Structure Guided Retrieval-Augmented Generation for Factual Queries

Anonymous ACL submission

Abstract

Retrieval-Augmented Generation (RAG) has been proposed to mitigate hallucinations in large language models (LLMs), where generated outputs may be factually incorrect. However, existing RAG approaches predominantly rely on vector similarity for retrieval, which is prone to semantic noise and fails to ensure that generated responses fully satisfy the complex conditions specified by factual queries, often leading to incorrect answers. To address this challenge, we introduce a novel research problem, named **EXACT RETRIEVAL PROBLEM** (ERP). To the best of our knowledge, this is the first problem formulation that explicitly incorporates structural information into RAG for factual questions to satisfy all query conditions. For this novel problem, we propose **STRUCTURE GUIDED RETRIEVAL-AUGMENTED GENERATION** (SG-RAG)[1], which models the retrieval process as an embedding-based subgraph matching task, and uses the retrieved topological structures to guide the LLM to generate answers that meet all specified query conditions. To facilitate evaluation of ERP, we construct and publicly release **EXACT RETRIEVAL QUESTION ANSWERING** (ERQA)[2] , a large-scale dataset comprising 120,000 fact-oriented QA pairs, each involving complex conditions, spanning 20 diverse domains. The experimental results demonstrate that SG-RAG significantly outperforms strong baselines on ERQA, delivering absolute gains of 20.68–50.88 percentage points, corresponding to 34%–450% relative improvements across metrics, while maintaining reasonable computational overhead.

1 Introduction

LLMs suffer from a well-documented limitation: they often produce hallucinations that are fluent but factually incorrect (Huang et al., 2025). Such hallucinations have been observed in medical QA (Pal et al., 2023), legal drafting (Curran et al., 2023), and scientific writing (Sui et al., 2024), where factual errors may mislead users and undermine trust (Luo et al., 2024). RAG alleviates hallucinations by incorporating external knowledge, shows strong performance in QA and assistant tasks. Current RAG methods can be categorized into two paradigms: chunk-based RAG, represented by NaiveRAG (Lewis et al., 2020), and graph-based RAG, exemplified by GraphRAG (Edge et al., 2024). Both paradigms have been adopted in real-world systems such as Dify (Arai, 2024) and LangChain (Topsakal and Akinci, 2023).

However, in real-world applications, many fact-oriented queries require that answers satisfy all conditions in the query. As shown in Fig 1, in a medical QA scenario, a user may ask: *"Which disease commonly uses massage as adjuvant therapy, is prone to cause hypertension and adrenal incidentaloma, and requires differential diagnosis from subclinical Cushing's syndrome?"* This query imposes four

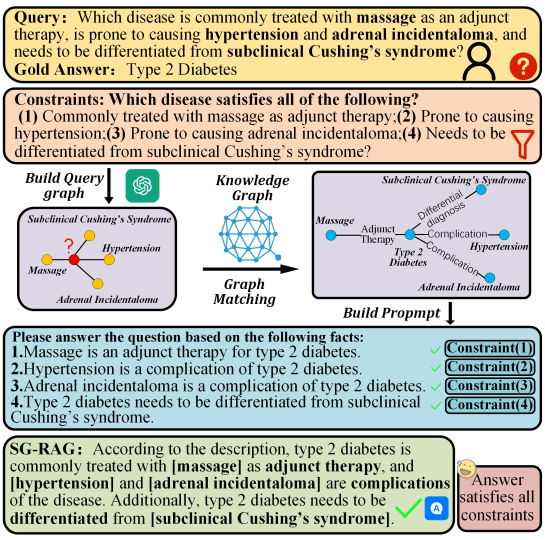

Figure 1: Multi-Condition QA Example.

[1]Code is publicly available at: https://anonymous.4open.science/r/SGRAG-23BB/

[2]Dataset is publicly available at: https://anonymous.4open.science/r/ERQA-0323/

distinct conditions on the target disease: (1) massage is used as adjuvant therapy; (2) it is prone to induce hypertension; (3) it tends to cause adrenal incidentaloma; (4) it requires differential diagnosis from subclinical Cushing's syndrome. Each condition represents a constraint, and collectively they define a complex query intent. Unlike single constraint queries, such multi-constraint queries demand that a correct answer satisfy all specified constraints. Current RAG methods such as NaiveRAG and GraphRAG often struggle to achieve this objective. The limitation lies in their reliance on semantic similarity to rank retrieval results and generate responses based on local text chunks or subgraphs from external knowledge sources. Thus, they fail to retrieve the full set of supporting information required to satisfy all the constraints in the query, leading to incomplete answers.

To address this gap, we define a novel research problem called **EXACT RETRIEVAL PROBLEM** (ERP). Given a user query involving multiple constraints, the goal of ERP is to retrieve information that precisely and comprehensively satisfies all specified conditions, from external knowledge sources, ensuring that the generated answer complies with every constraint in the query.

To solve ERP, we propose SG-RAG, **STRUCTURE GUIDED RETRIEVAL-AUGMENTED GENERATION**. Unlike existing RAG methods that rely on vector similarity, SG-RAG introduces a structure-driven retrieval mechanism that better respects the multi-constraint structure of the query. The core idea is to model knowledge retrieval as an embedding-based subgraph matching task, enabling the system to precisely and comprehensively retrieve information from external knowledge sources that satisfy the constraints specified in the query. The information of the retrieved subgraphs is then converted into prompts that guide the LLM in generating answers that satisfy all constraints. In the example of Figure 1, SG-RAG retrieves a subgraph that satisfies all query constraints and produces the correct answer, *Type 2 diabetes*.

To systematically evaluate the ability of SG-RAG to solve ERP, we construct and publicly release a large-scale benchmark dataset called ERQA (**EXACT RETRIEVAL QUESTION ANSWERING**). ERQA consists of three subsets built from diverse domains: (1) FB-ERQA, an English encyclopedic graph contributing 80,000 queries; (2) UD-ERQA, a multi-disciplinary English dataset built from textbooks covering 18 distinct domains, contributing 10,000 queries; and (3) CM-ERQA, a Chinese medical knowledge graph contributing 30,000 queries. Each query includes multiple constraints, with a ground-truth answer for evaluation. Based on ERQA, we compare SG-RAG with several RAG baselines. SG-RAG achieves significant and consistent gains across metrics, yielding 20.68–50.88 percentage-point absolute improvements, corresponding to 34% to over 450% relative gains, while maintaining reasonable computational efficiency. Our main contributions are as follows.

- To address the challenges observed in real-world applications, we propose a novel RAG query paradigm, called ERP.
- To solve ERP, we propose SG-RAG, a Structure Guided Retrieval-Augmented Generation method leveraging an embedding-based subgraph matching mechanism.
- To enable a systematic evaluation of SG-RAG, we construct and **publicly** release ERQA, a benchmark containing 120,000 factual QA pairs across 20 domains.
- On ERQA, SG-RAG outperforms strong baselines by 20.68–50.88 points, corresponding to 34% to over 450% relative gains across metrics.

2 Related Work

Retrieval-Augmented Generation was introduced by (Lewis et al., 2020) to improve factual QA by integrating vector similarity-based retrieval with the generation of language models. Subsequent research has progressed in two major directions: (i) query-controlled semantic alignment and (ii) structure-aware retrieval with explicit knowledge modeling. In the first line, HyDE (Gao et al., 2023) introduced hypothetical answer generation for backward retrieval. MEMORAG (Qian et al., 2024) incorporated a memory module for multi-turn coherence, while RQ-RA (Chan et al., 2024) improved multi-hop QA via structured query rewriting. In the second line, GraphRAG (Edge et al., 2024) pioneered entity graph integration and community-based paragraph retrieval. LightRAG (Guo et al., 2024) reduced the cost of graph construction through a simplified structure. HopRAG (Liu et al., 2025) introduced multi-hop subgraphs for long-range access, GRAG (Xu et al., 2025a) focused on dynamic graph evolution, and G-Refer (Li et al., 2025) employed dominant embeddings with contrastive learning to enforce structural consistency. For domain-specific QA,

MedRAG (Zhao et al., 2025) integrated medical ontologies, HippoRAG (Jimenez Gutierrez et al., 2024) mimicked hippocampal memory encoding, and AMAR (Xu et al., 2025b) enabled multi-view retrieval of entities, attributes, and paths. In summary, existing RAG methods effectively mitigate LLM hallucination, but their reliance on vector similarity limits their ability to precisely handle real-world queries with complex conditions. **Subgraph Matching** techniques fall into join-based and backtracking-based categories. Join-based approaches include BiGJoin and its variants (Ammar et al., 2018) achieving worst-case optimal joins even in dynamic graphs, fractional-cover-based joins (Ngo et al., 2018), and distributed matching via Timely Dataflow (Lai et al., 2019). Systems such as EmptyHeaded (Aberger et al., 2017) utilize SIMD joins with high-level queries, while cost-based optimizers refine join orders (Mhedhbi and Salihoglu, 2019). Hybrid methods such as RapidMatch (Sun et al., 2020) merge joins with exploration; SEED (Lai et al., 2016) applies clique/star units with bushy joins; and TwinTwigJoin (Lai et al., 2015) achieves optimality on MapReduce. The visual and partial matching are addressed by FERRARI (Wang et al., 2020) and PANDA (Xie et al., 2017), respectively. Backtracking-based methods include VF3 (Carletti et al., 2015) and VF2 Plus (Carletti et al., 2017) for dense and sparse graphs, QuickSI (Shang et al., 2008) with optimized orderings, and redundancy reduction strategies via Cartesian product postponement (Bi et al., 2016) or algebraic pruning (He and Singh, 2008). In addition, CECI (Bhattarai et al., 2019) leverages embedding clusters, while parallelized exploration (Sun et al., 2012) scales to billion-node graphs. Pruning-free methods (Bonnici et al., 2013) also perform well in biological networks. Recently, GNN-PE (Ye et al., 2024) enables exact subgraph matching using dominant path embeddings and cost-aware decomposition. Although these methods improve scalability and efficiency through pruning and join optimization, they typically assume fully labeled query graphs. However, in RAG scenarios, the query target is often unknown, making such methods difficult to apply.

3 Preliminaries and Problem Definition

In this section, we present the core notations and formal definitions. Symbols are summarized in Appendix.A.

Definition. A graph G is a quadruple (V, E, κ, L), where: V is a set of vertices, each $v_i \in V$ represented as $(\mathrm{id}(v_i), \ell(v_i), \xi(v_i))$, with unique id, semantic label, and textual description. E is a set of directed edges $e_{ij} = (v_i, v_j)$, each encoded as $(\mathrm{id}(e_{ij}), v_i, v_j, \xi(e_{ij}))$. $\kappa : V \times V \to E$ maps a pair of vertices to their edge. L is a labeling function assigning each vertex $v_i \in V$ a label $\ell(v_i)$. Given two graphs G_A and G_B, if they are isomorphic, we write $G_A \equiv G_B$ (Han et al., 2013).

Problem Definition. EXACT RETRIEVAL PROBLEM (ERP): Given: a natural language query q^o with a set of constraints $C = \{c_1, ..., c_k\}$ $(k \geq 2)$; a knowledge corpus $K = \{k_1, ..., k_n\}$ and a large language model Λ, ERP aims to retrieve from K the most accurate knowledge that satisfies all the constraints in C, and use it to prompt the Λ to generate a factual answer. Each constraint c_i typically corresponds to an entity related to the target answer. When such precise information cannot be retrieved, the goal is to identify relevant evidence to guide the Λ in generating a reliable answer.

4 Method

4.1 System Overview

To address ERP, q^o can be structured into a query graph q, and K can be represented as a knowledge graph G. This enables ERP to be transformed into a subgraph matching task: Find a subgraph $g \subseteq G$ such that $g \equiv q$, ensuring that all query constraints are structurally satisfied. However, since the subgraph isomorphism is **NP-complete**, applying it effectively in the RAG setting poses a significant challenge. To overcome this challenge, we propose SG-RAG. As shown in Figure 2, we formalize SG-RAG as a tuple:

$$\text{SG-RAG} = (\epsilon, \mathcal{M}, \phi, \psi, \sigma, \delta, \gamma) \quad (1)$$

where ϵ: document structuring module that converts knowledge corpus K into a knowledge graph G; $\mathcal{M}$: GNN model for generating path dominance embeddings; ϕ: index construction module that builds R*-Tree I_l over path embeddings; ψ: query graph construction module that extracts, normalizes, completes and decomposes q^o; σ: path-level retrieval module using I_l; δ: subgraph assembly module that forms exact subgraphs; γ: answer generation module that prompts LLM with retrieved subgraphs. The overall system execution is defined:

$$\text{SG-RAG}(q^o; K) = \gamma(q^o, \delta(\psi(q^o)), \sigma(\phi(\mathcal{M}, \epsilon(K)))) \quad (2)$$

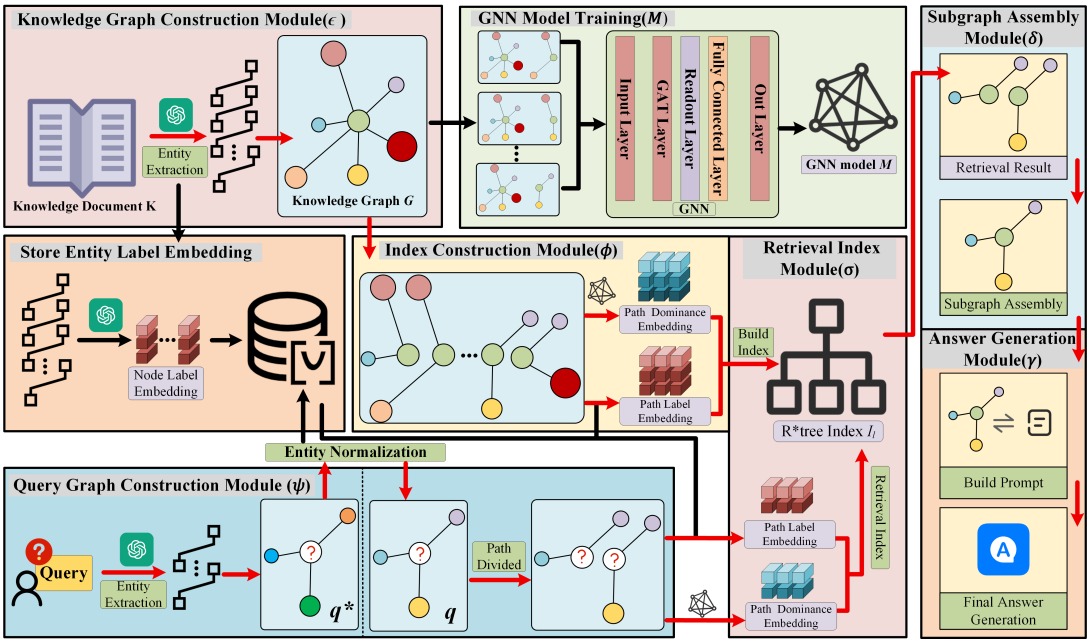

Figure 2: SG-RAG system architecture for structure guided retrieval and answer generation.

SG-RAG operates in three stages, as illustrated in Algorithm 1: **Index Construction** (Lines 1–6): the document corpus K is converted into a knowledge graph G via ϵ. Label embeddings of entity and path $o_0(v_i)$ and $o_0(p_z)$ are calculated, and the GNN model $\mathcal{M}$ is trained. The path dominance embeddings $o(p_z)$ are computed and indexed by ϕ. **Index-based Retrieval** (Lines 7–16): the user query q^o is parsed into a query graph q^* with label embeddings $o_0(q_i)$. Entity normalization, path decomposition, and label completion are performed by ψ. Each completed query path set $Q' \in P$ is embedded in $\mathcal{M}$, and exact path candidates are retrieved using σ. **Answer Generation** (Lines 17–27): subgraphs are assembled by δ, and used by γ to generate the answer a based on the best available subgraph.

4.2 Index Construction

This stage corresponds to the components ϵ, $\mathcal{M}$, and ϕ in the formal definition of the SG-RAG.

Generation of the Node and Path Label Embedding. SG-RAG first converts the raw corpus K into a structured knowledge graph G, where semantically meaningful entities are represented as nodes and explicit relations as edges (see Appendix.B for details). The node label embeddings are then computed using a pre-trained LLM. For each node $v_i \in V(G)$ with the label $\ell(v_i)$, the embedding is defined as $o_0(v_i) = \text{LLM}(\ell(v_i)) \in \mathbb{R}^F$. To represent the semantics of the path, we define the label embedding of a path $p_z = [v_1, v_2, \ldots, v_k]$ as the concatenation of its embeddings of the constituent node: $o_0(p_z) = [o_0(v_1), o_0(v_2), \ldots, o_0(v_k)] \in \mathbb{R}^{k \cdot F}$. It preserves both semantic content and node order for alignment during path-level retrieval.

Algorithm 1 SG-RAG Framework

Require: External documents K, User query q^o
Ensure: Answer a
1: Extract knowledge graph $G \leftarrow \epsilon(K)$
2: Generate label embeddings $o_0(v_i)$ for all $v_i \in V(G)$ via LLM
3: Generate path label embeddings $o_0(p_z)$ for all paths p_z of length l
4: Train GNN model $\mathcal{M}$
5: Compute path dominance embeddings $o(p_z)$ using $\mathcal{M}$
6: Build R*-Tree index I_l over $o_0(p_z)$ and $o(p_z)$
7: Extract query graph q^* from q^o and generate $o_0(q_i)$ for all $q_i \in V(q^*)$ via LLM
8: Normalize entities in q^* to obtain q
9: Decompose q into path set Q of length l
10: Compute $o_0(p_q)$ for each $p_q \in Q$
11: Predict possible labels using I_l
12: Complete Q into fully labeled path sets P
13: $S \leftarrow \emptyset$
14: **for** each query path set $Q' \in P$ **do**
15: Compute $o(p_q)$ for each $p_q \in Q'$ using $\mathcal{M}$
16: Retrieve cand_list
17: Assemble g from cand_list, add to S
18: **end for**
19: **if** $S \neq \emptyset$ **then**
20: Generate a using information of $g \in S$ as prompt
21: **else**
22: Construct fallback subgraph g'' by retrieving 1-hop neighbors of known entities in q
23: Generate a using information of g'' as prompt
24: **end if**
25: **return** a

Training of the Dominant Embedding Model. To enable exact subgraph matching, SG-RAG introduces a dominant embedding mechanism based on GNN. This mechanism learns structural representations of each node and its surrounding subgraph. For each node v_i, a 1-hop star subgraph g_{v_i} is constructed and encoded using a Graph Attention Network (GAT)-based architecture. The final dominant embedding $o(v_i)$ captures the topological context around v_i. To ensure structural containment, we enforce a dominance constraint: for any suitable substructure $s_{v_i} \subset g_{v_i}$, we require $o(s_{v_i}) \preceq o(g_{v_i})$. To implement this constraint during training, we introduce the following loss function:

$$\mathcal{L} = \sum \|\max(0, o(g_{v_i}) - o(s_{v_i}))\|_2^2 \tag{3}$$

This loss penalizes any violation of the dominance condition, encouraging the GNN to learn embeddings where substructures are embedded in semantically smaller regions than their supersets. For a path $p_z = [v_1, \ldots, v_k]$, the dominant embedding at the path level is computed as the concatenation $o(p_z) = [o(v_1), \ldots, o(v_k)]$. These embeddings are used for pruning and matching: a query path p_q is considered a match if $o(p_q) \preceq o(p_z)$ in all dimensions, indicating that the candidate structurally contains the query. This embedding formulation enables efficient structure-aligned filtering in the retrieval phase. See Appendix.C for full training details and architectural illustration.

Construction of the Path Index. To enable path-based subgraph retrieval, SG-RAG constructs an R*-Tree index over path embeddings.

$$I_l = \text{R*-Tree}\left(\{o_0(p_z),\ o(p_z) \mid |p_z| = l\}\right), \tag{4}$$

where each path p_z of length l is encoded by both its semantic embedding $o_0(p_z)$ and its structural embedding $o(p_z)$. These embeddings are derived as follows: $o_0(p_z)$ is the concatenation of the node label embeddings to capture the semantics of the path, while $o(p_z)$ is formed by concatenating the dominant embeddings of all nodes along the path p_z, where the dominant embedding of each node is calculated using the pre-trained GNN model $\mathcal{M}$. The resulting embedding pairs $(o_0(p_z), o(p_z))$ are indexed for efficient retrieval. Each index node stores different content based on type: **Leaf node:** stores $o_0(p_z)$ and $o(p_z)$; **Non-leaf node:** stores minimum bounding rectangles ($MBRs$) over $o_0(p_z)$ and $o(p_z)$ for semantic and structural filtering, respectively. At query time, the index is traversed in a heap-based best-first manner. Nodes whose $MBRs$ do not overlap with the query embedding region are pruned early, accelerating candidate filtering.

4.3 Index-Based Retrieval

This stage corresponds to the components ψ and σ in the formal definition of the SG-RAG.

Query Graph Extraction and Entity Normalization. As part of the ψ module, SG-RAG first transforms a query in natural language q^o into a structured query graph q^* using an LLM to extract entities and relations. It also computes the label embeddings $o_0(q_i^*) = \text{LLM}(\ell(q_i^*)) \in \mathbb{R}^F$ for each query node q_i^*. To ensure consistency with the knowledge graph G, SG-RAG performs entity normalization using FAISS (Douze et al., 2024): for each q_i^*, the most semantically similar node $q_i \in V(G)$ is retrieved and used to replace q_i^*, producing a normalized query graph q. Prompt and normalization algorithm are in Appendix.D.

Path Decomposition and Label Completion. To enable path-level retrieval, SG-RAG decomposes the query graph q into a set of linear query paths and completes unknown node labels to produce fully specified path sets P. We employ a cost-aware decomposition algorithm to iteratively select query paths of length l that minimize edge overlap and retrieval cost. Each path is scored using a degree-based path weight and the optimal path set Q is selected to fully cover $E(q)$. For paths containing unknown nodes, SG-RAG performs wildcard-based label completion by traversing the R*-Tree index I_l to identify candidate labels, generating a mapping U from unknown nodes to candidate label sets. All possible label combinations from U are instantiated in Q to form completed query path sets $P = \{Q'_1, Q'_2, \ldots, Q'_n\}$ for downstream matching. The complete procedures are in Appendix.E.

Path-level Retrieval. The module σ performs path-level retrieval on the I_l for each fully labeled query path $p_q \in P$. Each query path is encoded with: a semantic embedding $o_0(p_q)$ through the LLM; a structural embedding $o(p_q)$ via the GNN model $\mathcal{M}$. SG-RAG uses a heap-based best-first traversal strategy over I_l. During traversal: **At Non-leaf nodes**, for each query path p_q and each child node N_i, the system checks two constraints: **Semantic constraint**: whether $o_0(p_q)$ intersects with the label MBR $\text{MBR}_0(N_i)$; **Structural constraint**:

whether the dominance region

$$\mathrm{DR}(o(p_q)) = \{z \in \mathbb{R}^d \mid o(p_q)[i] \leq z[i], \forall i\} \quad (5)$$

overlaps with $\mathrm{MBR}(N_i)$. Only when both constraints are satisfied is p_q forwarded to N_i. **At leaf nodes**, for each stored path p_z, it is added to the result based on **Exact match**: if $o_0(p_q) = o_0(p_z)$ and $o(p_q) \preceq o(p_z)$. This dual filtering strategy ensures both semantic alignment and structural inclusion. All valid candidates are accumulated in p_q.cand_list. The algorithm is in Appendix.L.

4.4 Answer Generation

Subgraph Assembly. The module δ composes candidates for full subgraphs from path-level matches to produce a set: exact matches S that are isomorphic to the query graph. For exact matches, SG-RAG enumerates all combinations of candidate paths from p_q.cand_list and verifies whether they can be merged into a conflict-free subgraph. This process enables SG-RAG to reconstruct structurally consistent subgraphs that satisfy all query constraints. The assembly logic is in Appendix.M.

Generate Answer. The module γ generates answers based on matched subgraphs and supports two strategies. If the exact match set S is nonempty, SG-RAG extracts semantic labels and descriptions from all subgraphs $g \in S$ and combines them with the original query q^o to construct a structured prompt for the LLM, ensuring all constraints are satisfied. When exact subgraphs are not found, SG-RAG falls back to building a 1-hop neighborhood subgraph g'' around entities mentioned in q^o as the context of last resort. This ensures SG-RAG can still produce informative responses. The prompt construction process is in Appendix.I.

4.5 Time Complexity Analysis

We analyze the time complexity of the SG-RAG retrieval phase, which consists of three components: entity normalization, unknown label completion, and exact subgraph matching. The overall retrieval complexity is the following.

$$O\Bigg(|V(q)| \cdot \log N + \sum_{i=0}^{h} |Q_u| \cdot f^{\,h-i+1} \cdot (1 - PP_i) + \Big(\prod_{j=1}^{u} t_j\Big) \cdot \sum_{i=0}^{h} |Q| \cdot f^{\,h-i+1} \cdot (1 - PP_i)\Bigg). \quad (6)$$

where $|V(q)|$ is the number of nodes in the query graph, N is the number of entities in the knowledge graph, u is the number of unknown nodes, t_j is the candidate label count for node j, and h, f, PP_i denote the height, fan-out, and pruning ratio of the R*-Tree. In practice, most queries involve only one unknown node with a small number of candidates ($u = 1, t_1 \leq 10$), simplifying the complexity to:

$$O\Bigg(|V(q)| \cdot \log N + (t_1 + 1) \cdot \sum_{i=0}^{h} |Q| \cdot f^{\,h-i+1} \cdot (1 - PP_i)\Bigg) \quad (7)$$

This enables near-linear scalability, with modular execution and efficient index pruning ensuring low overhead. See Appendix.F for proof details.

5 Experimental Study

5.1 Dataset

Since most existing open RAG and QA benchmarks target single constraint queries (e.g., Natural Questions (NQ) (Kwiatkowski et al., 2019)) and rarely verify constraint satisfaction with verifiable evidence, we construct and publicly release ERQA to systematically evaluate SG-RAG on ERP. ERQA comprises three subsets: FB-ERQA contains 80,000 English QA pairs derived from FB15K-237 with 6.1 constraints per query on average; UD-ERQA contains 10,000 QA pairs from a cross-domain academic corpus spanning 18 disciplines with 4.7 constraints per query on average; CM-ERQA contains 30,000 Chinese QA pairs from the CPubMed-KG (Zhang et al., 2025) with 5.4 constraints per query on average. Detailed statistics, example queries, and the question-generation prompt are provided in Appendix.G and Appendix.J. We evaluate SG-RAG on ERQA for ERP and additionally on NQ dataset to verify its generalization to single-constraint queries.

5.2 Experimental Setting

Environment. All experiments were conducted on a local workstation running Ubuntu 22.04 LTS. The hardware configuration includes a 16 core, 32 thread Intel Core™ CPU and an NVIDIA GeForce RTX 4060 GPU (driver version 560.94, CUDA 12.6). **Model Configuration.** During dataset construction, we employed the `GLM-4-Flash` to generate structured queries. For all vector embedding tasks, we used `text-embedding-3-small` to maintain consistency between methods. We fix the model and the text preprocessing pipeline. In the answer generation stage, we adopted `GPT-4o` with a context window fixed at $1,200$ tokens

Table 1: Performance Comparison of SG-RAG and Baseline Methods on ERQA Subsets

	FB-ERQA					CM-ERQA					UD-ERQA				
	Hit@1	Precision	Recall	F1	Emp.S	Hit@1	Precision	Recall	F1	Emp.S	Hit@1	Precision	Recall	F1	Emp.S
GPT-5.1	19.1%	17.78%	19.23%	18%	0.412	6.5%	15.3%	6.5%	9%	0.192	10.02%	8.38%	11.06%	10%	0.287
NaiveRAG	14.8%	14.53%	15.06%	15%	0.781	14.2%	8.23%	17.24%	11%	0.564	14.41%	12.19%	15.66%	14%	0.681
RAPTOR	61.05%	60.55%	61.14%	61%	0.128	9.5%	8.73%	9.5%	9%	0.095	10.65%	9.73%	11.38%	10%	0.115
GraphRAG	61.8%	61.72%	61.83%	62%	1.432	20.2%	12.71%	22.28%	16%	1.007	18.79%	14.14%	20.04%	17%	1.204
LightRAG	20.3%	20.02%	20.42%	20%	0.934	14.1%	8.29%	17.0%	11%	0.734	17.64%	14.03%	18.79%	16%	0.795
SG-RAG	**82.48%**	**82.47%**	**82.51%**	**82%**	**1.855**	**61.1%**	**60.47%**	**61.2%**	**61%**	**1.641**	**61.83%**	**65.02%**	**67.49%**	**66%**	**1.611**

per query. **Baseline Methods.** To comprehensively evaluate the generation performance of SG-RAG, we compare it against five representative retrieval-augmented or generation-based baselines: NaiveRAG (Lewis et al., 2020), RAPTOR (Sarthi et al., 2024), GraphRAG (Edge et al., 2024), LightRAG (Guo et al., 2024), GPT-5.1. We additionally design two naive baselines to isolate the effect of structure guided exact retrieval; a constraint wise evidence union alternative can be strong, but we omit it because its performance is sensitive to evidence aggregation under a fixed context budget. *Entity-based Retrieval* returns chunks that mention any query entity. *2-hop Graph Retrieval* expands each query entity to its two-hop neighbors. **Evaluation Metrics.** We adopt a hybrid evaluation that combines objective metrics and subjective scoring to assess the accuracy and reasoning quality of the answer. We use four widely adopted metrics: Precision; Recall; F1 Score; Hit@1 as the objective metrics. We adopt *Empowerment Score* (Emp.S) as (Guo et al., 2024), a subjective metric, to evaluate the answers' reasoning quality (details in Appendix.K).

Table 2: Performance under Varying Constraints

	4 Constraints		5 Constraints		6 Constraints	
	Recall	Hit@1	Recall	Hit@1	Recall	Hit@1
GPT-5.1	9.19%	9.02%	11.01%	10.82%	9.07%	9.01%
NaiveRAG	16.06%	15.18%	16.20%	15.24%	15.41%	14.66%
RAPTOR	29.55%	29.14%	29.50%	29.31%	28.65%	28.38%
GraphRAG	37.66%	36.91%	37.58%	36.12%	36.29%	36.17%
LightRAG	21.52%	20.03%	17.89%	17.11%	17.54%	17.04%
SG-RAG	**72.14%**	**71.22%**	**69.10%**	**68.36%**	**70.48%**	**69.1%**

5.3 Experimental Results Analysis

Performance Analysis. Overall. As shown in Table 1, SG-RAG consistently outperforms strong baselines for each metric in all ERQA subsets. For Hit@1, SG-RAG achieves gains of 34% (FB-ERQA), 202% (CM-ERQA), and 229% (UD-ERQA), demonstrating a much higher probability of retrieving the correct answer on the first attempt. For Recall, improvements reach 35%, 175%, and 237%, indicating substantially stronger coverage of relevant knowledge. For F1, increases of 34% (FB-ERQA), 281% (CM-ERQA), and 282% (UD-ERQA) show that SG-RAG achieves a more balanced trade-off between precision and recall. Table 1 also reports the subjective evaluation results (Emp.S) on all subsets. We use GPT-5.2 in providing Emp.S. SG-RAG achieves the highest scores across all subsets, outperforming GraphRAG by nearly 0.5 and RAPTOR by more than 15×, indicating superior reasoning and explanatory quality by Emp.S. **Robustness.** As shown in Table 2, even under six-constraint queries, SG-RAG delivers a Hit@1 over 69%, representing a 91% improvement over GraphRAG, confirming its robustness in complex retrieval scenarios. **Ablation.** To verify that the gains of SG-RAG stem from structure-guided exact retrieval, we conduct a progressive comparison on CM-ERQA against naive variants spanning from purely semantic retrieval to shallow neighborhood expansion, including NaiveRAG, Entity-based Retrieval, LightRAG, and 2-hop graph retrieval. As shown in Table 3, enlarging the retrieval scope brings only limited improvements, and these naive methods still remain in a low Precision and low F1 regime overall. In contrast, SG-RAG consistently achieves substantially higher Precision, Recall, and F1 than the naive baseline, indicating that entity co-occurrence signals and local neighborhoods are insufficient

Table 3: Performance Comparison with Naive Baselines

Method	Precision	Recall	F1
NaiveRAG	8.23%	17.24%	11%
Entity-based Retrieval	9.31%	16.84%	12%
LightRAG (1-hop)	8.29%	17.00%	11%
2-hop Graph Retrieval	8.71%	20.10%	12%
SG-RAG	**60.47%**	**61.20%**	**61%**

Table 4: Performance on the NQ Dataset

Method	Precision	Recall	F1	Hit@1
NaiveRAG	79.51%	80.27%	79.89%	79.51%
GraphRAG	85.92%	87.88%	86.89%	85.92%
LightRAG	88.41%	89.05%	88.73%	88.41%
SG-RAG	**89.33%**	**89.49%**	**89.41%**	**89.33%**

for satisfying multi-constraint queries. **Generalization.** In addition, although SG-RAG is designed for multi-constraint queries, we further evaluate its robustness on single constraint queries using the NQ dataset, where queries can be represented as minimal graphs with two nodes and one edge. As shown in Table 4, SG-RAG achieves the best performance, reaching 89.33% in Hit@1 and 89.41% in F1, slightly surpassing LightRAG by 0.92 and 0.68 points, respectively, and outperforming GraphRAG and NaiveRAG by 3.41 and 9.82 points in Hit@1. These results indicate that SG-RAG remains stable and competitive even when the query contains one constraint. **Case Study.** We include a case study, details are in Appendix.H.

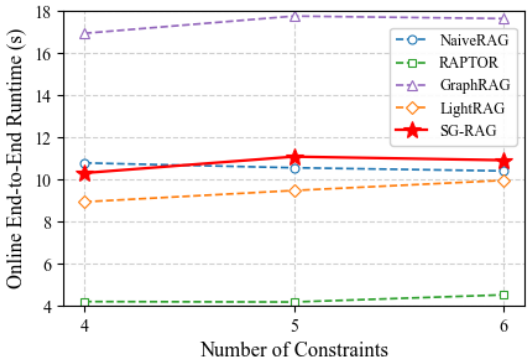

Figure 3: Online end-to-end run time comparison.

Efficiency Analysis. **Online.** Figure 3 shows the end-to-end run times of SG-RAG and the baselines under different levels of constraint. SG-RAG remains close to NaiveRAG and significantly outperforms GraphRAG. RAPTOR is the fastest, but its generation quality is the lowest, while SG-RAG offers a better balance between runtime and response quality. **Offline.** We further measure the index construction time of SG-RAG and other structure-aware RAG systems. As shown in Table 5, the offline construction time is approximately 8h for GraphRAG, 10h for LightRAG, and 11h for SG-RAG on CM-ERQA, indicating only a marginal overhead of about 1 hour over LightRAG.

Table 5: Offline index construction time across datasets.

Dataset	GraphRAG	LightRAG	SG-RAG
FB- ERQA	10h	13h	14h
UD-ERQA	21h	23h	28h
CM-ERQA	8h	10h	11h

5.4 Sensitivity Study

We analyze two key factors in the SG-RAG pipeline: path length l and LLM selection. **Path length** l affects query decomposition for retrieval. To ensure edge coverage, l should satisfy: $l \in \left[\left\lceil \frac{D}{2} \right\rceil, D\right]$, D is the diameter of query graph q. In ERQA most queries have $D = 2$, allowing $l = 1$ and $l = 2$. As shown in Table 6, accuracy remains nearly identical in both settings, since SG-RAG reconstructs the full subgraph prompts after matching. **LLM Selection.** On FB-ERQA, we compare

Table 6: Performance of Different Path Length l

	Hit@1	Precision	Recall	F1
l=1	71.12%	71.14%	71.28%	71.21%
l=2	70.45%	70.12%	70.31%	70.21%

six generation models in identical settings, covering representative flagship LLMs for both English and Chinese. As shown in Table 7, SG-RAG maintains consistent quality across different LLMs, with only marginal differences in both objective and subjective metrics, indicating high adaptability.

Table 7: Performance of Different Generation Models

Model	Recall	Hit@1	Precision	Emp.S
GPT-5.1	80.19%	79.32%	80.11%	2.01
GPT-4o-mini	79.86%	79.21%	79.64%	1.75
GLM-4V	80.22%	80.03%	80.17%	1.82
GLM-4-Flash	78.94%	78.32%	78.64%	1.73
Gemini-2.5	80.10%	79.56%	80.04%	1.88
Qwen-3	79.95%	79.12%	79.80%	1.81

6 Conclusion

We introduce a novel research problem ERP in RAG for factual queries with multiple conditions. To solve it, we propose SG-RAG, a structure guided RAG algorithm based on subgraph matching with path embeddings. We construct a large-scale benchmark ERQA for systematic ERP evaluation across domains. Experiments demonstrate that SG-RAG consistently outperforms strong baselines by large margins across metrics.

Limitations

SG-RAG is a multi-stage pipeline; errors in query graph extraction, entity normalization, or label completion may propagate to downstream retrieval and generation, potentially reducing reliability in noisy real-world inputs. In particular, parts of the pipeline rely on LLM-based information extraction during knowledge graph construction (e.g., entity/relation extraction), which can introduce upstream inaccuracies that are difficult to fully diagnose and may affect subsequent indexing and retrieval. In addition, our evaluation focuses on English and Chinese LLMs; we do not study other languages (e.g., Japanese, French, or German), and the observed trends may not directly transfer to those settings. Finally, while SG-RAG is designed as a general framework, we do not explore domain-specific optimizations; tailoring components such as entity normalization, indexing granularity, or prompt design to particular vertical domains may yield further gains beyond what is reported here.

References

Christopher R Aberger, Andrew Lamb, Susan Tu, Andres Nötzli, Kunle Olukotun, and Christopher Ré. 2017. Emptyheaded: A relational engine for graph processing. *ACM Transactions on Database Systems (TODS)*, 42(4):1–44.

Khaled Ammar, Frank McSherry, Semih Salihoglu, and Manas Joglekar. 2018. Distributed evaluation of subgraph queries using worstcase optimal lowmemory dataflows. *arXiv preprint arXiv:1802.03760*.

Kohei Arai. 2024. Design of on-premises version of rag with ai agent for framework selection together with dify and dsl as well as ollama for llm. *International Journal of Advanced Computer Science & Applications*, 15(12).

Bibek Bhattarai, Hang Liu, and H Howie Huang. 2019. Ceci: Compact embedding cluster index for scalable subgraph matching. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1447–1462.

Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. Efficient subgraph matching by postponing cartesian products. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1199–1214.

Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis Shasha, and Alfredo Ferro. 2013. A subgraph isomorphism algorithm and its application to biochemical data. *BMC bioinformatics*, 14(Suppl 7):S13.

Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. 2017. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):804–818.

Vincenzo Carletti, Pasquale Foggia, and Mario Vento. 2015. Vf2 plus: An improved version of vf2 for biological graphs. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 168–177. Springer.

Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. Rq-rag: Learning to refine queries for retrieval augmented generation. *arXiv preprint arXiv:2404.00610*.

Shawn Curran, Sam Lansley, and Oliver Bethell. 2023. Hallucination is the last thing you need. *arXiv preprint arXiv:2306.11520*.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281*.

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*.

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*.

Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. 2013. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, page 337–348, New York, NY, USA. Association for Computing Machinery.

Huahai He and Ambuj K Singh. 2008. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 405–418.

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

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*, 37:59532–59569.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and 1 others. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Longbin Lai, Lu Qin, Xuemin Lin, and Lijun Chang. 2015. Scalable subgraph enumeration in mapreduce. *Proceedings of the VLDB Endowment*, 8(10):2150–8097.

Longbin Lai, Lu Qin, Xuemin Lin, Ying Zhang, Lijun Chang, and Shiyu Yang. 2016. Scalable distributed subgraph enumeration. *Proceedings of the VLDB Endowment*, 10(3):217–228.

Longbin Lai, Zhu Qing, Zhengyi Yang, Xin Jin, Zhengmin Lai, Ran Wang, Kongzhang Hao, Xuemin Lin, Lu Qin, Wenjie Zhang, and 1 others. 2019. Distributed subgraph matching on timely dataflow. *Proceedings of the VLDB Endowment*, 12(10):1099–1112.

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.

Yuhan Li, Xinni Zhang, Linhao Luo, Heng Chang, Yuxiang Ren, Irwin King, and Jia Li. 2025. G-refer: Graph retrieval-augmented large language model for explainable recommendation. In *Proceedings of the ACM on Web Conference 2025*, pages 240–251.

Hao Liu, Zhengren Wang, Xi Chen, Zhiyu Li, Feiyu Xiong, Qinhan Yu, and Wentao Zhang. 2025. Hoprag: Multi-hop reasoning for logic-aware retrieval-augmented generation. *arXiv preprint arXiv:2502.12442*.

Junliang Luo, Tianyu Li, Di Wu, Michael Jenkin, Steve Liu, and Gregory Dudek. 2024. Hallucination detection and hallucination mitigation: An investigation. *arXiv preprint arXiv:2401.08358*.

Amine Mhedhbi and Semih Salihoglu. 2019. Optimizing subgraph queries by combining binary and worst-case optimal joins. *arXiv preprint arXiv:1903.02076*.

Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):1–40.

A Pal, LK Umapathi, and M Sankarasubbu. 2023. Med-halt: medical domain hallucination test for large language models. arxiv. *arXiv preprint arXiv:2307.15343*.

Hongjin Qian, Peitian Zhang, Zheng Liu, Kelong Mao, and Zhicheng Dou. 2024. Memorag: Moving towards next-gen rag via memory-inspired knowledge discovery. *arXiv preprint arXiv:2409.05591*, 1.

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

Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. 2008. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment*, 1(1):364–375.

Peiqi Sui, Eamon Duede, Sophie Wu, and Richard Jean So. 2024. Confabulation: The surprising value of large language model hallucinations. *arXiv preprint arXiv:2406.04175*.

Shixuan Sun, Xibo Sun, Yulin Che, Qiong Luo, and Bingsheng He. 2020. Rapidmatch: A holistic approach to subgraph query processing. *Proceedings of the VLDB Endowment*, 14(2):176–188.

Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. 2012. Efficient subgraph matching on billion node graphs. *arXiv preprint arXiv:1205.6691*.

Oguzhan Topsakal and Tahir Cetin Akinci. 2023. Creating large language model applications utilizing langchain: A primer on developing llm apps fast. In *International conference on applied engineering and natural sciences*, volume 1, pages 1050–1056.

Chaohui Wang, Miao Xie, Sourav S Bhowmick, Byron Choi, Xiaokui Xiao, and Shuigeng Zhou. 2020. Ferrari: an efficient framework for visual exploratory subgraph search in graph databases. *The VLDB Journal*, 29(5):973–998.

Miao Xie, Sourav S Bhowmick, Gao Cong, and Qing Wang. 2017. Panda: toward partial topology-based search on large networks in a single machine. *The VLDB Journal*, 26(2):203–228.

Derong Xu, Pengyue Jia, Xiaopeng Li, Yingyi Zhang, Maolin Wang, Qidong Liu, Xiangyu Zhao, Yichao Wang, Huifeng Guo, Ruiming Tang, and 1 others. 2025a. Align-grag: Reasoning-guided dual alignment for graph retrieval-augmented generation. *arXiv preprint arXiv:2505.16237*.

Derong Xu, Xinhang Li, Ziheng Zhang, Zhenxi Lin, Zhihong Zhu, Zhi Zheng, Xian Wu, Xiangyu Zhao, Tong Xu, and Enhong Chen. 2025b. Harnessing large language models for knowledge graph question answering via adaptive multi-aspect retrieval-augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25570–25578.

Yutong Ye, Xiang Lian, and Mingsong Chen. 2024. Efficient exact subgraph matching via gnn-based path dominance embedding. *Proceedings of the VLDB Endowment*, 17(7):1628–1641.

Ziheng Zhang, Zhenxi Lin, Yefeng Zheng, and Xian Wu. 2025. How much medical knowledge do llms have? an evaluation of medical knowledge coverage for llms. In *Proceedings of the ACM on Web Conference 2025*, pages 5330–5341.

Xuejiao Zhao, Siyan Liu, Su-Yin Yang, and Chunyan Miao. 2025. Medrag: Enhancing retrieval-augmented generation with knowledge graph-elicited reasoning for healthcare copilot. In *Proceedings of the ACM on Web Conference 2025*, pages 4442–4457.

A Appendix: Symbol Definitions

Symbol	Description
K	external knowledge documents
G	knowledge graph (data graph)
q^o	an user natural language query
q^*	initial query graph
q	the normalized query graph
a	the final answer generated by SG-RAG
g	exact matched subgraphs
g''	fallback matched subgraphs
S	Sets of matched subgraphs g
D	the diameter of the G
I_l	an R*-Tree index over paths of length l
p_z (p_q)	path in G (or q)
p_c (p'_c)	path in candidate set
v_i (q_i)	a node in G (or q)
e_{ij} ($e_{q_iq_j}$)	an edge in G (or q)
$\mathcal{M}$	a GNN model of G
Q	query paths before label completion
Q'	query paths after label completion
U	a mapping from unknown nodes to label
P	fully labeled query path sets
$p_q.cand\_list$	exact candidate paths
g_{v_i} (s_{v_i})	a star subgraph(substructure) centered at v_i
$o(v_i)$	a star subgraph embedding
$o_0(v_i)$ ($o_0(q_i)$)	a node label embedding
$o(p_z)$ ($o(p_q)$)	a path dominance embedding
$o_0(p_z)$ ($o_0(p_q)$)	a path label embedding

Table 8: Symbols and Description.

B Appendix: Graph Construction and Label Embedding Implementation Details

Entity and Relation Extraction Format. We employ a large language model (GPT-4o-mini) to extract graph components from input text, following a standardized prompt template (see Table 9). Each extracted node is formatted as a triplet: $(\mathrm{id}(v_i), \ell(v_i), \xi(v_i))$, and each edge as a quadruple: $(\mathrm{id}(e_{ij}), v_i, v_j, \xi(e_{ij}))$. This structured output ensures compatibility with downstream graph-based reasoning and retrieval modules.

Section	Content
Goal	Given a paragraph, identify all semantically meaningful entities and extract structured relations among them. Each entity becomes a node and each relation becomes an edge in the output graph.
Step1: Nodes	Identify all entities. For each entity, extract: • $\mathrm{id}(v_i)$: Unique node identifier • $\ell(v_i)$: Node label (entity name) • $\xi(v_i)$: Node description (its role or attributes) **Format:** (<$\mathrm{id}(v_i)$><|><$\ell(v_i)$><|><$\xi(v_i)$>)
Step2: Edges	For each related entity pair, extract: • $\mathrm{id}(e_{ij})$: Unique edge identifier • v_i, v_j: Start and end node IDs • $\xi(e_{ij})$: Description of the relation **Format:** (<$\mathrm{id}(e_{ij})$><|><v_i><|><v_j><|><$\xi(e_{ij})$>)
Step3: Output Format	List all nodes and edges using # as a separator. End with `<|COMPLETE|>`.
Input Variable	`Text: {input_text}`

Table 9: Prompt Template for Text-to-Graph Extraction

Label Embedding Computation. We use `text-embedding-3-small` to obtain semantic embeddings of node labels. These embeddings are stored for downstream path-level embedding composition and retrieval matching.

Path Embedding Concatenation. Given a path $p_z = [v_1, v_2, \ldots, v_k]$, its label embedding is defined as the concatenation of node-level embeddings:

$$o_0(p_z) = [o_0(v_1), o_0(v_2), \ldots, o_0(v_k)] \in \mathbb{R}^{k \cdot F}, \quad (8)$$

preserving order-sensitive semantic representation for high-fidelity matching.

C Appendix: GNN-Based Dominant Embedding Training

Architecture. This appendix describes the training process of the GNN-based dominant embedding module used in SG-RAG, including GAT layer formulation, subgraph aggregation, and structural dominance constraints. As shown in Fig 4, the GNN adopts a standard GAT-based architecture with an input projection layer, an attention-based message passing layer, a readout layer, and a fully connected projection head.

GAT Layer Formulation. Given input label embeddings x_j for node v_j, the attention coefficient between v_i and v_j is computed as:

$$a_{v_i v_j} = a(Wx_i, Wx_j) \tag{9}$$

where $a(\cdot,\cdot)$ is a shared attention function $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$, capturing the importance of v_j to v_i. Let $N(v_i)$ denote the neighbor set of v_i. The attention coefficients are normalized using softmax:

$$\alpha_{v_i v_j} = \text{softmax}(a_{v_i v_j}) = \frac{\exp(a_{v_i v_j})}{\sum_{v_k \in N(v_i)} \exp(a_{v_i v_k})} \tag{10}$$

Then the updated node representation is:

$$x_i' = \sigma\left(\sum_{v_j \in N(v_i)} \alpha_{v_i v_j} \cdot Wx_j\right) \tag{11}$$

where $\sigma(\cdot)$ is a nonlinear activation function, and $x_i' \in \mathbb{R}^{F'}$.

Star Subgraph Aggregation. The dominant embedding of a node is computed from its local star-shaped subgraph g_{v_i} via aggregation:

$$y_i = \sum_{v_j \in V(g_{v_i})} x_j' \tag{12}$$

$$o(g_{v_i}) = \sigma(Wy_i), \quad o(v_i) = o(g_{v_i}) \tag{13}$$

As shown in Fig 5, each star subgraph includes the central node and its neighbors, and all of its proper substructures are used for enforcing dominance.

Dominance Constraint. For any proper substructure $s_{v_i} \subset g_{v_i}$, the model enforces:

$$o(s_{v_i}) \preceq o(g_{v_i}) \tag{14}$$

$$\mathcal{L} = \sum \|\max(0, o(g_{v_i}) - o(s_{v_i}))\|_2^2 \tag{15}$$

This constraint ensures that smaller subgraphs embed into smaller vector regions, enabling efficient pruning.

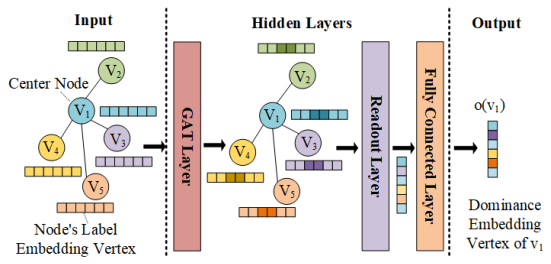

Figure 4: Architecture of the GNN used for learning dominant embeddings.

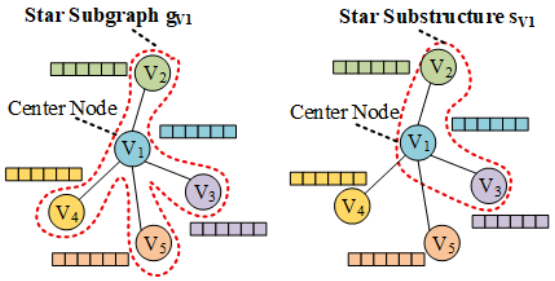

Figure 5: Illustration of a star-shaped subgraph and its substructures.

Algorithm 2 GNN Model Training

Require: training data D, learning rate η
Ensure: trained GNN model M
$D \leftarrow \emptyset$
for each vertex $v_i \in V(G)$ **do**
  extract star subgraph g_{v_i} and substructures s_{v_i}
  add all (g_{v_i}, s_{v_i}) to D
end for
shuffle D
repeat
  for each batch $B \subseteq D$ **do**
    compute embeddings and loss $\mathcal{L}(B)$
    update model: M.update$(\mathcal{L}(B), \eta)$
  end for
  $\mathcal{L}_e \leftarrow 0$
  for each batch $B \subseteq \mathcal{D}$ **do**
    $\mathcal{L}_e \leftarrow \mathcal{L}_e + \mathcal{L}(B)$
  end for
until $\mathcal{L}_e = 0$
return trained model $\mathcal{M}$

Path-Level Embedding. Given a path $p_z = [v_1, \ldots, v_k]$, its embedding is defined as:

$$o(p_z) = [o(v_1), \ldots, o(v_k)] \in \mathbb{R}^{k \cdot d} \tag{16}$$

Example Illustration. As shown in Fig 6, SG-RAG verifies whether a query path $q_2 q_1 q_3$ is contained in a candidate path $v_2 v_1 v_3$ by checking:

$$o(q_2 q_1 q_3) \preceq o(v_2 v_1 v_3) \quad \Rightarrow \text{Valid Match} \tag{17}$$

If the condition fails in any dimension, the match is rejected. This dominance check underpins the path-level pruning in the retrieval phase SG-RAG.

Training Algorithm. The full training process is shown in Algorithm 2. It uses a star-subgraph contrastive loss over multiple pairs to learn structure-aware embeddings that generalize to unseen queries.

D Appendix: Entity Normalization Details

Entity and Relation Extraction. The initial query graph q^* is constructed by prompting a large

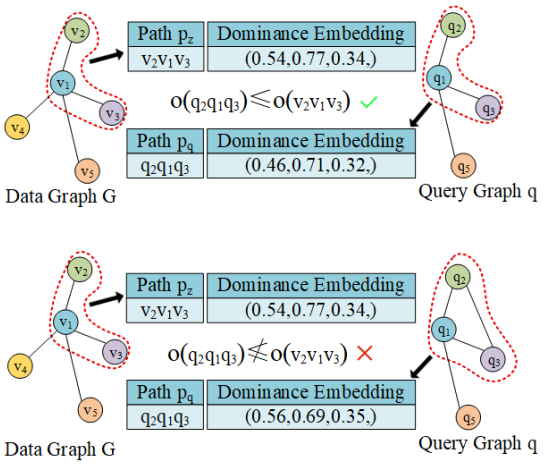

Figure 6: Path-level dominant embedding matching via element-wise comparison.

Section	Content
Goal	Given a user question, identify all semantically independent entities and extract their implicit structural relations to construct a query graph. If the question contains a target entity (i.e., the answer is unknown), it should be extracted as a node with label "UNK".
Step1: Nodes	Identify all entities. For each entity, extract: • $\mathrm{id}(q_i)$: Unique node identifier • $\ell(q_i)$: Node label (entity name), use "UNK" if it is the target to be queried • $\xi(q_i)$: Node description (optional, can be empty if unknown) **Format:** (<id(q_i)><|><$\ell(q_i)$><|><$\xi(q_i)$>)
Step2: Edges	For each related entity pair, extract: • $\mathrm{id}(e_{q_iq_j})$: Unique edge identifier • q_i, q_j: Start and end node IDs **Format:** (<id($e_{q_iq_j}$)><|><q_i><|><q_j><|>)
Step3: Output Format	List all nodes and edges in order, separated by #. End with <|COMPLETE|>.
Input Variable	`Text: {user_query}`

Table 10: Prompt Template for Query-to-Graph Extraction

language model to identify entities and their relationships from the natural language query q^o. The prompt format and examples are listed in Table 10.

Label Embedding. For each node $q_i^* \in V(q^*)$, the system computes its label embedding:

$$o_0(q_i^*) = \mathrm{LLM}(\ell(q_i^*)) \in \mathbb{R}^F. \tag{18}$$

These embeddings serve as semantic keys for aligning query nodes with their canonical counterparts in G.

Entity Normalization via FAISS. Due to possible lexical variations in user queries, SG-RAG employs a FAISS-based nearest neighbor search over the label embedding space of G:

- A deep copy of the initial graph q^* is made: $q \leftarrow \mathrm{DeepCopy}(q^*)$;
- Each $o_0(q_i^*)$ is used as a query vector;
- FAISS retrieves the most similar $q_i \in V(G)$ based on cosine similarity;
- The node q_i^* is replaced with q_i in q.

Formal Procedure. The normalization process is summarized in Algorithm 3.

Algorithm 3 Entity Normalization

Require: Initial query graph q^* with raw entities
Ensure: Normalized query graph q
$q \leftarrow \mathrm{DeepCopy}(q^*)$
for each vertex $q_i^* \in V(q^*)$ **do**
  $o_0(q_i^*) \leftarrow \mathrm{Embed}(\ell(q_i^*))$
  $q_i \leftarrow \mathrm{FAISS\_Search}(o_0(q_i^*)$, label embedding of $V(G))$
  Replace q_i^* with q_i in $V(q)$
end for
return q

E Appendix: Path Decomposition and Completion

Cost-Aware Path Decomposition. Given a query graph q and predefined path length l, we initialize $Q = \emptyset$ and $\mathrm{Cost}_Q(\phi) = +\infty$. Starting from the node with highest degree, we enumerate all initial paths of length l as PathSet, and iteratively construct the path set local_Q using:

- minimal edge overlap with existing local_Q;
- minimal path weight $w(p) = -\sum_{q_i \in p} \deg(q_i)$.

The optimal set Q minimizing $\mathrm{Cost}_Q(\phi) = \sum w(p_q)$ is retained.

Label Completion for Unknown Nodes. For each query path p_q with unknown nodes, we insert zero vectors to obtain wildcard label embeddings $o_0(p_q)$. We then traverse the R*-Tree index I_l using a max-heap, comparing known dimensions of $o_0(p_q)$ with MBRs to collect aligned candidate paths p_z, and extract candidate label values into a mapping U from node IDs to label sets.

Enumerating Completed Paths. Given:

$$U = \{q_1 : [\ell_1, \ell_2],\ q_2 : [\ell_3]\},$$

we enumerate the Cartesian product of label options, e.g., $(\ell_1, \ell_3), (\ell_2, \ell_3)$, and apply each com-

Algorithm 4 Cost-Model-Based Query Plan Selection

Require: Query graph q, path length l
Ensure: Query path set Q representing the query plan ϕ
$Q \leftarrow \emptyset$; $\text{Cost}_Q(\phi) \leftarrow +\infty$
Select starting vertex q_i with the highest degree
Obtain initial path set PathSet of length l starting from q_i
for each candidate initial path $p_q \in$ PathSet **do**
  $\text{local}_Q \leftarrow \{p_q\}$; local_cost $\leftarrow 0$
  while some edge in $E(q)$ is not covered by local_Q **do**
    Select path p of length l that connects with local_Q, minimizing:
    - Edge overlap with local_Q
    - Path weight $w(p)$
    $\text{local}_Q \leftarrow \text{local}_Q \cup \{p\}$
    local_cost $\leftarrow$ local_cost $+ w(p)$
  end while
  if local_cost $< \text{Cost}_Q(\phi)$ **then**
    $Q \leftarrow \text{local}_Q$
    $\text{Cost}_Q(\phi) \leftarrow$ local_cost
  end if
end for
return Q

bination to a deep copy of Q to obtain a label-complete set Q'. Repeating this for all combinations gives:

$$P = \{Q'_1, Q'_2, \ldots, Q'_n\}.$$

This ensures that all query paths are structurally complete and semantically instantiable for path-level matching. Algorithms 4; Algorithms 5 and Algorithms 6 formally describe the decomposition and completion process.

F Appendix: Complexity Analysis

Entity Normalization. Each query node requires FAISS-based ANN search over N entities:

$$O(|V(q)| \cdot \log N) \tag{19}$$

Unknown Label Completion. Let $|Q_u|$ be the number of query paths with unknown nodes, h the R*-Tree height, f the average fan-out, and PP_i the pruning ratio at level i:

$$O\left(\sum_{i=0}^{h} |Q_u| \cdot f^{h-i+1} \cdot (1 - PP_i)\right) \tag{20}$$

Exact Subgraph Matching. Let u be the number of unknown nodes, t_j the number of label candidates for node j, $|Q|$ the number of paths per query:

$$O\left(\left(\prod_{j=1}^{u} t_j\right) \cdot \sum_{i=0}^{h} |Q| \cdot f^{h-i+1} \cdot (1 - PP_i)\right) \tag{21}$$

Algorithm 5 Find Candidate Labels for Unknown Vertices in Query Graph

Require: Query path set Q; R*-Tree index I_l over data graph G
Ensure: U: a map from unknown vertex ID to candidate labels
$U \leftarrow \emptyset$
UnknownVertexPathset $\leftarrow \emptyset$ // Paths containing unknown nodes
while at least one unknown vertex is not covered **do**
  Select a path $p_q \in Q$ that contains an uncovered unknown vertex
  UnknownVertexPathset $\leftarrow$ UnknownVertexPathset $\cup \{p_q\}$
end while
for each query path $p_q \in$ UnknownVertexPathset **do**
  Compute $o_0(p_q)$ using LLM // All-zero embedding for unknown nodes
  root(I_l).list $\leftarrow$ UnknownVertexPathset
  Insert $(\text{root}(I_l), 0)$ into heap H
end for
while H is not empty **do**
  $(N, \text{key}(N)) \leftarrow H$.pop()
  if key(N) $< \min_{p_q \in \text{UnknownVertexPathset}} \|o_0(p_q)\|_1$ **then**
    break
  end if
  if N is not a leaf node **then**
    for each child $N_i \in N$ **do**
      for each query path $p_q \in N$.list **do**
        if $o_0(p_q)$ matches N_i.MBR$_0$ on known positions **then**
          N_i.list $\leftarrow N_i$.list $\cup \{p_q\}$
        end if
      end for
      if N_i.list $\neq \emptyset$ **then**
        Insert $(N_i, \text{key}(N_i))$ into heap H
      end if
    end for
  else
    for each stored path $p_z \in N$ **do**
      for each query path $p_q \in N$.list **do**
        if $o_0(p_q)$ and $o_0(p_z)$ match on known positions **then**
          Extract labels from p_z for unknown positions in p_q
          Update U[unknown vertex id] accordingly
        end if
      end for
    end for
  end if
end while
return U

Variable Definitions $V(q)$: nodes in the query graph; N: number of knowledge graph entities; Q: set of query paths; Q_u: set of query paths with unknown nodes; u: number of unknown nodes; t_j: number of label candidates for unknown node j; h: R*-Tree height; f: fan-out per index level; PP_i: pruning ratio at index level i

Algorithm 6 Populate the Unknown Vertices in Query Paths

Require: Query path set Q; candidate label map U
Ensure: $P \leftarrow \{Q' \mid Q'$ is a label-completed instantiation of $Q\}$

```
UnknownVertexIDs ← keys of U
LabelOptions ← [U[id] for id in UnknownVertexIDs]
LabelCombinations ← CartesianProduct(LabelOptions)
P ← ∅
for each combo ∈ LabelCombinations do
    LabelMap ← ∅
    for i = 0 to length(UnknownVertexIDs) − 1 do
        LabelMap[UnknownVertexIDs[i]] ← combo[i]
    end for
    Q' ← DeepCopy(Q)
    for each path in Q' do
        for each vertex v in path do
            if v.label = NULL and v.id ∈ LabelMap then
                v.label ← LabelMap[v.id]
            end if
        end for
    end for
    Append Q' to P
end for
return P
```

Practical Assumptions. When $u = 1$, $t_1 \leq 10$, $|Q| \approx 3$–5, the expression simplifies to:

$$O\left(|V(q)| \cdot \log N + (t_1 + 1) \cdot \sum_{i=0}^{h} |Q| \cdot f^{h-i+1} \cdot (1 - PP_i)\right) \tag{22}$$

This confirms that retrieval cost remains tractable even under scale.

G Appendix: ERQA Dataset Construction and Statistics

Subset	#Ent.	#Rel.	AvgE	AvgR	#d
CM-ERQA	1.66M	4.58M	5.4	5.7	3.6
FB-ERQA	14.5K	272.1K	6.1	5.4	3.1
UD-ERQA	314.4K	53.0K	4.7	5.1	3.3

Table 11: Structural Statistics of the ERQA Datasets

Dataset Statistics. "#Ent." and "#Rel." indicate the number of entities and relations, "AvgE" and "AvgR" are the average number of entities and relations per query graph, and "#d" is the average maximum path length. These statistics guide the selection of hyperparameters SG-RAG such as the path length l.

Bridge-Star Subgraph Construction. Each QA pair is constructed over a **Bridge-Star Subgraph**, defined as a pair of high-degree star nodes connected via shared bridge nodes (Figure 7). Each star node forms a 1-hop subgraph, while bridge nodes enable multi-hop constraint reasoning.

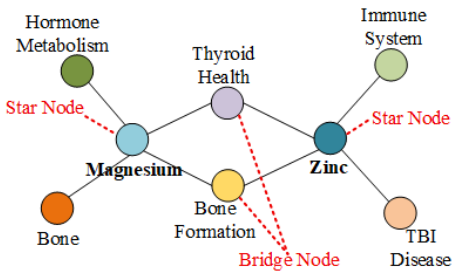

Figure 7: Illustration of a bridge-star subgraph structure.

Question Generation Process. For each subgraph: One star node is hidden and designated as the ground-truth answer; The visible star node's 1-hop neighbors and all bridge nodes are included in the question context. The structured input is then converted into a natural language question via LLM prompting. The prompt format in Appendix.J.

Example. In Figure 7, "Magnesium" and "Zinc" are star nodes. "Magnesium" links to "Bone" and "Hormone Metabolism"; "Thyroid Health" and "Bone Formation" are bridge nodes. A generated question is: *Which element is associated with bone formation, thyroid health, hormone metabolism, and immune system?* Here, "Magnesium" is the correct answer, while "Zinc" is a plausible distractor. This design supports precise multi-constraint evaluation of RAG systems.

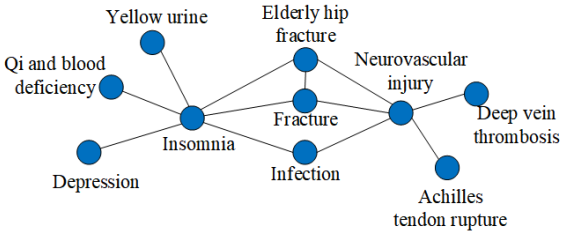

Figure 8: Subgraph structure used for query construction.

H Appendix: Case Study

To further illustrate the effectiveness of SG-RAG in handling constraint-rich queries, we present a representative case from the CM-ERQA subset : **Query:** *Which disease is likely to simultaneously cause deep vein thrombosis, acute closed Achilles tendon rupture, infection, and fracture as complications?* **Gold Answer:** *Neurovascular injury* The entities included in this query and their neighbors

Method	Answer	Key Supporting Output
GPT-5.1	Unable determine	"...could not identify a single disease associated with all listed complications..."
NaiveRAG	Unable determine	"...no disease found to be directly related to all the listed complications..."
RAPTOR	Sleep disorder	"Sleep disorder"
GraphRAG	Breast cancer	"...may experience thrombosis, infection, and fracture..."
LightRAG	Unable determine	"...no disease clearly causes all these complications simultaneously..."
SG-RAG	**Neurovascular injury**	"Answer: Neurovascular injury. Reasoning: Based on the description, this condition is associated with all listed complications..."

Table 12: Comparison of Answers

are shown in Figure 8. The comparison of the generation effect of SG-RAG and the baseline method is shown in Table 12

Analysis of Methods. Below we provide detailed observations for each method's performance:

- **GPT-5.1** generates generalized scenarios (e.g., trauma, diabetes) with no precise answer, failing to enforce multi-constraint reasoning.
- **NaiveRAG** retrieves texts related to individual entities, but lacks a mechanism to ensure global constraint satisfaction.
- **RAPTOR** retrieves some relevant candidates but is affected by noisy context, leading to inaccurate answer selection.
- **GraphRAG** partially matches constraints (e.g., thrombosis–infection–fracture), but ignores "Achilles tendon rupture," resulting in hallucination of "breast cancer."
- **LightRAG** covers all constraint terms but fails to reason over their intersection due to lack of co-occurrence modeling.
- **SG-RAG** successfully reconstructs the full constraint structure, retrieves a matching subgraph, and generates a precise answer.

I Appendix: Prompt Template for Answer Inference from Matched Subgraph

This prompt guides the model to infer an answer from a matched subgraph by reasoning over structured relations and answering a user's natural language query. The model is instructed to restrict its final answer to one of the entities explicitly mentioned in the graph, the detail shown in Table 13.

Section	Content
Goal	Given a structured subgraph and a user query, infer the most probable answer by reasoning over the entities and their relations. The answer must be one of the entities mentioned in the graph.
Step1: Input Subgraph	Each node contains: • $id(v_i)$ • $\ell(v_i)$ • $\xi(v_i)$ Each edge contains: • $id(v_i)$ •$id(v_j)$ • $\ell(e_{ij})$
Step2: Relation Paragraph	Convert each edge into a natural language sentence: "Node v_i is related to Node v_j via: [relation :$\ell(e_{ij})$]." Concatenate all such sentences into a coherent paragraph as background knowledge.
Step3: Prompt Composition	Combine the following parts into the final prompt: • Fixed instruction explaining the task • The relation paragraph from Step 2 • The user's question • A constraint: the answer must be one of the mentioned entity labels. If undecidable, return unable to determine.

Formatted Example	`Below is a paragraph describing the relationships among entities in a structured graph.`
	`This paragraph contains the answer to a user question. Read and reason carefully.`
	`Note: The final answer must be one of the entity labels mentioned in the paragraph.`
	–
	`Known Relations:`
	`Node1 is related to Node2 via: edge1.`
	`Node1 is related to Node3 via: edge2.`
	`Node1 is related to Node4 via: edge3.`
	–
	`User Question: Which entity is associated with multiple others?`
	`Answer: [Entity] or "Unable to determine"`

Input Variables	`Matched Subgraph: {nodes, edges}` `User Question: {query}`

Table 13: Prompt Template for Answer Inference Using Matched Subgraph

J Appendix: Prompt Template for Question Construction

This prompt is used to generate a natural and concise question based on a bridge-star subgraph. The question must start with "Which

Section	Content
Goal	Generate a fluent and concise natural language question that starts with "Which {CORE_TYPE}...". The question must simultaneously reference: • Unique neighbors of the core node {UNIQUE_DESC} • Shared bridge neighbors {COMMON_DESC} This ensures that the answer must satisfy all structural constraints while avoiding ambiguity caused by bridge entities.
Step1: Placeholder Filling	• {CORE_TYPE}: The type of the core entity (e.g., "element", "disease") • {UNIQUE_DESC}: Descriptions of neighbors exclusive to the core node • {COMMON_DESC}: Descriptions of shared bridge neighbors
Step2: Prompt to LLM	You are a Chinese language expert. Based on the placeholders provided, polish and generate a fluent, natural Chinese question that: • Starts with "Which{CORE_TYPE}..." • Mentions both: {UNIQUE_DESC} (directly related features) {COMMON_DESC} (commonly co-occurring context) • Do not reveal or explain the answer. Return only the question sentence.
Output Format	LLM should return a single sentence only, without any explanation or metadata.
Example Input	{CORE_TYPE} = Element {UNIQUE_DESC} = bone formation (promotes), hormone metabolism (related) {COMMON_DESC} = thyroid health (influences), immune system (associated)
Example Output	Which element is closely related to bone formation and hormone metabolism, and also influences thyroid health and participates in immune system regulation?
Input Variables	Bridge-Star Subgraph: {nodes, edges}

Table 14: Prompt for Natural Question Generation

{CORE_TYPE}..." and mention both the **unique neighbors** of the core node and the **bridge neighbors**, ensuring the structural constraints are embedded and bridge nodes are disambiguated, the detail shown in Table14.

K Appendix: Empowerment Score Definition and Evaluation

To assess the reasoning quality of generated answers, we adopt a subjective evaluation metric termed **Empowerment Score(Emp.S)**, designed to approximate human judgment. Each answer is scored by an LLM evaluator along two dimensions:

- **Logical Coherence** (0–2 **points)**: Does the response follow a clear and structured reasoning process?
- **Informational Value** (0–1 **point)**: Does the response provide useful insights, elaboration, or interpretative depth?

The total score ranges from 0 to 3. All answers from SG-RAG and baseline systems are assessed under the same prompt setting by the same LLM judge to ensure fairness. We report average Empowerment Scores across pairwise comparisons. Evaluation prompt format and scoring rubric are included in Table 15.

Section	Content
Goal	Automatically evaluate answers from multiple methods to the same question. The LLM acts as a reviewer and scores each answer independently according to the criteria below.
Step1:Input	• Question: {query} • Gold Answer: {gold_answer} • Answer List: [Method_A] {answer_A} [Method_B] {answer_B} ... [Method_F] {answer_F}
Step2: Evaluation Instruction	You are a senior evaluator. Please carefully read the question, the gold-standard answer, and the list of candidate answers. For each answer, assign scores based on the criteria below. Return the evaluation as a structured JSON.
Scoring Criteria	• **Logical Coherence (0–2)**: Is the reasoning clear, complete, and well-sequenced? • **Insight (0–1)**: Does the answer offer new insight or helpful suggestions?

Output Format	Return the scores as follows:
	{
	"Method_A": {"logic": L1, "insight": I1, "total": T1},
	"Method_B": {"logic": L2, "insight": I2, "total": T2},
	. . .
	"Method_F": {"logic": L3, "insight": I3, "total": T3}
	}

Table 15: Prompt Template for Subjective Evaluation

L Appendix: matching algorithm

Algorithm 7 Exact Subgraph Matching with GNN-based Path Dominance Embedding

Require: query graph q; fully labeled query path sets P; trained GNN model M; R*-Tree I_l over data graph G.
Ensure: exact match subgraph set S.

$S \leftarrow \emptyset$
for each query path set $Q'_i \in P$ **do**
  for each query path $p_q \in Q'_i$ **do**
    p_q.cand_list $\leftarrow \emptyset$
    Obtain $o(p_q)$ via GNN
    Obtain $o_0(p_q)$ via LLM
  end for
  root(I_l).list $\leftarrow Q'_i$
  Insert $(\text{root}(I_l), 0)$ into heap H
  while H is not empty **do**
    $(N, \text{key}(N)) \leftarrow H$.pop()
    if key$(N) < \min_{p_q \in Q'_i} \|o(p_q)\|_1$ **then**
      break
    end if
    if N is an internal node **then**
      for each child $N_i \in N$ **do**
        for each $p_q \in N$.list **do**
          if $o_0(p_q) \in \text{MBR}_0(N_i)$ **and** $DR(o(p_q)) \cap \text{MBR}(N_i) \neq \emptyset$ **then**
            N_i.list $\leftarrow N_i$.list $\cup \{p_q\}$
          end if
        end for
        if N_i.list $\neq \emptyset$ **then**
          Insert $(N_i, \text{key}(N_i))$ into H
        end if
      end for
    else
      for each $p_z \in N$ **do**
        for each $p_q \in N$.list **do**
          if $o_0(p_q) = o_0(p_z)$ **then**
            if $o(p_q) \preceq o(p_z)$ **then**
              p_q.cand_list $\leftarrow p_q$.cand_list $\cup \{p_z\}$
            end if
          end if
        end for
      end for
    end if
  end while
  Assemble S from all p_q.cand_list using Algorithm 7
end for
return S

Algorithm 8 Assemble Subgraphs

Require: Query graph q; Fully labeled query path set Q'; Each p_{q_i} is associated with p_{q_i}.cand_list
Ensure: Exact subgraph set S

$S \leftarrow \emptyset$
$F \leftarrow p_{q_1}$.cand_list $\times p_{q_2}$.cand_list $\times \cdots \times p_{q_k}$.cand_list
for each combination $\{p_{f_1}, \ldots, p_{f_k}\} \in F$ **do**
  $g \leftarrow$ empty graph
  query_node_map $\leftarrow$ empty map
  conflict $\leftarrow$ **False**
  for each path p_f in combination **do**
    for each query node v_f in p_f **do**
      $v_s \leftarrow$ mapped node of v_f
      if $v_f \in$ query_node_map **then**
        if query_node_map$[v_f] \neq v_s$ **then**
          conflict $\leftarrow$ **True**; **break**
        end if
      else
        query_node_map$[v_f] \leftarrow v_s$
        add v_s to g if not present
      end if
    end for
    add all edges in p_f to g
  end for
  if not conflict **then**
    $S \leftarrow S \cup \{g\}$
  end if
end for
return S

M Appendix: assembly algorithm