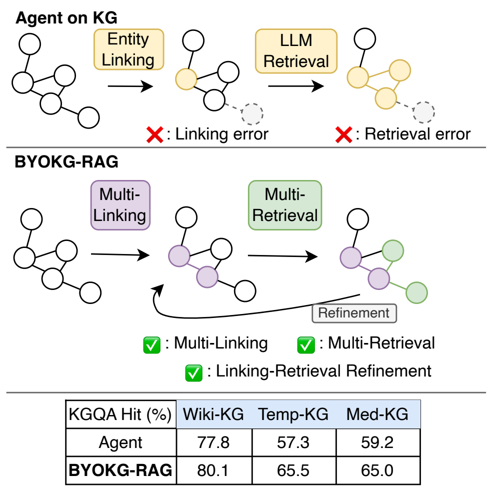
BYOKG-RAG: Multi-Strategy Graph Retrieval for Knowledge Graph Question Answering

Costas Mavromatis[*], **Soji Adeshina**, **Vassilis N. Ioannidis**, **Zhen Han**,
Qi Zhu, **Ian Robinson**, **Bryan Thompson**, **Huzefa Rangwala**, **George Karypis**
Amazon

Abstract

Knowledge graph question answering (KGQA) presents significant challenges due to the structural and semantic variations across input graphs. Existing works rely on Large Language Model (LLM) agents for graph traversal and retrieval; an approach that is sensitive to traversal initialization, as it is prone to entity linking errors and may not generalize well to custom ("bring-your-own") KGs. We introduce BYOKG-RAG, a framework that enhances KGQA by synergistically combining LLMs with specialized graph retrieval tools. In BYOKG-RAG, LLMs generate critical graph artifacts (question entities, candidate answers, reasoning paths, and OpenCypher queries), and graph tools link these artifacts to the KG and retrieve relevant graph context. The retrieved context enables the LLM to iteratively refine its graph linking and retrieval, before final answer generation. By retrieving context from different graph tools, BYOKG-RAG offers a more general and robust solution for QA over custom KGs. Through experiments on five benchmarks spanning diverse KG types, we demonstrate that BYOKG-RAG outperforms the second-best graph retrieval method by 4.5% points while showing better generalization to custom KGs. BYOKG-RAG framework is open-sourced at https://github.com/awslabs/graphrag-toolkit.



KGQA Hit (%)	Wiki-KG	Temp-KG	Med-KG
Agent	77.8	57.3	59.2
BYOKG-RAG	80.1	65.5	65.0

Figure 1: While agentic retrieval (top) may be prone to linking and retrieval errors, e.g., for aggregation queries, BYOKG-RAG (middle) employs multi-strategy linking and retrieval, along with iterative refinement, to mitigate these errors. As a result, BYOKG-RAG improves KGQA performance across diverse KGs (bottom).

1 Introduction

Knowledge Graphs (KGs) (Vrandečić and Krötzsch, 2014) are databases that store information and data in structured format, typically using entities and relationships. The structure of KGs enables efficient data updates and complex queries, making them valuable assets for enterprises across multiple sectors (Ozsoy et al., 2024). In the Large Language Model (LLM) era (Brown et al., 2020; Bommasani et al., 2021), KGs have been widely adapted as external information sources to ground LLM responses to semantics present in the graph and to solve KG Question Answering (KGQA) tasks (Pan et al., 2024). In KGQA, one prevalent approach is retrieval-augmented generation (RAG) that leverages graph retrieval methods to fetch relevant information from the KG as additional input for the LLM (Lewis et al., 2020; Peng et al., 2024).

Adapting RAG to custom ("bring-your-own") KGs presents significant challenges as KGs vary not only in their schema design and semantic representations, but also in how information must be retrieved, e.g., via complex aggregation queries. Current approaches (Mavromatis and Karypis, 2024; Luo et al., 2024a; Jiang et al., 2024) use graph retrievers specifically fine-tuned for a particular KG, but require training data, which may not be avail-

[*]Correspondence: mavrok@amazon.com.

able due to privacy concerns or practical limitations. Alternative methods utilize LLMs to traverse the graph step-by-step until arriving at the answers or until termination in an agentic fashion (Jiang et al., 2023a; Sun et al., 2024). These methods leverage the inherent knowledge within LLMs to handle 'bring-your-own' scenarios, where training data is unavailable for KGQA. However, such approaches have limitations: Extracting knowledge from custom KGs may require operations like performing complex graph aggregations which is not readily obtained via LLM-based traversal.

In this work, we introduce **BYOKG-RAG**, a framework that addresses these challenges through multi-strategy graph retrieval. Our key insight is to leverage LLMs as a KG-Linker that generates diverse graph artifacts - from entity mentions to executable queries. These artifacts are then processed by specialized graph tools, each designed to handle different retrieval scenarios effectively. Given a user query and the KG schema[1], the LLM is prompted to generate relevant graph artifacts, such as question and answer entities, relationship types, and executable graph queries. Subsequently, the graph toolkit takes these artifacts as input, links them to the KG and retrieves relevant graph context through different retrieval methods. The retrieved context can be used in an iterative fashion so that KG-Linker improves the accuracy of both the generated artifacts and subsequent retrieval, before final KGQA. Compared to agentic retrieval, which is restricted to the graph context that can be explored by the agent, BYOKG-RAG retrieves context from multiple linking and retrieval tools, including agentic retrieval, to enhance KGQA accuracy; see Figure 1 for a high-level comparison.

Our contributions are summarized below:

- We introduce BYOKG-RAG, a novel framework that improves KGQA across custom KGs via multi-strategy graph linking and retrieval.
- BYOKG-RAG outperforms the best graph retrieval method by 4.5% points in KGQA across four KGs. In addition, BYOKG-RAG achieves on-par or better performance than state-of-the-art KG agents on key benchmarks.

[1]The schema typically expresses the node and relation types present in the graph, among other properties, and is available in common graph databases via `graph.get_schema()` functionality.

- Our thorough evaluation across KGs with diverse semantics demonstrates the benefits of multi-strategy graph retrieval, and we open-source our toolkit.

2 Background & Related Work

2.1 Problem Formulation

Consider a Knowledge Graph (KG) $\mathcal{G}$ containing facts represented as triplets (h, r, t), where h represents the head entity, t represents the tail entity, and r denotes the relation between them. The task of KGQA involves extracting a set of entities $\mathcal{A} \in \mathcal{G}$ that correctly answer a given natural language question q. Since KGs typically contain millions of facts and nodes, retrieval-augmented generation retrieves relevant graph context $\mathcal{C}$, which is then used by LLM to generate the answers as $\mathcal{A} = \text{LLM}(q, \mathcal{C})$. To retrieve $\mathcal{C}$, common approaches combine entity linking techniques (Yih et al., 2015) to identify question entities e_q with graph retrieval methods that extract relevant information within an L-hop neighborhood of these entities (Sun et al., 2018).

2.2 Related Works

KGQA Methods. KGQA methods involve either fine-tuning or zero-shot/few-shot inference. Fine-tuning methods encompass several approaches. Some methods focus on parsing the given question into a logical form query executable over the KG (Sun et al., 2020; Lan and Jiang, 2020a; Ye et al., 2022; Gu et al., 2023; Agarwal et al., 2023). Others involve training specialized neural networks for handling KG data (Sun et al., 2018; He et al., 2021; Mavromatis and Karypis, 2022; Jiang et al., 2023b). Recent approaches aim at teaching LLMs to better understand KG semantics (Luo et al., 2024a; Ao et al., 2025) or instructing the LLM on how to explore the graph (Jiang et al., 2024; Luo et al., 2025a; Zhang et al., 2025). Zero-shot and few-shot methods typically rely on LLMs' knowledge and combine them with appropriate graph tools for graph traversal (Jiang et al., 2023a; Kim et al., 2023; Sun et al., 2024; Dong et al., 2025; Sui et al., 2024; Jo et al., 2025), graph querying (Li et al., 2023; Wang et al., 2023), and constrained decoding on KGs (Luo et al., 2024b; Li et al., 2024a). BYOKG-RAG is a novel framework for improving graph linking and retrieval in "bring-your-own-KG" (zero-shot) settings.

Graph RAG. Graph RAG refers to the frame-

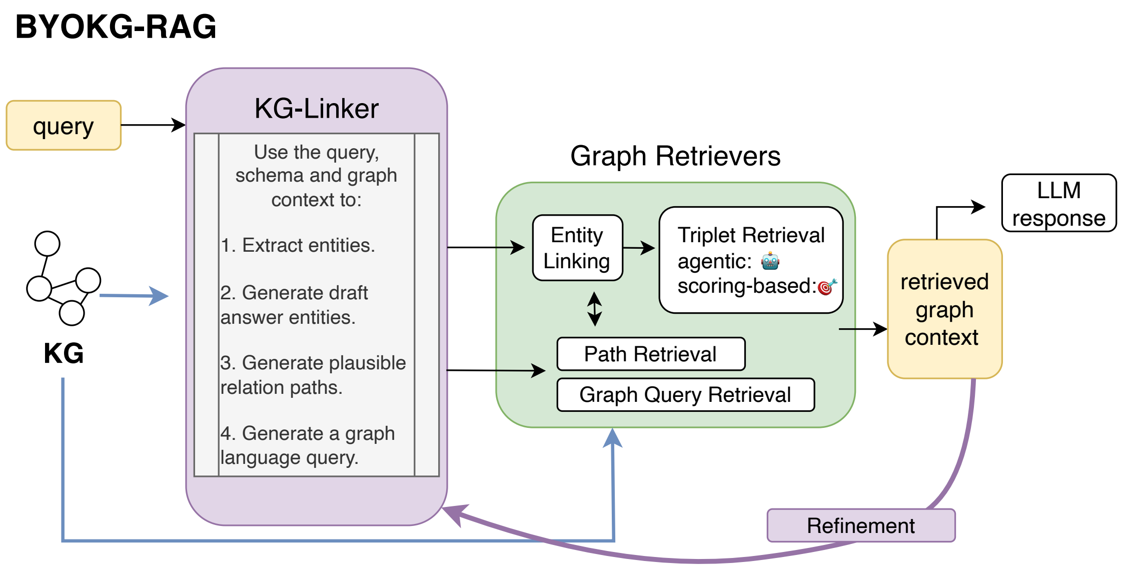


Figure 2: BYOKG-RAG prompts an LLM to generate critical graph artifacts for graph linking. Then, a toolkit of graph retrievers operates on the underlying graph and, based on the generated artifacts and query's context, retrieves relevant information for final KGQA or linking refinement.

work of using context originating from a graph or a KG within RAG (Edge et al., 2024; He et al., 2024; Mavromatis and Karypis, 2024). Recently, methods (Gutiérrez et al., 2024; Guo et al., 2024; Luo et al., 2025b; Chen et al., 2025) use LLMs for constructing KGs out of raw data to improve information retrieval with graph information (Lee et al., 2024; Sarmah et al., 2024), beyond text-based semantics. With the increased deployment of KG-powered applications (Peng et al., 2024), BYOKG-RAG offers a multi-strategy toolkit for retrieving task-specific graph information.

Graph Linking. Entity linking (Yadav and Bethard, 2019) in KGs maps mentions in natural language to their corresponding graph entities and has been extensively studied in KGQA systems (Yih et al., 2015; Oliya et al., 2021; Li et al., 2020). Beyond entity linking, some approaches focus on mapping natural language questions to graph paths (Saxena et al., 2020; Shi et al., 2021) and subgraphs via graph query execution (Lan and Jiang, 2020b; Gu and Su, 2022; Shu et al., 2022; Yu et al., 2022; Gao et al., 2025), typically employing training data for more accurate linking and retrieval. KG-Linker extends these ideas by combining multiple linking approaches with LLM capabilities, enabling robust graph retrieval without task-specific training data.

3 Bring-Your-Own-KG RAG (BYOKG-RAG)

In BYOKG-RAG, we address the challenges of graph retrieval in custom KGs through a novel two-stage approach. First, we prompt an LLM (KG-Linker) to generate graph artifacts - including entities, paths, and queries - that serve as anchor points for graph retrieval. Then, these artifacts are processed by specialized graph tools that link them to the underlying KG and retrieve relevant context through multiple complementary methods. This synergistic combination of LLM-generated artifacts and specialized graph tools enables more robust retrieval across custom KGs. The overall framework is illustrated in Figure 2.

3.1 KG-Linker

The key insight behind KG-Linker is to leverage LLMs' natural language understanding to generate diverse graph artifacts that can be linked to the KG through specialized tools. In KG-Linker, given an input a user query, a graph schema, and an optional graph context, an LLM is prompted to generate graph artifacts, including question entities, draft answers, useful relation paths, and graph queries. The LLM's prompt includes four main tasks: (1) Entity Extraction, (2) Relation Path Identification, and (3) Graph Query Generation, and (4): Draft

KG-Linker's Prompt p.

Given a question, schema, and optional graph context, your role is to perform the following tasks:

Task: Entity Extraction
Extract all topic entities from the question and useful intermediate entities from the graph context (if provided) within `<entities>` tags.

Task: Relationship Path Identification
Identify all relevant relationship paths that connect the entities and can be used to answer the question within `<paths>` tags.

Task: Graph Query Generation
Construct a complete, executable graph query statement that will retrieve the answer from a graph database. Format your query in {openCypher} language within `<opencypher>` tags.

Task: Draft Answer Generation
Answer the question using your existing knowledge base or the external information in the graph context (if provided) within `<answer>` tags.

Now, please analyze the following:

Question: {question}
Schema: {schema}
Graph Context: {graph_context}

Figure 3: The prompt template used in KG-Linker. Full prompts are provided in Appendix F.1.

Answer Generation.

The instruction template p is presented in Figure 3, where `{question}` is the input user query, `{schema}` is the KG schema, including node types and relations, and `{graph_context}` denotes the graph context retrieved from intermediate steps (if available). `{openCypher}` may be replaced with other graph query languages. Based on the LLM prompt, the generated graph artifacts are the following,

$$\tilde{\mathcal{E}} = \text{LLM(task = “Entity Extraction”)}, \quad (1)$$

$$\tilde{\mathcal{P}} = \text{LLM(task = “Path Identification”)}, \quad (2)$$

$$\tilde{\mathcal{Q}} = \text{LLM(task = “Graph Query Generation”)}, \quad (3)$$

$$\tilde{\mathcal{A}} = \text{LLM(task = “Draft Answering”)}, \quad (4)$$

where $\tilde{\mathcal{E}}$, $\tilde{\mathcal{P}}$, $\tilde{\mathcal{Q}}$, and $\tilde{\mathcal{A}}$ denote the generated entities, paths, graph query, and answer entities, respectively. Note that those graph artifacts can be generated by a single LLM call.

3.2 Entity Linking

Entity linking is fundamental to our framework, enabling graph traversal from accurate graph anchor points. Entity linking maps LLM-generated entities $\hat{\mathcal{E}} = \tilde{\mathcal{E}} \cup \tilde{\mathcal{A}}$ to actual entities $\mathcal{E}$ in the graph, e.g., “*Jamaican people*” $\mapsto$ “*Jamaica*”. We use the textual descriptions of the entities to perform entity linking and implement two complementary linking methods: fuzzy-string matching, and node embedding linking.

For each entity $\hat{e}_i \in \hat{\mathcal{E}}$, we retrieve top-$m$ entities present in the graph based on string match scores, node embedding similarity, or the union of both, where m is a hyperparameter, defaulting to $m = 3$. By default, we use string-based matching[2] and the encoder used for transforming entity names into embeddings is `bge-m3` (Chen et al., 2024). After performing entity linking, we obtain the linked entity set $\mathcal{E}$, where $|\mathcal{E}| \leq m \cdot |\hat{\mathcal{E}}|$ when using single linking method, or $|\mathcal{E}| \leq 2m \cdot |\hat{\mathcal{E}}|$ when using both string and embedding-based linking.

3.3 Path Retrieval

Path retrieval executes and validates the LLM-generated paths while discovering alternative routes in the graph that can lead to the required information. Formally, it takes as input the generated chain of relations $\tilde{\mathcal{P}}$, executes them on the graph, and returns the resulting intermediate entities along with the relation paths. We implement a `follow-paths` functionality which takes as input a sequence of relations $\tilde{\mathcal{P}}$ and source entities $\mathcal{E}$ and returns the executable paths $\mathcal{P}_f \subset \tilde{\mathcal{P}}$ over the graph $\mathcal{G}$ as

$$\mathcal{P}_f = \text{Follow-Paths}(\tilde{\mathcal{P}}, \mathcal{E}, \mathcal{G}). \quad (5)$$

We perform a breadth-first search starting from the source entities, exploring the graph until all valid paths generated by KG-Linker are found or no further valid paths remain. As source entities, we use the linked entities $\mathcal{E}$ of the entity linking step Section 3.2.

In addition, we retrieve the shortest paths connecting the extracted entities $\mathcal{E}$ and candidate answers $\mathcal{A}$ generated by the LLM (if non-empty). We implement a `shortest-paths` functionality which returns

$$\mathcal{P}_s = \text{Shortest-Paths}(\mathcal{E}, \mathcal{A}, \mathcal{G}) \quad (6)$$

[2]https://github.com/seatgeek/thefuzz

based on the candidate entities and answers, using the Dijkstra algorithm[3]. Path retrieval returns

$$\mathcal{P} = \mathcal{P}_f \cup \mathcal{P}_s. \tag{7}$$

3.4 Graph Query Retrieval

Graph query retrieval translates natural language into executable graph operations, crucial for handling complex queries in enterprise KGs. As a default, we prompt the LLM to generate a query $\tilde{Q}$ in openCypher language which is common in graph databases like Neptune and Neo4j. This step requires the graph to be stored in a format that supports the execution of the generated graph language queries. We implement an `execute-query` functionality which takes as input a query $\tilde{Q}$ and returns

$$\mathcal{Q}_a = \text{Execute-Query}(\tilde{Q}, \mathcal{G}), \tag{8}$$

which are the query execution results over $\mathcal{G}$.

3.5 Triplet Retrieval

Triplet retrieval complements the above methods by capturing relevant facts that might be missed through direct retrieval. Triplet retrieval operates on the triplet-level granularity of the KG, where triplets $\mathcal{T}$ are formatted as (h, r, t) and express natural language descriptions as "*head* $\rightarrow$ *relation* $\rightarrow$ *tail*", e.g., "*Jamaica* $\rightarrow$ *language_spoken* $\rightarrow$ *English*". We implement two approaches: agentic retrieval for step-by-step exploration, and scoring-based retrieval for more efficient semantic matching.

Agentic Retrieval. Given the linked entities $\mathcal{E}$ and the query q, agentic retrieval fetches the relevant triplets $\mathcal{T}_q$, using an LLM that iteratively traverses the KG based on q. We provide the algorithm in Algorithm 1 and the LLM prompts are in Appendix F.2.

At each iteration t, we obtain one-hop candidate triplets $\mathcal{T}_q^t$ and prompt the LLM to select the most relevant relations $\mathcal{R}_q^t$. The triplets are filtered based on the LLM's selection and subsequently we prompt the LLM to select the most relevant entities $\mathcal{E}_q^t$ to explore next. The process is terminated until self-termination (empty entity set to explore) or until we reach the maximum agentic iterations T_A, defaulting to $T_A = 3$. The agent returns the graph context $\mathcal{T}_q$ explored until termination.

Scoring-based Retrieval. Alternative to LLM-based traversal, scoring-based retrieval performs

[3]https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Algorithm 1 Agentic Retrieval.

```
1:  Input: Query $q$, Entities $\mathcal{E}$, Graph $\mathcal{G}$, Iterations $T_A$
2:  Optional: Context $\mathcal{T}_q^0$, else $\mathcal{T}_q^0 = \emptyset$
3:  $\mathcal{E}_q^0 \leftarrow \mathcal{E}$
4:  for all $t \in T_A$ do
5:      $\hat{\mathcal{T}}_q^t = \text{OneHop}(\mathcal{E}_q^{t-1}, \mathcal{G})$   {one-hop graph expansion}
6:      $\mathcal{R}_q^t = \text{LLM}(\text{"Select Relations"}, \hat{\mathcal{T}}_q^t, q)$   {relevant relations}
7:      $\hat{\mathcal{T}}_q^t \leftarrow \text{Filter}(\hat{\mathcal{T}}_q^t : r \in \mathcal{R}_q^t)$
8:      $\mathcal{T}_q^t \leftarrow \mathcal{T}_q^{t-1} \cup \hat{\mathcal{T}}_q^t$   {updated context}
9:      $\mathcal{E}_q^t = \text{LLM}(\text{"Select Entities"}, \mathcal{T}_q^t, q)$   {next entities}
10:     if $\mathcal{E}_q^t == \emptyset$ then
11:         break   {self-termination}
12:     end if
13: end for
14: $\mathcal{T}_q \leftarrow \mathcal{T}_q^t$.
15: Output: Return $\mathcal{T}_q$.
```

top-k triplet selection $\mathcal{T}_q$ based on question-triplet semantic similarity, where $|\mathcal{T}_q| = k$ and k is a hyperparameter. In Appendix A, we present an implementation that uses reranker models to select relevant triplets within L-hop distance from linked entities. Our framework also supports a more efficient embedding-based retrieval approach.

3.6 Iterative Process

BYOKG-RAG's iterative process enables progressive refinement of retrieved context. Each iteration combines information from multiple retrieval methods, allowing the LLM to generate more accurate artifacts based on accumulated context. BYOKG-RAG's iterative process is presented Algorithm 2. After graph linking and retrieval, we collect the retrieved context

$$\mathcal{C} = \mathcal{T}_q \cup \mathcal{P}_f \cup \mathcal{P}_s \cup \mathcal{Q}_a, \tag{9}$$

and verbalize it with predefined templates. As shown in Algorithm 2, the context $\mathcal{C}$ is used as input to KG-Linker's prompt (Figure 3) to iteratively refine its generations $\tilde{\mathcal{E}}$, $\tilde{\mathcal{P}}$, $\tilde{\mathcal{Q}}$, and $\tilde{\mathcal{A}}$. By default, we have $T_R = 2$ iterations, but we include self-termination when the KG-Linker does not provide new entities to link. Finally, we collect the graph context $\mathcal{C}$ as input for the final KGQA task.

Algorithm 2 BYOKG-RAG Framework.

```
1: Input: Prompt $p$, Query $q$, Schema $S$, Iterations $T_R$
2: $\mathcal{C} \leftarrow \emptyset$
3: for all $t \in T_R$ do
4:     $\tilde{\mathcal{E}}, \tilde{\mathcal{P}}, \tilde{\mathcal{Q}}, \tilde{\mathcal{A}} = \text{LLM}(p, q, S, \mathcal{C})$
5:     $\mathcal{E}, \mathcal{A} = \text{EntityLink}(\tilde{\mathcal{E}}, \tilde{\mathcal{A}})$
6:     if $\tilde{\mathcal{E}} = \emptyset$: break
7:     $\mathcal{P} = \text{PathRetrieve}(\tilde{\mathcal{P}}, \tilde{\mathcal{E}}, \tilde{\mathcal{A}})$
8:     $\mathcal{Q}_a = \text{QueryRetrieve}(\tilde{\mathcal{Q}})$
9:     $\mathcal{T}_q = \text{TripletRetrieve}(q, \mathcal{E} \cup \mathcal{A}, \mathcal{C})$
10:    $\mathcal{C} \leftarrow \mathcal{T}_q \cup \mathcal{P} \cup \mathcal{Q}_a \cup \mathcal{C}$
11: end for
12: $\mathcal{A} = \text{LLM}(q, \mathcal{C})$.
13: Output: Answers $\mathcal{A}$.
```

4 Experimental Setup

4.1 Datasets & Metrics

We evaluate KGQA benchmarks spanning different backbone KGs. The statistics are provided in Appendix B.

WebQSP (Yih et al., 2015) and ComplexWebQuestions (**CWQ**) (Talmor and Berant, 2018) involve questions answerable over the Freebase KG (Bollacker et al., 2008). WebQSP and CWQ contain 1,628 and 3,531 questions, respectively, and involve general knowledge about entities, requiring 1-2 hop and 1-4 hop KG traversal, respectively. WebQSP-IH and CWQ-IH refer to in-house subsets, containing 500 questions each from the original set. For pre-processing the Freebase KG, we follow standard steps from previous works (He et al., 2021).

CronQuestions (Saxena et al., 2021) are questions that require temporal reasoning over a KG subset of Wikidata, which contains temporal information (Lacroix et al., 2020). As the tempral KG contains quadruples (*head*, *relation*, *tail*, *timestamp*), we convert them to triplets (*head*, *relation: timestamp*, *tail*). We focus on CronQuestions's subset of complex questions and we sample 200 per question type (before/after, first/last, time join) resulting into 600 questions.

MedQA (Jin et al., 2021) consists of questions extracted from USMLE exams. We combine MedQA with Disease DrugBank KG (Wishart et al., 2018), following previous works (Yasunaga et al., 2021). Since the KG may not be relevant for all questions, we filter for questions whose answer candidates appear in the KG (via entity matching). Additionally, we remove MedQA's multiple choices from the LLM's context to increase the benchmark's difficulty.

The **Text2cypher** dataset (Ozsoy et al., 2024) emulates text2cypher queries encountered in enterprise KGs, such as Northwind. As not all ground-truth queries are executable over the KGs provided in the benchmark, we filter for questions whose ground-truth answers can be retrieved from the graphs.

Metrics. Similar to previous works, we report the *Hit* metric for WebQSP, CWQ, and CronQuestions, which measures if the LLM generates any correct answer via exact string matching. In MedQA, we report Hit@2 (*H@2*) by augmenting the LLM generation based on the retrieved KG context with its original prediction, and measure if correct answers are found. For Text2cypher, we use Claude-Sonnet-3.5 as an LLM-as-a-judge (*LLMaaJ*) that scores if the executed results match the ground-truth results (1.0: correct, 0.5: partially correct, 0.0: incorrect). We also evaluate retrieval accuracy by Recall@k which measures if correct answers are found within top-k retrieval results.

4.2 Competing Methods

Baselines. As key baselines, we include components present within BYOKG-RAG's framework. *Vanilla LLM* is the approach of direct QA without KG context. *LLM+graph-query* prompts the LLM to generate an executable graph query, and augments the LLM's direct predictions along with the execution results. For MedQA, we prompt the LLM to generate answers based on the execution results. *Text-based Retrieval* (Appendix A.1) retrieves top-k triplets based on embedding similarity, while *Graph Reranker* (Appendix A.2) uses a reranker to select top-k triplets within $L = 2$ hops of linked entities. We use $k = 50$ for Freebase, $k = 10$ for the rest KGs. *Agentic Traversal* (Algorithm 1) is the LLM-based KG traversal and retrieval as described in Algorithm 1. Note that Graph Reranker and Agentic Traversal require entity linking as a first step, and we also incorporate entity linking in LLM+graph-query for better query generation. We employ the same LLM across methods for downstream KGQA, after graph retrieval.

For a fair comparison, we focus on zero/few-shot methods and evaluate against established KG agents, including those using agentic traversal (Jiang et al., 2023a; Sun et al., 2024; Sui et al., 2024) and LLM-guided graph retrieval (Li et al.,

Table 1: Performance comparison of zero-shot methods across different KGQA benchmarks. We bold the best and underline the second-best method.

	Freebase-KG WebQSP-IH Hit (%)	Freebase-KG CWQ-IH Hit (%)	Temp-Wikidata CronQuestions Hit (%)	DiseaseDB-KG MedQA H@2 (%)	Cypher-KG Northwind LLMaaJ (%)
			Claude-Sonnet-3.5		
Vanilla LLM	70.4	54.2	19.2	57.9	0.0
LLM + graph-query$^{\alpha}$	75.2	54.3	19.2	<u>62.5</u>	<u>55.3</u>
Text-based Retrieval$^{\beta}$	70.8	56.8	<u>59.8</u>	58.1	0.7
Graph Reranker$^{\gamma}$	76.0	63.0	41.8	58.2	3.4
Agentic Traversal$^{\delta}$	<u>86.2</u>	<u>69.3</u>	57.3	59.2	3.4
BYOKG-RAG	**86.6**	**73.6**	**65.5**	**65.0**	**64.9**
			Claude-Haiku-3.5		
Vanilla LLM	67.2	48.2	15.0	44.7	0.0
LLM + graph-query$^{\alpha}$	76.4	48.4	15.0	<u>49.8</u>	<u>52.6</u>
Text-based Retrieval$^{\beta}$	74.6	58.4	<u>59.6</u>	47.6	0.0
Graph Reranker$^{\gamma}$	75.6	62.7	38.7	46.6	3.4
Agentic Traversal$^{\delta}$	<u>81.0</u>	<u>64.2</u>	40.8	46.2	3.4
BYOKG-RAG	**82.8**	**66.8**	**61.9**	**54.2**	**59.1**
			Llama-3.3-70B		
Vanilla LLM	68.2	51.8	14.8	45.1	0.0
LLM + graph-query$^{\alpha}$	69.2	51.8	14.8	<u>49.1</u>	<u>58.6</u>
Text-based Retrieval$^{\beta}$	73.2	60.1	<u>60.2</u>	47.3	0.0
Graph Reranker$^{\gamma}$	75.6	62.9	38.9	46.5	3.4
Agentic Traversal$^{\delta}$	<u>81.9</u>	<u>67.0</u>	45.8	45.1	3.4
BYOKG-RAG	**82.4**	**67.2**	**62.2**	**53.1**	**68.9**

$^{\alpha}$We combine LLM generation with text2cypher generation.
$^{\beta}$We retrieve top-k triplets based on question-triplet embedding similarity (Appendix A.1).
$^{\gamma}$We retrieve top-k triplets via graph-based reranking (Appendix A.2).
$^{\delta}$The LLM agent iteratively explores relevant entities and relations until self-termination (Algorithm 1).

2023; Dong et al., 2025).

BYOKG-RAG Implementation. We implement BYOKG-RAG with default hyperparameters as described within subsections of Section 3. We use entity linking with top-$m = 3$ via combining string and embedding-based matching. When executing graph queries, we store the KG in NeptuneAnalytics[4] and query the database[5] with openCypher[6]. We retrieve triplets via agentic-based retrieval, and we combine it with top-$k = 10$ efficient text-based retrieval. We also present results for 'BYOKG-RAG (scoring)', which substitutes the agentic retrieval with the scoring-based approach in Appendix A.2. For the LLMs used in KG-Linker, we experiment with Claude-Sonnet-3.5, Claude-Haiku-3.5 (Anthropic, 2024), and Llama-3.3-70B (Grattafiori et al., 2024), accessed via Bedrock API[7].

[4]https://aws.amazon.com/neptune/

[5]Our implementation also supports a local storage format, where the graph is represented as a dictionary mapping nodes to their one-hop triplets. When using local storage, the graph query generation prompt from KG-Linker is omitted, while all other functionalities remain unchanged.

[6]https://opencypher.org/

5 Experimental Results

5.1 Main Results

Table 1 demonstrates the effectiveness of BYOKG-RAG across diverse KGQA benchmarks. On the Freebase-KG benchmarks, BYOKG-RAG achieves 86.6% and 73.6% Hit rates on WebQSP-IH and CWQ-IH respectively, outperforming the best baseline methods (86.2% and 69.3%). The improvement is also pronounced on specialized KGs: for temporal reasoning on CronQuestions, BYOKG-RAG achieves a 65.5% Hit rate, showing a significant gain over text-based retrieval (59.8%). Similarly, on the medical domain MedQA benchmark, BYOKG-RAG improves H@2 performance to 65.0% compared to graph-query's 59.2%. The framework's generalization capability is further demonstrated on enterprise KGs, where it achieves 64.9% LLMaaJ score on Northwind, outperforming graph-query (55.3%). Table 1 also evalu-

[7]https://aws.amazon.com/bedrock/

Table 2: Performance comparison of BYOKG-RAG with state-of-the-art zero/few-shot KGQA methods on WebQSP and CWQ datasets. We provide performance numbers of competing methods as reported in the corresponding literature. In addition, we report average number of LLM calls as a metric in terms of LLM utilization (the lower, the better).

	Freebase-KG			
	WebQSP		CWQ	
	Hit (%)	#LLM Calls	Hit (%)	#LLM Calls
Vanilla LLM	62.0	1	42.1	1
CoT LLM	72.1	1	53.0	1
KD-CoT (Wang et al., 2023)	68.6	2	55.7	4
StructGPT (Jiang et al., 2023a)	72.6	4	54.2	4
KB-BINDER (Li et al., 2023)	74.4	6	–	–
ToG (Sun et al., 2024)	82.6	11.2	67.6	14.3
EffiQA (Dong et al., 2025)	82.9	4.4	69.5	6.5
FiDeLiS (Sui et al., 2024)	84.1	10.7	**71.4**	15.2
BYOKG-RAG (scoring)	85.4	2	68.7	3
BYOKG-RAG (agentic)	**87.1**	4.5	71.1	6.3

ates BYOKG-RAG using different LLM backbones. While Claude-Sonnet-3.5 performs best overall, the performance improvements are consistent across Claude-Haiku-3.5 and Llama-3.3-70B, indicating that our framework's benefits are robust across different LLMs. Using Claude-Sonnet-3.5, BYOKG-RAG outperforms the strongest baseline across benchmarks (agentic retrieval for WebQSP-IH/CWQ-IH, text-based retrieval for CronQuestions, and graph querying for MedQA/Northwind) by **4.5% points**, on average.

In Table 2, we compare BYOKG-RAG against state-of-the-art zero/few-shot KGQA methods on WebQSP and CWQ benchmarks. BYOKG-RAG achieves the best performance on WebQSP (87.1% Hit rate) and comparable results on CWQ (71.1%), while requiring fewer LLM calls than competing methods. Notably, BYOKG-RAG's more efficient scoring-based variant maintains strong performance while using only 2-3 LLM calls.

5.2 Ablation Studies

Our studies in Tables 1, 3, 4, and 8 analyze BYOKG-RAG's components.

Table 1 reveals the effectiveness of different retrieval methods in BYOKG-RAG across diverse KG types. On Freebase benchmarks (WebQSP-IH, CWQ-IH), agentic traversal shows strong performance (86.2%, 69.3% with Claude-Sonnet-3.5), indicating step-by-step graph exploration is well-suited for multi-hop queries. For temporal reasoning in CronQuestions, text-based retrieval proves effective (60.2% Hit with Llama-3.3-70B), suggesting temporal information can be captured through semantic matching. For the medical domain (MedQA), the combination of different retrieval methods in BYOKG-RAG yields notable improvements (65.0% H@2 with Claude-Sonnet-3.5). Most distinctively, on enterprise KGs (Northwind), the ability to generate and execute Cypher queries proves crucial, with BYOKG-RAG achieving 64.9% accuracy and LLM+graph-query achieving 55.3% with Claude-Sonnet-3.5.

Table 3: Ablation study on different linking methods (Claude-Sonnet-3.5 LLM).

	CWQ-IH Hit (%)	CronQuestions Hit (%)
Vanilla LLM	54.2	19.2
Scoring-based Retrieval:		
w/ Entity Linking only	63.0	59.8
w/ KG-Linker	71.6	63.2
Agentic Retrieval:		
w/ Entity Linking only	69.3	57.3
w/ KG-Linker	73.6	65.5

Table 4: Comparison of different retrieval methods[†] on CWQ. '#KG Tokens' denotes the median number of KG tokens[‡] retrieved as context for the LLM.

	Recall@10	#KG Tokens[‡]	#Train Data
RoG	54.5	201	30.5K
SubgraphRAG	58.7	1,442	27.6K
GNN-RAG	64.1	114	27.6K
BYOKG-RAG:			
- Scoring	60.6	1,483	0
- Agentic	70.5	396	0

[†]RoG (Luo et al., 2024a), SubgraphRAG (Li et al., 2024b), and GNN-RAG (Mavromatis and Karypis, 2024) are fine-tuned graph retrievers.
[‡]We count tokens via https://github.com/openai/tiktoken.

Table 3 demonstrates the value of multi-strategy linking beyond entity matching alone. On CWQ-IH, KG-Linker improves performance from 63.0% to 71.6% with scoring-based retrieval and from 69.3% to 73.6% with agentic retrieval. Similar gains are observed on CronQuestions.

Table 4 compares retrieval effectiveness across different methods. While supervised methods like RoG, SubgraphRAG, and GNN-RAG achieve Recall@10 scores of 54.5-64.1% using substantial training data (27.6K-30.5K examples), BYOKG-RAG performs competitively without any training data. BYOKG-RAG's agentic variant achieves

Table 5: Case study on ByoKG-RAG's iterative retrieval.

Question	In what years did Stan Kasten's organization win the World Series?
Answer	1963 World Series | 1988 World Series | 1965 World Series | 1981 World Series | 1959 World Series
ByoKG-RAG (1st iteration)	m.0_yv0g3 -> organization.leadership.person -> Stan Kasten m.0_yv0g3 -> organization.leadership.organization -> Los Angeles Dodgers Stan Kasten -> business.board_member.leader_of > m.0_yv0g3-> organization.leadership.organization -> Los Angeles Dodgers
ByoKG-RAG (2nd iteration)	Los Angeles Dodgers -> sports.sports_team.championships -> **1963 World Series | 1988 World Series | 1965 World Series | 1981 World Series | 1959 World Series**

70.5% Recall@10 while retrieving 396 tokens in total, while its scoring-based variant achieves 60.6% with 1,483 tokens, demonstrating effective zero-shot retrieval with moderate context sizes.

Furthermore, we provide a case study on how ByoKG-RAG iteratively improves its retrieval in Table 5.

Additional experiments and cases studies are in Appendix D and E.

6 Conclusion

We introduced ByoKG-RAG, a framework that enhances KGQA by combining LLMs with graph retrieval tools. Through extensive experiments across five benchmarks, we demonstrated that multi-strategy graph retrieval matters: ByoKG-RAG leverages multiple retrieval methods to achieve superior performance (4.5 percentage points improvement over the strongest baselines) while generalizing effectively to KGs with diverse semantics.

Limitations

ByoKG-RAG leverages diverse graph tools to retrieve context, enabling more effective KGQA across custom KGs. However, without proper context pruning mechanisms, the retrieved context may become too lengthy and potentially confuse LLMs with limited context-handling capabilities (Liu et al., 2023). Furthermore, our current work considers the KG as the sole source of external information. Future work could expand ByoKG-RAG's capabilities by incorporating additional information sources, such as text databases (Wu et al., 2024), thereby enhancing the framework's applications.

References

Dhruv Agarwal, Rajarshi Das, Sopan Khosla, and Rashmi Gangadharaiah. 2023. Bring your own kg: Self-supervised program synthesis for zero-shot kgqa. *arXiv preprint arXiv:2311.07850*.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku.

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*, volume 39, pages 23424–23432.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, and 1 others. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

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 preprint arXiv:2502.14902*.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*.

Zixuan Dong, Baoyun Peng, Yufei Wang, Jia Fu, Xiaodong Wang, Xin Zhou, Yongxue Shan, Kangchen Zhu, and Weiguo Chen. 2025. EffiQA: Efficient question-answering with strategic multi-model collaboration on knowledge graphs. In *Proceedings of* *the 31st International Conference on Computational Linguistics*.

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, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Jianqi Gao, Jian Cao, Ranran Bu, Nengjun Zhu, Wei Guan, and Hang Yu. 2025. Promoting knowledge base question answering by directing llms to generate task-relevant logical forms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23914–23922.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Yu Gu, Xiang Deng, and Yu Su. 2023. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada. Association for Computational Linguistics.

Yu Gu and Yu Su. 2022. Arcaneqa: Dynamic program induction and contextualized encoding for knowledge base question answering. *arXiv preprint arXiv:2204.08109*.

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

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *arXiv preprint arXiv:2405.14831*.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In *Proceedings of the 14th ACM international conference on web search and data mining*, pages 553–561.

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

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023a. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. 2024. Kg-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. *arXiv preprint arXiv:2402.11163*.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. 2023b. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. In *International Conference on Learning Representations*.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2021. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*.

Sumin Jo, Junseong Choi, Jiho Kim, and Edward Choi. 2025. R2-kg: General-purpose dual-agent framework for reliable reasoning on knowledge graphs. *arXiv preprint arXiv:2502.12767*.

Jiho Kim, Yeonsu Kwon, Yohan Jo, and Edward Choi. 2023. KG-GPT: A general framework for reasoning on knowledge graphs using large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*.

Timothée Lacroix, Guillaume Obozinski, and Nicolas Usunier. 2020. Tensor decompositions for temporal knowledge base completion. *arXiv preprint arXiv:2004.04926*.

Yunshi Lan and Jing Jiang. 2020a. Query graph generation for answering multi-hop complex questions from knowledge bases. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 969–974. Association for Computational Linguistics.

Yunshi Lan and Jing Jiang. 2020b. Query graph generation for answering multi-hop complex questions from knowledge bases. Association for Computational Linguistics.

Meng-Chieh Lee, Qi Zhu, Costas Mavromatis, Zhen Han, Soji Adeshina, Vassilis N Ioannidis, Huzefa Rangwala, and Christos Faloutsos. 2024. Hybgrag: Hybrid retrieval-augmented generation on textual and relational knowledge bases. *arXiv preprint arXiv:2412.16311*.

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.

Belinda Z Li, Sewon Min, Srinivasan Iyer, Yashar Mehdad, and Wen-tau Yih. 2020. Efficient one-pass end-to-end entity linking for questions. *arXiv preprint arXiv:2010.02413*.

Kun Li, Tianhua Zhang, Xixin Wu, Hongyin Luo, James Glass, and Helen Meng. 2024a. Decoding on graphs: Faithful and sound reasoning on knowledge graphs through generation of well-formed chains. *arXiv preprint arXiv:2410.18415*.

Mufei Li, Siqi Miao, and Pan Li. 2024b. Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation. *arXiv preprint arXiv:2410.20724*.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6966–6980. Association for Computational Linguistics.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

Haoran Luo, Yikai Guo, Qika Lin, Xiaobao Wu, Xinyu Mu, Wenhao Liu, Meina Song, Yifan Zhu, Luu Anh Tuan, and 1 others. 2025a. Kbqa-o1: Agentic knowledge base question answering with monte carlo tree search. *arXiv preprint arXiv:2501.18922*.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024a. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *International Conference on Learning Representations*.

Linhao Luo, Zicheng Zhao, Chen Gong, Gholamreza Haffari, and Shirui Pan. 2024b. Graph-constrained reasoning: Faithful reasoning on knowledge graphs with large language models. *arXiv preprint arXiv:2410.13080*.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Dinh Phung, Chen Gong, and Shirui Pan. 2025b. Gfm-rag: Graph foundation model for retrieval augmented generation. *arXiv preprint arXiv:2502.01113*.

Costas Mavromatis and George Karypis. 2022. ReaRev: Adaptive reasoning for question answering over knowledge graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2447–2458, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Armin Oliya, Amir Saffari, Priyanka Sen, and Tom Ayoola. 2021. End-to-end entity resolution and question answering using differentiable knowledge graphs. *arXiv preprint arXiv:2109.05817*.

Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. 2024. Text2cypher: Bridging natural language and graph databases. *arXiv preprint arXiv:2412.10064*.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921*.

Bhaskarjit Sarmah, Dhagash Mehta, Benika Hall, Rohan Rao, Sunil Patel, and Stefano Pasquali. 2024. Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 608–616.

Apoorv Saxena, Soumen Chakrabarti, and Partha Talukdar. 2021. Question answering over temporal knowledge graphs. *arXiv preprint arXiv:2106.01515*.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. 2021. Transfernet: An effective and transparent framework for multi-hop question answering over relation graph. *arXiv preprint arXiv:2104.07302*.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje F Karlsson, Tingting Ma, Yuzhong Qu, and Chin-Yew Lin. 2022. Tiara: Multi-grained retrieval for robust question answering over large knowledge bases. *arXiv preprint arXiv:2210.12925*.

Yuan Sui, Yufei He, Nian Liu, Xiaoxin He, Kun Wang, and Bryan Hooi. 2024. Fidelis: Faithful reasoning in large language model for knowledge graph question answering. *arXiv preprint arXiv:2405.13873*.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4231–4242. Association for Computational Linguistics.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph. In *International Conference on Learning Representations*.

Yawei Sun, Lingling Zhang, Gong Cheng, and Yuzhong Qu. 2020. Sparqa: skeleton-based semantic parsing for complex questions over knowledge bases. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8952–8959.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

Keheng Wang, Feiyu Duan, Sirui Wang, Peiguang Li, Yunsen Xian, Chuantao Yin, Wenge Rong, and Zhang Xiong. 2023. Knowledge-driven cot: Exploring faithful reasoning in llms for knowledge-intensive question answering. *arXiv preprint arXiv:2308.13259*.

David S Wishart, Yannick D Feunang, An C Guo, Elvis J Lo, Ana Marcu, Jason R Grant, Tanvir Sajed, Daniel Johnson, Carin Li, Zinat Sayeeda, and 1 others. 2018. Drugbank 5.0: a major update to the drugbank database for 2018. *Nucleic acids research*, 46(D1):D1074–D1082.

Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis Ioannidis, Karthik Subbian, James Y Zou, and Jure Leskovec. 2024. Stark: Benchmarking llm retrieval on textual and relational knowledge bases. *Advances in Neural Information Processing Systems*, 37:127129–127153.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

Vikas Yadav and Steven Bethard. 2019. A survey on recent advances in named entity recognition from deep learning models. *arXiv preprint arXiv:1910.11470*.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. Qa-gnn: Reasoning with language models and knowledge graphs for question answering. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 2022. RNG-KBQA: Generation augmented iterative ranking for knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6032–6043. Association for Computational Linguistics.

Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the Joint Conference of the 53rd Annual Meeting of the ACL and the 7th International Joint Conference on Natural Language Processing of the AFNLP*.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Wang, Zhiguo Wang, and Bing Xiang. 2022. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases. *arXiv preprint arXiv:2210.00063*.

Han Zhang, Langshi Zhou, and Hanfang Yang. 2025. Learning to retrieve and reason on knowledge graph through active self-reflection. *arXiv preprint arXiv:2502.14932*.

A Scoring-based Triplet Retrieval

Alternatively to the the LLM-based traversal of Section 3.5, scoring-based retrieval performs top-k triplet selection $\mathcal{T}_q$ based on question-triplet semantic similarity, where $|\mathcal{T}_q| = k$ and k is a hyperparameter.

A.1 Text-based Retrieval

Text-based retrieval converts KG triplets $\mathcal{T}$ into natural language statements and retrieves the top-k triplets based on their semantic similarity to the question. Since computing embeddings for all triplets in real-world KGs is computationally prohibitive, we adopt an efficient embedding decomposition approach (Li et al., 2024b). Specifically, we decompose triplets into head, relation, and tail components, pre-compute embeddings for individual nodes and relations, and aggregate their similarity scores with the question. Formally, this approach is expressed as:

$$\begin{aligned} \mathcal{T}_q = \underset{(h,r,t)\in\mathcal{T}}{\arg\text{top-}k} \big(&\mathsf{Embed}(q,h)+ \\ &\mathsf{Embed}(q,r)+ \\ &\mathsf{Embed}(q,t)\big), \end{aligned} \tag{10}$$

where Embed computes cosine similarity using pretrained embedding models (Chen et al., 2024), implemented efficiently via FAISS (Douze et al., 2024)[8] .

A.2 Graph Reranker

Given linked entities $\mathcal{E}$ (Section 3.2) and query q, the graph reranker module retrieves the top-k most

[8]https://huggingface.co/

relevant triplets by reranking (Gao et al., 2023) triplets within an L-hop neighborhood of $\mathcal{E}$.

First, we collect triplets $\mathcal{T}^{(L)}$ within an L-hop distance from $\mathcal{E}$ as

$$\mathcal{T}^{(L)} = \bigcup_{e\in\mathcal{E}} \{(h,r,t) \in \mathcal{G}_e^{(L)}\} \tag{11}$$

where $\mathcal{G}_e^{(L)}$ denotes the L-hop subgraph starting from entity e, and L is a hyperparameter that defaults to $L = 2$.

Since the number of triplets $|\mathcal{T}^{(L)}|$ can be excessively large in dense graphs, we implement a two-step pruning process: first filtering out triplets with irrelevant relations, then eliminating irrelevant triplets from the remaining set. We use a lightweight reranker model (`bge-reranker-base` cross-encoder (Xiao et al., 2023)) to maintain the most relevant relations

$$\mathcal{R}_q = \arg\,\text{top-}k_r\big(\texttt{Reranker}(q, \mathcal{R}^{(L)})\big), \tag{12}$$

which returns the top-k_r most relevant relations to the question q, where $\mathcal{R}^{(L)}$ is the relation set in $\mathcal{T}^{(L)}$, $|\mathcal{R}_q| = k_r$, and $k_r = 20$ by default. Next, we collect triplets of which relations are present in $\mathcal{R}_q$ t9 $\mathcal{T}_r^{(L)} = \{(h,r,t) \in \mathcal{T}^{(L)} : r \in \mathcal{R}_q\}$, and prune them again to $\mathcal{T}_q^{(L)}$ by

$$\mathcal{T}_q^{(L)} = \arg\,\text{top-}k_t\big(\texttt{Reranker}(q, \mathcal{T}_r^{(L)})\big), \tag{13}$$

where $k_t = 100$ by default. Here, we transform the triplets to a natural language description for the reranker with a predefined template. We use a more powerful reranker (`bge-reranker-v2-minicpm-layerwise` cross-encoder) to obtain the final top-k most relevant triplets $\mathcal{T}_q$ as

$$\mathcal{T}_q = \arg\,\text{top-}k\big(\texttt{Reranker2}(q, \mathcal{T}_q^{(L)})\big), \tag{14}$$

where $|\mathcal{T}_q| = k$.

B Dataset Statistics

We provide the dataset statistics in Table 6, along with question examples from each benchmark.

C Other Evaluation Metrics

LLMaaJ. We conducted preliminary evaluations using LLM-as-a-Judge (LLMaaJ) with Claude-Sonnet-3.5 on the WebQSP-IH and CWQ-IH datasets. The results show an alignment between the Hit@1 and LLMaaJ metrics for ByoKG-RAG: 86.6% Hit@1 vs. 86.8% LLMaaJ on WebQSP-IH, and 73.6% Hit@1 vs. 74.2% LLMaaJ on CWQ-IH.

F1. F1 metric is suitable when question have multiple answers, common in KGQA tasks. ByoKG-RAG is designed to support path-based retrieval and graph query execution, both of which naturally accommodate multiple valid answers. When ByoKG-RAG successfully generates a correct path or Cypher query, it retrieves all associated anchor nodes, returning the full set of valid answers for the question. We conducted a preliminary experiment on the WebQSP-IH dataset using path retrieval. On the subset of questions where the generated paths are executable, we found that the F1 score closely aligns with the Hit@1 metric (80.2% Hit@1 vs. 78.8% F1).

D Additional Experiments

Table 7 presents the refinement effectiveness (Section 3.6) of ByoKG-RAG. In all cases presented, ByoKG-RAG's refinement improves the original graph retrieval methods considerably, highlighting the importance of iterative retrieval. To further analyze the impact of iteration count T_R, we ran additional experiments with $T_R = 3$. KG-Linker includes a self-termination mechanism (Algorithm 2), allowing the retrieval process to stop early once sufficient information has been gathered. For WebQSP-IH, KG-Linker terminates after an average of 1.1 iterations, achieving the same performance as with $T_R = 2$. For CWQ-IH, it terminates after an average of 2.3 iterations, with a marginal performance gain (73.6% for $T_R = 2$ vs. 73.9% for $T_R = 3$). These findings suggest that the self-termination mechanism is effective in adapting to each query.

Table 8 presents a latency analysis, showing that while ByoKG-RAG introduces moderate overhead 2.4× relative to the baseline), it achieves superior performance than agentic retrieval (1.6×).

E Case Studies

We provide case studies on how ByoKG-RAG improves retrieval in Table 9. In CronQuestion, ByoKG-RAG retrieves the correct answer while Agentic retrieval does not. In CWQ, ByoKG-RAG finds the answers step-by-step (1st vs. 2nd iteration), while path retrieval also find relevant information. In MedQA, different components of ByoKG-RAG retrieve information relevant to the correct answer. In Northwind, ByoKG-RAG

Table 6: Summary of datasets.

	KG	#Triplets	# QA	Type
WebQSP	Freebase	16.3M	1,638	General Knowledge
WebQSP-IH	Freebase	1.5M	500	General Knowledge
CWQ	Freebase	35.3M	3,531	General Knowledge (multi-hop)
CWQ-IH	Freebase	1.5M	500	General Knowledge (multi-hop)
CronQuestions	Temp-Wikidata	325K	600	Temporal Reasoning
MedQA	Disease DrugBank	44.5K	227	Medical Knowledge
Northwind	Text2cypher KG	3K	178	In-Domain Query

	Examples
WebQSP	*"In which state was the battle of Antietam fought?"*
WebQSP-IH	*"What language do Jamaican people speak?"*
CWQ	*"Who was an actor in the film that Christopher Lee Foster was a crew member of as a kid?"*
CWQ-IH	*""Lou Seal is the mascot for the team that last won the World Series when?"*
CronQuestions	*"Who preceded Diana Laidlaw as Minister for Transport?"*
MedQA	*"A 55-year-old woman has [condition]. What is best treatment?"*
Northwind	*"Which categories have products with a unit price less than $10?"*

Table 7: Ablation study on retrieval refinement.

	CWQ-IH Hit (%)
Graph Reranker	63.0
+BYOKG-RAG refinement	68.8
	WebQSP-IH Hit (%)
Path Retrieval	75.2
+BYOKG-RAG refinement	80.2
	NorthWind LLMaaJ (%)
Graph-Query	55.3
+ BYOKG-RAG refinement	64.9

generates an executable cypehr query while graph-query does not.

F Prompts

F.1 KG-Linker

The prompts per task employed in KG-Linker (Figure 3) are presented in more details in Figure 4 (entity extraction), Figure 5 (path extraction), Figure 6 (graph query generation), and Figure 7 (answer generation).

F.2 Agentic Traversal

The prompts used in agentic KG traversal are presented in Figure 8 (relation selection) and Figure 9 (entity selection).

Table 8: Comparison of average latency overhead across different methods on CWQ-IH. Results are normalized relative to the fastest method's per-query latency time, using Claude-Sonnet-3.5 as the backbone LLM.

	Hit (%)	Latency unit[†]
LLM+graph-query	54.4	1.0×
Agentic Retrieval	69.3	1.6×
BYOKG-RAG	73.6	2.4×

[†]We measured system latency on an EC2 g5.16xlarge instance, using Bedrock API for LLM queries. We acknowledge potential optimizations, including faster entity linking, deploying reranker and embedding models on higher-performance hardware, batch API calls, and prompt caching.

Table 9: Case studies on ByoKG-RAG's benefits in KGQA.

Question (CronQuestions)	When Francis Pym, Baron Pym had been the Member of the 48th Parliament of the United Kingdom, who had been the Minister of Finance of Norway?
Answer	Per Kleppe
Agent Retrieval	Francis Pym, Baron Pym -> position held (1979 - 1983) -> Member of the 48th Parliament of the United Kingdom Francis Pym, Baron Pym -> position held (1974 - 1979) -> Member of the 47th Parliament of the United Kingdom [...]
ByoKG-RAG	Member of the 48th Parliament of the United Kingdom -> position held (1979 - 1983) -> Francis Pym, Baron Pym Minister of Finance of Norway -> position held (1973 - 1979) -> **Per Kleppe** [...]
Question (CWQ)	In what years did Stan Kasten's organization win the World Series?
Answer	1963 World Series | 1988 World Series | 1965 World Series | 1981 World Series | 1959 World Series
ByoKG-RAG (1st iteration)	m.0_yv0g3 -> organization.leadership.person -> Stan Kasten m.0_yv0g3 -> organization.leadership.organization -> Los Angeles Dodgers
ByoKG-RAG (2nd iteration)	Los Angeles Dodgers -> sports.sports_team.championships -> **1963 World Series | 1988 World Series | 1965 World Series | 1981 World Series | 1959 World Series**
ByoKG-RAG (path retrieval)	Stan Kasten -> business.board_member.leader_of > m.0_yv0g3-> organization.leadership.organization -> Los Angeles Dodgers
Question (MedQA)	A 59-year-old overweight woman presents to the urgent care clinic with the complaint of severe abdominal pain for the past 2 hours. She also complains of a dull pain in her back with nausea and vomiting several times. Her pain has no relation with food. Her past medical history is significant for recurrent abdominal pain due to cholelithiasis. Her father died at the age of 60 with some form of abdominal cancer. Her temperature is 37C (98.6F), respirations are 15/min, pulse is 67/min, and blood pressure is 122/98 mm Hg. Physical exam is unremarkable. However, a CT scan of the abdomen shows a calcified mass near her gallbladder. Which of the following diagnoses should be excluded first in this patient
Answer	Gallbladder cancer
ByoKG-RAG (agent)	Gallbladder cancer -> may cause -> Abdominal mass | Cholestatic jaundice | Liver metastases
ByoKG-RAG (path / query retrieval)	Aortic aneurysm, abdominal -> may cause > Abdominal mass-> may cause -> Gallbladder cancer
Question (Northwind)	What is the average 'unitPrice' of products that have been ordered in quantities greater than 10?
Graph-Query	Query: MATCH (p:Product)<-[:PART_OF]-(o:Order) WHERE o.quantity > 10 RETURN avg(p.unitPrice) AS averageUnitPrice Result: None
ByoKG-RAG	Retrieval: (638: orderID: 10666, shipName: Richter Supermarkt) -> ORDERS -> (unitPrice: 123.79, productName: Thfcringer Rostbratwurst) Query: MATCH (o:Order)-[r:ORDERS]->(p:Product) WHERE r.quantity > 10 RETURN avg(p.unitPrice) AS averageUnitPrice Result: avgPrice: 26.0989786683906

KG-Linker's Entity Extraction Prompt

Task: Entity Extraction
Extract all topic entities from the question (people, places, organizations, concepts, etc.) that will need to be matched in the graph database.
- Include all entities that are directly mentioned and relevant to answering the question
- Use exact names as they appear in the question
- If an entity reference is vague or ambiguous, include both the mention from the question and potential aliases or full names that might match in the database
- For organizations or entities with common abbreviations, include both full names and abbreviations when relevant
- Format your response as follows, where entities are separated by newlines:

```
<entities>
entity1
entity1_possible_alias
entity2
entity3
...
</entities>
```

If no topic entities are present in the question, return empty tags:

```
<entities>
</entities>
```

Task: Relevant Entity Extraction
Extract all relevant entities from the graph context and question (people, places, organizations, concepts, etc.) that need to explore next for answering the question.
Consider important entities only that are necessary for answering the question. Do not select entities, for which we already have all necessary information.
- [Same instructions as before]
- Format your response as follows, where entities are separated by newlines:

```
<entities>
next_entity1
next_entity1_possible_alias
next_entity2
next_entity3
...
</entities>
```

The entites should be sorted from the most important to the least important.
If we can answer the question directly based on the provided graph context, respond with:

```
<entities>
FINISH
</entities>
```

Figure 4: The entity extraction prompt in KG-Linker. We use the 'relative' entity extraction prompt when provided with graph context.

KG-Linker's Path Generation Prompt

Task: Relationship Path Identification
Identify all relevant relationship paths that connect the entities and can be used to answer the question.
- Only use relationships that are explicitly defined in the provided schema
- Paths may be single relationships or combinations of multiple relationships
- Generate at least 3 different meaningful relationship paths when possible, focusing on diversity rather than redundancy
- If graph context is provided, carefully analyze it to identify additional relevant relationship paths that might lead to the answer, especially focusing on paths to retrieve missing information that is not present in the context
- Use the context to determine which paths are most likely to yield correct answers based on the information provided
- Format your response as follows, where paths are separated by newlines:

```
<paths>
relation1
relation1 -> relation2
relation3
...
</paths>
```

For multi-step paths, use the "->" delimiter between relationships.
If the question does not require following any relationships, return empty tags:

```
<paths>
</paths>
```

Figure 5: The path generation prompt in KG-Linker.

KG-Linker's Graph Query Generation Prompt

Task: OpenCypher Query Generation
Construct a complete, executable OpenCypher statement that will retrieve the answer from a graph database.
- Your query must only use node types, relationship types, and properties defined in the provided schema
- Include appropriate MATCH patterns, WHERE clauses, and RETURN statements
- Handle any filtering, aggregation, or sorting required by the question
- If graph context is provided, ensure your query incorporates the relevant entities and relationships identified from the context

- Format your response as follows:

```
<opencypher>
MATCH ...
WHERE ...
RETURN ...
</opencypher>
```

Figure 6: The graph query generation prompt in KG-Linker.

KG-Linker's Draft Answer Generation Prompt

Task: Question Answering
Answer the question using your existing knowledge base or the external information provided in the graph context (if provided).
- Provide only direct entity answers that specifically address the question
- Each answer should be a distinct, well-defined entity (person, place, organization, concept, etc.)
- List multiple answers if appropriate, with each answer on a separate line
- Do not include explanations, reasoning, context, or commentary of any kind
- Do not preface or conclude your answer with statements like "Based on my knowledge..." or "The answers are..."
- If graph context is provided, prioritize answers that can be derived from the context over general knowledge
- If you genuinely cannot determine the answer from the provided context or your knowledge base, you may return empty answer tags
- Format your response exactly as follows, where answers are separated by newlines:

```
<answers>
answer_entity1
answer_entity2
...
</answers>
```

If no clear answer can be determined, provide empty tags:

```
<answers>
</answers>
```

Figure 7: The answer generation prompt in KG-Linker.

Relation Selection Prompt

Task: Relation Selection
Your task is to select the most appropriate relations based on their relevance to a given question.

Follow these steps:
1. Read the provided <question>question</question> carefully.
2. Analyze each relation in the <relation> list and determine its relevance to the question and relation.
3. Respond by selecting the most relevant relations within <select>relations</select> tags. Be both frugal on your selection and consider completeness.
4. The selected relations should be provided line-by-line.

Example format: <question>
Name the president of the country whose main spoken language was English in 1980?
</question>

<entity>
English
</entity>

<relations>
language.human_language.main_country
language.human_language.language_family
language.human_language.iso_639_3_code
base.rosetta.languoid.parent
language.human_language.countries_spoken_in
</relations>

<selected>
language.human_language.main_country
language.human_language.countries_spoken_in
base.rosetta.languoid.parent
</selected>

Explanation: [short explanation (deprecated due to figure length)]

Important Instructions: Always return at least one relation. Now it is your turn.

<question>
{question}
</question>

<entity>
{entity}
</entity>

<relations>
{relations}
</relations>

Remember to parse your response in <selected></selected> tags:

Figure 8: The relation selection prompt template used in agentic traversal.

Entity Selection Prompt

Task: Entity Selection
Given a question and the associated retrieved knowledge graph context (entity, relation, entity), you are asked to select the most important entities to explore in order to answer the question. Consider important entities only that are necessary for answering the question. Do not select entities, for which we already have all necessary information.
- Format your response exactly as follows: <next-entities>
relevant_entity1
relevant_entity2
...
</next-entities>

The selected entities must be provided line-by-line.
Example format:
Question: Name the president of the country whose main spoken language was English in 1980?
Graph Context: English -> countries_spoken_in -> England | USA

<next-entities>
England
USA
</next-entities>

The entities should be sorted from the most important to the least important. Important Instruction: If we can answer the question directly based on the provided graph context, respond with:
<next-entities>
FINISH
</next-entities>

Figure 9: The entity selection prompt template used in agentic traversal.