Quantifying Retrieval Quality in GraphRAG: A Schema-Agnostic Approach

Thibaud Vanmechelen, Alexandre Achten, Zaineb Gabsi, Sabri Skhiri
Euranova
{thibaud.vanmechelen, alexandre.achten, zaineb.gabsi, sabri.skhiri}@euranova.eu

Abstract

While LLMs have achieved significant success in natural language tasks, their tendency to hallucinate remains a critical challenge. RAG tries to address this issue by grounding models in external data; however, standard vector-based RAGs often fail when working with highly interconnected datasets. GraphRAG has emerged as a superior alternative in this setting by modelling the relational topology, yet evaluating GraphRAGs remains challenging. Current benchmarks predominantly focus on the final LLM-generated output frequently overlooking the structural accuracy of the underlying retrieval process. In this paper, we propose a novel schema-agnostic framework for the automated generation of synthetic evaluation datasets from KGs. Unlike previous approaches, our framework establishes a rigorous, deterministic ground truth to specifically quantify the retriever performance across nine distinct query categories, including multi-hop and aggregation tasks. We demonstrate the utility of this benchmark by applying it to a biochemical KG and evaluating four diverse retrieval architectures. Our results indicate that agentic, LLM-driven retrievers provide the highest recall and reasoning capacity, effectively navigating complex topologies where other methods struggle. This work provides a robust, scalable methodology for performance tracking, shifting the evaluation of GraphRAG toward a more topologically precise standard.

Keywords: GraphRAG, KGs, Benchmarking, Synthetic Dataset Generation, Retrieval Quality

1. Introduction

In recent years, Large Language Models (LLMs) have revolutionised the field of Artificial Intelligence through their impressive generalisation capabilities in a wide range of tasks, including text generation and question answering. However, an important limitation of LLMs is that they tend to hallucinate (Xu et al., 2024; Huang et al., 2025), particularly within domains where they lack sufficient knowledge.

To mitigate these issues, researchers introduced Retrieval-Augmented Generation (RAG) (Lewis et al., 2020). This architecture leverages domain-specific document retrieval, providing the LLM with relevant context to produce more grounded and accurate answers. Despite its success, standard RAG is not perfect; it is not adapted for datasets containing highly interconnected information. Indeed, the typical vector database focuses on semantic similarity rather than structural relationships, which causes RAG models to struggle to answer complex queries that require traversing links between different data entries, an issue already mentioned in (Peng et al., 2025).

To address these relational limitations and improve the reasoning capabilities of LLMs, a new framework has recently been introduced: GraphRAG (Edge et al., 2024). Using graph databases to model the relationships between data nodes explicitly, GraphRAG enables LLMs to directly navigate the information topology. This switch from standard RAG, where simple metrics and retrieval mechanisms can be used, to GraphRAG leads to a whole new set of challenges regarding the way we navigate the Knowledge Graph (KG) and retrieve information.

Indeed, the response quality of GraphRAGs highly depends on the quality of the information that is retrieved from the graph because an LLM cannot generate accurate answers from erroneous/incomplete information. Therefore, focusing on the retrieval and being able to quantify its quality is an essential aspect of GraphRAGs. Currently, few papers and benchmarks focus on the retrieval part of those GraphRAGs. Most often, they evaluate the final answer generated by the LLM rather than the accuracy of the retrieved elements (Xiao et al., 2025; Edge et al., 2024).

In this paper, we propose a novel approach for benchmarking GraphRAGs. We have designed a framework that enables the generation of synthetic evaluation datasets from KGs, while maintaining a rigorous ground truth to specifically assess the retrieval quality of the GraphRAG architectures. This solution provides an alternative evaluation methodology that focuses on classical performance dimensions compared to those explored in this previous work (Houimli et al., 2025). By addressing the potential limitations (lack of quantitative evaluation, inability to ensure correctness) of this earlier approach, our new process aims to deliver a complementary benchmark to assess the performance of graph-based retrieval more robustly.

The primary contributions of our work are the following:

Proceedings of the Knowledge Graphs and Large Language Models Workshop 2026 (KG-LLM@LREC26), pages 176–189
May 16, 2026.

- We introduce a novel methodology for the automated generation of evaluation datasets based on existing KGs, specifically designed to benchmark the retrieval performance of GraphRAG systems.
- We demonstrate our approach by generating a specialised evaluation dataset based on a scientific biochemical KG derived from Hetionet (Himmelstein et al., 2016).
- We perform a detailed performance analysis of four retrieval methods using this new benchmark.

2. Related Work

Recently, GraphRAG has occupied a prominent role in scientific research. This interest is driven by the potential of graph-based architectures to ground LLMs and facilitate reasoning over interconnected data, which is expected to significantly enhance the performance of LLM agents across diverse domains.

Current research has addressed various aspects of the GraphRAG pipeline, ranging from retrieval strategies (Sanmartin, 2024; Wen et al., 2024; Luo et al., 2023; Tian et al., 2024; Guo et al., 2024; Wu et al., 2023) and comprehensive surveys (Han et al., 2024; Xiang et al., 2025; Zhang et al., 2025), to KG construction (Choi and Jung, 2025; Chen et al., 2025; Min et al., 2025) and benchmark design (Houimli et al., 2025; Xiao et al., 2025; Tang and Yang, 2024).

2.1. Retrieval Strategies

Designing efficient retrieval strategies capable of navigating the complex topology of KGs is an essential aspect of GraphRAG research. To this end, researchers have explored diverse methodologies that vary significantly in their logic (Sanmartin, 2024; Wen et al., 2024; Luo et al., 2023; Tian et al., 2024; Guo et al., 2024; Wu et al., 2023). Generally, these retrieval methods can be categorised into three main families: rule-based or symbolic traversal, neural or embedding-driven retrieval, and hybrid or multi-step reasoning strategies.

Rule-Based and Symbolic Traversal retrievers rely on a predefined set of rules and deterministic algorithms. These methods generally involve an initial phase to identify entry points within the graph, typically through Entity Linking and Relational Matching. In this case, Entity Linking is used to map entities identified in the query to specific graph nodes (Shen et al., 2021; Lumer et al., 2025), while Relational Matching identifies predicates mentioned in the text and finds the corresponding edges (Gao et al., 2024; Kim et al., 2023). To further expand the search space, some methods incorporate k-hop subgraph extraction (Jiang et al., 2023; Tian et al., 2024) or the retrieval of paths up to a specified length (Feng et al., 2020; Zhang et al., 2022), though the implementation of these traversal strategies varies significantly across the literature.

Neural and Embedding-Driven retrievers aim to leverage deep semantic representations to enhance the retrieval performance, going beyond the literal matching typically used in rule-based approaches. These methods generally project graph components and the natural language query into a shared vector space, which enables direct similarity-based search (Edge et al., 2024; Min et al., 2025) or the potential conditioning of graph embeddings on the query (Tian et al., 2024). This category encompasses a wide and diverse range of methods, such as Graph Neural Network (GNN) (Scarselli et al., 2009), including more specialised frameworks such as GraphSAGE (Hamilton et al., 2017) and Graph Attention Networks (GATs) (Veličković et al., 2018). Within these models, the embeddings are generated via message-passing mechanisms, where the query can be integrated, such as in GNN-RAG (Mavromatis and Karypis, 2024). Once these enriched embeddings are generated, the system can perform latent reasoning over the graph elements to identify relevant information.

Hybrid and Multi-Step Reasoning Strategies integrate neural reasoning with symbolic or iterative logic to address the limitations of single-stage retrieval. This category encompasses a very diverse range of designs. For instance, frameworks such as G-Retriever (He et al., 2024) combine semantic search with structural graph optimisation, using Prize-Collecting Steiner Tree (Ahmadi et al., 2024) algorithms for subgraph construction. Other approaches include iterative retrieval, where an autonomous agent explores the graph incrementally, gathering evidence through successive hops to build an answer set (Sanmartin, 2024; Wen et al., 2024; Luo et al., 2023; Guo et al., 2024). Moreover, Hierarchical Strategies, such as ArchRAG (Wang et al., 2025), reorganise graph elements into thematic communities; this abstraction allows the retriever to operate at varying levels of granularity, significantly improving retrieval quality for global or high-level queries.

2.2. Performance Evaluation

Evaluating RAG and GraphRAG systems is challenging because performance can be assessed at multiple stages of the pipeline, and each stage exposes different failure modes. The most common paradigm is end-to-end evaluation, where the final answer is scored against a reference using lexical-overlap metrics (e.g., EM/F1, ROUGE) or human/LLM-based judgments. For example, "From

Local to Global" (Edge et al., 2024) adopts an LLM-as-a-judge protocol to assess overall pipeline outputs.

While practical, this setup conflates two distinct failure modes: (i) retrieval errors (missing or incorrect graph evidence) and (ii) generation errors (hallucination, poor reasoning, or weak grounding). As a result, the score conflates missing evidence with the LLM's failure to use retrieved context.

This limitation has motivated a growing corpus of evaluation frameworks that target specific components of the RAG system. General-purpose benchmarks such as RAGBench (Friel et al., 2024) provide large-scale datasets and structured evaluation signals intended to diagnose RAG behaviour across domains. Nevertheless, RAGBench focuses primarily on text-based relevance and veracity, failing to assess the correct use of the graph structure.

In parallel, MultiHop-RAG (Tang and Yang, 2024) explicitly stresses retrieval by requiring systems to combine multiple supporting pieces of information, and they commonly report metrics such as MAP@K, MRR@K, and HIT@K. GraphRAG-Bench (Xiao et al., 2025) proposes a holistic framework intended to assess the entire lifecycle of a GraphRAG system, with an emphasis on retrieval efficiency rather than retrieval quality. Other studies (Cahoon et al., 2025) benchmark graph-based retrieval mechanisms for open-domain QA using standard metrics such as Accuracy, Recall, and Precision on off-the-shelf public datasets.

Alongside dataset-driven benchmarks, judge-based evaluation frameworks, such as RAGElo (Rackauckas et al., 2024), RAGAS (Es et al., 2024), and ARES (Saad-Falcon et al., 2024), aim to reduce reliance on costly human annotations by using LLMs as evaluators. However, these efforts primarily target semantic similarity/relevance. They are insufficient because they cannot diagnose failure cases where the model does not retrieve the entirety of the graph elements. This work moves beyond these semantic proxies, addressing the need to verify that the retrieved subgraph contains all the elements necessary for generation.

3. Material

Building upon this previous research (Houimli et al., 2025), we used a subset of the biomedical KG Hetionet (Himmelstein et al., 2016) as our primary data source. It served as a base for generating the new evaluation dataset and benchmarking the performance of the various retrievers. The graph architecture comprises five node types and ten different relationship types, represented in Figure 1, and was deployed in a Neo4j environment to facilitate high-performance retrieval. Unlike common GraphRAG implementations that rely on text-heavy nodes (such as document chunks or detailed entity descriptions), our database is primarily structural. It represents biochemical entities where the nodes have minimal attributes; therefore, the essential information is encoded within the graph's topology itself rather than its textual content.

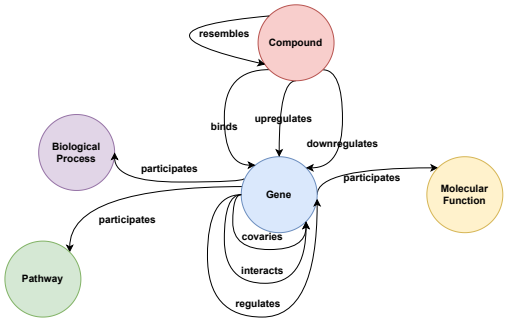

Figure 1: Schema-level overview of the filtered biochemical KG. Nodes correspond to biomedical entity types (*Gene*, *Compound*, *Molecular Function*, *Biological Process*, *Pathway*), and directed edges encode curated relations (*binds*, *up/downregulates*, *participates*, *interacts*, *regulates*, *covaries*). The graph contains 38,584 nodes and 1,488,879 edges (Houimli et al., 2025).

4. Methodology

4.1. Dataset Generation

The objective of this study is to develop a benchmark for the quantitative evaluation of various retrieval strategies. While previous work (Houimli et al., 2025) addressed the qualitative aspects of retrieval, leveraging the Ares framework (Saad-Falcon et al., 2024) and a fine-tuned BioBERT model (Lee et al., 2020) to assess the contextual relevance, this paper shifts the focus toward measuring "structural" alignment. More specifically, we aim to quantify the difference between the graph elements retrieved by the system and a predefined ground truth. Therefore, we need to establish a pipeline that generates datasets containing both natural language queries paired with their corresponding graph elements (that we denote as "ground truth elements").

This objective imposes three major constraints on the benchmark design:

- **Consistency**: we must ensure that the ground truth elements matched to each query indeed represent a verifiable ground truth.
- **Coverage**: To avoid evaluation bias, the ground truth must encompass the exhaustive set of relevant items for a specific query. Evaluating against an incomplete subset would just

measure the alignment with a specific selection rather than the global retrieval quality.

- **Generalisation**: While our study uses a specific biochemical KG, the benchmark that we develop is designed to be agnostic to the underlying schema, ensuring its usability across diverse KGs.

To achieve this objective, we implemented the pipeline illustrated in Figure 2.

The first step of our framework involves establishing a taxonomy of categories representing the fundamental operations that one could perform on a graph database. We assume these categories encompass the majority of query types encountered in practical KG applications. We define nine categories: *direct-node-retrieval*, *direct-edge-retrieval*, *aggregation-node*, *aggregation-edge*, *negative-node*, *negative-edge*, *out-of-scope*, *multi-hop*, and *intersection-node*. For each category, we define generalised Cypher[1] templates, which will serve as the basis for natural language question generation. By design, these templates target nodes or edges separately; therefore, the resulting answer sets will be homogeneous, containing only one type of element at a time.

The *direct-node-retrieval* category corresponds to questions requiring the direct extraction of nodes without the need for complex reasoning. Similarly, *direct-edge-retrieval* corresponds to the same retrieval objective but specifically for graph edges.

The *aggregation-node* and *aggregation-edge* categories represent question types that necessitate the quantification/counting of elements, such as counting nodes or edges that satisfy specific predicates, and are functionally similar to direct retrieval. However, the principal difference is that their output will be the number of elements satisfying the particular predicates rather than the complete list of ground truth elements.

Furthermore, the *negative-node* and *negative-edge* categories focus on "null-response" queries where there is no valid answer that exists within the graph. The motivation for these categories is to evaluate whether retrievers have a bias toward returning results in instances where no ground truth actually exists. The *out-of-scope* category has the same objective; however, the latter involves queries that are entirely unrelated to the KG. These are used to observe the retriever's behaviour when confronted with questions that lack both a valid answer and relevance to the underlying data.

Finally, the remaining two categories are specifically designed to highlight the advantages of KGs over traditional vector databases. A principal strength of GraphRAG is its ability for relational reasoning during the retrieval process. To evaluate this aspect, the *multi-hop* category targets questions that require the retriever to traverse specific paths across the graph topology to locate all the correct information. Similarly, the *intersection-node* category focuses on triangular structures, wherein a central node shares relationships with two other distinct nodes (either converging or diverging), requiring also structural reasoning to satisfy the query constraints.

Following the definition of these general categories, we established a set of sub-categories and corresponding Cypher templates for specific queries within each class. Each sub-category represents a distinct query template associated with the parent category. For example, within the *direct-node-retrieval* category, we defined the `exact-property-match`, `contain-property-match`, and `membership-match` sub-categories. The purpose of this sub-classification is to segment each category with greater precision. Furthermore, this granularity facilitates detailed performance tracking, which enables us to analyse how retrieval quality evolves across specific categories and sub-categories and to identify whether retrievers show shortcomings in certain areas. Template 1 presents one of the sub-categories that we used for the dataset generation.

```
template:
  "MATCH (n{node_type}) WHERE ALL
      (key IN keys($params[key])
      RETURN n"
type: "exact-property-match"
```

Template 1: Cypher template for the **exact-property-match** sub-category, retrieving nodes whose properties exactly match a parameter set (`$params`).

These categories and sub-categories generic templates provide the necessary framework for the generation of the evaluation dataset. Moreover, this design is sufficiently abstract to be transferred to any KG architecture, thereby satisfying the **Generalisation** constraint.

The second step of our pipeline involves populating the generic query templates with data from the KG to generate executable queries. To achieve this, we select at random information directly from the graph. This ensures data integrity, as the information comes from the existing database, thereby satisfying the **Consistency** constraint. With this retrieved information, we can complete the query parameters. For instance, in the *direct-node-retrieval* category, we randomly select a node and a subset of its properties; these properties and their corresponding values are then applied as filters within the generic query (illustrated in Template 1).

The *negative-node*, *negative-edge*, and *out-of-scope* categories are managed with a slightly different approach, as these classes contain no valid

[1]The query language used for Neo4j databases.

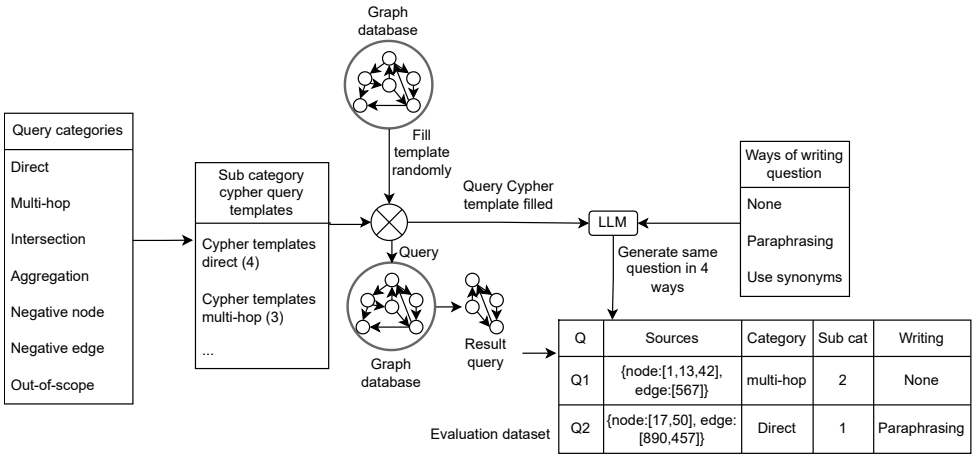

Figure 2: **Dataset generation framework**. This Figure presents the schema-agnostic dataset generation workflow that we propose in this paper. Note that both graph database icons represent the same database instance.

ground truth within the graph. For the former two, we deliberately select or generate incorrect elements, such as non-existent edges between nodes or non-existent nodes derived from a set of invalid values. On the other hand, the *out-of-scope* category does not use Cypher templates; instead, it is handled at the third step of our pipeline by a LLM.

Once the queries are fully populated, we execute them against the database. Indeed, relying only on the randomly selected elements used to fill the templates does not guarantee that they are the exhaustive set of answers for that specific Cypher query; thus, it would not satisfy the **Coverage** constraint. By executing the queries, we ensure that this property is met, as all ground truth elements are retrieved and will be used as the ground truth.

The third and final step of our dataset generation pipeline involves translating the populated Cypher queries, or generating them from scratch for the *out-of-scope* category, into natural language questions using a Large Language Model (LLM). In this step, the populated queries are provided as input to the LLM alongside a designated writing style. We aim to assess the sensitivity of retrievers to question formulation and measure, for instance, whether they are influenced by more abstract phrasing. To achieve this, we use three distinct writing styles: None, Paraphrasing, and Synonyms. The "None" style corresponds to a direct translation of the query. The "Paraphrasing" style reformulates the initial translation, while the "Synonyms" style substitutes key terms with their equivalents.

At the end of the pipeline, we produce a table similar to the one illustrated in Figure 2. This dataset includes a question identifier, the source graph elements (ground truth), the natural language question, the category, the sub-category, and the applied writing style.

4.2. Evaluation

To rigorously assess the performance of the retrievers, we implemented a structured evaluation workflow, illustrated in Figure 3, that measures the accuracy of retrieved contexts against the ground truth dataset. During the evaluation phase, the retrieval system processes each query to generate a set of predicted identifiers (nodes or edges), which are then compared against the expected ground truth using distinct metrics.

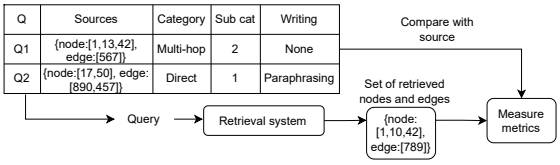

Figure 3: Evaluation framework.

For standard retrieval tasks where the objective is to identify specific nodes or edges, we use set theory that defines metrics as follows: let $G_{expected}$ be the set of ground-truth identifiers and $G_{retrieved}$ be the set of retrieved identifiers.

- **Precision (P)**: The proportion of retrieved identifiers that are relevant. If both sets of retrieved identifiers and expected identifiers are empty, the precision is considered maximal.

$$P = \frac{|G_{retrieved} \cap G_{expected}|}{|G_{retrieved}|} \quad (1)$$

- **Recall (R)**: The proportion of relevant identifiers that were successfully retrieved. If both sets of retrieved identifiers and expected identifiers are empty, the recall is considered maximal.

$$R = \frac{|G_{retrieved} \cap G_{expected}|}{|G_{expected}|} \quad (2)$$

- **F1-Score**: The harmonic mean of precision and recall.

For aggregation questions (e.g., "How many genes interact with X?"), we employ proxy metrics that measure numerical proximity rather than strict set overlap.

$$P_{proxy} = \min\left(\frac{C_{expected}}{C_{retrieved}}, 1.0\right) \quad (3)$$

$$R_{proxy} = \min\left(\frac{C_{retrieved}}{C_{expected}}, 1.0\right) \quad (4)$$

where C represents the count value. This formulation penalises deviations in either direction while capping the score at 1.0 for perfect matches.

For questions requiring the traversal of specific routes (e.g., "How are A and B connected?"), we introduce topological metrics to verify structural integrity:

- **Path Hit Rate**: A strict binary metric that returns 1 only if the entire set of ground-truth identifiers is contained within the retrieved subgraph, and 0 otherwise.
- **ROUGE-L**: We adapt the ROUGE-L score (Lin and Och, 2004) to evaluate the Longest Common Subsequence (LCS) of the retrieved IDs against the ground truth. This effectively measures the agent's ability to recover the correct sequence of connections in multi-hop scenarios.

In addition to standard quality metrics, the experimental setup captures the context retrieval time, measured in seconds, and the count of API calls made to the LLM. Together, these measurements give insights into the system's real-time responsiveness and its overall cost-efficiency.

4.3. Retrieval Strategies

Different retrieval strategies have been implemented and will be tested on this evaluation framework. They are inspired by the (Houimli et al., 2025) paper.

In the **Graph Traversal** strategy, a Text2Cypher model is used as described in (Ozsoy et al., 2025). This approach uses an LLM to translate natural language queries directly into executable, schema-aware Cypher statements. To address the potential for syntax errors in model-generated code, we incorporate a robust self-correcting feedback loop: if the initial query fails execution, the specific database error is fed back to the model to iteratively refine the syntax. The `gpt-4o` model is prompted, as it is mentioned to be the best-performing.

The Cypher query generated this way is then executed against the graph database, and raw results are stored.

The **Semantic Agent** leverages an LLM-driven[2] architecture initialised with a set of high-level tools designed for broad, content-based exploration. This agent process begins with a semantic vector search (via a `search_index` tool) to locate graph nodes that are conceptually similar to the input query, establishing relevant entry points into the graph. Then the agent traverses immediate connections using a relational neighbour retrieval tool (`get_neighbors_by_relationship`). This loop continues until either the maximum number of steps is reached or sufficient contextual data has been collected.

The **Chain-of-Thought Agent** acts as an extension of the Semantic Agent by providing a more extensive set of tools. It uses the same semantic search to find the seed nodes, then chooses through a reasoning process which tool is best adapted to fetch the information required to answer the question. These new tools allow the agent to count the number of neighbours, get common neighbours, and get specific nodes or edges. This extensive toolset enables the agent to dynamically navigate the graph structure, allowing it to synthesise information from disparate parts of the graph and address multi-hop queries that require an understanding of specific topological constraints.

For comparative analysis, we established a **Random baseline**. This baseline selects k random nodes and edges from the graph schema to serve as a lower-bound reference, allowing us to quantify the performance gain achieved by the semantic and agentic reasoning strategies.

5. Results & Discussion

To evaluate the performance of the various retrievers, we executed each of them against a dataset of 500 samples generated using our framework. To ensure that the assessment covered diverse retrieval challenges, this dataset was stratified across the distinct categories. We tracked performance metrics relative to the specific category, template, and writing style of each question. This granular result collection facilitates an advanced performance analysis, allowing us to identify potential bottlenecks and the specific domains in which each retriever excels.

5.1. Retrievers' Performance

The main objective of this evaluation is to evaluate the effectiveness of each technique with regard to the various performance metrics we use. Figure 4 displays the performance of the retrievers across our metrics: F1 score, recall, precision, hit rate, and ROUGE score.

[2] `gpt-5.2` was used as it was performing better.

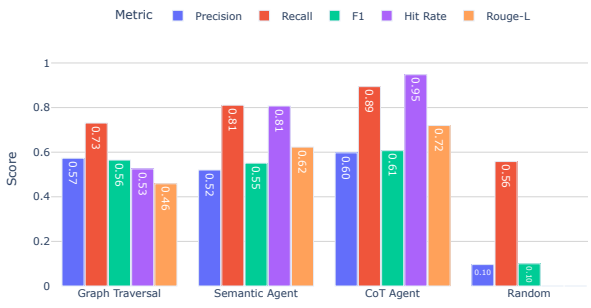

Figure 4: Retrievers performance on the generated dataset across different metrics.

Several observations can be drawn from the data in Figure 4. First, the baseline Random retriever consistently performs poorly. Given the scale of our KG, the probability of randomly selecting correct entities is negligible. Consequently, it obtained near-zero scores across precision, F1, hit rate, and ROUGE metrics. The 0.56 recall observed is an artefact of our metric definition, since in null-ground-truth cases (the ground truth is an empty set), we consider that the empty set matches by default the retriever's output. Moreover, the proxies that we use for the aggregation question evaluation also slightly bias recall, as they give non-null scores because the random baseline always returns k elements. These specific instances marginally inflate the recall despite the lack of true predictive precision (highlighted by the 0.1 precision, due to the numerical proximity used in the proxy measures).

The GraphTraversal Retriever achieved the second-highest precision, indicating that its retrieved elements are likely to belong to the ground truth. This retriever uses a Text-to-Cypher approach to translate natural language queries into executable code; thus, a successful translation ensures that the resulting execution focuses strictly on relevant entities, minimising the inclusion of external noise. However, this retriever struggles with multi-hop metrics. This suggests that while generating Cypher queries for direct retrieval is straightforward, synthesising logic for complex path traversals and multi-hop reasoning is significantly more difficult for the model, as confirmed by a deeper analysis of the results. Thus, while highly effective for simple queries, this approach performs more like a traditional RAG system.

On the other hand, the two Agentic Retrievers appear as the best retrievers for multi-hop queries. The Semantic Agent leverages a semantic search as an initial seeding process, augmenting its information pool by incrementally traversing the neighbours of current nodes afterwards. This iterative expansion closely mirrors the requirements for multi-hop reasoning over moderate path lengths. The Chain-of-Thought (CoT) Agent extends this capability thanks to a broader set of tools, such as direct node retrieval and counting tools. Our results confirm that the CoT agent outperforms the semantic variant due to this enhanced flexibility.

Despite their high recall, these agentic models generally demonstrate similar precision to the GraphTraversal approach. This discrepancy comes from the "over-inclusion" of elements during the reasoning process. Our ground truth elements are strictly defined by the output of the generic Cypher templates, which target either nodes or edges. In practice, when asked a question such as, "What is the relationship indicating that the Gene FOX04 participates in the Pathway Adaptive Immune System?", the CoT agent retrieves the correct relationship along with the IDs of the two terminal nodes. While these nodes are technically accurate in a broader context, they are not the specific targets of the query. Our "hard" precision metric penalises this behaviour, even though such additional context would likely enhance, rather than hinder, the quality of a final generated answer.

Therefore, applying a "hard" precision metric in this context may be overly pessimistic regarding the retriever's actual performance. The recall metric is better suited to our use case, as we hypothesise that at a later stage, during answer generation, an LLM could effectively filter the relevant context from the retrieved set when generating this final answer. This highlights a fundamental trade-off between our current benchmark and LLM-as-a-judge approaches. While a strict retrieval benchmark provides clear quantitative metrics, it may occasionally over-penalise a retriever for over-inclusion. Conversely, an LLM-as-a-judge might be more permissive of extra context, yet it is less capable of precisely assessing the total coverage of retrieved elements. This underscores the complementarity of these two benchmarking methodologies.

Overall, the performance of the agentic methods is highly satisfactory. They capture nearly all required ground truth elements, and the auxiliary information that they retrieve could likely contribute to a more comprehensive final response.

To extend our analysis, Figure 5 illustrates the F1 scores achieved by the CoT Agent across the different categories and writing styles.

The Figure 5 shows that the category type has a major influence on the performance. F1 scores vary drastically, ranging from near-perfection (0.9) to near-zero (0.1). The retriever performed best in the out-of-scope, direct-node-retrieval, negative-node, and multi-hop categories. These classes achieved F1 scores roughly between 0.7 and 0.9, which is a promising result suggesting that the agent is indeed capable of both standard retrieval and multi-hop reasoning.

On the other hand, the retriever performed worst

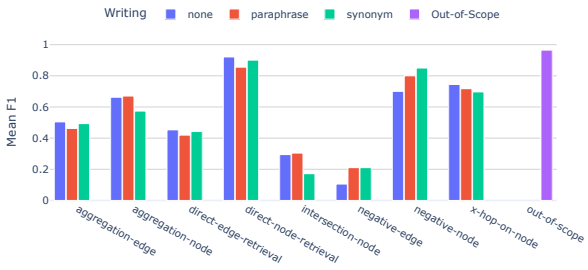

Figure 5: **Chain-of-Thought Agent Retriever.** Mean F1 score along categories.

in the *intersection-node* and negative-edge categories. This disparity is somewhat unsurprising; indeed, the KG contains a significant amount of intersections (hundreds of millions); thus, it is more difficult for the agent to navigate through the topology and find the correct ones. Additionally, for the negative-edge category, the retriever has to go through the complete topology to ensure that the edge does not exist, which is more error-prone than simply retrieving a direct edge and mainly affects the agentic retrievers.

Regarding the influence of writing styles, our findings suggest that they do not significantly impact the retriever's performance. F1 scores remained largely consistent across the "None", "Paraphrasing", and "Synonyms" styles within most categories. Even in instances where we observe variance, such as in the negative-edge, intersection-node, and negative-node categories, we do not observe any emerging trend. The "best" and "worst" performing styles were inconsistent across the board. We can thus conclude that the category type is the principal factor that influences the retriever's performance rather than the specific phrasing.

5.2. Retrievers' Resource Efficiency

Another important aspect of evaluating retrieval strategies is resource efficiency, more particularly regarding the retrieval time and the frequency of API calls. The performance metrics alone are not sufficient for determining the overall retriever's viability; a retriever that achieves marginally better accuracy at the cost of a significantly higher execution time or number of API calls may be unsuitable for real-world applications. Therefore, we also analysed the computational overhead associated with each retrieval method.

Table 1 details the average execution time and the mean number of API calls per query for each retriever, computed over the 500-question dataset.

Unsurprisingly, the Random Retriever is the most efficient, as it only selects k elements without requiring any external model interaction. Following this, the GraphTraversal Retriever is the second-most efficient strategy. It typically requires only a single API call to translate the natural language query into Cypher, with additional calls occurring only when the generation failed. Its relatively low execution time was partially influenced by a two-minute timeout threshold enforced on the database queries; while this threshold was reached only in a few isolated cases, it was used to maintain a consistent execution flow and mitigate the impact of extreme outliers (generally when cross-product queries were generated).

Retriever	Time (s)	API Calls
Random	2.683	0
GraphTraversal	4.256	1.078
Semantic Agent	6.893	2.096
CoT Agent	7.560	4.018

Table 1: Per-query resource consumption of the evaluated retrievers, expressed as average execution time (s) and average API call count.

Overall, each retriever demonstrated acceptable execution times and number of API calls. No single strategy showed extreme inefficiency; all evaluated methods have reasonable resource efficiency.

6. Conclusion

This paper introduces a novel framework for GraphRAG dataset generation, designed to be adaptable to any underlying graph database, alongside a rigorous evaluation methodology. A key difference from our previous work is the transition from a subjective LLM-as-a-judge approach to a complementary, significantly stricter evaluation framework. This methodology facilitates advanced performance tracking with a specific focus on multi-hop reasoning. We demonstrated the practical utility of our approach by applying it to a subset of the Hetionet (Himmelstein et al., 2016) dataset. Using the resulting benchmark, we evaluated four distinct retrieval methods and established that LLM-driven agentic retrievers exhibit superior performance. These techniques show an interesting capacity to reason over local context while strategically navigating the graph topology to retrieve the missing information. While our strict benchmark provides a rigorous baseline, it occasionally penalises 'over-inclusion', a behaviour that, while technically imprecise, may still offer value to the end-user. Therefore, to address this limitation and capture the full spectrum of a GraphRAG system quality, one should opt for a hybrid approach that balances deterministic metrics with subjective evaluation. Future work could apply this methodology to KGs across diverse domains to validate its generalisability.

7. Bibliographical References

Ali Ahmadi, Iman Gholami, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Mohammad Mahdavi. 2024. Prize-collecting steiner tree: A 1.79 approximation. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1641–1652.

Joyce Cahoon, Prerna Singh, Nick Litombe, Jonathan Larson, Ha Trinh, Yiwen Zhu, Andreas Mueller, Fotis Psallidas, and Carlo Curino. 2025. Optimizing open-domain question answering with graph-based retrieval augmented generation. In *Proceedings of the 1st workshop connecting academia and industry on Modern Integrated Database and AI Systems*, pages 1–11.

Haoting Chen, Sergio José Rodríguez Méndez, and Pouya Ghiasnezhad Omran. 2025. Open local knowledge graph construction from academic papers using generative large language models. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 2551–2559.

Seungmin Choi and Yuchul Jung. 2025. Knowledge graph construction: Extraction, learning, and evaluation. *Applied Sciences*, 15(7):3727.

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

Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. 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*, pages 150–158.

Yanlin Feng, Xinyue Chen, Bill Yuchen Lin, Peifeng Wang, Jun Yan, and Xiang Ren. 2020. Scalable multi-hop relational reasoning for knowledge-aware question answering. *arXiv preprint arXiv:2005.00646*.

Robert Friel, Masha Belyi, and Atindriyo Sanyal. 2024. Ragbench: Explainable benchmark for retrieval-augmented generation systems. *arXiv preprint arXiv:2407.11005*.

Yifu Gao, Linbo Qiao, Zhigang Kan, Zhihua Wen, Yongquan He, and Dongsheng Li. 2024. Two-stage generative question answering on temporal knowledge graph using large language models. *arXiv preprint arXiv:2402.16568*.

Tingfeng Guo, Qihui Yang, Chao Wang, Yulu Liu, Peng Li, Jian Tang, and Yuzhuo Wen. 2024. Knowledgenavigator: leveraging large language models for enhanced reasoning over knowledge graph. *Complex & Intelligent Systems*, 10(5):7063–7076.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, et al. 2024. Retrieval augmented generation with graphs (graphrag). *arXiv preprint arXiv:2309.15217*.

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

Daniel Himmelstein, Antoine Lizee, Faisal Alquaddoomi, Vincent Rubinetti, Dongbo Hu, Casey Greene, and Sergio Baranzini. 2016. hetio/hetionet: Data repository containing Hetionet downloads.

Asma Houimli, Zaineb Gabsi, and Sabri Skhiri. 2025. Evaluation of graphrag strategies for efficient information retrieval.

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

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

Jiho Kim, Yeonsu Kwon, Yohan Jo, and Edward Choi. 2023. Kg-gpt: A general framework for reasoning on knowledge graphs using large language models. *arXiv preprint arXiv:2310.11220*.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2020. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

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

Chin-Yew Lin and Franz Josef Och. 2004. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd annual meeting of the association for computational linguistics (ACL-04)*, pages 605–612.

Elias Lumer, Pradeep Honaganahalli Basavaraju, Myles Mason, James A Burke, and Vamse Kumar Subbiah. 2025. Graph rag-tool fusion. *arXiv preprint arXiv:2502.07223*.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2023. Reasoning on graphs: Faithful and interpretable large language model reasoning.

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

Congmin Min, Sahil Bansal, Joyce Pan, Abbas Keshavarzi, Rhea Mathew, and Amar Viswanathan Kannan. 2025. Towards practical graphrag: Efficient knowledge graph construction and hybrid retrieval at scale.

Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. 2025. Text2cypher: Bridging natural language and graph databases. In *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*, pages 100–108.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2025. Graph retrieval-augmented generation: A survey. *ACM Trans. Inf. Syst.*, 44(2).

Zackary Rackauckas, Arthur Câmara, and Jakub Zavrel. 2024. Evaluating rag-fusion with ragelo: an automated elo-based framework. *arXiv preprint arXiv:2406.14783*.

Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. 2024. Ares: An automated evaluation framework for retrieval-augmented generation systems. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 338–354.

Daniel Sanmartin. 2024. KG-RAG: Bridging the gap between knowledge and creativity.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Wei Shen, Yuhan Li, Yinan Liu, Jiawei Han, Jianyong Wang, and Xiaojie Yuan. 2021. Entity linking meets deep learning: Techniques and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 35(3):2556–2578.

Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries.

Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang wei Wang, Nitesh V. Chawla, and Panpan Xu. 2024. Graph neural prompting with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19208–19216.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks.

Shu Wang, Yixiang Fang, Yingli Zhou, Xilin Liu, and Yuchi Ma. 2025. Archrag: Attributed community-based hierarchical retrieval-augmented generation.

Yilin Wen, Zifeng Wang, and Jimeng Sun. 2024. Mindmap: Knowledge graph prompting sparks graph of thoughts in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10370–10388.

Yike Wu, Nan Hu, Guilin Qi, Sheng Bi, Jiaqi Ren, Aocheng Xie, and Wenting Song. 2023. Retrieve-rewrite-answer: A kg-to-text enhanced llms framework for knowledge graph question answering.

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

Yilin Xiao, Junnan Dong, Chuang Zhou, Su Dong, Qian-wen Zhang, Di Yin, Xing Sun, and Xiao Huang. 2025. Graphrag-bench: Challenging domain-specific reasoning for evaluating graph retrieval-augmented generation. *arXiv preprint arXiv:2506.02404*.

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. 2024. Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint arXiv:2401.11817*.

Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Hao Chen, Yilin Xiao, Chuang Zhou, Junnan Dong, Yi Chang, and Xiao Huang. 2025. A survey of graph retrieval-augmented generation for customized large language models.

Xikun Zhang, Antoine Bosselut, Michihiro Yasunaga, Hongyu Ren, Percy Liang, Christopher D Manning, and Jure Leskovec. 2022. Greaselm: Graph reasoning enhanced language models for question answering. *arXiv preprint arXiv:2201.08860*.

A. Additional Experiments

Appendix A presents additional results detailing the F1 score distribution across all query categories. These figures follow the format of Figure 5, and extend the evaluation to the Semantic Agent, Random Retriever, and Graph Traversal Retriever.

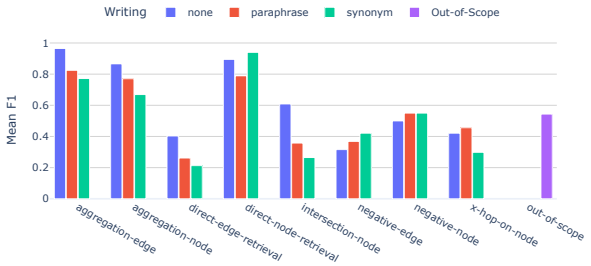

Figure 6: **GraphTraversal Retriever.** Mean F1 score along categories.

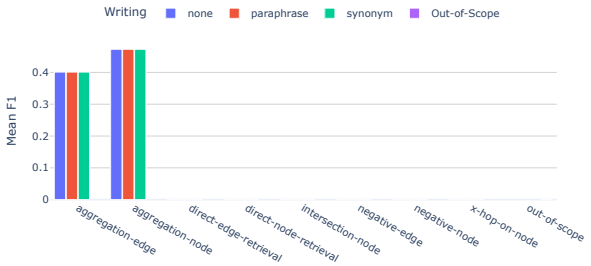

Figure 7: **Random Retriever.** Mean F1 score along categories.

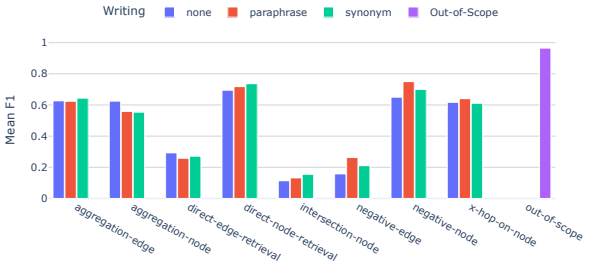

Figure 8: **Semantic Agent Retriever.** Mean F1 score along categories.

B. Categories Taxonomy

Appendix B further formalises the query taxonomy that we introduced in Section 4.1. Figure 9 provides a schematic representation of these categories, illustrating the structural patterns that define the retrieval challenges that we consider in our framework.

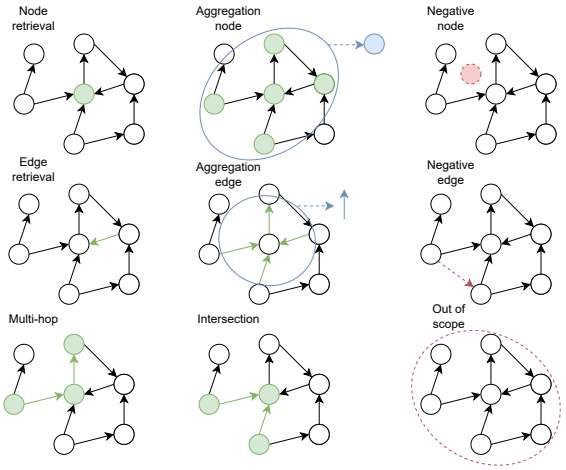

Figure 9: **Graphical representation of the query taxonomy.** Green highlights indicate the expected retrieval set for each category, while red highlights illustrate the lack of ground truth elements. Blue boundaries represent aggregation operators.

C. Cypher Query Templates

In Appendix C, we provide the detailed list of query templates used during the question generation process.

While the official Cypher specification prescribes the use of a colon (`:`) to denote the node labels, our templates omit this prefix to facilitate structural flexibility. By treating the label specification as a dynamic string injection within the brackets, the framework can handle more easily either typed or

untyped node searches without altering the underlying template logic.

```
direct-node-retrieval:
  1:
    template: "MATCH (n{node_type
       }) WHERE ALL(key IN keys(
       $params) WHERE n[key] =
       $params[key]) RETURN n" #
       Note: the format is n{
       node_type} and not n: {
       node_type} so that it is
       possible to use the same
       template for MATCH(n) and
       MATCH(n: type)
    type: "exact-property-match"

  2:
    template: "MATCH (n{node_type
       }) WHERE n.{prop} CONTAINS
        $params RETURN n"
    type: "contain-property-match
       "

  3:
    template: "MATCH (n{node_type
       }) WHERE n.{prop} IN
       $params RETURN n"
    type: "membership-match"

direct-edge-retrieval:
  1:
    template: "MATCH (n1{
       node_type1})-[r{rel_type
       }]->(n2{node_type2}) WHERE
        ALL(key IN keys($params)
       WHERE n1[key] = $params[
       key]) RETURN r"
    type: "known-source-matching"

  2:
    template: "MATCH (n1{
       node_type1})-[r{rel_type
       }]->(n2{node_type2}) WHERE
        ALL(key IN keys($params)
       WHERE n2[key] = $params[
       key]) RETURN r"
    type: "known-target-matching"

  3:
    template: "MATCH (n1{
       node_type1})-[r{rel_type
       }]-(n2{node_type2}) WHERE
       ALL(key IN keys($params)
       WHERE n1[key] = $params[
       key]) RETURN DISTINCT r"
    type: "known-source-matching-
       undirected"

  4:
    template: "MATCH (n1{
       node_type1})-[r{rel_type
       }]->(n2{node_type2}) WHERE
        ALL(key IN keys($params1)
        WHERE n1[key] = $params1[
       key]) AND ALL(key in keys(
       $params2) WHERE n2[key] =
       $params2[key]) RETURN r"
    type: "known-extremities-
       matching"

  5:
    template: "MATCH (n1{
       node_type1})-[r{rel_type
       }]-(n2{node_type2}) WHERE
       ALL(key IN keys($params1)
       WHERE n1[key] = $params1[
       key]) AND ALL(key in keys(
       $params2) WHERE n2[key] =
       $params2[key]) RETURN
       DISTINCT r"
    type: "known-extremities-
       matching-undirected"

x-hop-on-node: # to stay general,
     never precise the type of
    relationship
  1:
    template: "MATCH p = (n1{
       node_type1})-[{rel_type}*{
       x}]->(n2{node_type2})
       WHERE ALL(key IN keys(
       $params) WHERE n1[key] =
       $params[key]) RETURN nodes
       (p) LIMIT 1"
    type: "path-from-source-node"

  2:
    template: "MATCH p = (n1{
       node_type1})-[{rel_type}*{
       x}]->(n2{node_type2})
       WHERE ALL(key IN keys(
       $params1) WHERE n1[key] =
       $params1[key]) AND ALL(key
        in keys($params2) WHERE
       n2[key] = $params2[key])
       RETURN nodes(p) LIMIT 1"
    type: "path-from-extremities"

aggregation-node:
  1:
    template: "MATCH (n{node_type
       }) WHERE ALL(key IN keys(
       $params) WHERE n[key] =
       $params[key]) RETURN COUNT
       (n) AS count"
    type: "property-node-count"

  2:
    template: "MATCH (n1{
       node_type1})-[r{rel_type
       }]->(n2{node_type2}) WHERE
        ALL(k IN keys($params)
       WHERE n1[k] = $params[k])
       RETURN COUNT(n2) AS count"
    type: "source-neighbor-count"
```

```
    3:
      template: "MATCH (n1{
         node_type1})-[r{rel_type
         }]->(n2{node_type2}) WHERE
          ALL(k IN keys($params)
         WHERE n2[k] = $params[k])
         RETURN COUNT(n1) AS count"
      type: "target-neighbor-count"

    4:
      template: "MATCH (n1{
         node_type1})-[r{rel_type
         }]-(n2{node_type2}) WHERE
         ALL(k IN keys($params)
         WHERE n1[k] = $params[k])
         RETURN COUNT(DISTINCT n2)
         AS count"
      type: "neighbor-count"

    5:
      template: "MATCH (n{node_type
         }) WHERE n.{prop} CONTAINS
          $params RETURN COUNT(n)
         AS count"
      type: "contain-property-count
         "

    6:
      template: "MATCH (n{node_type
         }) WHERE n.{prop} IN
         $params RETURN COUNT(n) AS
          count"
      type: "membership-count"

  aggregation-edge:
    1:
      template: "MATCH (n1{
         node_type1})-[r{rel_type
         }]->(n2{node_type2}) WHERE
          ALL(key IN keys($params)
         WHERE n1[key] = $params[
         key]) RETURN COUNT(r) AS
         count"
      type: "known-source-count"

    2:
      template: "MATCH (n1{
         node_type1})-[r{rel_type
         }]->(n2{node_type2}) WHERE
          ALL(key IN keys($params)
         WHERE n2[key] = $params[
         key]) RETURN COUNT(r) AS
         count"
      type: "known-target-count"

    3:
      template: "MATCH (n1{
         node_type1})-[r{rel_type
         }]-(n2{node_type2}) WHERE
         ALL(key IN keys($params)
         WHERE n1[key] = $params[
         key]) RETURN COUNT(
         DISTINCT r) AS count"
      type: "known-source-count-
         undirected"

    4:
      template: "MATCH (n1{
         node_type1})-[r{rel_type
         }]->(n2{node_type2}) WHERE
          ALL(key IN keys($params1)
          WHERE n1[key] = $params1[
         key]) AND ALL(key in keys(
         $params2) WHERE n2[key] =
         $params2[key]) RETURN
         COUNT(r) AS count"
      type: "known-extremities-
         count"

    5:
      template: "MATCH (n1{
         node_type1})-[r{rel_type
         }]-(n2{node_type2}) WHERE
         ALL(key IN keys($params1)
         WHERE n1[key] = $params1[
         key]) AND ALL(key in keys(
         $params2) WHERE n2[key] =
         $params2[key]) RETURN
         COUNT(DISTINCT r) AS count
         "
      type: "known-extremities-
         count-undirected"

  negative-node:
    1:
      template: "MATCH (n{node_type
         }) WHERE n.{prop} =
         $params RETURN n"
      type: "property-inequality"

    2:
      template: "MATCH (n{node_type
         }) WHERE NOT n.{prop} IN
         $params RETURN n"
      type: "membership-inequality"

  negative-edge:
    1:
      template: "MATCH (n1{
         node_type1}), (n2{
         node_type2}) WHERE ALL(k
         IN keys($params1) WHERE n1
         [k] = $params1[k]) AND ALL
         (k IN keys($params2) WHERE
          n2[k] = $params2[k]) AND
         NOT (n1)-[{rel_type}]->(n2
         ) RETURN n1, n2"
      type: "missing-directed-edge"

    2:
      template: "MATCH (n1{
         node_type1}), (n2{
         node_type2}) WHERE ALL(k
         IN keys($params1) WHERE n1
         [k] = $params1[k]) AND ALL
         (k IN keys($params2) WHERE
```

```
          n2[k] = $params2[k]) AND
        NOT (n1)-[{rel_type}]-(n2)
          RETURN n1, n2"
    type: "missing-undirected-
        edge"

out-of-scope:
  1:
    template: "Totally out-of-
        scope question (about any
        potential domain).
        Examples: {examples}"
    type: "out-of-scope-general"

  2:
    template: "out-of-scope in
        the {domain_name} domain.
        Examples: {examples}"
    type: "out-of-scope-specific"

intersection-node:
  1:
    template: "MATCH (n1{
        node_type1})-[r1{rel_type1
        }]->(n{node_type_target})
        <-[r2{rel_type2}]-(n2{
        node_type2}) WHERE ALL(k
        IN keys($params1) WHERE n1
        [k] = $params1[k]) AND ALL
        (k IN keys($params2) WHERE
          n2[k] = $params2[k])
        RETURN DISTINCT n"
    type: "intersection-two-
        sources-converging"

  2:
    template: "MATCH (n1{
        node_type1})<-[r1{
        rel_type1}]-(n{
        node_type_target})-[r2{
        rel_type2}]->(n2{
        node_type2}) WHERE ALL(k
        IN keys($params1) WHERE n1
        [k] = $params1[k]) AND ALL
        (k IN keys($params2) WHERE
          n2[k] = $params2[k])
        RETURN DISTINCT n"
    type: "intersection-two-
        sources-diverging"
```

Cypher templates across all categorized query domains. These patterns illustrate the mapping between the theoretical taxonomy and the concrete database implementation.