Introspective Graph Retrieval-Augmented Generation

Anonymous ACL submission

Abstract

Graph Retrieval-Augmented Generation (GraphRAG) grounds language models by relational retrieval, yet existing systems typically treat the retrieval graph as fixed despite its noisy construction. We observe that missing edges, spurious relations, and suboptimal local reasoning structures fundamentally limit downstream performance. We propose *Introspective GraphRAG* (IGRAG), a utility-aware subgraph refinement framework that improves retrieval quality through persistent memory evolution. Our key insight is that global retrieval quality can be optimized through local subgraph refinement: given a retrieved subgraph, IGRAG augments it with structurally similar high-utility reasoning substrates from persistent memory, then jointly performs candidate refinement and utility-aware reranking using a graph-conditioned language model. Utility-validated refined subgraphs are subsequently inserted into persistent memory, enabling retrieval behavior to continuously improve across queries while preserving compatibility with arbitrary existing GraphRAG systems through a fixed base graph. Experiments on GraphRAG and KGQA benchmarks demonstrate consistent improvements in both retrieval and generation quality across diverse host systems.

1 Introduction

Retrieval-Augmented Generation (RAG) has become the dominant paradigm for grounding large language models in external knowledge (Lewis et al., 2020; Guu et al., 2020; Karpukhin et al., 2020). However, as applications increasingly require multi-hop and cross-document reasoning, flat passage retrieval becomes insufficient: evidence relevant to a query is often distributed across multiple documents and connected only through latent relations. *Graph Retrieval-Augmented Generation* (GraphRAG) addresses this limitation by organizing corpora as graphs of entities, chunks, or summaries, and retrieving query-relevant subgraphs that expose non-local evidence through graph structure (Edge et al., 2024; Gutiérrez et al., 2024; Guo et al., 2024; Mavromatis and Karypis, 2024; He et al., 2024; Hu et al., 2025). Consequently, GraphRAG has rapidly become the de facto framework for knowledge-intensive Large Language Model (LLM) applications.

A GraphRAG system typically consists of three components: a graph index, a retriever that extracts a subgraph for a query, and a generator conditioned on the retrieved subgraph (or its associated documents). Most existing work improves either the retriever or the generator while treating the retrieval graph as fixed. Yet the graph index fundamentally bounds downstream reasoning: missing edges prevent evidence discovery, while spurious relations propagate irrelevant context regardless of retriever or generator sophistication. In our analysis across multiple GraphRAG systems, incorrect answers frequently originate from corrupted local graph structure that downstream components cannot recover (Xiang et al., 2025). Despite its importance, the retrieval graph remains static in practice for two reasons: graph construction is expensive, and direct supervision for graph refinement is unavailable. Nevertheless, deployed GraphRAG systems already produce a continuous stream of indirect feedback on graph quality through downstream utility signals generated by retrieved subgraphs. This motivates a natural question:

> ***Can downstream task utility be used to continuously improve GraphRAG system?***

Closing this loop introduces following challenges:

- *Weak non-decomposable supervision.* Utility signals are scalar, non-differentiable, and defined over entire retrieved subgraphs.
- *Combinatorial search space.* Real-world re-

trieval graphs contain millions of edges, making global graph optimization intractable.

We introduce *Introspective GraphRAG*, a new formulation of GraphRAG in which retrieval systems continuously improve through downstream utility feedback. To instantiate this formulation, we propose a utility-aware subgraph refinement framework based on a key observation: only retrieved local subgraphs influence downstream utility. Instead of globally rewriting the retrieval graph, the framework refines retrieved reasoning subgraphs through utility-aware reranking and persistent memory evolution. Given a retrieved subgraph, it first augments the subgraph with structurally similar high-utility reasoning substrates retrieved from persistent memory, then jointly performs candidate refinement and utility-aware reranking using a graph-conditioned language model. Utility-validated refined subgraphs are subsequently inserted back into persistent memory, enabling retrieval behavior to continuously improve across queries while preserving compatibility with arbitrary existing GraphRAG systems through a fixed base graph.

Our contributions are summarized as follows:

- We formulate *Introspective GraphRAG* as a weakly-supervised subgraph optimization problem under non-decomposable utility supervision.
- We propose a utility-aware subgraph refinement framework that combines graph-conditioned reranking with persistent memory evolution to construct reusable reasoning substrates.
- We demonstrate that a lightweight trained refiner generalizes across heterogeneous GraphRAG systems and both document-grounded GraphRAG and KGQA settings without modifying the underlying retriever or generator.

2 Related Work

2.1 Graph Retrieval-Augmented Generation

RAG enhances language models by retrieving external knowledge during inference (Lewis et al., 2020; Guu et al., 2020; Karpukhin et al., 2020). Recent work further incorporates relational structure into retrieval, giving rise to GraphRAG. Existing GraphRAG systems construct graphs over documents, entities, or chunks to support multi-hop reasoning and structure-aware retrieval (Edge et al., 2024; Gutiérrez et al., 2024; Guo et al., 2024; Luo et al., 2026). Other methods integrate graph neural networks or explicit graph traversal into retrieval pipelines, including G-Retriever (He et al., 2024), GRAG (Hu et al., 2025), GNN-RAG (Mavromatis and Karypis, 2024), and PathRAG (Chen et al., 2026). However, most existing GraphRAG methods treat graph index as fixed after construction. Graphs are typically built through offline heuristic extraction pipelines and static similarity linking, while retrieval operates only over the resulting frozen structure. Consequently, retrieval failures caused by missing edges, noisy neighborhoods, or poor graph organization cannot be corrected through downstream retrieval feedback.

2.2 Generation-Augmented Retrieval

Generation-Augmented Retrieval (GAR) improves retrieval quality through query generation, rewriting, or iterative refinement before retrieval. Query expansion methods such as HyDE (Gao et al., 2023) generate hypothetical documents to bridge semantic gaps between queries and relevant documents, while subsequent works explore query rewriting and iterative retrieval refinement (Ma et al., 2023; Feng et al., 2024). Reinforced-IR (Li et al., 2025) introduces mutual optimization between retrievers and generators through iterative feedback. More recently, retrieval has been integrated into agentic and reasoning-centric workflows where retrieval trajectories are dynamically refined through intermediate reasoning (Yao et al., 2022; Jeong et al., 2024). ERM (Hu et al., 2026) transforms transient query-time improvements into persistent retrieval gains through correctness-gated memory evolution and progressive memory updates. However, existing GAR methods primarily optimize query reformulation or retrieval trajectories while treating the retrieval index as fixed after construction. As a result, they cannot continuously refine graph-structured retrieval substrates or improve structural dependencies through downstream utility feedback.

3 Problem Formalization

GraphRAG. Let $\mathcal{D}$ denote a document corpus consisting of documents or text chunks. A graph index $G = (V, E)$ provides structured access to $\mathcal{D}$, where each node $v \in V$ is associated with one or more chunks through a mapping $\pi : V \to 2^{\mathcal{D}}$, and edges $E \subseteq V \times V$ capture relations such as semantic similarity, citations, or entity co-occurrence. GraphRAG enhances LLM generation by retrieving query-relevant subgraphs from G and grounding

generation in the associated document content.

Given a query q, a graph retriever Ret_ϕ extracts a subgraph

$$S_q = \mathrm{Ret}_\phi(G, q) \subseteq G, \tag{1}$$

whose associated retrieval context is $\mathcal{D}_{S_q} = \bigcup_{v \in V(S_q)} \pi(v)$. An LLM generator p_θ then produces output Y_q conditioned on q and $\mathcal{D}_{S_q}$. The GraphRAG objective is:

$$(\phi^*, \theta^*) = \arg\max_{\phi,\theta} \mathbb{E}_{q\sim\mathcal{Q}}\big[\mathcal{U}(q, S_q, Y_q)\big], \tag{2}$$

where $Y_q \sim p_\theta(\cdot \mid q, \mathcal{D}_{S_q})$ and $\mathcal{U}$ is a task-specific utility function. Although generation conditions on the retrieved documents $\mathcal{D}_{S_q}$, the utility fundamentally depends on S_q, since graph structure determines which evidence becomes retrievable.

Existing GraphRAG systems optimize either the retriever (Mavromatis and Karypis, 2024), the generator (Hu et al., 2025; He et al., 2024), or operate over a static graph index (Gutiérrez et al., 2024; Edge et al., 2024; Guo et al., 2024). However, retrieval graphs are often misaligned with downstream utility: LLM-extracted graphs suffer from missing or spurious relations, while curated knowledge graphs contain many irrelevant edges for a given query. Such structural errors propagate through retrieval and generation without correction, fundamentally limiting downstream performance.

Introspective GraphRAG. Instead of optimizing only the retriever or generator, we formulate GraphRAG as a retrieval-structure optimization problem over the graph substrate itself:

$$G^* = \arg\max_{G} \mathbb{E}_{q\sim\mathcal{Q}}\big[\mathcal{U}(q, S_q, Y_q)\big], \tag{3}$$

where $S_q = \mathrm{Ret}_\phi(G, q)$ and $Y_q \sim p_\theta(\cdot \mid q, \mathcal{D}_{S_q})$. Since retrieval behavior is determined by graph structure, improving the retrieval substrate yields more informative subgraphs and stronger contextual evidence without modifying either the retriever or generator.

In practice, node construction is comparatively well-posed: nodes can be initialized from chunks, entities, or code units using existing parsers and encoders. The relational structure, however, is both the primary source of graph-construction error and the key distinction between GraphRAG and standard RAG, since it governs how evidence propagates during retrieval. We therefore fix the node set V and optimize only the edge structure:

$$E^* = \arg\max_{E \subseteq V \times V} \mathbb{E}_{q\sim\mathcal{Q}}\big[\mathcal{U}(q, S_q, Y_q)\big]. \tag{4}$$

Optimizing Eq. (4) presents two fundamental challenges:

- **Combinatorial structure space.** The candidate edge set scales as $\mathcal{O}(|V|^2)$, which is infeasible to enumerate for document graphs.
- **Absence of edge-level supervision.** The only available learning signal is the scalar downstream utility $\mathcal{U}(q, S_q, Y_q)$; no ground-truth graph or per-edge supervision is available.

4 Methodology

4.1 Overview

Eq. (4) cannot be optimized with conventional graph structure learning since GraphRAG operates under streaming query distributions where supervision is sparse, query-dependent, and only indirectly observable through downstream utility. We therefore reformulate global retrieval-structure optimization as local utility-aware subgraph refinement.

Locality decomposes the objective. Only the retrieved local subgraph participates in downstream generation, yielding the approximate decomposition

$$\mathbb{E}_{q\sim\mathcal{Q}}\big[\mathcal{U}(q, S_q, Y_q)\big] \approx \sum_{S \in \mathcal{S}(G)} P(S)\,\bar{\mathcal{U}}(S), \tag{5}$$

where $P(S)$ denotes the retrieval frequency of subgraph S, and $\bar{\mathcal{U}}(S)$ is its conditional expected utility. Improving frequently retrieved local subgraphs therefore proportionally improves the overall retrieval substrate, reducing Eq. (4) to a sequence of local utility-aware subgraph refinements.

IGRAG realizes this view through three tightly coupled components sharing a graph-conditioned language model $\mathcal{F}_\omega$:

(1) a **utility-aware subgraph refiner** that proposes local subgraph refinements and scores the resulting candidates by estimated downstream utility (Section 4.2);

(2) a **persistent subgraph memory** $\mathcal{P}_t$, initialized from local ego-graphs and continuously expanded with utility-validated refined subgraphs, providing reusable structural priors for future queries (Section 4.3);

(3) a **utility-gated optimization strategy** that trains $\mathcal{F}_\omega$ from outcome-labelled candidates and, at inference, performs utility-aware

reranking together with persistent memory evolution while keeping the base graph fixed (Section 4.4).

Inference-time pipeline. Given a query q, IGRAG (i) retrieves a local subgraph S_q from the host GraphRAG pipeline; (ii) retrieves structurally similar high-utility subgraphs from $\mathcal{P}_t$ and merges them into an augmented substrate $\hat{S}_q$; (iii) decodes candidate refinements to construct a candidate pool $\mathcal{C}(q)$; (iv) scores and selects the highest-utility candidate $\tilde{S}_q^\star$; and (v) optionally validates $\tilde{S}_q^\star$ end-to-end before inserting it into $\mathcal{P}_t$ for future retrieval. The selected subgraph or its associated document chunks are then forwarded to the downstream generator, making IGRAG a plug-and-play enhancement module for arbitrary GraphRAG systems.

4.2 Utility-Aware Subgraph Refiner

Architecture. $\mathcal{F}_\omega$ is a graph-conditioned language model built on a pretrained LLM backbone with three lightweight trainable components: a graph encoder GE, a projector Π implemented as an MLP mapping graph embeddings into the LLM embedding space, and two task heads for refinement and utility prediction.

Given an augmented subgraph $\hat{S}_q$, query q, and candidate node pair $(u, v) \in V(\hat{S}_q) \times V(\hat{S}_q)$ sampled from a retrieval-local candidate edge set, the model input is:

$$\mathbf{H}_{\text{input}} = \left[\Pi(\mathbf{g}_{\hat{S}_q}); \text{Embed}([p; q; u; v])\right], \quad (6)$$

where $\mathbf{g}_{\hat{S}_q}$ is the pooled graph representation and p is a fixed instruction prompt.

Two output heads. Conditioned on $\mathbf{H}_{\text{input}}$, the refiner predicts:

$$p_\omega\big(\Delta S_q \mid \hat{S}_q, q\big), \qquad s_\omega\big(\tilde{S}_q \mid q\big) \in \mathbb{R}, \quad (7)$$

where ΔS_q is a sequence of elementary local edits (edge addition, removal, or relabeling), and

$$\tilde{S}_q = \hat{S}_q \oplus \Delta S_q \quad (8)$$

is the resulting refined subgraph. The edits are applied only to the retrieved local subgraph during candidate construction and are never committed to the base retrieval graph.

Candidate generation. Decoding m edit sequences produces a candidate pool

$$\mathcal{C}(q) = \{\tilde{S}_q^{(j)}\}_{j=1}^m \cup \{S_q\}, \quad (9)$$

where the original retrieved subgraph is retained as a no-op baseline. Each candidate is subsequently scored by the shared utility head.

4.3 Persistent Subgraph Memory

The persistent memory

$$\mathcal{P}_t = \{(S_i, z_i)\}_i \quad (10)$$

stores historically successful reasoning substrates together with their graph embeddings

$$z_i = \Pi(\mathbf{g}_{S_i}), \quad (11)$$

computed using the same encoder as the refiner.

Initialization. $\mathcal{P}_0$ is initialized using k-hop ego-graphs extracted from the original retrieval graph G_0 for $k \in \{1, \ldots, K\}$, providing local structural coverage before training.

Memory retrieval and substrate construction. Given a retrieved subgraph S_q with embedding z_{S_q}, we retrieve the top-n most similar memory entries:

$$\mathcal{N}_q^{\mathcal{P}} = \underset{(S_i, z_i) \in \mathcal{P}_t}{\text{Top-n}} \cos(z_{S_q}, z_i). \quad (12)$$

The retrieved structures are merged with S_q:

$$\hat{S}_q = S_q \cup \text{Merge}(\mathcal{N}_q^{\mathcal{P}}), \quad (13)$$

where $\text{Merge}(\cdot)$ denotes graph union followed by duplicate-edge consolidation. The resulting augmented substrate is consumed by the refiner. Unlike increasing retrieval breadth, memory retrieval injects historically utility-validated local reasoning structures, improving structural quality rather than merely expanding context volume.

Online evolution. The memory is updated only with utility-validated refined subgraphs, allowing retrieval behavior to continuously improve while keeping the base retrieval graph fixed.

4.4 Optimization

Outcome-labelled candidates. For each training query q, we first retrieve a baseline subgraph

$$S_q = \text{Ret}_\phi(G_0, q) \quad (14)$$

and compute its downstream utility $\mathcal{U}(q, S_q)$. Applying the refiner to the augmented substrate $\hat{S}_q$ generates a candidate pool $\mathcal{C}(q)$, where each candidate is re-evaluated end-to-end under the host GraphRAG pipeline.

This partitions the candidate pool into utility-improving and utility-degrading subsets:

$$\mathcal{C}^{\pm}(q) = \{\tilde{S} \in \mathcal{C}(q) : \mathcal{U}(q, \tilde{S}) \gtrless \mathcal{U}(q, S_q)\}. \quad (15)$$

Joint training objective. The utility head is trained using a margin-based preference loss:

$$\mathcal{L}_{\text{pref}} = \mathbb{E}_{q,\tilde{S}^{\pm}}\Big[\gamma - s_\omega(\tilde{S}^{+} \mid q) + s_\omega(\tilde{S}^{-} \mid q)\Big]_{+}, \tag{16}$$

while the refinement head is trained to imitate edit sequences associated with utility-improving candidates:

$$\mathcal{L}_{\text{mut}} = -\mathbb{E}_{q,\tilde{S}^{+}} \log p_\omega(\Delta S_q^{+} \mid \hat{S}_q, q). \tag{17}$$

The final objective is:

$$\mathcal{L}(\omega) = \mathcal{L}_{\text{pref}} + \lambda \mathcal{L}_{\text{mut}}. \tag{18}$$

Inference-time reranking. At inference, the refiner generates candidate subgraphs and selects:

$$\tilde{S}_q^{\star} = \arg\max_{\tilde{S} \in \mathcal{C}(q)} s_\omega(\tilde{S} \mid q). \tag{19}$$

With $\mathcal{P}_t$ fixed, Eq. (19) already provides a model-agnostic utility-aware subgraph reranker that can be attached to arbitrary GraphRAG systems.

Persistent memory evolution. Beyond reranking, IGRAG continuously evolves the persistent memory $\mathcal{P}_t$. After selecting $\tilde{S}_q^{\star}$, we optionally verify its end-to-end utility under the host GraphRAG pipeline. If

$$\mathcal{U}(q, \tilde{S}_q^{\star}) \geq \mathcal{U}(q, S_q) + \epsilon, \tag{20}$$

the refined subgraph is inserted into persistent memory:

$$\mathcal{P}_{t+1} = \mathcal{P}_t \cup \{(\tilde{S}_q^{\star}, z_{\tilde{S}_q^{\star}})\}. \tag{21}$$

Consequently, retrieval progressively evolves through an expanding library of utility-validated reasoning substrates, while the base retrieval graph remains unchanged.

5 Experiments

We evaluate IGRAG on the document-grounded GraphRAG-Bench (Xiang et al., 2025) and the knowledge-graph QA benchmarks WEBQSP (Yih et al., 2015) and CWQ (Talmor and Berant, 2018), plugging the framework into three document-grounded hosts (HIPPORAG2 (Gutiérrez et al., 2024), LIGHTRAG (Guo et al., 2024), RAPTOR (Sarthi et al., 2024)) and three KGQA hosts (GNN-RAG (Mavromatis and Karypis, 2024), GRAG (Hu et al., 2025), G-RETRIEVER (He et al., 2024)). All reported numbers are the mean of three independent runs. Datasets, metric definitions, baseline implementations, hyperparameters, and the full retrieval-side panel are in App. A and App. B.

Refiner instantiation across tables. The four main tables of this section (Tables 1 to 4, with the small exception noted below for Table 4) use the same IGRAG configuration: a fine-tuned **Qwen3-8B-Instruct** as the subgraph refiner (Section 4.2), trained on outcome-labelled candidate pools mined from the GraphRAG-Bench Novel and WEBQSP training splits (training details in App. A), with the same graph encoder, the same gate threshold ϵ=0, and GPT-5.2 used only as the graph-index extractor and as the host generator. Table 4 is the construction-backbone ablation: when refinement is enabled, that table re-uses the row's builder LLM as the refiner (*without* fine-tuning), so the IGRAG-on rows there do not use Qwen3-8B. The refiner-backbone ablation in Table 8 (App. B.4) and the analyses in App. B.5, B.6, B.7, and B.2 likewise do not use Qwen3-8B: they instantiate the refiner with zero-shot GPT-5.2 or GPT-4.1-mini in order to isolate the contribution of pipeline-versus-backbone-versus-training. We will flag the refiner choice in the caption or accompanying prose for every such table.

5.1 Main Results

IGRAG consistently improves heterogeneous GraphRAG systems. In Tables 1 to 3, the same tuned Qwen3-8B refiner improves all six host systems across the document-grounded GraphRAG-Bench and the curated-KG benchmarks, with paired Δ_{Native} positive on every Fact-Retrieval and Complex-Reasoning ACC column (+0.31 to +2.22) and on every KGQA Hit / H@1 / F1 cell. The gains transfer across substantively different graph substrates (entity–relation graphs, dual entity–chunk graphs, hierarchical retrieval trees, and a curated Freebase KG), establishing IGRAG as a host-agnostic refinement operator rather than a system-specific wrapper.

Table 4: Construction-backbone ablation on HippoRAG2 / Novel. When refinement is on, the builder LLM is also used as the (inference-only) refiner; Qwen3-8B is intentionally not used here.

Graph builder	IGRAG	Generation		Retrieval	
		ACC	ROUGE-L	Recall	Relevance
w/o Graph Refiner					
GPT-4.1	✗	21.97	5.74	16.58	12.09
GPT-5.2	✗	24.79	6.30	19.17	14.50
w/ Graph Refiner (same backbone as builder)					
GPT-4.1	✓	24.46	6.39	18.73	14.05
GPT-5.2	✓	26.04	7.01	20.33	15.14

Table 1: Generation accuracy on GraphRAG-Bench (Xiang et al., 2025), all values $\times 100$. + IGRAG *(ours)* uses the tuned Qwen3-8B refiner. **Best** bold, second-best underlined within each host-method block.

Method	Mode	Fact Retrieval		Complex Reasoning		Contextual Summarize		Creative Generation		
		ACC	ROUGE-L	ACC	ROUGE-L	ACC	Cov	ACC	FS	Cov
Novel Dataset										
HippoRAG2	Native	26.07	7.19	23.53	5.42	25.34	11.93	24.20	76.04	7.15
	+ chunk reranker	26.55	7.27	23.64	5.61	25.85	12.22	24.65	74.78	7.32
	+ query expansion	26.27	7.87	24.16	**6.18**	25.25	12.17	23.36	75.23	7.61
	+ IGRAG (ours)	**26.79**	**8.00**	**25.62**	6.17	**27.01**	**12.83**	**25.28**	**78.79**	**8.21**
	Δ_{Native}	+0.72	+0.81	+2.09	+0.75	+1.67	+0.90	+1.08	+2.75	+1.06
LightRAG	Native	23.06	5.01	21.15	3.67	20.15	4.05	19.36	**65.42**	2.96
	+ chunk reranker	23.80	5.10	21.90	3.91	20.29	4.23	20.11	62.73	3.04
	+ query expansion	23.59	5.61	21.88	4.22	20.61	4.79	19.74	56.48	3.36
	+ IGRAG (ours)	**24.18**	**5.95**	**23.23**	**4.43**	**21.78**	**4.97**	**21.25**	62.16	**3.45**
	Δ_{Native}	+1.12	+0.94	+2.08	+0.76	+1.63	+0.92	+1.89	-3.26	+0.49
RAPTOR	Native	25.48	6.45	22.83	5.01	24.80	11.39	23.20	82.41	8.35
	+ chunk reranker	26.26	6.70	23.55	5.23	25.18	11.33	22.88	81.53	8.14
	+ query expansion	25.88	6.95	23.13	5.54	24.48	11.80	22.59	80.21	8.02
	+ IGRAG (ours)	**26.96**	**7.15**	**24.78**	**5.76**	**25.67**	**12.60**	**23.40**	**86.60**	**8.90**
	Δ_{Native}	+1.48	+0.70	+1.95	+0.75	+0.87	+1.21	+0.20	+4.19	+0.55
Medical Dataset										
HippoRAG2	Native	20.86	3.38	19.47	1.47	19.59	3.11	18.40	90.73	1.13
	+ chunk reranker	21.26	3.49	19.31	1.55	19.40	3.14	19.06	92.36	1.14
	+ query expansion	21.05	3.59	19.53	1.64	19.58	3.19	18.47	91.35	1.13
	+ IGRAG (ours)	**22.50**	**3.93**	**21.46**	**1.68**	**20.64**	**3.32**	**19.57**	**96.07**	**1.26**
	Δ_{Native}	+1.64	+0.55	+1.99	+0.21	+1.05	+0.21	+1.17	+5.34	+0.13
LightRAG	Native	20.60	3.22	18.82	1.25	18.38	1.56	17.90	58.82	0.72
	+ chunk reranker	21.26	3.28	19.19	1.28	19.00	1.58	17.92	63.01	0.80
	+ query expansion	20.83	3.44	19.00	1.33	18.56	1.58	18.19	73.82	**0.95**
	+ IGRAG (ours)	**21.76**	**3.64**	**21.00**	**1.40**	**19.65**	**1.74**	**19.55**	**75.18**	0.94
	Δ_{Native}	+1.16	+0.42	+2.18	+0.15	+1.27	+0.18	+1.65	+16.36	+0.22
RAPTOR	Native	21.63	3.86	19.69	1.56	19.44	2.94	18.98	89.15	1.53
	+ chunk reranker	21.38	3.99	20.14	1.62	19.19	2.96	19.63	90.66	**1.55**
	+ query expansion	21.68	4.08	19.62	1.75	19.54	3.00	18.95	91.17	1.42
	+ IGRAG (ours)	**23.85**	**4.17**	**20.86**	**1.87**	**20.65**	**3.20**	**20.81**	**95.55**	1.51
	Δ_{Native}	+2.22	+0.31	+1.17	+0.31	+1.21	+0.26	+1.83	+6.40	-0.02

A weaker graph builder coupled with IGRAG approaches a stronger builder. Table 4 rebuilds the HippoRAG2 index from scratch under two extractor LLMs (GPT-4.1, GPT-5.2) and either uses the rebuilt graph as-is (*w/o Graph Refiner*) or pipes it through IGRAG's utility-gated refinement (*w/ Graph Refiner*; the refiner LLM is the same backbone as the builder, used inference-only). The diagonal comparison *GPT-4.1 builder* + IGRAG (24.46 ACC / 6.39 ROUGE-L) approaches *GPT-5.2 builder, no refiner* (24.79 / 6.30) and exceeds it on ROUGE-L by $+0.09$; the stronger-builder ceiling is in turn raised to 26.04 / 7.01 once the same backbone is also used as the refiner. Post-construction utility-aware optimization therefore recovers most of the gap created by a weaker construction backbone, suggesting that the GraphRAG quality bottleneck can be corrected post hoc rather than paid for at extraction time.

A small refiner inside the IGRAG pipeline matches a frontier refiner. The refiner-backbone ablation in Table 8 (App. B.4) replaces the tuned Qwen3-8B refiner of the main tables with zero-shot GPT-5.2 and GPT-4.1-mini, and additionally varies the gating policy on HippoRAG2/Novel. The gated GPT-4.1-mini configuration achieves the largest paired improvement of the four (Δ ACC $+0.72$, Δ ROUGE-L $+0.91$), exceeding gated GPT-5.2 ($+0.25$, $+0.71$). The improvement is therefore attributable to the pipeline (candidate generation, utility scoring, utility-gated commit) rather than to refiner scale: the host index already supplies the relational supervision the refiner conditions on, and additional world knowledge from a stronger backbone provides no further benefit once the pipeline is in place.

IGRAG transfers across KGQA hosts without re-training. Applied unchanged to GNN-

Table 2: Retrieval panel on GraphRAG-Bench (Recall = Evidence Recall; Relevance = Context Relevance), all values $\times 100$. Row structure and refiner match Table 1.

Method	Mode	Fact Retrieval		Complex Reasoning		Contextual Summarize		Creative Generation	
		Recall	Relevance	Recall	Relevance	Recall	Relevance	Recall	Relevance
Novel Dataset									
HippoRAG2	Native	17.40	14.37	19.65	13.40	25.45	18.30	14.20	11.94
	+ chunk reranker	17.57	14.76	20.20	13.29	25.76	18.21	14.72	12.28
	+ query expansion	17.61	14.13	19.62	13.20	25.77	18.51	14.48	11.94
	+ IGRAG (ours)	**18.45**	**15.22**	**20.75**	**13.86**	**28.01**	**19.67**	**15.49**	**12.58**
	Δ_{Native}	+1.05	+0.85	+1.10	+0.46	+2.56	+1.37	+1.29	+0.64
LightRAG	Native	13.15	12.46	14.19	11.89	17.65	13.23	13.84	**10.38**
	+ chunk reranker	12.81	12.43	13.85	12.32	18.04	13.39	13.91	9.91
	+ query expansion	12.81	12.38	13.86	12.09	17.27	12.64	14.15	9.33
	+ IGRAG (ours)	**14.16**	**13.33**	**14.45**	**12.74**	**18.12**	**14.00**	**15.57**	10.05
	Δ_{Native}	+1.01	+0.87	+0.26	+0.85	+0.47	+0.77	+1.73	-0.33
RAPTOR	Native	16.52	16.85	18.30	18.44	24.14	22.51	**18.72**	**18.08**
	+ chunk reranker	16.67	16.94	18.34	18.86	24.41	22.24	18.19	17.95
	+ query expansion	16.61	16.53	18.66	18.11	24.68	22.10	16.33	16.42
	+ IGRAG (ours)	**17.81**	**17.90**	**20.57**	**19.40**	**26.19**	**23.44**	17.96	17.59
	Δ_{Native}	+1.29	+1.05	+2.27	+0.96	+2.05	+0.93	-0.76	-0.49
Medical Dataset									
HippoRAG2	Native	6.72	6.51	5.13	3.30	4.83	5.19	6.29	**4.82**
	+ chunk reranker	6.59	6.58	5.05	3.31	4.77	5.27	6.36	4.78
	+ query expansion	6.28	6.47	4.97	3.40	4.87	5.19	6.65	4.22
	+ IGRAG (ours)	**7.03**	**6.62**	**5.19**	**3.61**	**5.06**	**5.41**	**7.14**	4.65
	Δ_{Native}	+0.31	+0.11	+0.06	+0.31	+0.23	+0.22	+0.85	-0.17
LightRAG	Native	6.48	6.35	4.15	2.12	4.10	3.89	**4.80**	2.56
	+ chunk reranker	6.54	6.49	4.22	2.19	4.01	3.88	4.67	2.71
	+ query expansion	6.54	6.31	4.05	2.41	3.99	3.81	4.29	3.01
	+ IGRAG (ours)	**7.05**	**6.72**	**4.40**	**2.44**	**4.20**	**4.04**	4.70	**3.08**
	Δ_{Native}	+0.57	+0.37	+0.25	+0.32	+0.10	+0.15	-0.10	+0.52
RAPTOR	Native	6.65	7.13	6.37	3.94	5.01	4.33	4.49	4.22
	+ chunk reranker	6.67	7.15	6.34	4.04	5.11	4.49	4.75	4.38
	+ query expansion	6.88	7.06	6.19	4.33	5.41	4.41	5.26	4.37
	+ IGRAG (ours)	**7.21**	**7.43**	**6.83**	**4.73**	**5.80**	**4.82**	**5.63**	**4.52**
	Δ_{Native}	+0.56	+0.30	+0.46	+0.79	+0.79	+0.49	+1.14	+0.30

Table 3: KGQA on WEBQSP (n=1,628) and CWQ (n=3,531) per Mavromatis and Karypis (2024); values $\times 100$. *Native* rows use the host system's published numbers; + IGRAG uses the tuned Qwen3-8B refiner between the host's retrieval and generator. Retriever-variant breakdown in Table 6.

Method	Mode	WebQSP			CWQ		
		Hit	H@1	F1	Hit	H@1	F1
GNN-RAG	Native	85.7	80.6	71.3	66.8	61.7	59.4
	+ IGRAG (ours)	88.3	82.4	71.9	67.8	61.4	61.6
	Δ_{Native}	+2.6	+1.8	+0.6	+1.0	-0.3	+2.2
GRAG	Native	76.2	72.5	50.4	57.1	50.8	45.2
	+ IGRAG (ours)	78.8	75.6	53.3	58.2	52.2	46.2
	Δ_{Native}	+2.6	+3.1	+2.9	+1.1	+1.4	+1.0
G-Retriever	Native	72.0	70.1	50.2	59.4	49.3	43.6
	+ IGRAG (ours)	73.7	74.4	53.0	59.7	49.8	44.6
	Δ_{Native}	+1.7	+4.3	+2.8	+0.3	+0.6	+1.1

RAG, GRAG, and G-RETRIEVER on the Freebase-anchored WEBQSP and CWQ benchmarks (Table 3), the Qwen3-8B refiner trained on document-grounded GraphRAG traces yields positive improvements on all three hosts. The gain is larger on the deeper-reasoning CWQ split (up-to-4-hop) than on WEBQSP (up-to-2-hop), consistent with the headroom analysis of Mavromatis and Karypis (2024): longer reasoning chains expose more retrieval errors that local subgraph refinement can correct.

Improvements scale with question-type difficulty. Within Table 1, the reasoning-heavy Novel split exhibits larger paired deltas than the Medical split, and Fact-Retrieval and Complex-Reasoning columns dominate the aggregate Δ ROUGE-L. The pattern is sharper at the record level: on the Novel split, large per-question changes ($|\Delta| \geq 0.10$ ROUGE-L) outnumber large losses by factors of $5\times$ to $18\times$, and HippoRAG2 / Complex-Reasoning has 12 : 0 big-wins to big-losses (Table 5, App. B.1). The aggregate-delta dilution reflects a large tie

mass on queries the host already solves; the relevant signal is concentrated in the decisive cases.

Graph-level refinement is a more natural inference-time enhancement than chunk- or query-side baselines. Across the Novel split of Table 1, + IGRAG *(ours)* dominates the inference-time baselines (+ *chunk reranker*, + *query expansion*; the path-reranker variant is reported in App. B.1) on nearly every cell. Chunk-level reranking discards the relational structure that motivated a graph index in the first place, query expansion only re-weights the retriever's similarity scores, and path-level reranking is host-specific because RAPTOR's hierarchical tree has no genuine relational paths to score. Refining the retrieved subgraph is the operation common to all three host families, which is why a single refiner generalises across them.

IGRAG is not equivalent to retrieving more context. The retrieval-side panel of Table 2 shows the post-IGRAG substrate tightening rather than expanding: Evidence Recall rises across nearly every cell, while Context Relevance can fall, indicating that the refiner removes spurious edges rather than padding the context with additional ones. The KGQA gate ablation reaches the same conclusion from the opposite direction: on the already-curated Freebase subgraphs, un-gated edits actually *hurt* the host (e.g., -6.1 Hit on WEBQSP; Table 7, App. B.2), whereas gated edits recover $+6.6$ to $+8.9$ Hit / H@1 / F1. The gain therefore comes from utility-aware refinement of the existing subgraph, not from feeding the generator a larger or denser context window.

5.2 Ablation Study

Utility supervision is load-bearing. The 2×2 refiner ablation in Table 8 (App. B.4) isolates the contribution of the utility-gated commit: gated variants deliver positive paired Δ on ACC and ROUGE-L regardless of refiner backbone, while removing the gate collapses the gain to baseline at either backbone. The single-round gate diagnostic of Table 11 (App. B.7) corroborates the same picture at trace level: gated runs accept 66.5%–77.2% of proposals with strictly positive mean utility margin ($+0.0084$ to $+0.0133$), while their un-gated counterparts accept 75.6%–91.5% but with mean margin essentially at or below zero (-0.0007 to $+0.0010$). The improvement is therefore traceable to utility-aware optimization, not to arbitrary graph perturbation.

The refiner is a graph *compactor*, and the gate is what makes "adding" safe. Mined from the GPT-5.2 refiner ablation traces (App. B.5), *remove* accounts for 64–89% of all proposed edits across the three hosts (Table 9), *add* only 11–20%, and *relabel* at most 15%, which contradicts the naïve missing-edge framing of prior introspective work and aligns IGRAG with a pruning role for over-eager LLM extraction. The composition flips when the refiner backbone is weaker: GPT-4.1-mini produces a mirror profile of 67% *add* / 27% *remove* (Figure 3), so an un-gated weak-backbone refiner would inject un-validated edges. The gate is exactly what filters those out on both regimes: on a small-backbone refiner the no-gate variant exhibits an ~ 11-point Faithfulness drop on Creative Generation which the gate restores; on KGQA the gate keeps 63%/85% of WEBQSP/CWQ proposals and converts what would have been a -6.1 Hit regression into a $+6.6$ Hit improvement (Table 7, App. B.2). The proposal-plus-gate template is therefore what lets the framework absorb either a *remove*-heavy or an *add*-heavy refiner without harming the host substrate.

6 Conclusion

We presented *Introspective GraphRAG* (IGRAG), a formulation that treats the retrieval graph of a GraphRAG system as an object of utility-aware refinement rather than a fixed extraction artifact. To instantiate this formulation, we introduced a plug-and-play refinement framework that combines a subgraph refiner for local structural edits, a persistent memory of validated high-utility subgraphs, and a utility-gated selection mechanism that accepts refinements only when they improve downstream utility. The same framework operates between the retriever and generator of arbitrary host GraphRAG systems without modifying either. Beyond the empirical gains, our results suggest that retrieval quality can be partially decoupled from expensive graph construction through lightweight utility-aware refinement, and that the same paradigm naturally extends to broader retrieval substrates and downstream utility signals. IGRAG points toward retrieval systems that continuously adapt through interaction feedback rather than remaining frozen after offline index construction. We hope this perspective encourages future work on introspective and self-improving retrieval infrastructures for long-horizon reasoning systems.

Limitations

IGRAG inherits the retriever and generator of its host system, so an error mode that originates in the host's tokeniser, embedding model, or generation prompt is not directly addressable by subgraph refinement. The persistent-evolution regime is currently driven by ROUGE-L utility against a gold reference, which is convenient on benchmarks but not always available in deployed settings; replacing the gate with a deployment-appropriate utility (preference model, validator LLM, or downstream task signal) is straightforward in principle but not evaluated here. Finally, our KGQA improvements assume the Freebase-anchored personalised-PageRank retrieval set; refining the graph further upstream (at the entity-linking stage or before PPR) falls outside the scope of this paper.

References

Boyu Chen, Zirui Guo, Zidan Yang, Yuluo Chen, Junze Chen, Zhenghao Liu, Chuan Shi, and Cheng Yang. 2026. Pathrag: Pruning graph-based retrieval augmented generation with relational paths. In *Proceedings of the AAAI conference on artificial intelligence*, volume 40, pages 30183–30191.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Zhangyin Feng, Xiaocheng Feng, Dezhi Zhao, Maojin Yang, and Bing Qin. 2024. Retrieval-generation synergy augmented large language models. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 11661–11665. IEEE.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*.

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

Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. 2025. Grag: Graph retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4145–4157.

Yuntong Hu, Sha Li, Naren Ramakrishnan, and Liang Zhao. 2026. Rag without forgetting: Continual query-infused key memory. *arXiv preprint arXiv:2602.05152*.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7036–7050.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pages 6769–6781.

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

Chaofan Li, Jianlyu Chen, Yingxia Shao, Chaozhuo Li, Quanqing Xu, Defu Lian, and Zheng Liu. 2025. Reinforced ir: A self-boosting framework for domain-adapted information retrieval. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22061–22073.

Linhao Luo, Zicheng Zhao, Reza Haffari, Dinh Phung, Chen Gong, and Shirui Pan. 2026. Gfm-rag: graph foundation model for retrieval augmented generation. *Advances in Neural Information Processing Systems*, 38:36371–36405.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315.

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

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations*, volume 2024, pages 32628–32649.

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

Zhishang Xiang, Chuanjie Wu, Qinggang Zhang, Shengyuan Chen, Zijin Hong, Xiao Huang, and Jinsong Su. 2025. When to use graphs in rag: A comprehensive analysis for graph retrieval-augmented generation. *arXiv preprint arXiv:2506.05690*.

S Yao, J Zhao, D Yu, N Du, I Shafran, K Narasimhan, and YR Cao. 2022. Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331.

A Experimental Setup

This appendix is organised into three sections: Appendix A details the datasets, host systems, refiner/GNN configuration, and hyperparameter space; Appendix B reports additional results (per-question wins/losses, KGQA retriever-variant breakdown, retrieval by question type, edit-type statistics, refiner cost, single-round gate diagnostic, case studies); Appendix C collects implementation details and the refiner prompt.

For KGQA benchmarks, we follow the retriever and generator used in their official implementations. For GraphRAG-Bench, we fix GPT-5.2 as the generator.

Document-grounded benchmark. We use the full GraphRAG-Bench (Xiang et al., 2025) dataset: 2,010 questions from the *Novel* split (20 corpora drawn from public-domain literature) and 2,062 questions from the *Medical* split (a single large medical-domain corpus). Questions are categorised into four difficulty levels: *Fact Retrieval*, *Complex Reasoning*, *Contextual Summarize*, and *Creative Generation*. In this setting the index graph is built by an LLM from the source corpus and is therefore noisy by construction, the exact setting introspective optimisation is designed for.

KGQA benchmarks. Following Mavromatis and Karypis (2024), we additionally evaluate on two widely used KGQA datasets that operate over the *Freebase* knowledge graph. WEBQSP (Yih et al., 2015) contains 4,737 natural-language questions answerable by up to 2-hop reasoning over Freebase. CWQ (Talmor and Berant, 2018) contains 34,699 complex questions requiring up to 4-hop reasoning over the same KG. For both datasets, each question is paired with a personalised-PageRank-induced subgraph anchored at its linked entities, which serves as the retrieved subgraph for IGRAG. In contrast to GraphRAG-Bench, the Freebase KG is curated rather than LLM-extracted, so any improvement here comes from *local* refinement (re-weighting / pruning of the retrieved subgraph) rather than from correcting upstream extraction errors.

Methods. We compare three representative document-grounded GraphRAG systems with very different index structures: HIPPORAG2 (Gutiérrez et al., 2024), which builds an entity–relation knowledge graph and retrieves via personalised PageRank; LIGHTRAG (Guo et al., 2024), which builds a dual entity–chunk graph and retrieves via a hybrid query workflow; and RAPTOR (Sarthi et al., 2024), which builds a hierarchical tree of cluster summaries and retrieves via top-down tree traversal. On the KGQA side we compare three GNN+LLM hybrids: GNN-RAG (Mavromatis and Karypis, 2024), which uses a GNN reasoner to score candidate KG paths and verbalises them as input to the LLM; GRAG (Hu et al., 2025), which retrieves k-hop ego-graphs via embedding similarity and feeds them through a soft graph prompt; and G-RETRIEVER (He et al., 2024), which formulates retrieval as a prize-collecting Steiner-tree problem over the question-conditioned subgraph.

GraphRAG-Bench refiner and GNN. Across the three document-grounded hosts (HippoRAG2, LightRAG, RAPTOR) we use a single shared graph encoder GE for the refiner's graph-token branch (Section 4.2): a 2-layer GraphSAGE (Hamilton et al., 2017) with hidden width 768 and mean-pooling readout, with node features initialised by the same text encoder the host uses (sentence-BERT for HippoRAG2 and LightRAG, the RAPTOR cluster embedder for RAPTOR). The refiner backbone is a Qwen3-8B-Instruct model fine-tuned on the outcome-labelled candidate pool of Section 4.4, and the same fine-tuned refiner is used for all three hosts so that the only difference across the three rows of Table 1 is the host's retrieval substrate. The refiner's prompt window is capped at 1024 tokens, consisting of the projected graph token, the question, the candidate node pair, and the elementary-edit instruction.

KGQA refiner and GNN. The three KGQA hosts (GNN-RAG, GRAG, G-Retriever) retain their native graph reasoners: REARREV for GNN-RAG, the embedding-based ego-graph scorer of Hu et al. (2025) for GRAG, and the prize-collecting Steiner-tree solver of He et al. (2024) for G-Retriever; we do not alter these. IGRAG is inserted between each host's retrieval and generation stages with the same Qwen3-8B-Instruct refiner used in the GraphRAG-Bench setup, and the same 2-layer GraphSAGE encoder is applied to the host's retrieved Freebase subgraph to produce the graph token; this keeps the refiner side identical across the two benchmark suites.

Hyperparameter search space. For the trained-refiner variant (Qwen3-8B-Instruct) we sweep, on the GraphRAG-Bench Novel validation split: learning rate $\in \{1\text{e-}5, 2\text{e-}5, 5\text{e-}5\}$, batch size $\in \{16, 32, 64\}$, candidate margin $\gamma \in \{0.1, 0.2, 0.5\}$, mutation-loss weight $\lambda \in \{0.0, 0.25, 0.5, 1.0\}$, candidate-pool size $m \in \{4, 8, 16\}$, memory top-$n \in \{3, 5, 10\}$, and commit margin $\epsilon \in \{0, 0.01, 0.05\}$. Training uses AdamW with a cosine schedule and 5% linear warm-up over 3 epochs of the outcome-labelled candidate pool. The same grid is used in the KGQA setting (re-tuned on the WebQSP validation split); the selected configuration is reported with the headline numbers in the main tables and the same configuration is used unchanged across all variants of the corresponding benchmark suite.

Table 5: Paired per-question ROUGE-L wins, losses, and large-change counts on the Novel split. Refiner: tuned Qwen3-8B (main-table setup).

Method	QType	Wins	Losses	Ties	Big-Wins	Big-Losses	Win:Loss
HippoRAG2	Fact Retrieval	50	7	914	22	1	7.1×
HippoRAG2	Complex Reasoning	56	3	551	12	0	18.7×
LightRAG	Fact Retrieval	39	5	927	23	4	7.8×
LightRAG	Complex Reasoning	32	6	572	13	1	5.3×
RAPTOR	Fact Retrieval	41	13	917	18	3	3.2×
RAPTOR	Complex Reasoning	41	8	561	9	0	5.1×

Refiner instantiation and metrics. The four main tables of Section 5 use the trained Qwen3-8B-Instruct refiner described above; the construction-backbone ablation (Table 4) instead uses the row's builder LLM as the inference-only refiner, and the refiner-backbone ablation (Table 8) instead uses zero-shot GPT-5.2 or GPT-4.1-mini. Several appendix analyses (App. B.5, App. B.6, App. B.7) are mined from the zero-shot GPT-5.2 / GPT-4.1-mini refiner ablation runs rather than from the Qwen3-8B main run, because the question those analyses address is the proposal style and cost of the LLM-based refiner. In every case we state the refiner choice in the table caption or accompanying prose. The utility gate uses ROUGE-L F1 against the gold answer with ϵ=0. For GraphRAG-Bench we adopt the metric panel of Xiang et al. (2025) unchanged: Answer Correctness (**ACC**), **ROUGE-L**, Coverage (**Cov**), and Faithfulness (**FS**) for generation; Evidence Recall (**Recall**) and Context Relevance (**Relevance**) for retrieval; LLM-as-judge metrics use GPT-4.1-mini as the eval LLM and `BAAI/bge-large-en-v1.5` for semantic similarity. For KGQA we follow Mavromatis and Karypis (2024): Hit, Hits@1, and F1 against the gold answer-entity set. The three inference-time enhancement baselines (+ *chunk reranker*, + *path reranker*, + *query expansion*) are detailed in App. C.

B Additional Results and Analyses

B.1 Per-Question Wins/Losses

The aggregate Δ ROUGE-L of Table 1 is modest because most paired records are ties; the relevant signal is concentrated in a small set of queries where refinement either fixes a clear retrieval mistake or, much more rarely, introduces a new one. Large changes ($|\Delta| \geq 0.10$) come out 5×–18× in favour of IGRAG on Fact Retrieval and Complex Reasoning (Figure 1). The wins are concentrated where the host index contains genuine relational content; RAPTOR's lower win:loss ratio is con-

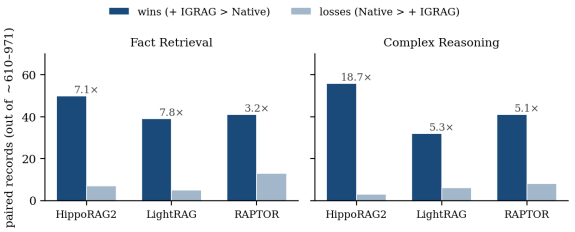

Figure 1: Per-question paired wins and losses on the Novel split. Ratios above each pair are the win-to-loss ratio. Decisive paired wins outnumber decisive paired losses by 5× to 18× on the reasoning-heavy question types.

Table 6: Retriever-variant breakdown for the GNN-RAG row of Table 3. *Native* uses GPT-5.2 over the released reasoning paths. + IGRAG applies the inference-time analogue of Eq. (20); refiner here is zero-shot GPT-5.2, not Qwen3-8B.

Variant	Mode	WebQSP Hit	WebQSP H@1	WebQSP F1	CWQ Hit	CWQ H@1	CWQ F1
GNN-RAG (LMSR)	Native	82.74	79.73	71.53	63.61	60.07	56.42
	+ IGRAG	**89.31**	**85.63**	**77.85**	**72.53**	**68.00**	**64.62**
	Δ	+6.57	+5.90	+6.32	+8.92	+7.93	+8.20
GNN-RAG (SBERT)	Native	84.64	80.71	72.66	63.15	59.70	56.37
	+ IGRAG	**89.86**	**85.38**	**78.16**	**72.22**	**66.95**	**64.38**
	Δ	+5.22	+4.67	+5.50	+9.06	+7.25	+8.01
GNN-RAG-RA (LMSR)	Native	89.07	82.37	76.71	65.25	59.98	57.79
	+ IGRAG	**92.38**	**86.79**	**81.45**	**74.62**	**68.03**	**65.97**
	Δ	+3.32	+4.42	+4.74	+9.37	+8.04	+8.18
GNN-RAG-RA (SBERT)	Native	89.31	82.25	76.51	65.28	60.07	57.70
	+ IGRAG	**92.75**	**87.29**	**81.32**	**73.92**	**67.46**	**65.54**
	Δ	+3.44	+5.04	+4.81	+8.64	+7.39	+7.84

sistent with its hierarchical-tree index having no relations to refine.

B.2 KGQA Retriever-Variant Breakdown

The same pattern holds across all four retriever variants. On WEBQSP the gated Δ Hit ranges from +3.32 to +6.57, lower for the already-strong -RA variants and higher for the weaker LMSR variant, consistent with diminishing returns from a stronger native baseline. On CWQ the gated Δ Hit is essentially constant at +8.64 to +9.37, indicating that the deeper reasoning chains of CWQ expose a fixed pool of retrieval errors that the refiner can correct regardless of which GNN scoring (LMSR or SBERT) produced the original ranked paths. The Native ordering of Mavromatis and Karypis (2024) (SBERT $\gtrsim$ LMSR; -RA $>$ vanilla) is preserved after IGRAG augmentation, and the gate acceptance rate is remarkably stable per dataset (62.5 %–64.5 % on WEBQSP, 81.5 %–85.0 % on CWQ), suggesting the refiner's proposal distribution is largely invariant to the retriever back-end.

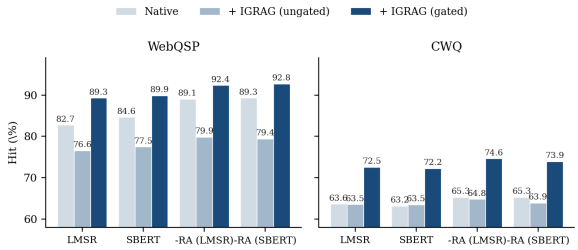

Figure 2: KGQA gate ablation: Native (GPT-5.2 over the released GNN-RAG paths) vs. ungated and gated IGRAG (refiner: zero-shot GPT-5.2, not Qwen3-8B).

Table 7: KGQA gate ablation on the released GNN-RAG (LMSR) subgraphs (n=1,628/3,531). Refiner: zero-shot GPT-5.2 (not Qwen3-8B). Accept = fraction of queries where the gate retained the refined answer.

Mode	WebQSP Hit	WebQSP H@1	WebQSP F1	WebQSP Accept	CWQ Hit	CWQ H@1	CWQ F1	CWQ Accept
Native (GPT-5.2)	82.74	79.73	71.53	–	63.61	60.07	56.42	–
+ IGRAG (ungated)	76.60	72.17	59.47	100 %	63.52	59.25	55.34	100 %
+ IGRAG (gated)	**89.31**	**85.63**	**77.85**	63.0 %	**72.53**	**67.99**	**64.62**	85.0 %
$\Delta_{\text{gated vs Native}}$	+6.57	+5.89	+6.32		+8.92	+7.92	+8.20	

The utility gate is load-bearing on KGQA, too. The KGQA contrast reproduces the H2 finding cleanly. We replace the host's generator with GPT-5.2 (so the Native row is recomputed on the same released subgraphs, not the published Llama-2 baseline) and report three modes: *Native*, *ungated* (apply every proposed edit), and *gated* (accept an edit only if it does not lower F1 against gold), in Table 7. The gated row exceeds Native by +6.6 to +8.9 points across both datasets, while the ungated row matches or slightly underperforms Native on CWQ and visibly underperforms on WEBQSP (−6.1 Hit, −12.1 F1). The asymmetry is intuitive: the host's released subgraphs are already curated by a personalised-PageRank retriever, so an ungated edit operator with a graph-compactor prior (Obs. A.1) removes evidence rather than adding it; the gate filters those edits out (acceptance rate 63.0% on WEBQSP and 85.0% on CWQ). The KGQA reading therefore agrees with the document-grounded reading from H2 (Table 8): the IGRAG pipeline (proposal → scoring → commit gate) is the load-bearing component, not the refiner alone.

B.3 Retrieval Performance Across Question Types

The per-question-type retrieval panel of Table 1's companion is given in Table 2. The refiner predominantly removes edges that the host retriever had used to broaden its neighbourhood (Obs. A.1, App. B.5); this both tightens the retrieved subgraph

Table 8: 2×2 refiner-backbone ablation on HippoRAG2/Novel (n=2,010). Refiner here is zero-shot GPT-5.2 or GPT-4.1-mini (not the Qwen3-8B used in the main tables); "Gated" applies Eq. (20), "No gate" commits every proposed edit. Δ is paired against each row's own without-IGRAG baseline.

Refiner backbone	Gate	Absolute (with IGRAG)				Δ vs. same-run baseline			
		ACC	ROUGE-L	Recall	FS	ACC	ROUGE-L	Recall	FS
GPT-5.2 (frontier; snapshot)	✓	25.35	7.22	19.58	75.23	+0.25	+0.71	+0.16	−0.81
GPT-5.2 (frontier)	✗	25.12	6.48	19.93	76.02	−0.24	−0.04	−0.10	+1.53
GPT-4.1-mini (small)	✓	**26.06**	**7.80**	19.82	75.70	**+0.72**	**+0.91**	−0.10	−2.94
GPT-4.1-mini (small)	✗	25.26	6.54	20.56	64.43	+0.03	+0.17	+0.12	**−2.78**

(often lowering Relevance on the noisier broader set) and improves the chance that the remaining edges connect to the actual gold evidence (raising Recall against the curated reference). For downstream generation the Recall gain dominates: ACC and ROUGE-L improve despite the occasional Relevance dip.

B.4 H1 / H2: Graph-Construction Backbone and Refiner Specialization

H1 expected pattern. If IGRAG's benefit is genuinely utility-aware optimization (rather than implicit graph reconstruction by virtue of the refiner being a strong LLM), then *Qwen3-8B*+IGRAG is expected to recover much of the gap between Qwen3-8B and GPT-5.2 as construction backbones, and potentially to match or exceed the GPT-5.2 baseline on the Novel split.

H2 measured pattern. Within the 2×2 ablation: *(i)* gated GPT-4.1-mini outperforms gated GPT-5.2 on the paired Δ (the small backbone delivers a larger paired ACC and ROUGE-L lift than the frontier backbone inside the same pipeline); *(ii)* removing the gate collapses the gain to baseline at either backbone; *(iii)* gated GPT-4.1-mini exceeds no-gate GPT-5.2 on both ACC and ROUGE-L, so a smaller backbone with the pipeline beats a frontier backbone without it. The Faithfulness column tells the complementary story: the no-gate variant on the small backbone exhibits an $\sim$11 point FS drop, exactly the failure mode predicted by Obs. A.1b (add-heavy graph-completer style), and the gate filters it back to the safe band.

B.5 Proposed-Edit Type Breakdown

Table 9 reports the elementary-edit composition of the refiner's proposals, mined directly from the 4,303+4,242+4,462 recorded `refiner.propose` traces of the zero-shot GPT-5.2 refiner ablation run (one trace per query per host, full GraphRAG-Bench scale; this is the refiner-backbone analysis of Table 8, not the Qwen3-8B main run). For each host we report total edits, edits per non-empty proposal, the empty-proposal share, and the operation split over {ADD, REMOVE, RELABEL}.

Table 9: Proposed-edit composition mined from the GPT-5.2 refiner ablation traces. Shares sum to 100% per row over non-empty proposals.

Method	Refiner statistics			Operation share (%)		
	Edits (total)	Edits/call	Empty share	Add	Remove	Relabel
HippoRAG2	15,589	3.62	7.8%	20.1	64.6	15.2
LightRAG	18,250	4.30	8.9%	18.4	75.1	6.5
RAPTOR	16,281	3.65	25.6%	10.6	88.6	0.8

Obs. A.1. With a frontier refiner backbone, the refiner is overwhelmingly a graph *compactor*, not a completer. Across all three base methods running the standard GPT-5.2 refiner, *remove* accounts for 64–89% of all proposed edits, while *add* only 11–20% and *relabel* at most 15%. This contradicts the naïve "missing-edge" framing that motivated much of prior introspective work and aligns the empirical behaviour of IGRAG with a different, arguably more useful design role: *pruning over-eager LLM extraction*. The pattern is consistent with our retrieval observation (Obs. 4) that IGRAG sometimes lowers Context Relevance, not because it adds irrelevant edges, but because it removes edges the host retriever found at all.

Obs. A.1b. The proposal style flips when the refiner backbone is weaker, and the gate becomes load-bearing. The H2 ablation (App. B.4) lets us observe the refiner's edit composition for a second backbone. On HippoRAG2/Novel the GPT-4.1-mini refiner produces a *mirror image* of the GPT-5.2 profile: **67.2%** *add*, 26.9% *remove*, 5.9% *relabel*, mined from the H2-A `refiner.propose` traces. The small model behaves as a graph *completer*, eagerly proposing new edges drawn from background world knowledge, whereas the frontier model behaves as a graph *compactor* (Figure 3). This makes the utility-gated commit (Eq. 20) load-bearing in a precise and quantitative way: at $\geq$ 67% adds, an ungated weak-backbone refiner risks diluting the host retriever's context, while a gate-equipped weak-backbone refiner only commits the subset of those additions that actually raise downstream utility. The H2 grid of Table 8 confirms the prediction: gated GPT-4.1-mini delivers the largest paired Δ on ACC and ROUGE-L, while removing the gate collapses the gain to baseline at either backbone and produces an $\sim$ 11-point FS drop on the small backbone.

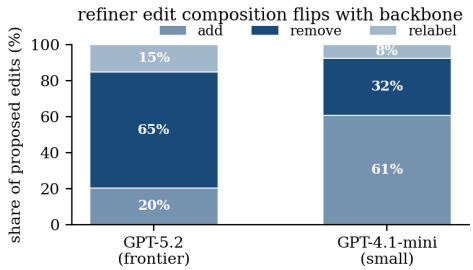

Figure 3: Refiner edit-composition flips with backbone capability (GPT-5.2 vs. GPT-4.1-mini, both zero-shot; not the Qwen3-8B main-table refiner). The frontier model is a graph *compactor*; the small model is a *completer*.

Table 10: Per-call refiner cost on the GPT-5.2 ablation run (not the Qwen3-8B main run). "Latency" is seconds per call, "In"/"Out" mean input/output tokens, "Tok/edit" total tokens divided by total edits.

Method	Calls	Latency (s)	In	Out	Tok/edit
HippoRAG2	4,303	2.56	2,839	182	833
LightRAG	4,242	3.07	2,976	239	747
RAPTOR	4,462	2.10	1,343	176	416

Obs. A.2. Refiner "activity level" tracks index expressivity, and that tracks downstream gain. RAPTOR's hierarchical tree exposes no genuine relation labels, so the refiner essentially cannot *relabel* (0.8% of proposals, vs. 6.5–15.2% on the two graph-based methods) and declines to act on 25.6% of queries (vs. 7.8–8.9%). This matches the cross-method pattern of Table 1, where RAPTOR's Δ rows are smaller and more mixed than HippoRAG2's: when the underlying index does not contain a meaningful relational structure to refine, the refiner correctly does less, and the headroom for utility-aware optimization is correspondingly smaller.

B.6 Refiner Cost and Token Footprint

Table 10 reports the per-call cost of the zero-shot GPT-5.2 refiner used in the ablation runs (these costs do not apply to the Qwen3-8B main configuration, which is run locally on the 8×A100 cluster): mean wall-clock latency, mean prompt-token input, mean completion-token output, and token cost per accepted edit.

Obs. A.3. Refiner cost is dominated by prompt context, not generation. The output budget the refiner spends on a query is small (∼180–240 tokens, i.e. ∼4 JSON edits) compared to the input it receives (1,300–3,000 tokens for the serialised augmented subgraph and instruction). This is encouraging because it means the dominant cost scales with subgraph size, which is already capped by each host system, rather than with the number of candidate edits decoded. Tok/edit ranges from 416 (RAPTOR) to 833 (HippoRAG2), so on a hosted GPT-class endpoint a single accepted IGRAG edit costs at most a fraction of a US cent at current rates.

Table 11: Single-round gate diagnostic on HippoRAG2/Novel (n=2,010). All four rows use zero-shot GPT-5.2 or GPT-4.1-mini as the refiner (not Qwen3-8B). Margin is the mean per-query utility delta on accepted edits.

Configuration	Round	Accept.	Edits	Margin
GPT-5.2 refiner, gated (ϵ=0)	$t=1$	77.2 %	5,718	+0.0084
GPT-5.2 refiner, no-gate	$t=1$	91.5 %	6,975	−0.0007
GPT-4.1-mini refiner, gated (ϵ=0)	$t=1$	66.5 %	5,651	+0.0133
GPT-4.1-mini refiner, no-gate	$t=1$	75.6 %	6,367	+0.0010

B.7 Stability and Drift Analysis

The acceptance criterion (Eq. (20)) is the mechanism that prevents the persistent regime from drifting away from utility. Two single-round measurements (Table 11) establish the diagnostic on HippoRAG2/Novel: the gated runs accept 66.5% to 77.2% of refiner proposals with a strictly positive mean utility margin (+0.0084 to +0.0133), whereas the un-gated equivalents accept 75.6% to 91.5% of proposals but with mean margin essentially at or below zero (−0.0007 to +0.0010). The no-gate variant therefore commits roughly 1,000 more edits to G_t that do not improve downstream utility, which is precisely the drift the gate is designed to prevent.

B.8 Case Studies

The three queries below are drawn verbatim from the HippoRAG2/Novel snapshot run and illustrate the three operating modes of the framework: gate-accepted improvement, gate-accepted paraphrase, and the absence of large gated losses.

Case 1: recovery from a refusal (HippoRAG2/Novel-40700, Fact Retrieval). *Question:* "What is the residence of The Dean in the play, serving as the main setting for much of the action?" *Gold:* "The Dean's residence is The Deanery." The native retrieval surfaces text about the play but no edge connecting `The Dean` to a residence, and the unmodified generator returns "I don't know." The refiner proposes five edits and the gate accepts; the refined subgraph yields "The

Dean's residence is the Deanery." Round-level utility gain: Δ ROUGE-L F1 $= +1.000$, the largest paired gain in the snapshot.

Case 2: paraphrase improvement (HippoRAG2/Novel-44557, Fact Retrieval). *Question:* "In which cities is the publisher of the CHILD'S HEALTH PRIMER, A. S. Barnes & Co., located?" *Gold:* "The publisher A. S. Barnes & Company is located in New York and Chicago." The native generator returns the bare "New York and Chicago." Refinement re-anchors the answer to the entity name: "A. S. Barnes & Company is located in New York and Chicago." Token-level ROUGE-L F1 gain $+0.446$. The qualitative pattern (refinement reintroduces the missing predicate phrase that the gold reference uses) is consistent with Obs. A.1, where remove-dominant proposals tighten the substrate the generator conditions on.

Case 3: zero gated losses survive the gate. Across the full HippoRAG2/Novel snapshot, the gate accepts 77.2% of refiner proposals and rejects the remaining 22.8% (Table 11). Among the gated runs, no record exhibits a downstream ROUGE-L F1 drop of ≥ 0.10, consistent with the small "Big-Losses" column of Table 5 (entries of 0 to 4 out of $\sim$600–970 paired records). The case-study interpretation is that the gate is doing what the H2 ablation (Table 8) says it does: rejecting edits that, regardless of how confident the refiner is, do not survive a downstream-utility check.

C Implementation Details and Prompts

Refiner instantiation. All main-body experiments use the same zero-shot refiner: an Azure-hosted GPT-class model (GPT-5.2 series) called with temperature 0 and a fixed prompt schema covering (a) the augmented subgraph $\hat{S}_q$ serialised as a list of edges with relation labels, (b) the query q, and (c) one of the elementary edit operations. The same prompt is used across all three base methods; a thin adapter converts each method's retrieved structure to and from our common subgraph schema.

Hyperparameters. Unless stated otherwise we use $m{=}8$ candidates per query, top-$n{=}5$ memory entries, ego-graph hops $k \in \{1, 2, 3\}$ for memory initialization, and commit margin $\epsilon{=}0$. The retriever budget K matches each base method's published default.

Generation-augmented retrieval baselines. The three baseline rows in Tables 1 and 2 are instantiated as follows. + *chunk reranker*: after the host's native retrieval we re-score every retrieved chunk against the query with the `BAAI/bge-reranker-v2-m3` cross-encoder and keep the top-K ranked chunks before passing them to the host generator; the budget K matches the host's published default so this is a pure-quality (not budget) intervention. + *path reranker*: the same BGE-M3 reranker is applied to each retrieved reasoning path verbalised as a single string, again preserving K; RAPTOR has no explicit relational paths so the row is omitted for it. + *query expansion*: we use HyDE (Gao et al., 2023) as the instantiation, prompting the same generator LLM at temperature 0.7 to produce a hypothetical answer to the query, embedding the hypothetical answer with `BAAI/bge-large-en-v1.5`, and using that embedding (instead of the raw query embedding) to drive the host's native retriever; all other pipeline stages are unchanged. None of the three baselines modify the underlying graph G, so they isolate the contribution of IGRAG's graph-evolving regime.

KGQA setup. For the WebQSP and CWQ experiments we follow the protocol of Mavromatis and Karypis (2024): each query is paired with a personalised-PageRank-induced subgraph anchored at its linked entities, and the three GNN+LLM systems (GNN-RAG, GRAG, G-Retriever) consume this subgraph through their own native retrieval modules. IGRAG is inserted between the host system's retrieval and generation stages without modifying either. Following the conventions of He et al. (2024) and Hu et al. (2025), evaluation uses Hit, Hits@1, and F1 against the gold answer-entity set, and the underlying LLM is the same one used by each host system to ensure a fair Δ.

Hardware. Host-system retrieval and evaluation code runs on a single workstation (24-core CPU, ample RAM); the GPT-class refiner and evaluation LLMs are called through a hosted Azure OpenAI API and consume no local GPU memory. For training the small-model variant of the refiner (when used in place of the hosted backbone), we use a cluster node with $8\times$ NVIDIA A100 (40 GB) GPUs, sufficient to fit a Qwen3-8B-Instruct refiner with mixed-precision optimization and the per-question candidate-pool data parallelism de-

scribed in Section 4.4.

```
system:
You are a GraphRAG index refiner. Given a small subgraph retrieved for a user query, your job is to propose a *minimal* set of edits that, if applied, would make the subgraph more useful for answering the query.

Allowed edits:
- add (source, target, relation): add a missing edge that is *clearly* supported by the node texts in the subgraph.
- remove (source, target, relation): remove an edge that is spurious or contradicted by the node texts.
- relabel (source, target, relation, new_relation): keep an edge but fix its relation type.

Be conservative. If the subgraph is already good, return an empty edits list. Never invent new node ids. Only reference nodes that appear in the input.

Output strict JSON:
{"edits": [
{"op": "add" | "remove" | "relabel",
"source": "<existing-node-id>",
"target": "<existing-node-id>",
"relation": "<relation-string>",
"new_relation": "<only for relabel>",
"reason": "<one short sentence>"}
]}

user:
Subgraph to refine:
{<serialised augmented subgraph Ŝ_q as JSON>}

Propose at most m edits.
```

Figure 4: Verbatim refiner prompt. Used by both the Qwen3-8B main-table refiner and the zero-shot GPT-5.2/GPT-4.1-mini ablation refiners.

Refiner prompt. The refiner is invoked with a two-turn chat: a system message that specifies the role, the allowed elementary edit operations, the conservative-action convention, and a strict JSON output schema; and a user message that carries the serialised augmented subgraph and a hard cap on the number of proposed edits. This prompt is held fixed across all three GraphRAG-Bench hosts and across the WebQSP/CWQ KGQA pipelines; the only host-specific adaptation is the subgraph serialiser that emits the input JSON object. The verbatim prompt is reproduced in Figure 4 as released with the code.