Evaluation of GraphRAG Strategies for Efficient Information Retrieval

Asma Houimli
Email: asma.houimli@euranova.eu

Zaineb Gabsi
Email: zaineb.gabsi@euranova.eu

Sabri Skhiri
Email: sabri.skhiri@euranova.eu

***Abstract*—Retrieval Augmented Generation (RAG) has emerged as a solution to enhance large language models (LLMs) and avoid hallucination by grounding their answers in external data. However, traditional RAG systems struggle to capture relationships and cross-references between different sources unless explicitly mentioned. This challenge is common in real-world scenarios, where information is often distributed and interlinked, making graphs a more effective representation. Recent research has focused on using GraphRAG systems that retrieve subgraphs instead of isolated passages, improving both interpretability and retrieval precision. Our work provides a technical contribution through a comparative evaluation of retrieval strategies within GraphRAG, focusing on context relevance rather than abstract metrics. We aim to offer practitioners actionable insights into the retrieval component of the GraphRAG pipeline. We experimented with a biochemical knowledge graph, benchmarking four graph-based retrieval strategies. To assess retrieval quality, we developed a refined framework inspired by ARES, minimizing the reliance on reference-based evaluations.**

1. Introduction

Large Language Models (LLMs) [1] [2] demonstrated high performance in generation and question-answering tasks. However, they are prone to hallucinations [3] [4], generating plausibly sounding yet incorrect or fabricated information, particularly when they lack sufficient knowledge to accurately answer a question. To address this, Lewis et al. presented in their paper *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* [5], RAG, an architectural framework that grounds the answers of an LLM, using external data sources. This data is usually stored in vector databases, where passages are indexed and retrieved when relevant to enhance the LLM's answer generation.

Alternative versions of RAG have been developed to overcome its limitations; for instance, Corrective RAG [6] enhances information retrieval by sourcing data from the web when external sources are insufficient. Another enhanced version is proposed by Rackauckas et al. [7], called Rag-Fusion, which first reformulates the query, extracts relevant passages for each version, and fuses them to produce the final answer.

Although these improved versions enhance retrieval quality, RAG systems struggle to extract relationships and cross-references either within the same data sources or across different sources, unless they are explicitly stated [8]–[10]. This occurs because the retriever selects individual passages that are deemed semantically relevant to the query.

Recent research has increasingly focused on connecting LLMs to knowledge graphs [4] [8] [10]. Knowledge graphs are structured representations where entities such as people, events, places, or concepts are modeled as nodes, and relationships between them as edges. This structured format is particularly useful in domains where information is distributed across multiple highly interconnected sources, making it difficult to capture through isolated text passages.

GraphRAG utilizes the rich connections and semantic relationships found in knowledge graphs to surpass the limitations of vector-only RAG, ultimately providing more accurate and explainable responses.

The construction of the knowledge graph is also extremely important [10], as it can significantly affect the system's performance and the overall quality. There are various types of knowledge graphs, including community summary graphs [8], embedding graphs [11], and the most commonly used ones, which are basic text-attributed knowledge graphs [12].

A RAG or a Graph RAG pipeline, consists mainly of two components: the retriever and the generator. The retriever is responsible for extracting relevant data for answering a query, while the generator utilizes the extracted information to produce the final answer. Therefore, the retriever is the most important component, and the quality of the final answer is heavily dependent on its effectiveness.

This work does not aim to introduce a new theoretical framework or retrieval architecture. Rather, it offers a practical and implementation-driven benchmark of graph-based retrieval strategies, either available through existing frameworks or easily built on top of them. Our goal is to provide practitioners with a clear and actionable comparison of four different retrieval strategies in Graph-RAG systems, ranging from simple methods to more advanced reasoning techniques, evaluating them across three key dimensions: context relevance, runtime, and computational cost.

To that end, our contributions are threefold:

- We implemented and/or enhanced four GraphRAG retrieval strategies, covering both commonly used and

emerging approaches.
- We developed a domain-specific evaluation framework inspired by ARES [13], which includes a fine-tuned context relevance classifier trained on synthetic biomedical queries.
- We conducted experiments on a biochemical knowledge graph extracted from Hetionet [14], and evaluated each strategy's retrieval quality under controlled conditions.

2. State of the Art

2.1. Existing Works for retrieval strategies

According to the survey by Han et al. [10], current retrieval methods in GraphRAG can be broadly categorized into heuristic-based, learning-based, and advanced strategies that combine multiple techniques to handle complex queries. Heuristic-based Retrievers rely on predefined rules and algorithms. A common first step is Entity Linking, where entities in the query are mapped to nodes in the graph using vector-based similarity [15]–[17] or lexical features [18]. Similarly, Relational Matching identifies edges in the graph that align with relations mentioned in the query by selecting the top-K most similar edges [19], [20].

Once seed nodes and edges are identified, Graph Traversal algorithms are used to retrieve context. Some methods extract all paths up to a certain length between the initial entities [21]–[25], while others retrieve the k-hop subgraph surrounding the seed entities [26]–[29]. Another emerging type of graph traversal uses LLMs to generate Graph Query Language (GQL) queries that extract information from the graph database relevant to answering the natural language query [30]. To manage the risk of information overload from traversal, some methods use LLMs to prune irrelevant paths [15], [18], [31], [32] or employ pre-defined rules and templates to guide the traversal [33]–[35].

Other heuristic methods include Graph Kernels, which measure the structural similarity between a query graph and candidate graphs for tasks like document [36] or image retrieval [37], [38].

Learning-based Retrievers leverage machine learning to capture semantic relationships by embedding graph components and the query into a shared vector space for similarity search. Graph Neural Networks (GNNs) have become key in this approach, often integrating the query into the GNN's message-passing mechanism to make the process question-aware. For example, GNN-RAG [39] incorporates the query embedding into the message computation and retrieves the shortest paths to high-probability candidate nodes. Other works use conditional GNNs where only query-linked entities are initialized [40] or employ query-conditional attention weights on edges to guide the aggregation process [41].

Advanced Retrieval Strategies have been developed to address a variety of complex queries that require multi-hop reasoning.

- Integrated Retrieval combines symbolic and neural approaches. Some methods first expand neighbors in a symbolic knowledge graph and then use neural matching for path retrieval [16], [32]. While others use GNNs for neural retrieval of seed entities before symbolically extracting shortest paths [39]. Another approach is to symbolically fetch the k-hop neighborhood and then use neural attention to compute the relevance of candidates [23], [24], [29], [32], [42].
- Iterative Retrieval builds context through a multi-step process. This can involve an LLM agent that alternates between generating evidence and selecting the next neighbor to explore [43], iteratively expanding reasoning paths from seed entities [44], or repeatedly calling pre-defined graph interfaces to collect information [17], [26]. For instance, in their paper [4], the authors extend the chain-of-thought paradigm by enabling iterative reasoning over graph-structured data.
- Adaptive Retrieval identifies the required external information for a query by training models to predict the needed reasoning depth (number of hops) for a given query and retrieving content accordingly, ensuring that retrieved content is neither insufficient nor overly noisy [31], [45].

2.2. Context relevance evaluation

Since the focus is on the retrieval quality, the key metric to assess is Context relevance. In the survey *"Benchmarking of retrieval augmented generation"* [46], the authors presented a variety of methods.

The most basic approach involves using lexical matching metrics such as **Recall@K**,**MRR@K**, **MAP@K**, **Precision@K**. Each one provides a different insight into the effectiveness of the retrieval process.

While **Recall@K** measures how many of the truly relevant items appear in the top K, **Precision@K** measures how many of the top K items are truly relevant.

Both metrics are effective and used, for instance, the *Graph RAG-Tool Fusion* paper [17] reports Precision@K at $K = 10, 20, 30$; however, these metrics neglect results ranking. To address this, researchers use rank-aware metrics. For example, **MAP@K (Mean Average Precision@K)** measures how many relevant items are retrieved and how early they appear, while **MRR@K (Mean Reciprocal Rank@K)** measures how soon the most relevant item appears. It was employed in the study *Evaluating rag-fusion with ragelo* [7] to assess the context retrieved.

Although these metrics are popular due to their simplicity and ease of computation, one of their main limitations is the need for annotated reference data. Creating such annotations is often time-consuming and costly, which can significantly slow down experimentation.

Additionally, they rely on lexical matching without considering semantic equivalence, which leads to the underestimation of evidence expressed differently but holding the same meaning as the gold truth.

To address this issue, more recent methods of evaluation rely on using an **"LLM as a judge"** approach, where LLMs are tasked with assessing context relevance. Here,

we present some of the most commonly used evaluation frameworks that adopt this approach.

RAGAS: RAG Assessment [47] is a well-known framework that leverages LLMs to evaluate the performance of RAG systems. The evaluation covers three metrics: Context Relevance, Answer Faithfulness, and Answer Relevance. To assess the context relevance, the LLM is prompted to identify which statements in the retrieved context are relevant to the query. The final context relevance score is calculated as the ratio of the relevant statements found within the context.

RAGElo: RAGElo [7] is another "LLM as a judge" evaluation framework that uses an Elo-style rating system. Instead of assessing systems individually, it pits two systems against each other, with the LLM determining which one performs better for answer correctness and faithfulness. For context relevance in particular, it uses **MRR@5** by prompting an LLM to classify documents as **Not relevant**, **Somewhat relevant**, or **Very relevant** to the query. The labels generated by the LLM are then used to compute the **MRR@5** score.

Both RAGAS and RAGElo frameworks minimize reliance on human-annotated datasets, but have significant limitations: the results heavily depend on the LLM and the prompts used, leading to less stability compared to reference-based metrics. They may also struggle with specific domains that require specialized terminology that the LLM was not likely trained on.

ARES: This Framework [13] enhances RAG evaluation systems by using fine-tuned LLMs specifically trained on a specific domain. It covers three key metrics: context relevance, answer faithfulness, and answer relevance, each evaluated by a dedicated expert judge. By fine-tuning the models for evaluation, ARES improves upon generic LLMs, enhancing their ability to assess the specific metrics reliably.

Across the literature, GraphRAG benchmarks generally adopt traditional metrics such as accuracy, precision, and F1-score, focusing on end-task performance over retrieval quality [48]–[50].

Overall, existing work mostly evaluates end tasks, either using traditional metrics that require annotation or relying on generic LLMs prompting. To the best of our knowledge, currently, no benchmark compares context relevance across various GraphRAG retrieval strategies. In response, we evaluate retrieval separately from generation, using a domain-specific judge and PPI calibration, while also reporting efficiency (runtime and API calls) to highlight performance–efficiency trade-offs.

3. Biochemical Knowledge graph

The biochemical knowledge graph used for evaluating retrieval strategies is derived from Hetionet [14], a large-scale, heterogeneous biomedical knowledge graph that integrates data from various curated sources.

For our research, we focused on a subset of the graph highlighting key biochemical entities and their interactions. Specifically, the graph includes five node types: genes, compounds, biological processes, molecular functions, and pathways, along with directed relationships between them.

As illustrated in Fig. 1, this curated subset captures semantic and functional dependencies among biochemical concepts, making it suitable for evaluating multi-hop reasoning and structural awareness in retrieval strategies. We uploaded the graph to a Neo4j database for easier communication.

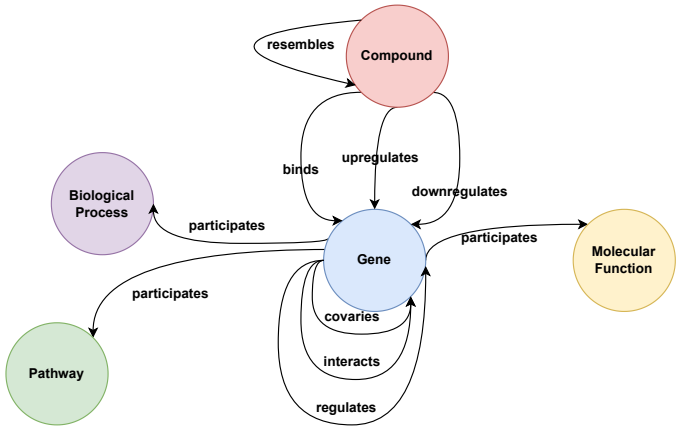

Figure 1: The final biochemical knowledge graph contains 38,584 nodes and 1,488,879 edges.

4. Design of Retrieval Strategies

We focused on four distinct retrieval strategies as illustrated in Fig. 2, either developing them from scratch or adapting existing implementations. In the following sections, we explain how each strategy operates given a natural language query, along with a brief description of its implementation.

4.1. Graph Traversal

When stored in a graph database, knowledge graphs can be easily queried using Graph Query Languages (GQL) such as Cypher, GraphQL, or SPARQL. However, using GQL requires specialized knowledge, which can be challenging for non-technical users. Neo4j's Text2Cypher [30] aims to overcome this challenge by automatically translating natural language queries into Cypher query language, making the interaction with the knowledge graph easier for non-experts.

Building on this, this strategy uses Neo4j's GraphRAG Text2Cypher feature. Given a natural language query, a description of the ontology or the schema of the knowledge graph (including node labels, edge labels, and properties), and a few-shot set of natural language-Cypher examples, the LLM generates a Cypher query. This query is then executed on the graph database, where the knowledge graph is stored, retrieving items or subgraphs deemed relevant. We have also implemented a retry mechanism: if the query fails, the LLM will regenerate a new one.

For the choice of the text to cypher LLM, in their paper *Text2Cypher: Bridging Natural Language and Graph Databases* [30], the authors outlined various approaches, ranging from simply prompting an LLM for the generation of the Cypher query to fine-tuning models specifically for this task. For our implementation, we used the `O3-mini` model as the Text2Cypher LLM. Since this is a reasoning task, this model appears to perform better than other `OpenAI` models.

Once the items are retrieved from the graph, they may initially appear disorganized and messy. To address this, we apply a cleaning step followed by graph pruning, where we select the first $k = 10$ items. This ensures that the context remains short and focused. The final step is the context refinement, in which the retrieved context is organized by prompting `Gpt-4o` with the generated cypher query and the cleaned retrieved items. This process produces a more refined and coherent context that presents the relevant information in an organized manner.

4.2. Hybrid Retrieval

This hybrid retrieval strategy combines simple Cypher-based traversal methods with semantic similarity ranking to prioritize semantically relevant subgraphs. Similar methods have been proposed in prior work, including Cheng et al [51]. In our implementation, we build upon the previous strategy by incorporating an additional semantic-similarity reranking of the retrieved items from the graph.

Instead of basic graph pruning, we apply **semantic-based pruning**. We embed the cleaned retrieved items in batches of 256, alongside the original query, using `OpenAI`'s model, `text-embedding-ada-002`. This batching process reduces the number of API calls and decreases overall latency compared to embedding each item individually. By calculating the cosine similarity between each item's embedding and the embedding of the query, we narrow the selection down to the top $k = 10$ most semantically relevant items. Finally, the selected items undergo the same **Context Refinement** step, as described in the graph traversal section, to ultimately achieve a coherent context.

4.3. Agentic semantic-first traversal

This retrieval strategy combines semantic search with graph-based reasoning. It begins with a one-time vector search to identify optimal starting nodes for traversal, followed by an iterative graph traversal.

Recent research has explored such hybrid agent-based approaches. For instance, in their paper *Graph RAG-Tool Fusion* [17], the authors present a novel plug-and-play method that effectively combines vector-based retrieval with efficient graph traversal. This technique identifies all relevant nodes (or tools) and then extracts full dependencies of these tools, allowing for further graph traversal to extract dependencies and gather additional information. The items obtained are then ranked, and the top results are used to generate the final answer. Another similar strategy is presented in the paper titled *"Graph Chain of Thoughts: Augmenting Large Language Models by Reasoning on Graphs"* [4]. In this work, the authors propose a framework called *"Graph chain of Thought"*, an iterative process that allows LLMs to navigate the graph step-by-step to identify the essential information needed for answering the query; Starting with vector search to identify initial search nodes, followed by step-by-step graph traversal to explore specific neighborhoods of these semantically relevant nodes.

Inspired by previous work, we adapt this paradigm in our implementation, combining vector search with guided graph traversal through a **tool-augmented LLM agent**.

The initial step in this implementation involves embedding the nodes of the knowledge graph. Each node is represented by a descriptive sentence that captures its key attributes. For example, a gene node might be described with a sentence detailing its name, whole description, and chromosome location (e.g., "Gene AGT is located on chromosome 1. Its function involves angiotensinogen (serpin peptidase inhibitor, clade A, member 8)."). These sentences are embedded using OpenAI's `text-embedding-ada-002` model, and the embedding vectors are ultimately stored as a node property and indexed by node type using Neo4j indexes.

The core of our implementation is the Langchain agent powered by `Gpt-4o`, which interprets the natural language query, plans the retrieval process, and orchestrates tool use throughout the reasoning process. The agent is provided with a system prompt providing high-level guidance on the expected behavior of the agent, as well as formatting rules and a few-shot set of examples demonstrating how queries should be decomposed and mapped to tool calls. The process begins with an **initial, mandatory, and one-time** semantic search step using node embeddings to identify semantically relevant starting points.

The agent then enters an iterative reasoning loop, calling specialized tools to retrieve neighbors or additional node information, analyzing the retrieved data and the intermediate steps at each iteration, and deciding how to proceed until enough information is collected or a maximum iteration limit is reached.

After retrieval is complete, the context refinement step is performed, where the raw results are refined to transform the returned nodes and edges into a coherent and concise context.

The implemented tools are the following:

- `Search_INDEX(keyword, node_type)`: This is the first tool to be used, which performs vector searches to retrieve nodes of a specific type semantically relevant to a particular part of the query, returning their IDs.
- `Get_neighbors_by_relationship(node_id, relationship, target_label)` and `Extract_genes_regulated_by_a_Gene (node_id)`: Used to fetch specific neighbors of a node given its ID.
- `Extract_gene_info(node_id)`: extract additional information from a particular gene node.

These tools serve as the agent's interface to the underlying knowledge graph and are used for translating reasoning based on natural language into concrete and executable operations.

4.4. Agentic traversal

Similar to the Agentic semantic-first traversal strategy, this strategy uses a step-by-step reasoning approach on graphs to extract relevant information needed to answer queries. It is also implemented through a **tool-augmented LLM agent**. The key difference is that this strategy **relies entirely on agentic iterative graph traversal**, skipping the mandatory initial step of semantic search found in the previous approach.

To implement this strategy, we developed both basic tools as well as advanced ones that facilitate operations like **list merging and set intersection** to integrate information from different graph regions when dealing with complex questions. These are the implemented tools:

- `RetrieveNode(name)`: Get a node's ID from its name.
- `Check_node(NodeID, FeatureName)`: Retrieve a node's attribute (e.g., name, chromosome).
- `NeighbourCheck(NodeID, NeighbourType)`: Retrieve a node's neighbors via a given relationship.
- `NodeDegree(NodeID, NeighborType)`: Returns the count of neighbors associated with a specific relationship.
- `InverseNeighbourCheck(nodeID, relation)`: Retrieves upstream (incoming) neighbours through a given relationship. We introduced this tool to support our unidirectional graph design.
- `FilterNodes(label, property, value)`: Filter nodes that satisfy a specific property constraint.
- `CombineLists(list1, list2)` and `IntersectLists(list1, list2)`: Perform set operations on lists. `CombineLists` is used for logical `OR`, whereas, `IntersectLists` support logical `AND`.

Initially, the LLM agent powered by `Gpt-4o` is given a prompt containing the natural language query, a textual description of the knowledge graph's ontology, a curated set of few-shot examples illustrating how this strategy is applied to answer natural language queries, and the full list of available tools it can use during execution. The LLM agent iteratively explores the graph using the given tools until enough evidence is found or a maximum number of iterations is reached, while maintaining an internal scratchpad that tracks intermediate reasoning steps and tool outputs, which is later refined into a concise and coherent context.

5. Evaluation Framework for Context Relevance

Building on the Context relevance evaluation methods detailed in the State of the Art 2.2, we adopt ARES as our backbone for context-relevance evaluation. We chose ARES because its modular architecture allows us to focus only on the evaluation of retrieved context, and given our biochemical setting, the fine-tuning yields a robust, domain-aware judge. Additionally, ARES is label-efficient and calibrated via synthetic question–context pairs and the use of PPI, which reduces reliance on large annotated QA corpora while correcting model bias. We therefore adapt ARES to our needs with a set of targeted changes, detailed in the following sections.

5.1. Synthetic Data Generation

To train a context-relevance model, we need a dataset of query–context pairs. Such datasets are often drawn from the text documents used to build the underlying knowledge graph; since no such corpus was available, we constructed one from two complementary sources:

- **PubMed articles.** We chose 24 PubMed papers on biochemistry and biomedicine. From each paper, we kept the main text and removed references, figures/tables, and metadata.
- **Graph linearizations.** We converted the biochemical knowledge graph into natural-language passages in two styles: *fan-out*, a central entity and its immediate neighbors (e.g., TP53 regulates MDM2, CDKN1A, and BAX"); and *chain-based*, multi-hop relations reflecting longer reasoning (e.g., Molindone binds HTR2A, which covaries BRSK1").

These passages were chunked into seven-sentence windows with a two-sentence overlap to preserve local context and continuity.

With the corpus ready, we built query–context pairs that the evaluator will be trained on. For each passage, we asked `GPT-3.5-Turbo` to generate at least two relevant questions. Passages from PubMed and fan-out linearizations produced simple questions focused on one entity or relation (e.g., "What compounds resemble Caffeine?"). Chain-based passages led to more complex questions that connect several entities (e.g., "What is the biological process that involves the gene covaried by NHLRC1?"). These formed our initial set of positive question–context pairs.

We generated negatives by pairing each passage with questions from unrelated passages. We removed duplicates and fixed a minimum of two negatives per passage to maintain class balance.

Finally, we expanded the generated dataset with a semantic augmentation step. We embedded all questions and passages using `text-embedding-ada-002`. We randomly paired a subset of questions with the nearest passages to create additional positive pairs and paired another subset with the farthest passages to create additional negative pairs. After de-duplication, we obtained a balanced dataset of query–context pairs, ready to use for downstream tasks.

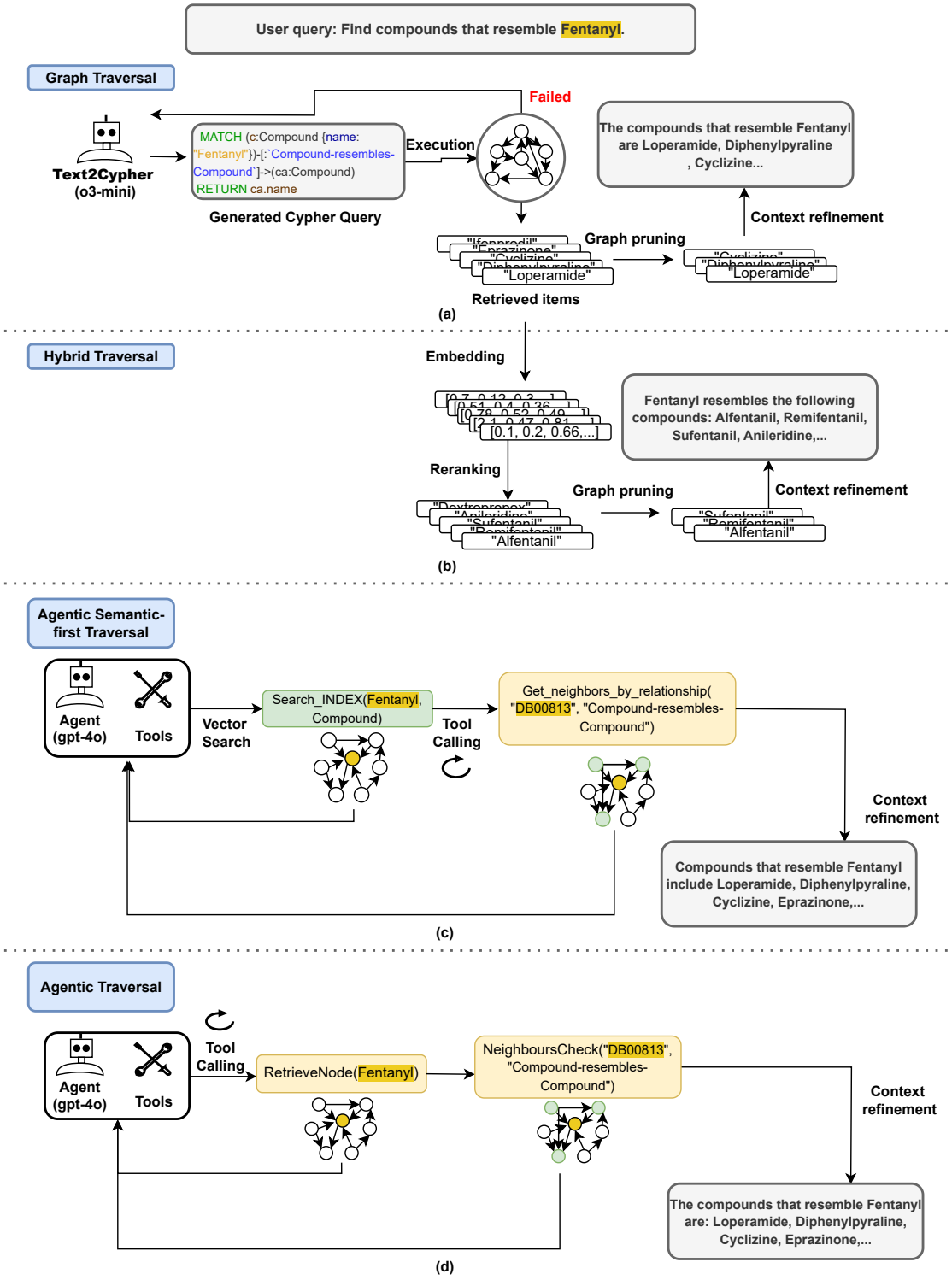

Figure 2: Representative workflow of the four retrieval strategies. (a)Graph Traversal (b)Hybrid Traversal (c)Agentic Semantic-First Traversal (d)Agentic Traversal.

5.2. Fine-Tuning BioBERT with LoRA

Once the synthetic dataset of query–context pairs was ready, we began training a model capable of classifying whether a given context is relevant to answering a query. In the original ARES framework, this role is fulfilled by a DeBERTa-V3 Large model. Because our work operates in the biochemical domain, we replace it with BioBERT introduced by Lee et al. [52]. In fact, BioBERT is a BERT model pretrained on PubMed abstracts and PMC full-text articles, which equips it with a stronger grasp of biochemical terminology and semantic relationships, making it better suited for evaluating context relevance in our setting.

Instead of full fine-tuning, which updates all of BioBERT's parameters, making it computationally expensive and risking overwriting its valuable biochemical knowledge, we adopt a Low-Rank Adaptation (LoRA) [53] approach. LoRA adapts the model by updating a small parameter set while keeping the base weights frozen.

This approach reduces catastrophic forgetting, which is a limitation of full fine-tuning [54] and tackles ARES's [13] concern of high GPU cost and long training, while maintaining strong performance.

To achieve robust results, we explore different training configurations and refine them using Bayesian optimization via Weights & Biases Sweeps. This process produces a well-tuned context-relevance judge capable of assessing context relevance in the biochemical domain.

5.3. Prediction Powered Inference

Given a batch of queries and their corresponding context retrieved from the biochemical knowledge graph using a specific strategy, the fine-tuned model estimates the probability that the context is relevant to the query for each pair. The final context relevance score is calculated as the average of these probabilities across the entire batch.

This approach assumes that the model's predictions are entirely trustworthy. In practice, small systematic biases in the predictions could misrepresent the true performance of a retrieval method. To address this, the ARES framework employs Prediction-Powered Inference (PPI) [55], a statistical framework that combines the speed of automated predictions with the precision of human judgment. The process begins by manually annotating a small representative subset of query–context pairs that are used to calibrate the model's outputs, correcting any bias and improving the alignment between predicted and actual relevance. The calibrated scores are then applied to the much larger unlabeled set, which allows the estimation of the mean context relevance with statistically valid confidence intervals. This approach is particularly well-suited to the biochemical domain, where labeling costs are high but rigorous evaluation is essential. By leveraging PPI, we grounded our large-scale retrieval assessments in human validated knowledge without the cost of labeling every example.

In summary, our evaluation framework adapts ARES, which was originally designed for evaluating **RAG** systems, to the **GraphRAG** setting. The process combines synthetic data generation, BioBERT Lora fine-tuning, and calibration via Prediction-Powered Inference. This design balances scalability with accuracy, enabling the evaluation of retrieval strategies without the cost of fully annotated datasets, the high GPU consumption, or the long training process.

The next section is dedicated to the experimental setup, where we apply this framework to the retrieval strategies presented in section 4 and compare their performance.

6. Experiments

6.1. Experimental Setup

To evaluate the performance of each retrieval strategy, we prepared two types of queries:

- **Simple Questions**: Typically focus on a specific node, its features, or direct neighbors; requiring only one-hop on the knowledge graph. (e.g., "What biological processes does the gene UBASH3B participate in?", "Does Tizanidine bind to any genes?"...)
- **Complex Questions**: These questions are more complex, requiring multiple hops on the knowledge graph, either asking about indirect neighbors or global questions. (e.g., "Find two compounds that regulate the gene that participates in the biological process response to gravity?", "Give me genes located on chromosome 8 that participate in at least one pathway and are regulated by another gene.", "Find two compounds that resemble, and both bind to a gene that participates in the biological process double-strand break repair via break-induced replication."...)

The purpose of these questions is to evaluate and benchmark the four retrieval strategies developed, assessing their ability to maintain performance even with complex inquiries. We curated and prepared a total of 300 simple questions and 200 complex questions from the biochemical knowledge graph, ensuring comprehensive coverage of all node types and relationships.

6.2. Evaluation Metrics

To evaluate the retrieval strategies, we focused on two key and complementary dimensions: **performance** and **efficiency**.

6.2.1. Performance. In the batches of simple and complex questions we created, each strategy retrieves and produces its candidate context. To evaluate their performance in extracting relevant information from the graph, we used the **Context Relevance** score on an entire batch of query-context pairs using predictions from the model alone as well as PPI-adjusted.

We calculate the average probability that the context is relevant to the query based on the predictions from the fine-tuned BioBERT model across an entire batch.

Regarding the PPI adjustment, we labeled a set of 150 query-context pairs. These pairs are used to correct the model's predictions, addressing any bias by taking the midpoint of the confidence intervals produced by PPI.

6.2.2. Efficiency. To evaluate the efficiency of the retrieval strategies, we focused on two key metrics: The **runtime** and the **number of API calls**.

Runtime is measured as the average time required to process a single query.

The number of API calls reflects the average number of external LLM invocations, including generation and embedding requests.

These two measures capture the computational and monetary overhead of each strategy, providing a practical view of their efficiency.

7. Results & Discussion

TABLE 1: **Performance of implemented retrieval strategies.** Average context relevance scores across simple and complex questions, reported as model-only predictions and PPI-adjusted values.

Retrieval Strategy	Simple		Complex	
	Model-only	PPI	Model-only	PPI
Graph Traversal	**0.942**	**0.840**	0.8395	0.744
Hybrid Traversal	**0.937**	**0.840**	**0.8409**	0.743
Agentic Semantic-first	0.921	0.820	0.688	0.588
Agentic Traversal	0.910	0.810	**0.8396**	0.740

TABLE 2: **Efficiency of retrieval strategies.** Average runtime (in seconds) and number of API calls per for simple and complex questions.

Retrieval Strategy	Simple		Complex	
	Runtime	API calls	Runtime	API calls
Graph Traversal	6.11	2.0	10.92	2.0
Hybrid Traversal	9.39	4.85	34.69	21.02
Agentic Semantic-first	4.94	4.5	7.72	5.89
Agentic Traversal	8.17	7.47	14.50	11.3

As illustrated in Table 1, all retrieval strategies achieve a high context relevance score (greater than 0.9) for simple questions, with **Graph Traversal** and **Hybrid Traversal** strategies obtaining the top scores for both model predictions and PPI-adjusted values. This reflects the effectiveness of all strategies in retrieving relevant information from the graph for simple questions that do not require multiple hops.

However, for complex questions, **Graph Traversal**, **Hybrid Traversal**, and **Agentic Traversal** all achieve high context relevance scores that are closer to each other, demonstrating consistent performance and stability even with more complex queries that necessitate multiple hops on the graph to find relevant information. In contrast, the agentic semantic-first traversal shows a significant decline in context relevance score. This decline can be attributed to the initial step of the semantic search, which retrieves the first nodes in a manner that can be misleading, especially for general questions. For example, consider the query: *"Give me all genes located on chromosome 8 that participate in at least one pathway and are regulated by another gene"*. This strategy works by taking a portion of the query, for example, *"located on chromosome 8, that participate in at least one pathway and are regulated by another gene"*, and using it to conduct a semantic search to retrieve relevant nodes. However, since the embeddings of each node contain information that is specific only to that node and not its neighborhood, the initial nodes retrieved may not be relevant. Therefore, the first step of the semantic search can be misleading if the query does not focus on a specific node. Additionally, the choice of which portions of the query to use for the semantic search is crucial to the effectiveness of the search.

It is important to note that while the PPI-adjusted values align with the trends observed in the model predictions, they are generally lower for both simple and complex questions. This can be attributed to the relatively small size of the evaluation set, which makes the PPI correction more sensitive to variability, potentially leading to slight underestimations of the model's true performance.

After examining the context relevance scores of each strategy, we now turn our attention to their efficiency as illustrated in Table 2, a clear contrast emerges: Graph traversal maintains a low and lightweight runtime with exactly two API calls per query. Whereas Hybrid traversal introduces a significant rise in both runtime and API calls due to the additional reranking step, making it the slowest option with the highest number of API calls for complex questions.

The agentic strategies fall in between these two extremes. Agentic semantic-first traversal achieves the fastest runtime for both simple and complex questions, while also requiring a low number of API calls. On the other hand, Agentic traversal has a longer runtime and involves more API calls, making it an expensive option for simple questions, and relatively slower and more costly for complex questions.

Taken together, these results highlight a clear trade-off between performance and efficiency. Graph Traversal stands out as the most optimal strategy, consistently achieving a high context relevance score while remaining lightweight in both runtime and API calls. Hybrid Traversal, although competitive in performance, pays a high cost in efficiency due to the semantic reranking, making it less practical in time or budget-constrained settings. In contrast, Agentic semantic-first traversal offers excellent efficiency but declines in performance on complex queries, reflecting the risk of relying on embedding that neglects neighborhood context. Finally, Agentic traversal delivers high performance across both simple and complex questions, but at the expense of increased API calls and longer runtime.

8. Conclusion

In this work, we adapt the ARES evaluation framework to the GraphRAG setting, conducting experiments

on a biochemical knowledge graph to assess four retrieval strategies. Our evaluation framework serves as a reliable, nearly reference-free context relevance classifier capable of capturing domain-specific nuances. We also examined the efficiency of these strategies by measuring runtime and API calls for each strategy. Results show that Graph Traversal performs best with low runtime and a minimal number of API calls. Hybrid and Agentic traversal methods also perform well but come with higher computational and monetary costs, making them effective yet expensive options. In contrast, the Agentic semantic-first traversal struggles with complex queries, reflecting the limitations of embeddings that only capture local node information.

This paper represents a practical benchmark. It is designed to guide practitioners by offering a clear and actionable comparison of different retrieval strategies in GraphRAG systems. This way, we provide not only insights into the performance and efficiency trade-offs but also concrete guidance for system builders who aim to balance performance with scalability and budget constraints.

It's important to highlight that these findings are closely tied to the specific experimental setup. So generalization to other domains and knowledge graph schemas may vary. As future work, we plan to explore neighborhood-aware or GNN-enhanced node embeddings, such as the method introduced in *"Augment to Interpret"* [56], to reduce the risk of relying on misleading starting points for retrieval. Another promising research direction is integrating an inference layer on top of the knowledge graph to evaluate its added value. We also plan to inspect emerging parameter-efficient fine-tuning methods like QLoRA, RoSA, DoRA, and related variants, to study their impact on judge quality and compute/memory cost.

References

[1] M. U. Hadi, R. Qureshi, A. Shah, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu, S. Mirjalili *et al.*, "A survey on large language models: Applications, challenges, limitations, and practical usage," *Authorea Preprints*, 2023.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[3] G. Perković, A. Drobnjak, and I. Botički, "Hallucinations in llms: Understanding and addressing challenges," in *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*. IEEE, 2024, pp. 2084–2088.

[4] B. Jin, C. Xie, J. Zhang, K. K. Roy, Y. Zhang, Z. Li, R. Li, X. Tang, S. Wang, Y. Meng *et al.*, "Graph chain-of-thought: Augmenting large language models by reasoning on graphs," *arXiv preprint arXiv:2404.07103*, 2024.

[5] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[6] S.-Q. Yan, J.-C. Gu, Y. Zhu, and Z.-H. Ling, "Corrective retrieval augmented generation," 2024.

[7] Z. Rackauckas, A. Câmara, and J. Zavrel, "Evaluating rag-fusion with ragelo: an automated elo-based framework," *arXiv preprint arXiv:2406.14783*, 2024.

[8] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, "From local to global: A graph rag approach to query-focused summarization," *arXiv preprint arXiv:2404.16130*, 2024.

[9] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, and Q. Li, "A survey on rag meeting llms: Towards retrieval-augmented large language models," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 6491–6501.

[10] H. Han, Y. Wang, H. Shomer, K. Guo, J. Ding, Y. Lei, M. Halappanavar, R. A. Rossi, S. Mukherjee, X. Tang *et al.*, "Retrieval augmented generation with graphs (graphrag)," *arXiv preprint arXiv:2309.15217*, 2024. [Online]. Available: https://arxiv.org/abs/2309.15217

[11] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE transactions on knowledge and data engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[12] X. He, Y. Tian, Y. Sun, N. Chawla, T. Laurent, Y. LeCun, X. Bresson, and B. Hooi, "G-retriever: Retrieval-augmented generation for textual graph understanding and question answering," *Advances in Neural Information Processing Systems*, vol. 37, pp. 132 876–132 907, 2024.

[13] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, "Ares: An automated evaluation framework for retrieval-augmented generation systems," *arXiv preprint arXiv:2311.09476*, 2023.

[14] D. S. Himmelstein, A. Lizee, C. Hessler, L. Brueggeman, S. L. Chen, D. Hadley, A. Green, P. Khankhanian, and S. E. Baranzini, "Systematic integration of biomedical knowledge prioritizes drugs for repurposing," *elife*, vol. 6, p. e26726, 2017.

[15] D. Sanmartin, "Kg-rag: Bridging the gap between knowledge and creativity," *arXiv preprint arXiv:2405.12035*, 2024.

[16] Y. Wen, Z. Wang, and J. Sun, "Mindmap: Knowledge graph prompting sparks graph of thoughts in large language models," *arXiv preprint arXiv:2308.09729*, 2023.

[17] E. Lumer, P. H. Basavaraju, M. Mason, J. A. Burke, and V. K. Subbiah, "Graph rag-tool fusion," *arXiv preprint arXiv:2502.07223*, 2025.

[18] Y. Wang, N. Lipka, R. A. Rossi, A. Siu, R. Zhang, and T. Derr, "Knowledge graph prompting for multi-document question answering," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 38, no. 17, 2024, pp. 19 206–19 214.

[19] Y. Gao, L. Qiao, Z. Kan, Z. Wen, Y. He, and D. Li, "Two-stage generative question answering on temporal knowledge graph using large language models," *arXiv preprint arXiv:2402.16568*, 2024.

[20] J. Kim, Y. Kwon, Y. Jo, and E. Choi, "Kg-gpt: A general framework for reasoning on knowledge graphs using large language models," *arXiv preprint arXiv:2310.11220*, 2023.

[21] Y. Feng, X. Chen, B. Y. Lin, P. Wang, J. Yan, and X. Ren, "Scalable multi-hop relational reasoning for knowledge-aware question answering," *arXiv preprint arXiv:2005.00646*, 2020.

[22] X. Jiang, R. Zhang, Y. Xu, R. Qiu, Y. Fang, Z. Wang, J. Tang, H. Ding, X. Chu, J. Zhao *et al.*, "Hykge: A hypothesis knowledge graph enhanced framework for accurate and reliable medical llms responses," *arXiv preprint arXiv:2312.15883*, 2023.

[23] M. Yasunaga, H. Ren, A. Bosselut, P. Liang, and J. Leskovec, "Qa-gnn: Reasoning with language models and knowledge graphs for question answering," *arXiv preprint arXiv:2104.06378*, 2021.

[24] M. Yasunaga, A. Bosselut, H. Ren, X. Zhang, C. D. Manning, P. S. Liang, and J. Leskovec, "Deep bidirectional language-knowledge graph pretraining," *Advances in Neural Information Processing Systems*, vol. 35, pp. 37 309–37 323, 2022.

[25] X. Zhang, A. Bosselut, M. Yasunaga, H. Ren, P. Liang, C. D. Manning, and J. Leskovec, "Greaselm: Graph reasoning enhanced language models for question answering," *arXiv preprint arXiv:2201.08860*, 2022.

[26] J. Jiang, K. Zhou, Z. Dong, K. Ye, W. X. Zhao, and J.-R. Wen, "Structgpt: A general framework for large language model to reason over structured data," *arXiv preprint arXiv:2305.09645*, 2023.

[27] Y. Kim, E. Kang, J. Kim, and H. H. Huang, "Causal reasoning in large language models: A knowledge graph approach," *arXiv preprint arXiv:2410.11588*, 2024.

[28] M. Niu, H. Li, J. Shi, H. Haddadi, and F. Mo, "Mitigating hallucinations in large language models via self-refinement-enhanced knowledge retrieval," *arXiv preprint arXiv:2405.06545*, 2024.

[29] Y. Tian, H. Song, Z. Wang, H. Wang, Z. Hu, F. Wang, N. V. Chawla, and P. Xu, "Graph neural prompting with large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 17, 2024, pp. 19 080–19 088.

[30] M. G. Ozsoy, L. Messallem, J. Besga, and G. Minneci, "Text2cypher: Bridging natural language and graph databases," *arXiv preprint arXiv:2412.10064*, 2024.

[31] T. Guo, Q. Yang, C. Wang, Y. Liu, P. Li, J. Tang, D. Li, and Y. Wen, "Knowledgenavigator: Leveraging large language models for enhanced reasoning over knowledge graph," *Complex & Intelligent Systems*, vol. 10, no. 5, pp. 7063–7076, 2024.

[32] L. Luo, Y.-F. Li, G. Haffari, and S. Pan, "Reasoning on graphs: Faithful and interpretable large language model reasoning," *arXiv preprint arXiv:2310.01061*, 2023.

[33] X. Dai, Y. Hua, T. Wu, Y. Sheng, and G. Qi, "Counter-intuitive: Large language models can better understand knowledge graphs than we thought," *CoRR*, 2024.

[34] R. Liao, X. Jia, Y. Li, Y. Ma, and V. Tresp, "Gentkg: Generative forecasting on temporal knowledge graph with large language models," *arXiv preprint arXiv:2310.07793*, 2023.

[35] C. Wang, Y. Xu, Z. Peng, C. Zhang, B. Chen, X. Wang, L. Feng, and B. An, "keqing: knowledge-based question answering is a nature chain-of-thought mentor of llm," *arXiv preprint arXiv:2401.00426*, 2023.

[36] B. Wu, B. Lang, and Y. Liu, "Gksh: Graph based image retrieval using supervised kernel hashing," in *Proceedings of the International Conference on Internet Multimedia Computing and Service*, 2016, pp. 88–93.

[37] G. Glavaš and J. Šnajder, "Event-centered information retrieval using kernels on event graphs," in *Proceedings of TextGraphs-8 graph-based methods for natural language processing*, 2013, pp. 1–5.

[38] J. Lebrun, S. Philipp-Foliguet, and P.-H. Gosselin, "Image retrieval with graph kernel on regions," in *2008 19th International Conference on Pattern Recognition*. IEEE, 2008, pp. 1–4.

[39] C. Mavromatis and G. Karypis, "Gnn-rag: Graph neural retrieval for large language model reasoning," *arXiv preprint arXiv:2405.20139*, 2024.

[40] G. Liu, Y. Zhang, Y. Li, and Q. Yao, "Explore then determine: A gnn-llm synergy framework for reasoning over knowledge graph," *arXiv e-prints*, pp. arXiv–2406, 2024.

[41] J. Fang, Z. Meng, and C. Macdonald, "Reano: Optimising retrieval-augmented reader models through knowledge graph generation," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 2094–2112.

[42] Y. Wang, A. Javari, J. Balaji, W. Shalaby, T. Derr, and X. Cui, "Knowledge graph-based session recommendation with session-adaptive propagation," in *Companion Proceedings of the ACM Web Conference 2024*, 2024, pp. 264–273.

[43] S. Wang, Y. Zhu, H. Liu, Z. Zheng, C. Chen, and J. Li, "Knowledge editing for large language models: A survey," *ACM Computing Surveys*, vol. 57, no. 3, pp. 1–37, 2024.

[44] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, L. M. Ni, H.-Y. Shum, and J. Guo, "Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph," *arXiv preprint arXiv:2307.07697*, 2023.

[45] Y. Wu, N. Hu, S. Bi, G. Qi, J. Ren, A. Xie, and W. Song, "Retrieve-rewrite-answer: A kg-to-text enhanced llms framework for knowledge graph question answering," *arXiv preprint arXiv:2309.11206*, 2023.

[46] S. Knollmeyer, O. Caymazer, L. Koval, M. U. Akmal, S. Asif, S. G. Mathias, and D. Großmann, "Benchmarking of retrieval augmented generation: A comprehensive systematic literature review on evaluation dimensions, evaluation metrics and datasets," in *Proceedings of the 16th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2024)-Volume 3*. SciTePress, 2024, pp. 137–148.

[47] S. Es, J. James, L. E. Anke, and S. Schockaert, "Ragas: Automated evaluation of retrieval augmented generation," in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, 2024, pp. 150–158.

[48] H. Han, H. Shomer, Y. Wang, Y. Lei, K. Guo, Z. Hua, B. Long, H. Liu, and J. Tang, "Rag vs. graphrag: A systematic evaluation and key insights," *arXiv preprint arXiv:2502.11371*, 2025.

[49] Y. Xiao, J. Dong, C. Zhou, S. Dong, Q.-w. Zhang, D. Yin, X. Sun, and X. Huang, "Graphrag-bench: Challenging domain-specific reasoning for evaluating graph retrieval-augmented generation," *arXiv preprint arXiv:2506.02404*, 2025.

[50] Q. Zeng, X. Yan, H. Luo, Y. Lin, Y. Wang, F. Fu, B. Du, Q. Xu, and J. Jiang, "How significant are the real performance gains? an unbiased evaluation framework for graphrag," *arXiv preprint arXiv:2506.06331*, 2025.

[51] K. Cheng, G. Lin, H. Fei, L. Yu, M. A. Ali, L. Hu, D. Wang *et al.*, "Multi-hop question answering under temporal knowledge editing," *arXiv preprint arXiv:2404.00492*, 2024.

[52] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "Biobert: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.

[53] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[54] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey," *arXiv preprint arXiv:2403.14608*, 2024.

[55] A. N. Angelopoulos, S. Bates, C. Fannjiang, M. I. Jordan, and T. Zrnic, "Prediction-powered inference," *Science*, vol. 382, no. 6671, pp. 669–674, 2023.

[56] G. Scafarto, M. Ciortan, S. Tihon, and Q. Ferre, "Augment to interpret: Unsupervised and inherently interpretable graph embeddings," in *Asian Conference on Machine Learning*. PMLR, 2024, pp. 1183–1198.