# GraFine: Retrieval-Time Refinement for Efficient Graph RAG over Corpus Graphs

Seonho An*
KAIST
Daejeon, Republic of Korea
asho1@kaist.ac.kr

Chaejeong Hyun*
KAIST
Daejeon, Republic of Korea
hchaejeong@kaist.ac.kr

Min-Soo Kim†
KAIST
Daejeon, Republic of Korea
minsoo.k@kaist.ac.kr



## Abstract

Graph RAG on corpus graphs enhances retrieval by leveraging intermediate node content as contextual clues to uncover unretrieved oracle nodes. However, existing methods suffer from two critical blind spots, namely semantically blind graph expansion and topology blind pruning, or else rely on prohibitively slow retrieval and generation interleaving. To address this, we formalize these limitations through an operational taxonomy and propose a retriever design that couples semantics-*aware* adding with graph-*aware* pruning under efficiency constraints. We instantiate this design as **GraFine**, which alternates between two refinement stages: Semantic Proximity eXpansion (SPX) for semantics-aware node addition, and a Graph Smoothing Reranker (GSR) for graph-aware pruning. Experiments on reference networks and text-rich knowledge graphs show that GraFine improves retrieval accuracy and generation quality while maintaining time-efficiency. Furthermore, we introduce Topological Recall (TR), a metric that quantifies the topological proximity between retrieved and oracle nodes. Our analysis using TR confirms that GraFine more effectively steers refinement toward undiscovered oracle nodes by leveraging intermediate contextual clues. Our code is available at this link: **github.com/asmath472/GraFine**.

## CCS Concepts

• **Information systems** → **Retrieval models and ranking**; *Language models*; *Rank aggregation*; • **Computing methodologies** → *Knowledge representation and reasoning*.

## Keywords

Retrieval-Augmented Generation, Graph Retrieval, Reranking

**ACM Reference Format:**
Seonho An, Chaejeong Hyun, and Min-Soo Kim. 2026. GraFine: Retrieval-Time Refinement for Efficient Graph RAG over Corpus Graphs. In *Proceedings of the 35th ACM International Conference on Information and Knowledge Management (CIKM'26).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

*Co-first author.
†Corresponding author.


*CIKM'26, Rome, Italy*

ACM ISBN 978-1-4503-XXXX-X/2026/06
https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

Retrieval-Augmented Generation (RAG) has emerged as a widespread solution to mitigate the inherent limitations of Large Language Models (LLMs), such as hallucinations and outdated parametric knowledge [11]. However, RAG methods that rely primarily on vector search (which we refer to as Vector RAG) often struggle to capture structural dependencies and non-textual relations, such as reference links between documents, because retrieval is largely driven by vector similarity alone [8, 12, 14, 38]. To address this, recent studies have proposed **Graph RAG** methods [5, 12, 14, 18, 21, 26, 33, 38], which incorporate a *graph* structure into the retrieval process to capture relationships via edges [15].

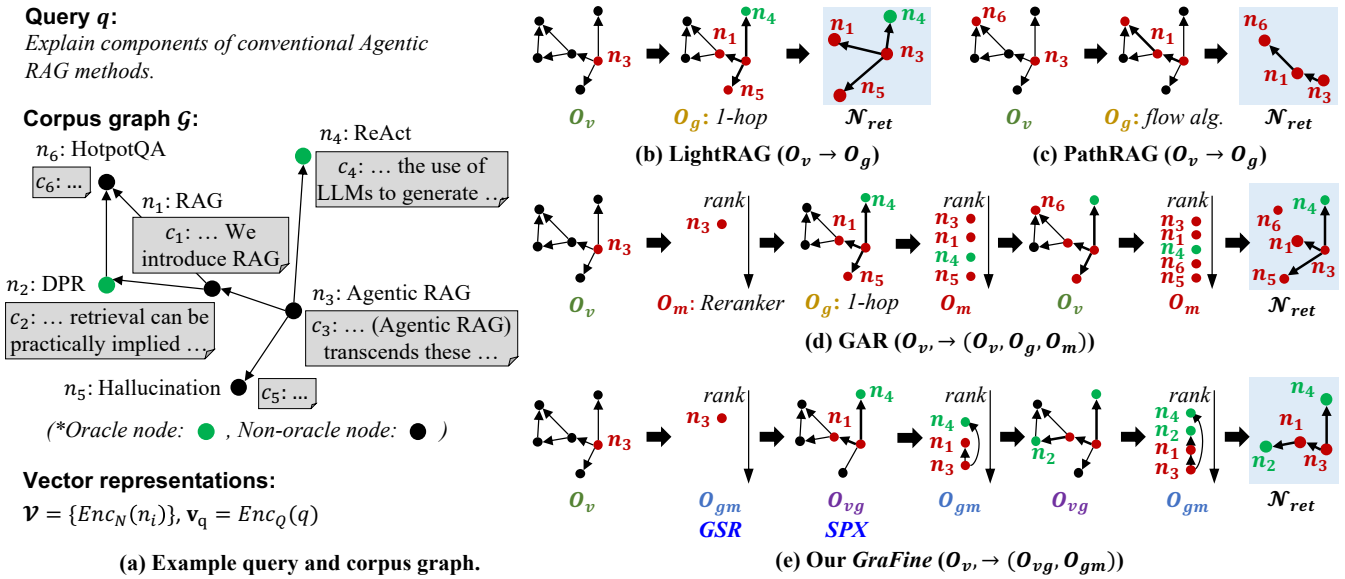
Recently, increasing attention has been paid to *corpus graphs*—such as reference networks [1]—in which each node contains rich textual information [5, 13, 27, 30]. Unlike conventional knowledge graphs (KGs), nodes in corpus graphs generally encapsulate rich contextual clues that can guide the retrieval process. Accordingly, retrievers on corpus graphs can use such clues at *retrieval time* to refine the retrieved set to move toward relevant yet initially unretrieved regions of the graph, thereby improving Graph RAG performance. For example, in Figure 1, given a query about *Agentic RAG components*, once the retriever has retrieved $n_3$ *(Agentic RAG)*, it can use the corresponding content $c_3$ as a clue to infer that $n_4$ *(ReAct)* is the next node to retrieve. This raises the following research question:

*How can we efficiently refine the intermediate retrieved set to reach previously unexplored but relevant regions on a corpus graph?*

Fundamentally, retrieval refinement can be described as an iterative process consisting of two steps: adding new elements to the current set of retrieved nodes (the *adding* step) and pruning existing ones from it (the *pruning* step).

Conventional approaches to address this fall into two broad categories. One line of work performs these two steps through retrieval-generation interleaving, including Agentic RAG methods that rely on repeated LLM reasoning [23, 26, 33, 35]. However, such methods incur high latency, often reaching up to tens of seconds [7, 32]. In interactive settings, such delays can substantially reduce usability [4, 22] and hinder practical deployment in enterprise environments.

A second line of work instead combines relatively fast model-based *pruning* (e.g., rerankers [27]) with graph-search-based *adding* methods, such as one-hop expansion [12], personalized PageRank [21], or PCST-based construction [16]. While these methods are effective on many datasets, they fundamentally have two inherent *informational* blind points: (1) **model-based pruning is topologically blind**, and (2) **graph search is semantically blind** in the current retriever design. We later formalize this observation




(a) Example query and corpus graph.

**Figure 1: Conceptual comparison of graph retrieval workflows based on retrieval operations. (a) illustrates the inputs for graph retrieval: $q$, $\mathcal{G}$, $\mathbf{v}_q$ and $\mathcal{V}$. (b)–(d) depict representative graph retrieval methods, while (e) presents our *GraFine* method.**

through *an operational taxonomy* of graph retrieval methods, using it to clarify these blind points. Together, these observations suggest two requirements for *retrieval-time refinement* on corpus graphs: **topology-aware pruning** and **semantics-aware adding**, while remaining **time-efficient**.

To address this, we **expand the design of retrievers** and propose **GraFine**, a fast and effective realization of the proposed design. After a seed retrieval step, GraFine alternates two lightweight refinement stages. The first is **Semantic Proximity eXpansion (SPX)**, which uses semantic signals to guide which nodes are added during expansion rather than relying on topology alone. The second is the **Graph Smoothing Reranker (GSR)**, which incorporates graph context during pruning and reranking rather than judging nodes from text alone. By interleaving these two stages, GraFine improves *retrieval-time refinement* without incurring the latency of retrieval-generation interleaving.

To evaluate the effectiveness of GraFine, we conduct both retrieval and generation experiments across two types of corpus graphs: **reference networks** [1, 31, 37] and **text-rich knowledge graphs** [5, 6, 8, 12]. Across these settings, GraFine improves retrieval accuracy and downstream generation quality while remaining time-efficient, achieving an average gain of 9.9% in Recall@10, generation win rates above 55%, and a consistently better effectiveness–latency trade-off.

To further examine whether retrieval refinement moves the retrieved set closer to relevant but initially missed targets, we introduce a new retrieval metric, **Topological Recall**. Unlike standard Recall, which only measures whether oracle nodes are retrieved, Topological Recall additionally quantifies the *topological proximity* between retrieved nodes and oracle nodes. Using this metric, we find that GraFine progressively moves intermediate retrieval results closer to oracle nodes, and that this progress is accompanied by improved recall. Our main contributions are summarized as follows:

- We formulate fast retrieval-time refinement on corpus graphs as **semantics-aware adding** and **topology-aware pruning** under efficiency constraints, and unify existing graph retrieval methods through an operational taxonomy.
- We propose **GraFine**, a lightweight retriever that alternates **SPX** for semantics-guided expansion and **GSR** for graph-aware pruning.
- We show on reference networks and text-rich knowledge graphs that GraFine improves retrieval and generation efficiently, and introduce **Topological Recall** to measure progress toward unretrieved oracle nodes.

## 2 Background and Problem Formulation

### 2.1 Graph RAG on Corpus Graphs

In this paper, we focus on *corpus graphs*, defined as follows:

**Definition 1** (Corpus Graph). A graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is classified as a *corpus graph* if and only if every node $n \in \mathcal{N}$ is associated with a pair $(k_n, c_n)$: $k_n$ serves as a node identifier (key), and $c_n$ provides descriptive textual information (textual content) about the node.

Graph RAG methods for *corpus graphs* take a query $q$ and a corpus graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ as inputs, and generate an answer $a$ through the following two steps:

(1) **Graph Retrieval Step:** Given $q$ and $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, the objective is to retrieve a set of nodes $\mathcal{N}_{\text{ret}} \subseteq \mathcal{N}$ that are *relevant* to $q$.

(2) **Generation Step:** An answer $a$ is generated based on $q$ and the retrieved node contents $\mathcal{C}_{\text{ret}} = \{c_i \mid (k_i, c_i) \in \mathcal{N}_{\text{ret}}\}$ using a generative language model $P_\theta$. Conventionally, $P_\theta$ processes $q$ and $\mathcal{C}_{\text{ret}}$ via in-context learning.

In this study, we assume corpus graphs that may contain edge directions and cycles; and do not explicitly leverage edge-level textual information. Other graph types (e.g., graphs that contains textual descriptions on edges) are outside the scope of this work.

## 2.2 Retrieval-Time Refinement

In corpus graphs, intermediate nodes encapsulate rich textual clues vital for moving toward structurally distant oracle nodes. We formalize this process of leveraging intermediate content to iteratively refine the retrieved node set as follows:

**Definition 2** (Retrieval-time refinement)**.** Given a query $q$, a corpus graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, a set of initial seed nodes $\mathcal{N}_{\text{seed}} = \mathcal{N}^{(0)} \subseteq \mathcal{N}$, and a set of oracle nodes $\mathcal{N}_q^\star \subseteq \mathcal{N}$ for $q$, *retrieval-time refinement* is the iterative process of transforming the current node set $\mathcal{N}^{(t)}$ into a refined node set $\mathcal{N}_{\text{ret}} = \mathcal{N}^{(T)}$ through a sequence of query-conditioned update steps:

$$\mathcal{N}^{(t+1)} = \phi_t(q, \mathcal{N}^{(t)}, \mathcal{G}), \qquad t = 0, \ldots, T-1.$$

Specifically, the update function $\phi_t$ performs two fundamental operations to approach the oracle nodes: (1) the *adding* step, which incorporates new nodes $n^+ \in \mathcal{N}^{(t+1)} \setminus \mathcal{N}^{(t)}$, and (2) the *pruning* step, which filters out existing nodes $n^- \in \mathcal{N}^{(t)} \setminus \mathcal{N}^{(t+1)}$ from the current node set.

The objective is to transform $\mathcal{N}_{\text{seed}}$ into a final set of nodes $\mathcal{N}_{\text{ret}}$ that maximizes the inclusion of previously missed oracle nodes in $\mathcal{N}_q^\star$. For example, to successfully retrieve the oracle node $n_4$ for query $q$ in Figure 1(a) about *components* of Agentic RAG, the retrieval must look beyond the isolated target content $c_4$ or its vector $\mathcal{V}_4$ and instead leverage the vital intermediate content $c_3$ of node $n_3$, which indicates that *Agentic RAG employs multi-step prompting strategies such as ReAct.*

# 3 Rethinking Graph Retrieval: An Operational Taxonomy

## 3.1 Fundamental Retrieval Operators

In a corpus graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, a *retrieval operator* $O$ is defined as a composite function $\mathcal{P} \circ \mathcal{R}$, where a **ranking function** $\mathcal{R}$ scores nodes and a **pruning function** $\mathcal{P}$ selects the final subset $\mathcal{N}_{\text{ret}}$. To instantiate the retrieval-time refinement process defined in Section 2.2, we classify retrieval operators into three categories based on input sources and scoring mechanisms: Vector Search ($O_{\text{v}}$), Graph Search ($O_{\text{g}}$), and Model-based Search ($O_{\text{m}}$). Specifically, $O_{\text{v}}$ initializes seed nodes, allowing $O_{\text{g}}$ and $O_{\text{m}}$ to execute actual retrieval-time refinement via adding and pruning, respectively. We detail their definitions below.

*3.1.1 Vector Search Operator.* The **Vector Search Operator** ($O_{\text{v}}$) serves as the seed initialization step. Instead of operating within the retrieval-time refinement loop, $O_{\text{v}}$ is responsible for generating the initial seed nodes that subsequent adding and pruning steps refine. Taking a query vector $\mathbf{v}_q$ and the set of all nodes $\mathcal{N}$ (associated with pre-indexed vectors $\mathcal{V}$) as inputs, it employs a ranking function $\mathcal{R}_{\text{V}}$ to compute vector similarity. This is followed by a pruning function $\mathcal{P}_{\text{V}}$ that identifies the top-$k$ nodes. Formally, $O_{\text{v}}$ is defined as follows:

**Definition 3** (Vector Search, $O_{\text{v}}$)**.** The Vector Search operator retrieves a node set $\mathcal{N}_{\text{VS}}$ by

$$\begin{aligned}\mathcal{N}_{\text{V}} &= O_{\text{v}}(\mathbf{v}_q, \mathcal{N}, k) = \mathcal{P}_{\text{V}}(\{\mathcal{R}_{\text{V}}(\mathbf{v}_q, n) \mid n \in \mathcal{N}\}, k) \\ &= \arg\operatorname*{topk}_{n \in \mathcal{N}} \ \text{sim}(\mathbf{v}_q, \mathbf{v}_n)\end{aligned}$$

where $\mathbf{v}_n$ denotes the vector representation of node $n$, and $\text{sim}(\cdot,\cdot)$ denotes a vector similarity metric.

**Example 1** (Dense Vector Search)**.** *In dense retrieval, the node vectors lie in a continuous latent manifold $\mathbb{R}^d$. The similarity is typically defined as the cosine similarity:*

$$\text{sim}(\mathbf{v}_q, \mathbf{v}_n) = \frac{\mathbf{v}_q^\top \mathbf{v}_n}{|\mathbf{v}_q| \cdot |\mathbf{v}_n|}$$

*3.1.2 Graph Search Operator.* The **Graph Search Operator** ($O_{\text{g}}$) functions as the *adding* step during refinement in retrieval-time. It takes a set of seed nodes $\mathcal{N}_{\text{seed}} \subset \mathcal{N}$, seed node features $\mathbf{H}_{\text{seed}}$, and the edge set $\mathcal{E}$ as inputs. It employs a ranking function $\mathcal{R}_{\text{G}}$ to calculate topological scores $\mathbf{H}^{(L)}$ via $L$-step signal propagation on $\mathcal{E}$, generalizing Graph Neural Networks (GNNs) message-passing. By depending on $\mathcal{N}_{\text{seed}}$ and $\mathbf{H}_{\text{seed}}$, $O_{\text{g}}$ to perform expansion based on the intermediate retrieved set, making it the primary mechanism for *adding* new nodes during iterative refinement. We define $O_{\text{g}}$ as follows:

**Definition 4** (Graph Search, $O_{\text{g}}$)**.** The Graph Search operator retrieves a node set $\mathcal{N}_{\text{G}}$ by

$$\begin{aligned}\mathcal{N}_{\text{G}} &= O_{\text{g}}(\mathcal{N}_{\text{seed}}, \mathbf{H}_{\text{seed}}, \mathcal{E}) = \mathcal{P}_{\text{G}}(\mathcal{R}_{\text{G}}(\mathcal{N}_{\text{seed}}, \mathbf{H}_{\text{seed}}, \mathcal{E})) \\ &= \mathcal{P}_{\text{G}}\left(\mathbf{H}^{(L)}\right)\end{aligned}$$

Here, $\mathbf{H}^{(L)}$ denotes the node scores in $\mathcal{N}$ after $L$ steps. With $\mathbf{H}^{(0)} = \mathbf{H}_{\text{seed}}$ initialized by the preceding retrieval, the $l$-th update rule is:

$$\mathbf{H}^{(l)} = f_{\text{prop}}^{(l)}\left(\mathcal{N}_{\text{seed}}, \mathbf{H}^{(l-1)}, \mathcal{E}\right)$$

**Example 2** (1-hop Traversal)**.** *In 1-hop traversal-based retrieval [12], $\mathcal{R}_{\text{G}}$ is a single-layer propagation ($L = 1$) where $f_{prop}^{(1)}$ is defined as:*

$$\mathbf{H}^{(1)} = f_{prop}^{(1)}(\mathcal{N}_{seed}, \mathbf{H}^{(0)}, \mathcal{E}) = \mathbf{A}\mathbf{H}^{(0)}$$

*where $\mathbf{A}$ is the adjacency matrix derived from $\mathcal{E}$. Here, $\mathbf{H}^{(0)}$ is initialized such that $[\mathbf{H}^{(0)}]_n = \mathbb{I}(n \in \mathcal{N}_{seed})$. The pruning function $\mathcal{P}_{\text{G}}$ selects immediate neighbors by excluding the seed nodes themselves:*

$$\mathcal{P}_{\text{G}}(\mathbf{H}^{(1)}) = \{n \mid [\mathbf{H}^{(1)}]_n > 0\} \setminus \mathcal{N}_{seed}$$

**Example 3** (PageRank Retrieval)**.** *In PageRank-based retrieval [14, 21], $\mathcal{R}_{\text{G}}$ performs iterative propagation using the update function $f_{prop}^{(l)}$ until $\mathbf{H}^{(L)}$ converges, where $f_{prop}^{(l)}$ is defined as:*

$$\mathbf{H}^{(l)} = f_{prop}^{(l)}(\mathcal{N}_{seed}, \mathbf{H}^{(l-1)}, \mathcal{E}) = (1-\alpha)\mathbf{M}\mathbf{H}^{(l-1)} + \alpha\mathbf{H}^{(0)}$$

*where $\alpha$ denotes the restart probability and $\mathbf{M}$ is the column-normalized transition matrix derived from $\mathcal{E}$. Here, $\mathbf{H}^{(0)}$ serves as the personalized restart distribution. The pruning function $\mathcal{P}_{\text{G}}$ selects the top-k nodes with the highest scores as follows:*

$$\mathcal{P}_{\text{G}}(\mathbf{H}^{(L)}) = \arg\operatorname*{topk}_{n \in \mathcal{N} \setminus \mathcal{N}_{seed}} [\mathbf{H}^{(L)}]_n$$

*3.1.3 Model-based Search.* Third, the **Model-based Search ($O_{\text{m}}$)** operator serves as the *pruning* step within the refinement process. While $O_{\text{g}}$ refines the set by adding candidate nodes, $O_{\text{m}}$ evaluates and *prunes* this intermediate retrieved set to retain only highly relevant nodes. It takes a textual query $q$ and a set of seed nodes $\mathcal{N}_{\text{seed}} \subseteq \mathcal{N}$ as inputs. In its ranking function $\mathcal{R}_{\text{M}}$, it utilizes a computationally intensive model $P_\phi$ (e.g., a language model) to process the

raw textual node contents and assess their semantic relevance. By enabling late query–content interaction, $O_{\mathrm{m}}$ can identify and downrank unhelpful nodes that are semantically irrelevant. Due to the high computational cost of this interaction, the operator is typically restricted to evaluating the smaller subset $\mathcal{N}_{\mathrm{seed}} \subset \mathcal{N}$ generated by the preceding steps. Formally, $O_{\mathrm{m}}$ is defined as follows:

**Definition 5** (Model-based Search, $O_{\mathrm{m}}$)**.** The Model-based Search operator selects a node set $\mathcal{N}_{\mathrm{M}}$ from $\mathcal{N}_{\mathrm{seed}} \subset \mathcal{N}$ by

$$\begin{aligned}\mathcal{N}_{\mathrm{M}} &= O_{\mathrm{m}}(q, \mathcal{N}_{\mathrm{seed}}, k) = \mathcal{P}_{\mathrm{M}}(\{\mathcal{R}_{\mathrm{M}}(q,n) \mid n \in \mathcal{N}_{\mathrm{seed}}\}, k) \\ &= \underset{n \in \mathcal{N}_{\mathrm{seed}}}{\arg\operatorname{topk}}\ P_\phi(q,n)\end{aligned}$$

## 3.2 Unifying Existing Graph Retrieval Methods

The three retrieval operators defined in Section 3.1 serve as the foundational units shared by the representative retrieval methods. As shown in Table 1, existing methods that appear algorithmically diverse can be expressed as different compositions of $O_{\mathrm{v}}$, $O_{\mathrm{g}}$ and $O_{\mathrm{m}}$.

For example, a standard Retrieve-then-Rerank pipeline (Example 4) is formally expressed as $O_{\mathrm{v}} \rightarrow O_{\mathrm{m}}$: vector search first produces a candidate set, which is subsequently pruned by a model-based reranker. Similarly, G-Retriever (Example 5) is expressed as $O_{\mathrm{v}} \rightarrow O_{\mathrm{g}}$, where initial seed nodes are identified via vector search and then expanded into the final retrieved set via graph-based search. Our proposed taxonomy provides a unified lens that encompasses a broad spectrum of representative graph retrieval methods, such as LightRAG [12], PathRAG [5], and GAR [27] shown in Figure 1(b–d). Ultimately, the decomposition of existing methods in Table 1 demonstrates that *seemingly disparate methods can be rewritten using our fundamental operator taxonomy.*

**Example 4** (Retrieve-then-Rerank Pipeline)**.** *Consider a standard pipeline employing a Bi-encoder (e.g., Contriever) for retrieval and a Cross-encoder (e.g., BERT) for reranking. While the overall process is neither $O_{\mathrm{v}}$ nor $O_{\mathrm{m}}$, we can divide the pipeline into two subprocesses:*

*(1) Candidate Retrieval: The Bi-encoder retrieves the top-100 candidates ($\mathcal{N}_{cand}$) based on cosine vector similarity with query vector $\mathbf{v}_q = Contriever(q)$. This instantiates $O_{\mathrm{v}}$:*

$$\mathcal{N}_{cand} = \underset{n \in \mathcal{N}}{\arg\operatorname{top100}}\ \frac{\mathbf{v}_q^\top \mathbf{v}_n}{|\mathbf{v}_q| \cdot |\mathbf{v}_n|} = O_{\mathrm{v}}(\mathbf{v}_q, \mathcal{N}, 100)$$

*(2) Reranking: The Cross-encoder scores $\mathcal{N}_{cand}$ given the query $q$ to select the final top-10 nodes. This instantiates $O_{\mathrm{m}}$:*

$$\mathcal{N}_{final} = \underset{n \in \mathcal{N}_{seed}}{\arg\operatorname{topk}}\ MLP(BERT(q \oplus n)) = O_{\mathrm{m}}(q, \mathcal{N}_{cand}, 10)$$

**Example 5** (G-Retriever)**.** *The retrieval in G-Retriever [16] comprises two stages: (1) Vector-Edge Retrieval ($O_{\mathrm{v}}$) and (2) PCST-based subgraph construction ($O_{\mathrm{g}}$), as detailed below:*

*(1) Vector-Edge Retrieval ($O_{\mathrm{v}}$): It retrieves the top-$k$ $\mathcal{N}_{sub} \subset \mathcal{N}$ and $\mathcal{E}_{sub} \subset \mathcal{E}$ via vector similarity. For the node and edge at rank $i$, we assign rank-based prizes $k - i$ to initialize $\mathbf{H}_{seed}$.*

*(2) PCST Construction ($O_{\mathrm{g}}$): The operator selects the final subgraph $\mathcal{N}_{ret}$ and $\mathcal{E}_{ret}$ by optimizing the PCST ranking function:*

$$\mathcal{R}_{PCST}(\mathcal{N}_{ret}, \mathcal{E}_{ret}) = \sum_{n \in \mathcal{N}_{ret}} p(n) + \sum_{e \in \mathcal{E}_{ret}} p(e) - c(\mathcal{N}_{ret}, \mathcal{E}_{ret})$$

*Here, $p(n)$ and $p(e)$ correspond to the values in $\mathbf{H}_{seed}$, and $c(\mathcal{N}_{ret}, \mathcal{E}_{ret})$ only depends on $\mathcal{E}$.*

**Table 1: Comparison of representative Graph RAG methods based on target database and retrieval operators.**

| Target database | Methods | Basic operations | | | Fusion operators | |
|---|---|---|---|---|---|---|
| | | $O_{\mathrm{v}}$ | $O_{\mathrm{g}}$ | $O_{\mathrm{m}}$ | $O_{\mathrm{vg}}$ | $O_{\mathrm{gm}}$ |
| Knowledge graph | HyKGE [20] | ✓ | × | × | × | × |
| | HippoRAG [21] GNN-RAG [28] G-Retriever [16] | ✓ | ✓ | × | × | × |
| | SubgraphRAG [24] | ✓ | × | ✓ | × | × |
| | LightPROF [2] | ✓ | ✓ | ✓ | × | × |
| | ToG [33] | ✓ | ✓ | ✓ | × | × |
| Corpus graph | LightRAG [12] PathRAG [5] | ✓ | ✓ | × | × | × |
| | GRAG [17] | ✓ | × | × | × | × |
| Corpus graph + KG | KG2RAG [38] | ✓ | ✓ | ✓ | × | × |
| Documents + KG | ToG 2.0 [26] | ✓ | ✓ | ✓ | × | × |
| | HippoRAG 2 [14] | ✓ | ✓ | × | × | × |
| Corpus graph | **GraFine (ours)** | ✓ | - | - | ✓ | ✓ |

More importantly, this decomposition exposes the **characteristic blind spots** that limit retrieval-time refinement in current methods. Since each operator can only rank nodes based on the information in its inputs, current operators are inherently restricted in how it performs the crucial *adding* and *pruning* of nodes.

- $O_{\mathrm{g}}$ **performs semantically-blind *adding***: While $O_{\mathrm{g}}$ effectively propagates signals over graph topology, it lacks direct query–content interaction when identifying new candidate nodes. Thus, during the *adding* step, $O_{\mathrm{g}}$ often drifts toward topologically reachable but semantically irrelevant nodes (e.g., node $n_5$ representing *Hallucination* in Figure 1(d)), failing to expand in a contextually relevant direction.
- $O_{\mathrm{m}}$ **performs topologically-blind *pruning***: Conversely, while $O_{\mathrm{m}}$ enables rich query–content interaction, it evaluates nodes in isolation from the graph structure. *Pruning* through $O_{\mathrm{m}}$ relies solely on local semantic similarity, causing it to assign misleading low scores to crucial nodes (e.g., node $n_4$ in Figure 1(d)) whose relevance is only established in the structural links and contextual clues provided by surrounding intermediate nodes.

Consequently, existing methods composed of these foundational operators inherit these blind spots as systemic limitations. Methods relying on $O_{\mathrm{g}}$ for adding are prone to drifting toward *topologically reachable but semantically unhelpful* nodes, while methods using $O_{\mathrm{m}}$ for pruning often fail to retain nodes whose relevance emerges *only through graph context.* Since any composition of these operators inherently suffers from these blind spots, current methods are restricted in their ability to perform effective retrieval-time refinement.

# 4 GraFine: Retrieval-Time Refinement for Graph RAG

## 4.1 Design: Semantic–Topological Refinement

To achieve effective retrieval-time refinement and move beyond the systemic limitations of existing methods, it is important to resolve the blind spots inherent in the fundamental operators. To this end, we propose two novel fusion operators which leverage

both semantic and topological inputs, ensuring the *adding* and *pruning* steps are semantically guided and topologically aware, respectively.

First, to overcome the *semantic blindness* inherent in $O_g$, we define a **Vector-Graph Search** ($O_{vg}$) operator. Rather than drifting toward topologically reachable but *unhelpful* nodes, $O_{vg}$ expands the retrieved set by utilizing both the query vector $\mathbf{v}_q$ and graph structure $\mathcal{E}$. This dual integration ensures that the addition of new nodes is steered toward semantically valuable regions.

**Definition 6** (Vector-Graph Search, $O_{vg}$)**.** $O_{vg}$ retrieves nodes using both vector representations $\mathcal{V}$ and graph topology $\mathcal{E}$:

$$\mathcal{N}_{VG} = O_{vg}(\mathbf{v}_q, \mathcal{V}, \mathcal{N}_{seed}, \mathcal{E})$$

Second, to overcome the *topological blindness* inherent in $O_m$, we define a **Graph-Model Search** ($O_{gm}$) operator. Instead of prematurely deleting nodes that lack standalone semantic similarity, $O_{gm}$ integrates the structural context of surrounding nodes $\mathcal{E}$ directly into relevance scoring. This ensures the pruning process safely retains nodes whose true value emerges only through the structural context.

**Definition 7** (Graph-Model Search, $O_{gm}$)**.** $O_{gm}$ selects nodes from $\mathcal{N}_{seed} \subset \mathcal{N}$ by incorporating topological context $\mathcal{E}$:

$$\mathcal{N}_{GM} = O_{gm}(q, \mathcal{N}_{seed}, \mathcal{E}, k) = \mathcal{P}_{GM}(\mathcal{R}_{GM}(q, \mathcal{N}_{seed}, \mathcal{E}), k)$$

## 4.2 GraFine Framework

The **GraFine** framework is the practical realization of our proposed retriever design, developed to execute effective retrieval-time refinement by resolving blind spots. GraFine takes the following inputs: a query $q$, a corpus graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, a query vector $\mathbf{v}_q$, a set of node vectors $\mathcal{V}$, and a set of *hyperparameters* ($BATCH$, $\alpha$, $\beta$, and $b_{max}$). The output is a final, refined set of retrieved nodes, $\mathcal{N}_{ret}$.

The execution flow of GraFine is outlined in Algorithm 1. In this algorithm, each colored box represents a retrieval operator: $O_v$ (Vector Search), $O_{gm}$ (Graph Model-based Search), and $O_{vg}$ (Vector-Graph Search). The process comprises two primary phases: the **initial setup step** and the **iterative retrieval step**. We explain these steps in detail below. Here, SPX and GSR stand for Semantics-Guided Graph Expansion (Section 4.3) and Graph-Aware Reranking (Section 4.4), respectively, which are proposed in this paper.

*Initial setup step (Lines 1–2).* In this step, the retriever establishes the starting nodes for the subsequent iterative process.

- **(L1)** First, a dot product-based vector search is performed on the node vectors $\mathcal{V}$ to retrieve a top-$BATCH$ list of candidates, $\mathcal{N}_{ret}$, sorted by their dot product scores.
- **(L2)** The **GSR** method is applied to assign initial relevance scores to these nodes.

By employing this iterative process *(Lines 4–7)*, GraFine enhances the quality of the input nodes $\mathcal{N}_{ret}$ fed into GSR, allowing us to produce better retrieval outcomes while minimizing the usage of GSR (i.e., minimize computational cost).

- **(L4)** In each iteration, our **SPX** algorithm ($O_{vg}$ operator) identifies potential new nodes ($\mathcal{N}_{add}$) based on the current $\mathcal{N}_{ret}$.
- **(L5–L6)** The retriever then incorporates up to $BATCH$ new nodes into $\mathcal{N}_{ret}$ (strictly adhering to the budget cap $b_{max}$).

**Algorithm 1** The GraFine framework

**Input:** Query $q$, Graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, Node vectors $\mathcal{V} = \{Enc_N(n_i) \mid n_i \in \mathcal{N}\}$, Query vector $\mathbf{v}_q = Enc_Q(q)$, Batch size $BATCH$, Smoothing factor $\alpha$, Score ratio $\beta$, Budget $b_{max}$
**Output:** $\mathcal{N}_{ret}$ (Set of retrieved nodes)

1: $\mathcal{N}_{ret} \leftarrow \arg\text{top}\, BATCH_{n \in \mathcal{N}}\ \text{sim}(\mathbf{v}_q, \mathcal{V}_n)$ ▷ 1. Initial $O_v$
2: $\mathcal{N}_{ret} \leftarrow \textbf{GSR}(q, \mathcal{N}_{ret}, \mathcal{E}, \alpha)$ ▷ 2. $O_{gm}$ using *GSR*
3: **while** $|\mathcal{N}_{ret}| < b_{max}$ **do** ▷ 3. Expansion Loop with $O_{vg}$
4: $\quad \mathcal{N}_{add} \leftarrow \textbf{SPX}(\mathbf{v}_q, \mathcal{V}, \mathcal{E}, \mathcal{N}_{ret}, \beta)$ ▷ $O_{vg}$ using *SPX*
5: $\quad k_{remain} \leftarrow \min(|\mathcal{N}_{ret}| + BATCH, b_{max}) - |\mathcal{N}_{ret}|$
6: $\quad \mathcal{N}_{ret} \leftarrow \mathcal{N}_{ret} \cup \mathcal{N}_{add}[:k_{remain}]$ ▷ Apply hard budget cap
7: $\quad \mathcal{N}_{ret} \leftarrow \textbf{GSR}(q, \mathcal{N}_{ret}, \mathcal{E}, \alpha)$ ▷ $O_{gm}$ using *GSR*
8: **end while**
9: **return** $\mathcal{N}_{ret}$

- **(L7)** For the next iteration, the retriever re-applies **GSR** to update the rankings of the retrieved nodes $\mathcal{N}_{ret}$.

The hyperparameters control the algorithm's behavior as follows: $BATCH$ denotes the number of nodes added to $\mathcal{N}_{ret}$ during a single iteration; $\alpha$ represents the smoothing factor for GSR (detailed in Section 4.4); $\beta$ represents the score ratio for SPX (in Section 4.3) and $b_{max}$ defines the maximum node budget, serving as the stopping criterion for the iterative loop.

## 4.3 SPX: Semantics-Guided Graph Expansion

We propose **Semantic Proximity eXpansion (SPX)**, a lightweight implementation of the $O_{vg}$ that identifies candidates to *add* by leveraging both topological structure and semantic representations, unlike conventional topology-only methods. As detailed in Algorithm 2, the procedure ranks candidates in $\mathcal{N}_{SPX}$ using a composite score—a $\beta$-weighted sum of structural importance ($I_{Struct}$) and semantic similarity ($I_{Sim}$):

- $I_{Struct}$ **(Lines 3–12):** This score combines *rank proximity* and *bridging capability*. It captures proximity to high-ranking context by favoring candidates connected to the highest-ranked nodes ($r_{best}$) in $\mathcal{N}_{ret}$. Additionally, it includes a bridging factor $|A(n)|$ that favors nodes linked to multiple retrieved nodes across the graph structure.
- $I_{Sim}$ **(Line 13):** This is the dot product similarity between the query $\mathbf{v}_q$ and the candidate vector $\mathcal{V}_n$. This helps preserve semantically relevant nodes even if not topologically central.

## 4.4 GSR: Graph-Aware Reranking

As a practical implementation of the $O_{gm}$ operator defined in Definition 7, we propose the **Graph Smoothing Reranker (GSR)**. To overcome the *topological blindness* in conventional model search, we interpret the initial cross-encoder embeddings $\mathbf{H}$ as structurally incomplete, noisy signals, as they rely solely on isolated text semantics. Consequently, GSR treats the task as a *graph signal denoising problem*, aiming to smooth $\mathbf{H}$ by leveraging $\mathcal{E}$. This corresponds to minimizing the Laplacian-regularized objective:

$$\mathcal{L}(\mathbf{H}') = \frac{1}{2}\|\mathbf{H}' - \mathbf{H}\|_F^2 + \frac{\lambda}{2}\text{Tr}(\mathbf{H}'^{\top}\mathbf{L}_{rw}\mathbf{H}')$$

**Algorithm 2** Our SPX method for $O_{\text{vg}}$

**Input:** Query vector $\mathbf{v}_q$, Node vectors $\mathcal{V}$, Edges $\mathcal{E}$, Retrieved $\mathcal{N}_{\text{ret}}$, Score ratio $\beta$.
**Inner function:** rankCheck$(n, \mathcal{N}_{\text{ret}})$ checks the rank of $n$, $\deg_{\mathcal{E}}(n)$ calculates the *total* degree of the node $n$.
**Output:** Set of nodes to add $\mathcal{N}_{\text{add}}$

$\mathcal{N}_{\text{SPX}} \leftarrow \{n_j \mid \exists_{n_i \in \mathcal{N}_{\text{ret}}}(n_i, n_j) \in \mathcal{E}\} \setminus \mathcal{N}_{\text{ret}}, R_{max} \leftarrow |\mathcal{N}_{\text{ret}}|$
**for** $n \in \mathcal{N}_{\text{SPX}}$ **do**
  $I_{Struct} \leftarrow 0$
  $A(n) \leftarrow \{v \in \mathcal{N}_{\text{ret}} \mid (v, n) \in \mathcal{E}\}$ ▷ Adjacent retrieved nodes
  **if** $R_{max} > 1$ **then**
    $r_{best} \leftarrow \min\{\text{rankCheck}(v, \mathcal{N}_{\text{ret}}) \mid v \in A(n)\}$
    $I_{Struct} \leftarrow 1 - \frac{r_{best}-1}{R_{max}-1}$
  **end if**
  $C_{max} \leftarrow \min(\deg_{\mathcal{E}}(n), R_{max})$
  **if** $C_{max} > 1$ **then**
    $I_{Struct} \leftarrow I_{Struct} + \frac{|A(n)|-1}{C_{max}-1}$ ▷ 1. *Structural score* ($I_{Struct}$)
  **end if**
  $I_{Sim} \leftarrow \mathbf{v}_q \cdot \mathcal{V}_n$ ▷ 2. *Similarity score* ($I_{Sim}$)
  $\mathbf{S}_n \leftarrow I_{Sim} + \beta \cdot (I_{Struct})$
**end for**
$\mathcal{N}_{\text{add}} \leftarrow \texttt{argsort}(\mathcal{N}_{\text{SPX}}, \texttt{score} = \mathbf{S})$
**return** $\mathcal{N}_{\text{add}}$

where $\mathbf{L}_{rw} = \mathbf{I} - \mathbf{P}$ is the random-walk Laplacian. Instead of the computationally expensive closed-form solution, we employ a *first-order approximation* via a single gradient descent step. This yields our efficient update rule:

$$\mathbf{H}' \leftarrow \mathbf{H} - \eta \nabla \mathcal{L}(\mathbf{H}) = (1-\alpha)\mathbf{H} + \alpha(\mathbf{PH})$$

where $\alpha = \eta\lambda$. Algorithm 3 details this process, where Lines 2–7 introduce our modifications to the standard *pruning* workflow. The detailed procedure of these steps is as follows:

- **(L2–L5) Propagation Matrix Construction:** GSR constructs a normalized propagation matrix $\mathbf{P}$ from subgraph adjacency ($\mathbf{A}$) and degree ($\mathbf{D}$) matrices. Using reciprocal node degrees, $\mathbf{P}$ moderates high-degree node influence during aggregation.
- **(L6) Latent Graph Fusion:** Initial latent vectors $\mathbf{H}$ are smoothed with neighbor-aggregated context ($\mathbf{P}\cdot\mathbf{H}$) via convolution. The factor $\alpha$ controls the trade-off between semantics and structural support, yielding fused representations $\mathbf{H}'$.
- **(L7) Semantic Scoring:** The final relevance scores $\mathbf{S}$ are computed by passing $\mathbf{H}'$ through the MLP head, so the ranking reflects both semantic relevance and topological evidence.

## 4.5 Efficiency Analysis

In this section, we analyze the efficiency of GraFine with a focus on the two refinement components, SPX for $O_{\text{vg}}$ and GSR for $O_{\text{gm}}$. Overall, after the initial retrieval, a query performs $T$ executions of SPX and $T+1$ executions of GSR, where $T$ is the number of refinement iterations until the budget cap is reached. We compare the practical efficiency of these components with related methods on real hardware in Section 6.2.1.

*For $O_{\text{vg}}$.* SPX scores only the candidates reachable from the current retrieved nodes. Under an adjacency-list implementation with precomputed node degrees and a lookup table for the current ranks,

**Algorithm 3** Our GSR method for $O_{\text{gm}}$

**Input:** Query $q$, Retrieved $\mathcal{N}_{\text{ret}}$, Edges $\mathcal{E}$, Smoothing factor $\alpha$
**Output:** Reranked list of nodes $\mathcal{N}_{\text{ret}}$
**Inner function:** Encoder$(\cdot)$ for latent vector extraction, MLP$(\cdot)$ for scoring.

1: $\mathbf{H} \leftarrow [\text{Encoder}(q, n_i)]_{n_i \in \mathcal{N}_{\text{ret}}}$ ▷ Extract Latent Vectors
2: $\mathbf{A} \in \{0,1\}^{|\mathcal{N}_{\text{ret}}| \times |\mathcal{N}_{\text{ret}}|}$ where $\mathbf{A}_{ij} = \mathbb{I}((n_i, n_j) \in \mathcal{E})$
3: $\mathbf{D} \in \mathbb{R}^{|\mathcal{N}_{\text{ret}}| \times |\mathcal{N}_{\text{ret}}|}$ where $D_{ii} = \sum_j A_{ij}$
4: $\mathbf{W} \leftarrow \mathbf{A} \cdot \mathbf{D}^{-1}$ ▷ Weighting by reciprocal of degrees
5: $\mathbf{P} \leftarrow \text{diag}(\mathbf{W} \cdot \mathbf{1})^{-1} \cdot \mathbf{W}$ ▷ Normalized Propagation Matrix
6: $\mathbf{H}' \leftarrow (1-\alpha)\mathbf{H} + \alpha(\mathbf{PH})$ ▷ **Latent Graph Fusion**
7: $\mathbf{S} \leftarrow \text{MLP}(\mathbf{H}')$ ▷ Scoring via Classifier Head
8: $\mathcal{N}_{\text{ret}} \leftarrow \texttt{argsort}(\mathcal{N}_{\text{ret}}, \texttt{score=S})$
9: **return** $\mathcal{N}_{\text{ret}}$

each execution scans the relevant outgoing edges once to construct $N_{\text{add}}$, computes the structural scores, and then evaluates vector similarities for those candidates. Therefore, one SPX call is linear in the size of the explored nodes, up to the candidate-sorting step, which adds $O(|N_{\text{add}}| \log |N_{\text{add}}|)$. Accordingly, the total cost of $O_{\text{vg}}$ is the sum of these costs across all iterations, which is relatively small in our experimental observation.

*For $O_{\text{gm}}$.* Each GSR execution reranks only the current retrieved set. The dominant semantic computation remains the PLM-based encoding in Line 1 of Algorithm 3, but we cache the latent vectors $\mathbf{H}$, so each node is encoded only when it first enters $\mathcal{N}_{\text{ret}}$; hence, across the whole query, the total encoding cost grows with the number of distinct retrieved nodes, which is bounded by $b_{\max}$, rather than with the number of GSR calls. The additional graph operations incur a cost of $O(|\mathcal{N}_{\text{ret}}|^2)$ per execution, but they are applied only to the current retrieved set. We examine the practical effect of these local overheads in Section 6.2.1.

# 5 Experimental Evaluation

We conduct two types of experiments: (1) *a retrieval experiment*, which aims to retrieve $\mathcal{N}_{\text{ret}}$ for a given query $q$, and (2) *a RAG experiment*, which focuses on generating responses based on $\mathcal{N}_{\text{ret}}$. Unless otherwise specified, we use OpenAI's text-embedding-3-small as the embedding model, OpenAI's gpt-5-mini as the generative LLM, and bge-reranker-v2-m3 as the reranker. For validating efficiency, we use two server configurations: (a) eight NVIDIA 24GB TITAN GPUs and (b) six NVIDIA 80GB A100 GPUs. We use these two hardware configurations to demonstrate that the efficiency trends of the compared methods remain robust across different system environments. All other experiments, except those for efficiency evaluation, are conducted via configuration (a).

## 5.1 Baselines

*5.1.1 Retrieval Baselines.* We evaluate retrieval performance using five document retrieval baselines and four graph retrieval baselines. While many Graph RAG methods designed for KGs are incompatible with corpus graphs, we include HippoRAG 2 [14] by adapting it to the corpus graph setting. Other methods that cannot be applied to corpus graphs are excluded.

Five document retrieval baselines rely on vector search ($O_{\text{v}}$) or model-based reranking ($O_{\text{m}}$). We include (1) **Vector Search** ($O_{\text{v}}$):

retrieves nodes based on dot product similarity; (2-3) **SPLADE** [9] ($O_v$) and **Contriever** [19] ($O_v$): representative sparse and dense retrieval methods, respectively; (4) **HyDE** [10] ($O_v$): performs retrieval using generated hypothetical documents; and (5) **Retrieve-then-Rerank (Re2)** ($O_v, O_m$): reranks the top-100 candidates retrieved by Vector Search.

Four graph retrieval baselines and GraFine incorporate graph topology via the graph search operator ($O_g$). We include (1) **GAR** [27] ($O_v, O_g, O_m$): dynamically interleaves vector and graph search with iterative reranking ($b_{max} = 100, BATCH = 10$); (2-3) **LightRAG/PathRAG** [5, 12] ($O_v, O_g$): refine queries using keywords generated by gpt-4o-mini, prior to executing $O_v$ and $O_g$; and (4) **HippoRAG 2** ($O_v, O_g$): a KG-based method that we adapted to perform retrieval on corpus graph topology. Finally, our **GraFine** ($O_v, O_{vg}, O_{gm}$): utilizes frozen [CLS] features extracted from the reranker (pre-MLP) to initialize **H**, leverages the last two layers of its classification head as $MLP(\cdot)$, and sets $b_{max}=100, BATCH=10, \alpha=0.2$, and $\beta=1$.

*5.1.2 RAG Baselines.* We generate responses by feeding the nodes retrieved by each retriever in the retrieval experiment into OpenAI's GPT-5-nano model. Consequently, the RAG experiment includes a total of 10 methods, nine baselines and GraFine.

## 5.2 Datasets

*5.2.1 The ACL-OCL dataset.* ACL OCL [31] is a text corpus derived from ACL Anthology, comprising approximately 80k academic papers with references and full texts. To evaluate baseline models on reference networks, we transform this corpus into the *ACL-OCL dataset*, which will be publicly released, specifically designed to assess both retrieval and generation performance. Unlike existing datasets, ACL-OCL emphasizes scenarios where retrieving the correct answer requires understanding the semantic and structural context of intermediate nodes. We construct the dataset through two stages: *reference network construction* and *synthetic query generation*.

*Reference Network Construction.* Each paper is divided into chunks of 4,096 characters, which constitute the node set $\mathcal{N}$. To construct the edge set $\mathcal{E}$ while mitigating spurious edges caused by paper-level metadata, we employ a reference detection model that identifies explicit citations within each chunk $n$. For each detected citation, we create an edge $(n, n_i)$ linking the chunk to the cited paper's corresponding node(s). This model is implemented using GPT-5-nano via in-context learning. Detailed statistics are in Table 2.

*Synthetic Query Generation.* We generate synthetic query-gold node pairs by selecting connected node pairs and prompting an LLM to formulate questions requiring information from both. This design intentionally targets the evaluation of retrieval-time refinement, as answering these queries necessitates interpreting node contents to bridge the semantic gap between the query and the target answer. In total, we generated 753 queries, as illustrated in Figure 2.

*5.2.2 Datasets for Experiments.* To evaluate retrieval performance, we use a total of five datasets spanning two types of corpus graphs: ACL-OCL and LACD [1] for reference networks, and BSARD-G, SciFact-G, and NFcorpus-G for text-rich knowledge graphs. As there are currently no established IR benchmarks specifically designed for text-rich knowledge graphs, we adapt three widely used IR benchmarks—BSARD [25], SciFact [36], and NFcorpus [3]—into graph-based formats following the graph construction procedure used in LightRAG. For ground-truth relevance, we define all nodes constructed from the original gold documents as oracle nodes.

**Query**

What visualization approach and export formats does the web-based annotation tool mentioned in **LIDA** use to render complex, overlapping text annotations and produce figures for publications?

**Related Nodes (Oracle Nodes)**

***LIDA node chunk #2:***
**BRAT (Stenetorp et al., 2012)** and Doccano 3 are web-based annotation tools [...]. **LIDA** aims to fill these gaps by providing [...]

***BRAT node chunk #1:***
BRAT is based on our previously released opensource STAV text annotation visualiser[...] Both tools share a **vector graphics-based visualisation** component [...] **BRAT integrates PDF and EPS image format** export [...]

**Figure 2: Example of synthetic query generation. Red indicates textual reference to the BRAT node, while blue indicates the answer.**

For the RAG experiment, we use ACL-OCL for reference networks and two datasets from *UltraDomain* [29] for text-rich knowledge graphs. We excluded LACD, SciFact and NFCorpus from the RAG evaluation, as they are not formatted as QA datasets. Detailed statistics for all datasets are summarized in Table 2.

**Table 2: Dataset statistics for retrieval and RAG experiments. UD: UltraDomain datasets; Gray shading: *reference networks*.**

| Datasets | Purpose | Domain | $\lvert\mathcal{N}\rvert$ | $\lvert\mathcal{E}\rvert$ | $\lvert Q\rvert$ |
|---|---|---|---|---|---|
| ACL-OCL | Ret. & Gen. | CS | 402,742 | 5,840,449 | 753 |
| LACD | Retrieval | Legal | 192,974 | 339,666 | 89 |
| BSARD-G | Ret. & Gen. | Legal | 56,728 | 92,672 | 222 |
| SciFact-G | Retrieval | Science | 36,438 | 43,557 | 1,109 |
| NFcorpus-G | Retrieval | Medical | 23,468 | 27,805 | 3,237 |
| UD-agriculture | Generation | Agriculture | 46,561 | 82,088 | 100 |
| UD-mix | Generation | Mix | 11,812 | 10,384 | 130 |

## 5.3 Evaluation Metrics

*5.3.1 Conventional Metrics.* To evaluate retrieval performance, we use Capped Recall score ($R@k$) and Normalized Discounted Cumulative Gain (nDCG), both evaluated at **top-10**. Capped Recall [34] normalizes the maximum achievable recall to 1 when selecting the top-$k$ nodes. For generation evaluation, we adopt a pairwise LLM-as-a-Judge approach, following the evaluation protocol and prompts used in LightRAG [12], where a generative LLM serves as the evaluator.

*5.3.2 Topological Recall.* To quantify the effectiveness of *retrieval-time refinement*, we introduce **Topological Recall (TR)**, a new metric defined over the range of $[0, 1]$. Unlike conventional Recall, TR captures the topological proximity of oracle nodes to the retrieved set $\mathcal{N}_{ret}$ by modeling shortest-path *uncertainty*.

**Definition 8** (Topological Recall)**.** For given $\mathcal{E}, \mathcal{N}_{ret}, \mathcal{N}_{oracle}$:

$$TR(\mathcal{E}, \mathcal{N}_{ret}, \mathcal{N}_{oracle}) = \text{avg}_{n_i \in \mathcal{N}_{oracle}} \left( \frac{1}{1 + u(\mathcal{E}, \mathcal{N}_{ret}, n_i)} \right)$$

where the uncertainty function $u$ is defined as the accumulated log-degree along the shortest path:

$$u(\mathcal{E}, \mathcal{N}_{\text{ret}}, n_i) = \min_{n_j \in \mathcal{N}_{\text{ret}}} \sum_{n_k \in \text{SP}(\mathcal{E}, n_j, n_i), n_k \neq n_i} \ln(1 + \deg_{\mathcal{E}}(n_k))$$

Here, $\text{SP}(\mathcal{E}, n_j, n_i)$ denotes the set of nodes on the shortest path from the seed node $n_j$ to the oracle node $n_i$.

The intuition is that TR extends conventional Recall to capture the *refinement potential* of a retrieved set. An unexplored oracle node is considered highly accessible if it is structurally close to the intermediate $\mathcal{N}_{\text{ret}}$ via low-degree neighbors. Conversely, high-degree nodes increase branching uncertainty, making it more likely that iterative *adding* will introduce many irrelevant candidates that make it harder for iterative *pruning* step to maintain well-*refined* retrieved set. Thus, by penalizing the log-degree, TR measures how well the current node set is positioned to uncover oracle nodes via refinement.

To formalize this relationship and provide a theoretical foundation for its application in future research, we present the following decomposition as a corollary along with its proof.

**Corollary 1** (Decomposition). *For given $\mathcal{E}, \mathcal{N}_{ret}, \mathcal{N}_{oracle}$, TR and Recall for $\mathcal{N}_{ret}$ have the following relationship:*

$$TR = Recall + \frac{|\mathcal{N}_{oracle} \setminus \mathcal{N}_{ret}|}{|\mathcal{N}_{oracle}|} \cdot TR(\mathcal{E}, \mathcal{N}_{ret}, \mathcal{N}_{oracle} \setminus \mathcal{N}_{ret})$$

Proof. Let $\mathcal{N}_{\text{found}} = \mathcal{N}_{\text{oracle}} \cap \mathcal{N}_{\text{ret}}$ and $\mathcal{N}_{\text{miss}} = \mathcal{N}_{\text{oracle}} \setminus \mathcal{N}_{\text{ret}}$. Note that for any $n_i \in \mathcal{N}_{\text{found}}$, the uncertainty $u(\mathcal{E}, \mathcal{N}_{\text{ret}}, n_i) = 0$. By decomposing the summation in the definition of TR:

$$\begin{aligned} TR &= \frac{1}{|\mathcal{N}_{\text{oracle}}|} \left( \sum_{n_i \in \mathcal{N}_{\text{found}}} 1 + \sum_{n_i \in \mathcal{N}_{\text{miss}}} \frac{1}{1 + u(\mathcal{E}, \mathcal{N}_{\text{ret}}, n_i)} \right) \\ &= \underbrace{\frac{|\mathcal{N}_{\text{found}}|}{|\mathcal{N}_{\text{oracle}}|}}_{\text{Recall}} + \frac{|\mathcal{N}_{\text{miss}}|}{|\mathcal{N}_{\text{oracle}}|} \underbrace{\left( \frac{1}{|\mathcal{N}_{\text{miss}}|} \sum_{n_i \in \mathcal{N}_{\text{miss}}} \frac{1}{1 + u(\mathcal{E}, \mathcal{N}_{\text{ret}}, n_i)} \right)}_{TR(\mathcal{E}, \mathcal{N}_{\text{ret}}, \mathcal{N}_{\text{miss}})} \quad \square \end{aligned}$$

Hereafter, we refer to $\frac{|\mathcal{N}_{\text{oracle}} \setminus \mathcal{N}_{\text{ret}}|}{|\mathcal{N}_{\text{oracle}}|} \cdot TR(\mathcal{E}, \mathcal{N}_{\text{ret}}, \mathcal{N}_{\text{oracle}} \setminus \mathcal{N}_{\text{ret}})$ as *MissTR*, meaning the partial credit for missing oracle nodes. In Section 6.3, we analyze GraFine's capability for retrieval-time refinement using TR and MissTR.

# 6 Experimental Results

## 6.1 Retrieval and Generation Performance

*6.1.1 Retrieval Experiments.* Table 3 presents the performance of GraFine and nine baselines for the retrieval pipeline across five graph retrieval datasets. Overall, our method demonstrates robust and consistent performance improvements across all evaluated datasets. **GraFine outperforms the most competitive baseline by 9.9% in R@10 and 9.1% in nDCG@10** in terms of overall average performance.

Specifically, GraFine significantly outperforms Re2, the strongest document retrieval baseline, by an average of **20.0% in R@10 and 18.7% in nDCG@10**. Our method also yields substantial gains over GAR, the most competitive graph-based baseline, particularly in reference networks. For instance, in the ACL-OCL dataset, GraFine outperforms GAR by a relative margin of **27.5% in R@10 and 30.5% in nDCG@10**. Notably, on this same dataset, PathRAG fails to complete the retrieval within a reasonable time limit due to the high complexity of its flow algorithm. These results collectively highlight the effectiveness of our approach in navigating complex graph structures.

*6.1.2 RAG Experiments.* Table 4 presents overall win rates comparing baselines and our method over four datasets. Here, GraFine demonstrates superior performance in the RAG setting, consistently achieving *average win rates* **exceeding 55% against all baselines.** To investigate this improvement, we analyze the Pearson correlation between retrieval accuracy (R@10) and generation quality (win rate) on the ACL-OCL and BSARD-G datasets, as they are the only ones that support both retrieval and generation tasks. As shown in Figure 3, we observe a significantly strong positive correlation with $p < 0.05$. It suggests that the retrieval enhancements from our method translate into more effective generation, indicating that employing GraFine as a retriever strengthens the overall Graph RAG capability.

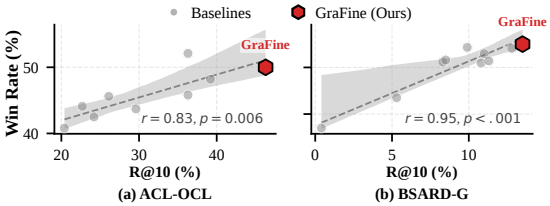



**Figure 3: Correlation between R@10 and Win Rate. GraFine is the self-reference baseline (50% win rate). Dashed lines and grey areas denote linear regression fits and 95% CIs.**

## 6.2 Efficiency and Latency Analysis

*6.2.1 Query Processing Time Analysis.* To demonstrate time-efficiency, we compare the *Query Processing Time (QPT)* and R@10 of GraFine against baselines Re2 and GAR across TITAN and A100 GPUs. Figure 4 illustrates the QPT and R@10 trade-off on ACL-OCL and SciFact-G. Each data point represents a different number of $O_{\text{m}}$ operators ($b_{\text{max}}$), from 10 to 100. As shown by the curves, GraFine achieves a **Pareto improvement**, consistently delivering higher R@10 without compromising efficiency on both datasets.

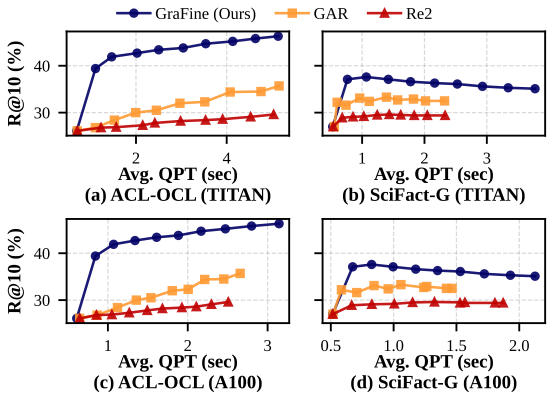



**Figure 4: Scatter plots illustrating the trade-off between Average QPT and R@10 on (a,c) ACL-OCL, and (b,d) SciFact-G.**

**Table 3: Retrieval results on five corpus graph datasets. All metrics are reported in percentage (%). The best results are highlighted in bold, and the second-best results are underlined. *Out-of-time* means that the method takes more than one hour per query.**

| Methods | Reference Networks | | | | Text-rich Knowledge Graphs | | | | | | Average | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ACL-OCL | | LACD | | BSARD-G | | SciFact-G | | NFCorpus-G | | Overall | |
| | R@10 | nDCG@10 | R@10 | nDCG@10 | R@10 | nDCG@10 | R@10 | nDCG@10 | R@10 | nDCG@10 | R@10 | nDCG@10 |
| **Document Retrieval** | | | | | | | | | | | | |
| Vector Search | 26.1 | 19.6 | 38.1 | 26.4 | 8.3 | 9.5 | 27.0 | 32.4 | 32.9 | 35.5 | 26.5 | 24.7 |
| SPLADE [9] | 39.2 | 33.2 | 0.0 | 0.0 | 5.3 | 5.8 | 27.2 | 32.5 | 33.8 | 36.1 | 21.1 | 21.5 |
| Contriever [19] | 20.4 | 16.0 | 1.1 | 0.9 | 0.4 | 0.3 | 27.7 | 32.5 | 35.2 | 37.7 | 17.0 | 17.5 |
| HyDE [10] | 22.7 | 17.1 | 39.2 | 26.3 | 9.9 | 11.5 | 29.8 | 35.7 | 37.0 | **40.0** | 27.7 | 26.1 |
| Re2 | 29.6 | 24.3 | 47.9 | 33.1 | 10.8 | 12.2 | 29.4 | 34.7 | 34.6 | 37.1 | 30.5 | 28.3 |
| **Graph Retrieval** | | | | | | | | | | | | |
| LightRAG [12] | 24.2 | 15.7 | 19.3 | 12.0 | 11.0 | 10.2 | 31.4 | 32.0 | 34.3 | 35.4 | 24.0 | 21.1 |
| PathRAG [5] | *Out-of-time* | | 0.0 | 0.0 | 11.3 | 11.7 | 14.4 | 14.8 | 31.6 | 33.4 | 14.3 | 15.0 |
| HippoRAG 2 [14] | 28.8 | 21.6 | 38.2 | 26.8 | 8.5 | 9.8 | 27.0 | 32.4 | 32.9 | 35.5 | 27.1 | 25.2 |
| GAR [27] | 36.3 | 30.8 | 48.6 | 33.8 | 12.8 | 13.8 | 32.5 | 37.1 | 36.4 | 38.6 | 33.3 | 30.8 |
| **GraFine (Ours)** | **46.3** | **40.2** | **50.3** | **35.0** | **13.7** | **13.9** | **35.1** | **39.4** | **37.6** | 39.3 | **36.6** | **33.6** |

**Table 4: Overall Win Rates (%) of Baselines v.s. GraFine across Four Datasets and Average.**

| Baselines | ACL-OCL | | BSARD-G | | UltraDomain-agriculture | | UltraDomain-mix | | Average | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Baseline | **GraFine** | Baseline | **GraFine** | Baseline | **GraFine** | Baseline | **GraFine** | Baseline | **GraFine** |
| Vector Search | 45.6 | **53.3** | 42.3 | **57.2** | 43.0 | **55.0** | 40.8 | **57.7** | 42.9 | **55.8** |
| SPLADE | 48.2 | **51.7** | 27.0 | **72.5** | 43.0 | **57.0** | 45.4 | **54.6** | 40.9 | **59.0** |
| Contriever | 40.8 | **58.2** | 14.0 | **85.6** | 42.0 | **56.0** | 46.2 | **53.1** | 35.8 | **63.2** |
| HyDE | 44.1 | **55.4** | 48.6 | **51.4** | 44.0 | **56.0** | 41.5 | **58.5** | 44.5 | **55.3** |
| Re2 | 43.7 | **55.1** | 41.9 | **57.7** | 3.0 | **95.0** | 49.2 | **50.8** | 34.5 | **64.6** |
| LightRAG | 42.5 | **57.1** | 45.9 | **53.6** | 39.0 | **60.0** | 45.4 | **53.1** | 43.2 | **55.9** |
| PathRAG | *Out-of-time* | | 42.8 | **56.8** | 38.0 | **62.0** | 23.1 | **76.2** | 34.6 | **65.0** |
| HippoRAG 2 | 45.8 | **53.5** | 43.2 | **56.8** | 46.0 | **53.0** | 35.4 | **63.1** | 42.6 | **56.6** |
| GAR-RAG | **52.1** | 47.1 | 48.2 | **51.8** | 38.0 | **61.0** | 39.2 | **60.0** | 44.4 | **55.0** |

These results are also aligned with our discussion in Section 4.5. Specifically, under the same budget $b_{\max} = 100$, the runtime of GraFine on ACL-OCL is comparable to that of GAR and Re2. Given that ACL-OCL contains 11× more nodes, 134× more edges, and 12× more edges per node than SciFact-G, this result suggests that the overall graph size is not the primary bottleneck of our method. Moreover, on both ACL-OCL and SciFact-G, the latency of GraFine increases linearly as the budget grows. This observation supports our complexity analysis in Section 4.5, indicating that the additional linear algebra computations are not the dominant bottleneck.

*6.2.2 GraFine versus Conventional Interleaving Retrieval.* To demonstrate GraFine's efficiency over computationally intensive interleaving retrieval, we compare it against IRCoT + Vector Search [35] and ToG-2 [26] on SciFact-G. To examine QPT fairly, we use a locally hosted Gemma 3 (12B) via Ollama in TITAN and A100 GPUs. IRCoT is configured with a 2-step process, retrieving 5 nodes per step.

As shown in Table 5, IRCoT slightly improves over Vector Search in terms of accuracy, but incurs substantial latency due to iterative LLM inference. ToG-2 further amplifies this limitation, exhibiting significantly higher latency with degraded retrieval performance. In contrast, GraFine effectively overcomes these bottlenecks. Compared to IRCoT, our method reduces **query processing time by 42–58% while improving R@10 by 11.7%**, and achieves both **higher accuracy and around 10× faster latency** than ToG-2, validating it as a time-efficient alternative to interleaving retrieval methods.

**Table 5: Time efficiency on SciFact-G: GraFine vs. LLM interleaving retrieval methods.**

| Method | R@10 | QPT (TITAN) | QPT (A100) |
|---|---|---|---|
| IRCoT + Vector Search | 31.4 | 6.54 sec | 5.03 sec |
| ToG-2 | 12.3 | 32.3 sec | 20.7 sec |
| **GraFine (Ours)** | **35.1** | **3.77 sec** | **2.12 sec** |

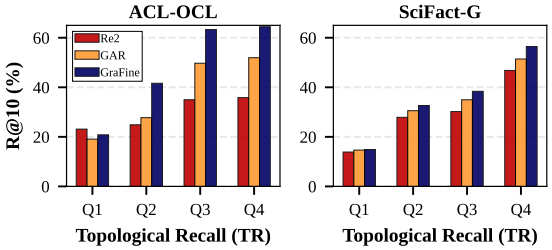



**Figure 5: Impact of Topological Recall (TR) on Retrieval Performance (R@10).**

## 6.3 Understanding GraFine

*6.3.1 Topological Recall (TR) Analysis.* In this section, we utilize the Topological Recall (TR) metric defined in Section 5.3.2 to analyze how effectively GraFine refines the retrieved node set. We validate whether TR effectively quantifies the topological proximity to oracle nodes and how our method exploits this proximity relative to baselines.

*Impact of Topological Proximity.* We first investigate the relationship between TR and the standard Recall (R@10) to understand how topological proximity translates to retrieval performance. Figure 5 illustrates the distribution of R@10 across TR quartiles (Q1–Q4). As expected, all methods demonstrate improved Recall with increasing TR, confirming that higher topological proximity creates

a favorable state for uncovering remaining oracle nodes through refinement. Critically, graph traversal methods (GraFine and GAR) exhibit steeper performance gains in the Q3-Q4 intervals compared to the vector-only baseline, Re2. This indicates that graph-based methods successfully exploit the graph structure to retrieve oracle nodes that remain unreachable through vector similarity alone. In particular, GraFine outperforms Re2 by a larger margin than GAR in the structurally difficult Q2 interval of ACL-OCL, demonstrating its ability to effectively navigate towards oracle nodes even when the initial search starts in a poorly connected region of the graph.

*Correlation Analysis.* To substantiate these observations, we examine which component of TR captures topological proximity to oracle nodes. Based on Corollary 1, we analyze the correlation between the two components–*Recall* and *MissTR*–against the *marginal recall gain* ($\Delta R = R@100_{total} - R@10_{vs}$ where $R@100_{total}$ is the recall after complete retrieval and $R@10_{vs}$ is the recall obtained from initial $O_v$). Here, $\Delta R$ quantifies the retriever's success in uncovering undiscovered oracle nodes during graph retrieval.

**Table 6: Correlation coefficient between two metrics and $\Delta R$.**

| Methods | ACL-OCL | | BSARD-G | | SciFact-G | |
|---|---|---|---|---|---|---|
| | Recall | MissTR | Recall | MissTR | Recall | MissTR |
| GAR | 0.48 | **0.50** | 0.46 | **0.79** | 0.12 | **0.65** |
| GraFine | 0.65 | **0.66** | 0.32 | **0.57** | 0.10 | **0.55** |

As shown in Table 6, MissTR consistently exhibits a stronger correlation with $\Delta R$ than standard Recall across all datasets and methods. While Recall reflects the success of the initial $O_v$ retrieval, it shows weak correlation with future discoveries. In contrast, the strong correlation in MissTR suggests that this component effectively captures the *refinement potential* of the current node set $N_{sel}$. These findings support our hypothesis that performance gains in graph-based methods are driven by their ability to navigate toward undiscovered oracle nodes through the graph topology.

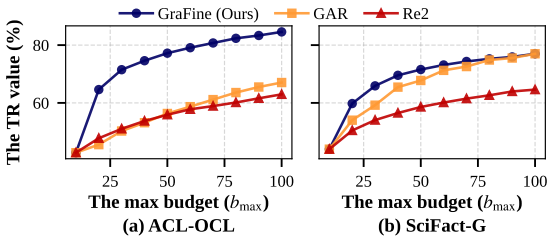



**Figure 6: Evolution of Topological Recall (TR) as a function of retrieval budget ($b_{max}$)**

*Evolution of TR in retrieval.* Figure 6 illustrates the evolution of TR as the retrieval budget $b_{max}$ increases. While all methods exhibit an upward trend, the rise in Re2 is largely a trivial consequence of the increasing Recall term within the decomposed TR equation (Corollary 1). More critically, we distinguish the trajectories of the graph traversal methods. Unlike the gradual ascent of GAR, GraFine demonstrates a rapid initial surge in TR. This sharp trajectory validates the efficacy of our proposed operators: SPX actively *steers* node selection towards oracle-rich neighborhoods in the early retrieval stages, while GSR effectively prioritizes candidates by interpreting their topological context. Consequently, these results provide strong empirical evidence that GraFine's mechanisms

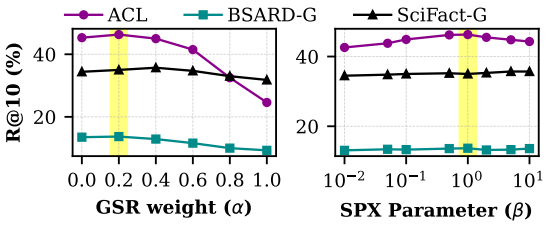



**Figure 7: R@10 sensitivity to GSR weight (left) and SPX parameter (right). Yellow bands mark the default values.**

support faster retrieval-time refinement toward high topological proximity than competing approaches.

*6.3.2 Sensitivity Analysis.* Figure 7 illustrates the sensitivity of GraFine's R@10 performance to variations in hyperparameters $\alpha$ and $\beta$ across three datasets. The results show consistently high performance across all datasets at our chosen settings of $\alpha = 0.2$ and $\beta = 1$, justifying our parameter selection. Conversely, we observe a performance degradation when $\alpha$ approaches 0 (i.e., relying solely on $O_m$ rather than $O_{gm}$) or when $\beta$ approaches extreme values of 0 or $\infty$ (i.e., using only $O_v$ or $O_g$ instead of $O_{vg}$). These findings demonstrate the contribution of both GSR and SPX to the overall effectiveness of our proposed method.

## 7 Related Work

Recently proposed retrieval–LLM interleaving methods for retrieval-time refinement [23, 26, 28, 33, 35], while effective for problems that go beyond single-step retrieval, rely on frequent LLM invocations, which incur substantial computational overhead and latency, making them impractical for Graph RAG on corpus graphs. Other recent graph retrieval methods aim to perform effective retrieval-time refinement by combining two or more operators from $\{O_v, O_g, O_m\}$ [5, 12, 14, 21]. However, these methods are fundamentally based on a sequential composition of operators and therefore inherit the limitations of those operators—the topology blindness of $O_m$ and the semantic blindness of $O_g$. Moreover, we observe that existing *fusion* approaches, such as G-Retriever [16], can also be formally expressed as compositions of our proposed operators $\{O_v, O_g, O_m\}$. Specifically, Examples 3 and 5 demonstrate how the PPR algorithm in HippoRAG 2 [14] and G-Retriever [16] can be represented within our taxonomy, respectively.

## 8 Conclusion

In this paper, we presented **GraFine**, a novel graph retrieval method designed to enable time-efficient retrieval-time refinement for Graph RAG on corpus graphs. Specifically, we identify two blind points of existing retrieval operations and overcome them by interleaving two novel fusion operators: **the Graph Smoothing Reranker (GSR)** for $O_{gm}$ and **Semantic Proximity eXpansion (SPX)** for $O_{vg}$. Extensive experiments across five corpus graph datasets demonstrate that GraFine outperforms state-of-the-art baselines by an average of **+9.9% in R@10 and +9.1% in nDCG@10.** Furthermore, compared to conventional LLM interleaving methods, our approach achieves a significant Pareto improvement, **reducing query processing time by 42–58% while simultaneously improving R@10 by 11.7%.**

## GenAI Usage Disclosure

Generative AI were used in a limited manner during the preparation of this work, including linguistic assistance during drafting, code debugging assistance during implementation, certain stages of ACL-OCL dataset construction (Section 5.2.1), and the LLM-as-a-Judge evaluation (Section 5.3.1). Except for the linguistic assistance and code debugging, all GenAI-assisted components are described in detail in the paper. No generative AI tools were used in any other part of the research or writing process.

## References


[1] Seonho An, Young-Yik Rhim, and Min-Soo Kim. 2025. GReX: A Graph Neural Network-Based Rerank-then-Expand Method for Detecting Conflicts Among Legal Articles in Korean Criminal Law. In *Proceedings of the Natural Legal Language Processing Workshop 2025*, Nikolaos Aletras, Ilias Chalkidis, Leslie Barrett, Cătălina Goanță, Daniel Preoțiuc-Pietro, and Gerasimos Spanakis (Eds.). Association for Computational Linguistics, Suzhou, China, 408–423. doi:10.18653/v1/2025.nllp-1.30

[2] Tu Ao, Yanhua Yu, Yuling Wang, Yang Deng, Zirui Guo, Liang Pang, Pinghui Wang, Tat-Seng Chua, Xiao Zhang, and Zhen Cai. 2025. Lightprof: A lightweight reasoning framework for large language model on knowledge graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 23424–23432.

[3] Vera Boteva, Demian Gholipour Ghalandari, Artem Sokolov, and Stefan Riezler. 2016. A Full-Text Learning to Rank Dataset for Medical Information Retrieval.. In *ECIR (Lecture Notes in Computer Science, Vol. 9626)*, Nicola Ferro, Fabio Crestani, Marie-Francine Moens, Josiane Mothe, Fabrizio Silvestri, Giorgio Maria Di Nunzio, Claudia Hauff, and Gianmaria Silvello (Eds.). Springer, 716–722. http://dblp.uni-trier.de/db/conf/ecir/ecir2016.html#BotevaGSR16

[4] Jake D Brutlag, Hilary Hutchinson, and Maria Stone. 2008. User preference and search engine latency. *JSM Proceedings, Qualtiy and Productivity Research Section* (2008).

[5] Boyu Chen, Zirui Guo, Zidan Yang, Yuluo Chen, Junze Chen, Zhenghao Liu, Chuan Shi, and Cheng Yang. 2025. PathRAG: Pruning Graph-based Retrieval Augmented Generation with Relational Paths. arXiv:2502.14902 [cs.CL] https://arxiv.org/abs/2502.14902

[6] Huan Chen, Gareth JF Jones, and Rob Brennan. 2024. An Examination of Embedding Methods for Entity Comparison in Text-Rich Knowledge Graphs. In *Proceedings of the Proceedings of the 32nd Irish Conference on Artificial Intelligence and Cognitive Science (AICS 2024). CEUR Workshop Proceedings.*

[7] Thomas Cook, Richard Osuagwu, Liman Tsatiashvili, Vrynsia Vrynsia, Koustav Ghosal, Maraim Masoud, and Riccardo Mattivi. 2025. Retrieval Augmented Generation (RAG) for Fintech: Agentic Design and Evaluation. *arXiv preprint arXiv:2510.25518* (2025).

[8] 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* (2024).

[9] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2288–2292.

[10] 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)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 1762–1777. doi:10.18653/v1/2023.acl-long.99

[11] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* 2, 1 (2023).

[12] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2025. LightRAG: Simple and Fast Retrieval-Augmented Generation. arXiv:2410.05779 [cs.IR] https://arxiv.org/abs/2410.05779

[13] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2025. LightRAG: Simple and Fast Retrieval-Augmented Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (Eds.). Association for Computational Linguistics, Suzhou, China, 10746–10761. doi:10.18653/v1/2025.findings-emnlp.568

[14] Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From RAG to Memory: Non-Parametric Continual Learning for Large Language Models. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=LWH8yn4HS2

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

[16] 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 (2024), 132876–132907.

[17] 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*, Luis Chiruzzo, Alan Ritter, and Lu Wang (Eds.). Association for Computational Linguistics, Albuquerque, New Mexico, 4145–4157. doi:10.18653/v1/2025.findings-naacl.232

[18] Yiqian Huang, Shiqi Zhang, and Xiaokui Xiao. 2025. Ket-rag: A cost-efficient multi-granular indexing framework for graph-rag. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 1003–1012.

[19] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Transactions on Machine Learning Research* (2022). https://openreview.net/forum?id=jKN1pXi7b0

[20] Xinke Jiang, Ruizhe Zhang, Yongxin Xu, Rihong Qiu, Yue Fang, Zhiyuan Wang, Jinyi Tang, Hongxin Ding, Xu Chu, Junfeng Zhao, and Yasha Wang. 2024. HyKGE: A Hypothesis Knowledge Graph Enhanced Framework for Accurate and Reliable Medical LLMs Responses. arXiv:2312.15883 [cs.CL] https://arxiv.org/abs/2312.15883

[21] Bernal Jimenez Gutierrez, 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 (2024), 59532–59569.

[22] Kaeun Kim, Ghazal Shams, and Kawon Kim. 2025. From Seconds to Sentiments: Differential Effects of Chatbot Response Latency on Customer Evaluations. *International Journal of Human–Computer Interaction* (2025), 1–17.

[23] Myeonghwa Lee, Seonho An, and Min-Soo Kim. 2024. PlanRAG: A Plan-then-Retrieval Augmented Generation for Generative Large Language Models as Decision Makers. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 6537–6555. doi:10.18653/v1/2024.naacl-long.364

[24] Mufei Li, Siqi Miao, and Pan Li. 2025. Simple is Effective: The Roles of Graphs and Large Language Models in Knowledge-Graph-Based Retrieval-Augmented Generation. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=JvkuZZ04O7

[25] Antoine Louis and Gerasimos Spanakis. 2022. A Statutory Article Retrieval Dataset in French. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 6789–6803. doi:10.18653/v1/2022.acl-long.468

[26] Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. 2025. Think-on-Graph 2.0: Deep and Faithful Large Language Model Reasoning with Knowledge-guided Retrieval Augmented Generation. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=oFBu7qaZpS

[27] Sean MacAvaney, Nicola Tonellotto, and Craig Macdonald. 2022. Adaptive Re-Ranking with a Corpus Graph. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (Atlanta, GA, USA) *(CIKM '22)*. Association for Computing Machinery, New York, NY, USA, 1491–1500. doi:10.1145/3511808.3557231

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

[29] 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. In *Proceedings of the ACM on Web Conference 2025* (Sydney NSW, Australia) *(WWW '25)*. Association for Computing Machinery, New York, NY, USA, 2366–2377. doi:10.1145/3696410.3714805

[30] Mandeep Rathee, Sean MacAvaney, and Avishek Anand. 2025. Guiding retrieval using llm-based listwise rankers. In *European Conference on Information Retrieval*. Springer, 230–246.

[31] Shaurya Rohatgi, Yanxia Qin, Benjamin Aw, Niranjana Unnithan, and Min-Yen Kan. 2023. The ACL OCL Corpus: Advancing Open Science in Computational Linguistics. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 10348–10361. doi:10.18653/v1/2023.emnlp-main.640

[32] Michael Shen, Muhammad Umar, Kiwan Maeng, G. Edward Suh, and Udit Gupta. 2024. Towards Understanding Systems Trade-offs in Retrieval-Augmented Generation Model Inference. arXiv:2412.11854 [cs.AR] https://arxiv.org/abs/2412.11854

[33] Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=nnVO1PvbTv

[34] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, Joaquin Vanschoren and Sai-Kit Yeung (Eds.). https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/65b9eea6e1cc6bb9f0cd2a47751a186f-Abstract-round2.html

[35] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 10014–10037. doi:10.18653/v1/2023.acl-long.557

[36] David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. 2020. Fact or Fiction: Verifying Scientific Claims. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 7534–7550. doi:10.18653/v1/2020.emnlp-main.609

[37] Jiasheng Zhang, Ali Maatouk, Jialin Chen, Ngoc Bui, Qianqian Xie, Leandros Tassiulas, Hua Xu, Jie Shao, and Rex Ying. 2025. LitFM: A Retrieval Augmented Structure-aware Foundation Model For Citation Graphs. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2* (Toronto ON, Canada) *(KDD '25)*. Association for Computing Machinery, New York, NY, USA, 3728–3739. doi:10.1145/3711896.3737028

[38] Xiangrong Zhu, Yuexiang Xie, Yi Liu, Yaliang Li, and Wei Hu. 2025. Knowledge Graph-Guided Retrieval Augmented Generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Luis Chiruzzo, Alan Ritter, and Lu Wang (Eds.). Association for Computational Linguistics, Albuquerque, New Mexico, 8912–8924. doi:10.18653/v1/2025.naacl-long.449