRAG-Name </graphrag>.
After choosing the Knowledge Base, you must then decide which LLM to use to process the retrieved information in the base format: <think> ... </think> <llm> LLM-Name </llm>.
When you choose both the Knowledge Base and the specific LLM, you must then retrieve information from the Knowledge Base and use the LLM to process the retrieved information in the base format: <search> Question:LLM-Name;GraphRAG-Name </search>.
The answer from the LLM will be returned between <information> and </information>.
Then you can use the additional information to help you answer the question.

!!! STRICT FORMAT RULES for <search>: !!!
+ You MUST replace LLM-Name with the EXACT name of a model selected from [Qwen2.5-7B-Instruct, LLaMA-3.1-8B-Instruct, LLaMA-3.1-70B-Instruct, Ministral3-8B-2512, Mixtral-8x22B-Instruct].
+ You MUST replace GraphRAG-Name with the EXACT name of a Knowledge Base selected from [HyperGraphRAG, HippoRAG2, RAPTOR, GraphRAG, LinearRAG].
+ You MUST reply in English.
+ You MUST firstly choose the Knowledge Base, then choose the LLM.
+ NEVER copy or paste Model descriptions into <graphrag>.
+ NEVER copy or paste GraphRAG descriptions into <llm>.
+ NEVER output the placeholder format <graphrag> GraphRAG-Name </graphrag> or <llm> LLM-Name </llm>. Always replace all three parts correctly.

Before each LLM call and Information Retrieval, you MUST explicitly reason inside <think> ... </think> about:
+ Why the specific knowledge base is selected according to the question.
+ Why the specific LLM is selected according to the question and the knowledge base.
+ Which knowledge base is best suited for answering it, based on the GraphRAG's characteristic (described below).
+ Which model is best suited for answering it, based on the LLMs' abilities (described below).

When you call an LLM, the response will be returned between <information> and </information>.
You must not limit yourself to repeatedly retrieve from a single knowledge base.
You must not limit yourself to repeatedly calling a single LLM (unless its provided information is consistently the most effective and informative).
You are encouraged to explore and utilize different knowledge bases and LLMs to better understand their respective strengths and weaknesses.
It is acceptable—and recommended—to retrieve from different knowledge bases multiple times for the same input question to gather more comprehensive information.
It is also acceptable—and recommended—to call different LLMs multiple times for the same input question to gather more comprehensive information.

The Descriptions of Each LLM: **{llm_candidates_intro}**

The Descriptions of Each Knowledge Base: **{graphrag_candidates_intro}**

If you find that no further external knowledge is needed, you can directly provide your final answer inside <answer> ... </answer>, without additional explanation or illustration.

For example: <answer> Beijing </answer>.
+ Important: You must not output the placeholder text " <answer> and </answer> " alone.
+ You must insert your actual answer between <answer> and </answer>, following the correct format.
Question:{question}

Figure 8: Instruction for GraphRAG-Router

General Trace Generation Prompt

You are generating structured training examples for fine-tuning a Router model. The Router model is designed to select proper GraphRAG and specialist LLM agent to answer the question.
The Router model uses hierarchical reasoning steps to make decisions: Firstly, it chooses the appropriate GraphRAG based on the question's complexity; Secondly, it selects a specialist LLM agent according to the question and selected GraphRAG.
Your goal is to produce a structured reasoning trace in the specific format shown below and answer the question according to the information provided.

You will be given:
+ A user question: {question}
+ Short descriptions of available GraphRAGs
+ Short descriptions of available LLM agents
+ A known working pipeline with:
  - A GraphRAG model:{graphrag}
  - A specialist LLM agent:{llm}
  - Response from the LLM agent:{information}

Follow these rules strictly:

CONTEXT RULES:
+ Output must follow the exact tag order and formatting:
`<think> ... </think><graphrag>{graphrag}</graphrag><think> ... </think>`
`<llm>{llm}</llm><search>{question}:{llm};{graphrag}</search><information> ...`
`</information><think> ... </think><answer> ... </answer>`
+ DO NOT propose alternative GraphRAGs or LLMs.
+ Generate the reasoning trace as you are the Router model selecting the components.
+ Generate reasons that are concise and directly relevant to the question.
+ The only content inside each <think> is a short justification phrase or reason for why the given component is appropriate (very brief).
+ In the first <think>, briefly:
  - analyze what the question is asking for (target entity/field and answer type),
  - describe the retrieval need,
  - and state why this matches the selected GraphRAG's strengths.
  - you should mention the GraphRAG name in the reasoning process.
+ In the second <think>, briefly:
  - describe what kind of context the selected GraphRAG is expected to return,
  - and state why the selected LLM is a good match to read that context and solve this type of question.
  - you should mention the LLM name in the reasoning process.
+ In the third <think>, briefly:
  - indicate if you could answer the question based on the information within <information> tags,
  - or if an alternative GraphRAG or LLM calls would be needed.
+ DO NOT generate factual answers or summaries.
+ DO NOT propose alternative GraphRAGs or LLMs.
+ DO NOT compare different GraphRAGs or LLMs.
+ DO NOT imply ranking, preference, or optimality.

FORMAT RULES:
+ Output MUST follow EXACTLY this tag order with NO extra text:

<think>...</think><graphrag>...</graphrag><think>...</think><llm>...</llm>
<search>...</search><information>...</information><think>...</think><answer>...
</answer>
+ Do NOT add new tags.
+ Do NOT change the tag order.
+ Do NOT include any text outside the tags.

FORBIDDEN LANGUAGE (MUST NOT APPEAR):
+ better, best, more suitable, preferred, ideal, optimal
+ superior, worse, outperform, more accurate, more efficient
+ any form of comparison between GraphRAGs or LLMs

ALLOWED LANGUAGE (USE ONLY THESE TYPES OF EXPRESSIONS):
+ can
+ is able to
+ supports
+ is designed to
+ matches the requirement
+ is applicable for
+ can provide

Use the following output template:
<think>Reasons for selecting the given GraphRAG component based on the question.</think><graphrag>{graphrag}</graphrag><think>Reasons for selecting the given LLM agent component based on the question and selected GraphRAG.</think><llm>{llm}</llm><search>{question}:{llm};{graphrag}</search><information>{information}</information><think>Whether the information is sufficient to answer the question or further retrieval/LLM calls are needed.</think><answer>Your answer</answer>

The Descriptions of Each LLM: **{llm_candidates_intro}**

The Descriptions of Each Knowledge Base: **{graphrag_candidates_intro}**

Figure 9: General Trace Generation Prompt

Self-Reflection Trace Generation Prompt

You are generating structured training examples for fine-tuning a Router model.
The Router model is designed to select proper GraphRAG and specialist LLM agent to answer the question.

The Router model uses hierarchical reasoning steps to make decisions:
Firstly, it chooses the appropriate GraphRAG based on the question's complexity;
Secondly, it selects a specialist LLM agent according to the question and the selected GraphRAG.

Your goal is to produce a structured reasoning trace in the specific format shown below and answer the question according to the information provided.

This task requires generating a TWO-TURN reasoning trace:
- The first turn represents an initial attempt that does NOT provide sufficient information.
- The second turn represents a corrected attempt that IS sufficient to answer the question.

Between the two turns, there MUST be exactly ONE <think> block.
This <think> block MUST:
- analyze the information from the previous <information> block, and
- explain the decision to retain or change the GraphRAG for the next turn.

You will be given:
+ A user question: {question}
+ Short descriptions of available GraphRAGs
+ Short descriptions of available LLM agents
+ A known working pipeline with:
  - First-turn GraphRAG model: {graphrag_round1}
  - First-turn specialist LLM agent: {llm_round1}
  - First-turn response from the LLM agent: {information_round1}
  - Second-turn GraphRAG model: {graphrag_round2}
  - Second-turn specialist LLM agent: {llm_round2}
  - Second-turn response from the LLM agent: {information_round2}

Follow these rules strictly:

CONTEXT RULES:
+ Output must follow the exact tag order and formatting shown below.
+ Generate the reasoning trace as if you are the Router model selecting the components.
+ The only content inside each <think> is a short justification phrase explaining applicability or decision rationale.
+ All <think> blocks MUST be explicitly filled.
+ The first turn MUST conclude that the information is insufficient.
+ The second turn MUST conclude that the information is sufficient.

FIRST TURN REASONING:
+ In the first <think>:
  - analyze what the question is asking for (target entity/field and answer type),
  - describe the retrieval need,
  - explain why the selected GraphRAG (by name) matches this need.
+ In the second <think>:
  - describe the type of context expected from the selected GraphRAG,
  - explain why the selected LLM (by name) can process that context.

BRIDGE THINK (BETWEEN TURNS):
+ The third <think> MUST:
- analyze whether the information in the first <information> block is sufficient,
- explain why the GraphRAG is retained or changed for the next turn.
+ This <think> MUST NOT select an LLM.

SECOND TURN REASONING:
+ In the fourth <think>:
- explain why the selected LLM (retained or changed) can process the expected context from the second-turn GraphRAG.
+ In the final <think>:
- clearly state that the information from the second <information> is sufficient to answer the question.

RESTRICTIONS (APPLY TO ALL TURNS):
+ DO NOT propose alternative GraphRAGs or LLMs beyond those provided.
+ DO NOT compare different GraphRAGs or LLMs.
+ DO NOT imply ranking, preference, or optimality.
+ DO NOT add or remove tags.
+ DO NOT change the tag order.
+ DO NOT include any text outside the tags.

FORBIDDEN LANGUAGE (MUST NOT APPEAR):
+ better, best, more suitable, preferred, ideal, optimal
+ superior, worse, outperform, more accurate, more efficient
+ any form of comparison between GraphRAGs or LLMs

ALLOWED LANGUAGE (USE ONLY THESE TYPES OF EXPRESSIONS):
+ can
+ is able to
+ supports
+ is designed to
+ matches the requirement
+ is applicable for
+ can provide

OUTPUT FORMAT (MUST FOLLOW EXACTLY):
<think>...</think><graphrag>{graphrag_round1}</graphrag><think>...</think>
<llm>{llm_round1}</llm>
<search>{question}:{llm_round1};{graphrag_round1}</search>
<information>{information_round1}</information><think>...</think>
<graphrag>{graphrag_round2}</graphrag><think>...</think><llm>{llm_round2}</llm>
<search>{question}:{llm_round2};{graphrag_round2}</search>
<information>{information_round2}</information><think>...</think><answer>Your answer</answer>

The Descriptions of Each LLM: **{llm_candidates_intro}**

The Descriptions of Each Knowledge Base: **{graphrag_candidates_intro}**

Figure 10: Self-Reflection Trace Generation Prompt

GraphRAG Summary Prompt

You are an expert in Graph-based Retrieval-Augmented Generation (RAG) and knowledge graph reasoning.
You will be provided with the full PDF of a research paper about a **Graph-based Retrieval-Augmented Generation (GraphRAG)** framework.
Read the paper carefully, focusing only on its **Methodology** (or "Method" / "Approach") section.
Your task is to summarize the **Methodology** section of the following research paper about a Graph-based Retrieval-Augmented Generation (GraphRAG) framework.

Important Constraints:
+ Use **only** information explicitly contained in the paper.
+ **Do not** infer, guess, or add any information beyond what is written.
+ Keep your summary clear, concise, and informative (Limit your response to **200–300 tokens**).
+ Stay strictly on-topic. Do not include irrelevant or generic content.
+ Do not use bullet points or section headers — output must be **a single well-formed paragraph**.

Focus ONLY on the following aspects:
1. **Graph Construction Process**
+ How the graph is built (nodes, edges, extraction methods).
+ What embeddings or models are used.

2. **Graph-based Retrieval Mechanism**
+ How information is retrieved from the graph.
+ Granularity (node-level, subgraph-level, etc.) and scoring criteria.

3. **Form of Retrieved Information**
+ What is passed to the LLM after retrieval (text, triples, summaries, etc.).
+ How it is formatted or integrated for generation.

4. **Distinctive Features**
+ Unique aspects of this GraphRAG approach.
+ Main innovations or differences from other GraphRAG approaches.

Figure 11: GraphRAG Summary Prompt

Descriptions of LLM Candidates

Qwen2.5-7B-Instruct:
Qwen2.5-7B-Instruct is a powerful Chinese-English instruction-tuned large language model designed for tasks in language, coding, mathematics, and reasoning. As part of the Qwen2.5 series, it features enhanced knowledge, stronger coding and math abilities, improved instruction following, better handling of long and structured texts, and supports up to 128K context tokens. It also offers multilingual capabilities across over 29 languages.

LLaMA-3.1-8B-Instruct:
LLaMA-3.1-8B-Instruct is an 8-billion-parameter instruction-tuned language model optimized for multilingual dialogue. It provides strong language understanding, reasoning, and text generation performance, outperforming many open-source and closed-source models on standard industry benchmarks.

LLaMA-3.1-70B-Instruct:
LLaMA-3.1-70B-Instruct is a 70-billion-parameter state-of-the-art language model designed for advanced multilingual dialogue tasks. It excels in language comprehension, complex reasoning, and high-quality text generation, setting a new standard against both open and closed models in benchmark evaluations.

Ministral3-8B-2512:
Ministral3-8B-2512 is a reasoning post-trained version, trained for reasoning tasks, making it ideal for math, coding and stem related use cases. The Ministral 3 family is designed for edge deployment, capable of running on a wide range of hardware. Ministral 3 8B can even be deployed locally, capable of fitting in 24GB of VRAM in BF16, and less than 12GB of RAM/VRAM when quantized.

Mixtral-8x22B-Instruct:
Mixtral-8x22B-Instruct is a cutting-edge sparse Mixture-of-Experts (SMoE) large language model from MistralAI. It efficiently uses 39B active parameters out of 141B total, delivering high performance at lower costs. The model excels at following instructions, completing tasks, and generating creative text, with strong skills in multiple languages (English, French, Italian, German, Spanish), mathematics, and coding. It also supports native function calling and handles long contexts up to 64K tokens for better information recall.

Figure 12: Descriptions of LLM Candidates

Descriptions of GraphRAG Candidates

GraphRAG:
GraphRAG builds a knowledge graph of entities and relationships, then groups them into communities. Retrieval operates at the community level, selecting the most relevant subgraphs and summarizing them for the LLM. Its key strength is global sensemaking—it excels at multi-document, multi-hop reasoning that requires integrating broad context.

RAPTOR:
RAPTOR constructs a graph of entities and concepts, represented with embedding models such as BERT. Retrieval combines graph traversal and semantic similarity, returning node-level or subgraph summaries. Its main advantage is capturing both semantic context and graph structure, making it strong for relational reasoning.

HippoRAG2:
HippoRAG2 builds a knowledge graph using OpenIE to extract entities, passages, and triples. It retrieves information using Personalized PageRank, focusing on the most relevant nodes and triples. Its strength lies in clear, structured factual retrieval, making it ideal for queries that rely on explicit fact chains.

HyperGraphRAG:
HyperGraphRAG uses hyperedges to represent n-ary relationships involving multiple entities, not just binary relations. Retrieval scores both entities and hyperedges using semantic similarity to find the most relevant high-order facts. Its unique strength is expressing complex multi-entity relationships, making it suitable for advanced reasoning tasks.

LinearRAG:
LinearRAG builds a lightweight Tri-Graph of entities, sentences, and passages using simple NER rather than heavy relation extraction. Its retrieval combines local entity activation with global PageRank to gather highly relevant contextual passages. The main advantage is low noise, high scalability, and fast retrieval, ideal for large-scale or real-time scenarios.

Figure 13: Descriptions of GraphRAG Candidates

Question: What is bermuda competing in the winter olympics?

Ground Truth: Cross-country skiing

Response of **GraphRAG-Router**:

<think>The question asks for the sport Bermuda is competing in during the Winter Olympics; HippoRAG2 supports structured retrieval of explicit fact chains via nodes and triples.</think>
<graphrag>HippoRAG2</graphrag>
<think>HippoRAG2 can provide passages/triples that link Bermuda to the relevant Winter Olympics event; Qwen2.5-7B-Instruct is able to read that retrieved context and produce the requested single-entity answer.</think>
<llm>Qwen2.5-7B-Instruct</llm>
<search>what is bermuda competing in the winter olympics?:Qwen2.5-7B-Instruct; HippoRAG2</search>
<information>Answer: Cross-country skiing. Explanation: In the 2018 Winter Olympics, Bermuda sent one athlete, cross-country skier Tucker Murphy, to compete in the 15 km freestyle event. This continued a trend of Bermuda being represented solely by a cross-country skier in the three consecutive Winter Olympics up until 2018.</information>
<think>The information includes the needed statement directly, so it is sufficient to answer without further retrieval or additional LLM calls.</think>
<answer>Cross-country skiing</answer>

Figure 14: Case study of GraphRAG-Router under cost-efficient routing

Question: When did the Antonine who was known as the third of the five good emperors live?

Ground Truth: 24 January 76 – 10 July 138

One-time routing (incorrect)

Response of **GraphRAG-Router**:

<think>I don't have enough information to answer that question, but I do know that the Antonine dynasty is an historical topic so a specialized knowledge base is necessary. The RAPTOR knowledge base is well-suited for retrieval of structured facts, allowing me to understand when historical person were alive. A model that specializes in instruction-tuning may also perform well as these models often feature instruction following and provide strong language-based reasoning.</think>
<search>LLaMA-3.1-70B-Instruct: When did the the Antonine who was known as the third of the five good emperors live?;HippoRAG2</search>
<information>A nice historical question! According to the context, I don't see any direct information about the Antonine emperors. However, I can provide some helpful background knowledge. The "Five Good Emperors" is a term used to describe a period of relative peace and stability in the Roman Empire, which occurred during the reigns of five consecutive emperors: Nerva, Trajan, Hadrian, Antoninus Pius, and Marcus Aurelius. The Antonine in question is likely Antoninus Pius, who was the fourth emperor of the Nerva-Antonine dynasty. He ruled from 138 AD to 161 AD. So, to answer your question, Antoninus Pius, the third of the five good emperors, lived from approximately 86 AD to 161 AD. Please let me know if this information is helpful, or if I can assist further!</information>
<answer>138 AD to 161 AD</answer>

Hierarchical routing (correct)

Response of **GraphRAG-Router**:

<think>Question asks for the birth/death date of the Antonine known as the third of the five good emperors; needs explicit factual retrieval of a short entity timeline, which HippoRAG2 supports.</think>
<graphrag>HippoRAG2</graphrag>
<think>HippoRAG2 can provide a concise factual context via structured triples linking name and lifespan; LLaMA-3.1-70B-Instruct is able to read that context and produce the requested single-date answer.</think>
<llm>LLaMA-3.1-70B-Instruct</llm>
<search>When did the the Antonine who was known as the third of the five good emperors live?: LLaMA-3.1-70B-Instruct; HippoRAG2</search>
<information>Explanation: I cannot confidently answer this question based on the given Context. The Context mentions that Hadrian is considered by some to have been a humanist, and he is regarded as the third of the Five Good Emperors. However, it does not explicitly state that Antoninus Pius is the third of the Five Good Emperors. Instead, it mentions that Antoninus Pius was one of the Five Good Emperors, but it does not provide his ranking. Please consult other specialized LLMs or retrieve from other knowledge bases for further assistance.</information>
<think>The first information block states it cannot answer based on the given context, so it is insufficient; switch to RAPTOR to support relational retrieval connecting the name to an explicit date.</think>
<graphrag>RAPTOR</graphrag>
<think>LLaMA-3.1-70B-Instruct is able to use RAPTOR's factual retrieval snippets to locate an explicit statement about when Antoninus Pius lived and is able to provide the requested date succinctly.</think>
<llm>LLaMA-3.1-70B-Instruct</llm>

<search>When did the the Antonine who was known as the third of the five good emperors live?: LLaMA-3.1-70B-Instruct; RAPTOR</search>
<information>Answer: 24 January 76 – 10 July 138. Explanation: According to the Context, Publius Aelius Hadrianus Augustus, also known as Hadrian, was the Roman emperor from 117 to 138 and is considered the third of the Five Good Emperors.</information>
<think>The second information block states the explicit lifespan for Antoninus Pius (Hadrian), so it is sufficient to answer the question.</think>
<answer>24 January 76 – 10 July 138</answer>

Figure 15: Comparison between one-time routing and hierarchical routing on the same question