Reducing Hallucinations in Complex Question Answering using Simple Graph-based Retrieval-Augmented Generation (long version)

Christopher J. Wedge, Joshua Stutter, Danny Dixon, Jacek Cała
National Innovation Centre for Data, Newcastle University
Newcastle upon Tyne, UK
{Chris.Wedge,Joshua.Stutter,D.J.Dixon2,Jacek.Cala}@ncl.ac.uk

ABSTRACT

Large language models (LLMs) have fundamentally transformed the landscape of Natural Language Processing (NLP), although they remain susceptible to errors. Retrieval-augmented generation (RAG) systems have emerged as a common deployment scenario seeking to both avoid the well known risk of the LLM "hallucinating" information, and to enable reasoning and question answering over proprietary information that the LLM did not have access to during training without resorting to expensive model fine-tuning.

In this work, we explore the idea of using a lightweight graph structure with a relatively simple graph schema, to support the RAG subsystem via a dedicated toolset. We design an agentic system with a variety of vector search and graph query tools operating over a structured dataset based on a curated subset of English Wikipedia articles, and evaluate its performance on questions from MoNaCo, a challenging Wikipedia based benchmark of complex question answering (QA) tasks.

Our results show that the introduction of graph-based tools can significantly increase the precision and recall of factual correctness, can halve the number of hallucinated answers, and achieves the highest fine-grained truthfulness score among the three evaluated scenarios. All this with a modest increase in token usage.

1 INTRODUCTION

Since the advent of the transformer architecture many competing Large language models (LLMs) have been developed with strong capabilities in understanding, reasoning, summarisation, and question answering. Their ability to generate coherent and contextually relevant responses has led to widespread adoption in many domains. Despite these advances, LLMs and LLM-based systems remain prone to a variety of failure modes.

Type I (false positive) and Type II (false negative) errors are not only core concepts in statistical inference, but also provide a useful framework for describing fundamental failure modes in statistical, analytical, and machine-learning models, including LLMs. In the case of LLMs and LLM-based systems, the ramifications of these failures can be severe and multifaceted. False positives may manifest as hallucinated content, retrieval of the incorrect context, or the use of inappropriate tools, whereas false negatives may lead to omissions, missed relevant context, or failure to invoke the correct tools. In practice, such errors often occur simultaneously in multiple stages of system operation [12]. Therefore, any method that improves the quality of a model or a model-based system is critical, insofar as it reduces both types of error simultaneously rather than merely shifting performance along the precision-recall tradeoff. This consideration becomes even more vital for complex question answering problems, such as multi-hop (MHQA), cross-document (CDQA) and multi-entity question answering (MEQA) [5, 38, 57], where knowledge from multiple sources must be retrieved, aggregated, and reasoned over in order to provide an answer.

As an example, consider the following question:

> *Can you name all the battles between the Dutch and English in the First, Second and Third Anglo-Dutch Wars, and list the victor of each battle?*

To answer this question, one needs to perform a sophisticated retrieval and reasoning process. In fact, it requires multi-entity and multi-hop reasoning, and cross-document access, all at once [50].

First, as with MEQA, the process must operate on a set of entities (in this case nations, wars, battles and victors) and needs to perform per-entity analysis and filtering. Second, as with MHQA, it requires hierarchical traversal, repeated relational expansion and nested decomposition. Finally, as with CDQA, it needs to fetch information from multiple sources: war documents, battle descriptions, and historical records. A single document is unlikely to answer this question; the information required is typically dispersed across dozens of documents.

These types of questions pose a significant challenge to current state-of-the-art LLM-based systems. There are multiple reasons for this, but in this work we focus on one aspect: Retrieval-Augmented Generation (RAG) [36]. RAG is a common element of many LLM-based systems, specifically those used for document processing, analytics, and question answering. In this context, we address complex QA problems, like the one presented above, using a unified vector and graph database with a series of pre-defined tools to improve retrieval from a knowledge base (KB). We compare our solution, vector+graph RAG, with simple vector RAG and zero-shot approaches. Using LLM agents prompted for safe refusal, that is models explicitly instructed to state 'unknown' when the necessary information could not be found in the provided KB, our system more than halved the proportion of complex questions that the agent refused to answer. At the same time, it improved the ratio of correct over incorrect answers compared to the zero-shot approach. Thus, it shows a promising direction of research in which graphs are coupled with more traditional vector-based retrieval methods.

The use of knowledge retrieval systems (RAG) to extend the capability of LLM agents is the preferred method by which general-purpose LLMs are applied to reason over specialised and / or proprietary knowledge [16]. By augmenting a user query with data queried by a knowledge retrieval system – data which may not be in the LLM's training data – the LLM can answer the question using in-context learning, extending its capabilities and reducing hallucinations [41].

Whilst early RAG systems consisted of a single retrieval-then-answer pipeline [22], it is now typically expected that RAG systems should be agentic for all but the simplest queries, so as to allow the LLM to practise question decomposition, reflection, and follow-up or validation queries using Chain of Thought (CoT) reasoning. Limiting ourselves to systems querying a defined KB (rather than Internet search), the most commonly-used tool within agentic RAG is vector search, where a vector database of text chunks from source documents is queried by an embedding model to assess semantic similarity [4, 35]. Beyond this, a variety of other tools have been developed that purport to improve retrieval in terms of its accuracy and / or efficiency.

Among these new retrieval methodologies, those that employ graphs are gaining popularity. Rather than operating over unstructured, chunked text documents, a knowledge graph (KG) is either harnessed directly or created using an LLM from a set of related documents (commonly known as *GraphRAG* [13]). However, a major challenge with graph-based solutions is the creation of a useful KG. On the one hand, KGs with rich structures are potentially desirable, but they necessitate a complex schema and thus risk filling the fixed context window of an LLM [7]. On the other hand, small schemas are more easily digestible by an LLM, but may not faithfully and comprehensively capture the true status of the KB.

In this work, we explore the latter idea, i.e. using a lightweight graph structure, with a relatively simple graph schema, to support the RAG subsystem via a dedicated toolset. We adopt this very practical approach in which a KB is constructed from semi-structured documents, including only a high-level structure of document titles, section titles, and constituent paragraphs, with links to other paragraphs or documents. This scenario is well suited to many real-world KBs that comprise various document types but lack a sophisticated and / or rich KG representation, such as those proposed by other work [11].

The tools we developed use the *Neo4j* graph database engine and its accompanying *Cypher* query language[1] to traverse the graph of Wikipedia articles, article sections and section paragraphs. We compare our graph-based approach with a typical vector-based RAG system on a set of sophisticated queries that require gathering and reasoning over information from multiple sources (articles, sections, and paragraphs).

Although work has been done to evaluate vector RAG against *GraphRAG* methods [17, 33, 39, 55], in this work we are concerned instead with evaluating vector RAG against knowledge base question answering (KBQA) from the perspective of unstructured and semi-structured data. There is as yet no research that directly evaluates vector RAG against KBQA on structured data. To address this deficiency, we aim to answer the following research question:

> *Do queries over structured knowledge bases, such as knowledge graphs, improve the performance of complex QA over unstructured RAG?*

We design an agentic system with a variety of vector search and graph query tools operating over a structured dataset based on a curated subset of English Wikipedia articles, and evaluate its performance on a challenging Wikipedia QA benchmark (MoNaCo, *vide infra*). Our experiments measure both answer accuracy and token usage, with the objective of a strong system being to maximise answer accuracy while minimising token usage.

Additionally, we compare both solutions against a zero-shot approach that relies purely on the knowledge acquired during model training. Our results show that the introduction of graph-based RAG significantly reduces hallucinated content and improves truthfulness scores, all with only a modest increase in token usage compared to the more conventional vector RAG approach, and using a lightweight and easy to construct graph knowledge base.

In summary, our main contributions are as follows. We:

- Identify a complex-QA benchmark suitable for comparing structured and unstructured retrieval.
- Design a unified *Cypher*-based toolset that affords efficient navigation over a knowledge graph.
- Conduct extensive experiments on the efficacy of RAG and KBQA tools.
- Release an evaluation KG based on a significant proportion of English Wikipedia, suitable for both RAG and KBQA.[2]

2 BACKGROUND AND SCOPE

As has recently been observed, the expectations of LLM-based and agentic systems are increasing beyond simple question answering problems [16]. Indeed, real-world queries often require retrieving information from multiple sources, summarising and reasoning before an answer can be produced. In this context, our reliance on systems that may hallucinate their answers, even if only at some of the substages of the whole answering pipeline, is problematic and undermines trustworthiness of these systems. Therefore, the need to design and develop solutions that can reduce hallucination (confabulation) and improve trust are critical.

Recently, there have been various attempts to address this problem. They revolve around improving four main aspects: query understanding and decomposition, evidence gathering (i.e. information retrieval), reasoning, and response synthesis and grounding (cf. Figure 1).

In this work we focus on the second step – information retrieval – and aim to verify the extent to which the introduction of a knowledge graph store with a simple document graph structure within a complex QA system can improve overall system performance. To accomplish this, we embed our QA system within an evaluation framework that includes a benchmark dataset and an evaluation step. Our goal is to evaluate the end-to-end factual correctness and truthfulness of the QA system. Our comparison is deliberately scoped to generic vector RAG versus vector+graph RAG over a simple document graph; more sophisticated *GraphRAG* [13] approaches and richer, LLM-constructed KG representations are beyond the scope of this study. Our simple graph is intended as an *enabler* of

[1] https://neo4j.com

[2] https://github.com/NICD-UK/graph-based-rag-qa/; to be shared upon publication.

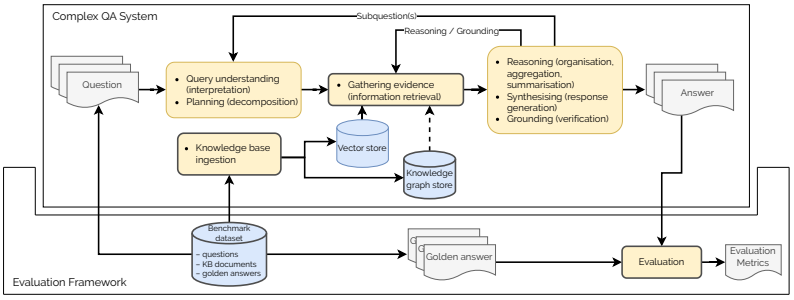


Figure 1: A Graph-based Question Answering pipeline embedded within an evaluation framework. Highlighted in black are items of our focus.

structure-aware retrieval rather than a competing heavyweight knowledge graph.

2.1 Information Retrieval

In the domain of information retrieval for LLM agents, there are two main approaches: parametric and non-parametric [8]. Parametric methods, such as fine-tuning, are beneficial for encoding knowledge directly within a model's weights, but they can be incredibly resource intensive and are inflexible to new knowledge [48]. Non-parametric solutions, such as RAG, on the other hand, are used directly with base models. By augmenting a user query with data returned by a knowledge retrieval system, i.e. the data that may not be in the LLM's training data, the LLM can answer the question using in-context learning, extending its capabilities and reducing hallucinations [41].

Several paradigms have emerged within state-of-the-art RAG systems, to effect this application of knowledge retrieval to LLMs. Vector search is still the most common method, as it can be applied directly to unstructured text, such as a corpus of PDF documents on a topic. As a result, most RAG benchmarks and evaluations begin from the assumption that the initial data will be of an unstructured form [25]. For natively structured data, text-to-SQL [10, 37] or text-to-SPARQL [34, 47] are common techniques, where the LLM is tasked with directly querying a relational or graph database respectively.

For the case of structured data querying over graphs, methods either:

- use graph metrics, such as community detection, alongside vector retrieval, to improve results by retrieving small subgraphs [13],
- verbalise entities and relationships into text for embedding [1], or
- generate structured queries, e.g. in SPARQL or *Cypher*, that are then executed directly against the graph [18].

Here, we propose an alternative approach. We expose the agent to specific tools, backed by handwritten queries, which enable flexible structured data retrieval while, at the same time, relieve the agent from the task of generating valid data queries – a potential source of failure and security vulnerability.

2.2 Benchmark dataset

An external knowledge base offers three advantages over relying on an LLM's parametric memory: it can be updated continuously rather than being frozen at training time [36, 45]; it supplies domain-specific or proprietary knowledge that the model lacks [32]; and it sidesteps the degraded performance of LLMs on long or complex inputs, since even large context windows are usably shorter than advertised [29]. Regardless of context size or parameter count, there will always be datasets too large or specialised for an LLM to reason over unaided.

Typical vector RAG and automatically-constructed *GraphRAG* both ingest *unstructured* documents. By contrast, a graph database operates on already-structured data, and its key advantage is a controlled schema: rather than *searching* for semantically similar text, it lets us *query* structure directly. Using this lens, we reviewed recent datasets and benchmarks to find one usable in a knowledge-graph RAG (KG RAG) system, employing the following criteria:

(1) Large dataset (>10,000 records); for a realistic scenario in which the KB does not fit into the LLM context window.
(2) End-to-end QA; in contrast to retrieval only or generation only evaluation.
(3) Data appropriate for both structured and unstructured retrieval; for fair comparison between simple vector-based RAG and KG RAG.
(4) Multi-hop, multi-entity reasoning; to tackle the challenging task of complex QA which the current LLM-based systems struggle to solve.

A search within the literature for benchmark datasets for RAG, especially graph RAG, yielded many candidates. A summary of this survey is presented in Appendix A.

In the end we selected the *MoNaCo* benchmark dataset [53]. Although it does not directly include a structured KB we built a simple knowledge graph to provide a structured index to the source documents (Wikipedia articles) necessary as context. This benchmark dataset, contains 1315 complex questions, one of which was shown earlier in the introduction. They are human written in natural language rather than being LLM-generated as in some alternatives. The answer to each question is a result to be determined by combining or reasoning over multiple pieces of information. The published, yet preliminary, studies of this dataset show poor performance by modern LLMs in a zero-shot or basic RAG context, which requires

the use of LLM reasoning approaches [53]. Within an agentic vector RAG implementation, this might require multiple queries to retrieve the necessary information, but we postulate that using a KG RAG system it may be possible to retrieve information directly via a suitable graph query.

All this provides a very realistic scenario. It is relevant to answering questions from a store of documents based on a simple and easy to create graph rather than a more comprehensive graph database. The latter might not be present in many application settings. Although this dataset is less than ideal in that the QA pairs are based on publicly available Wikipedia data, the complexity of questions, and the reported failure of zero-shot LLMs to answer these accurately without a retrieval subsystem [53], ensure that the agent cannot rely solely on parametric knowledge.

2.3 Evaluation

The decision on the most appropriate RAG evaluation metrics must take into consideration the benchmark dataset in use, to ensure that the required information is available to compute the chosen metrics in both vector- and KG-RAG scenarios. For definitions of key metrics see the two recent reviews of RAG evaluation methods in [20, 58]. Here, we consider which of the many available metrics are appropriate.

RAG-supported systems comprise separate retrieval and generation components [36]. As a result, any evaluation experiment has the option to focus on only one of the two individual components, or the entire end-to-end pipeline [58]. For the results to be widely explainable, in this study we focus on metrics that are relevant to the end-to-end RAG system. We report a headline difference in overall performance when adding a KG to a RAG system, rather than focusing on a specific sub-component.

2.3.1 Traditional metrics. The simplest metrics for evaluating generated answers come from the domain of NLP and tend to work at a character level. These include Exact Match, BLEU and ROUGE. The challenge when applying such metrics to LLM output is that a correct answer, identical semantically but phrased differently to the reference answer, would be given a low score. They are, therefore, ineffective when evaluating free-form LLM responses [20, 56].

2.3.2 Embedding-based metrics. Metrics based on semantic similarity are more appropriate for our purpose, as they seek to evaluate whether the meaning of the generated and reference answers are the same. A typical example would be BERTScore, which generates embeddings from a BERT pre-trained transformer for both generated and reference answers, allowing token level similarity to be addressed [60]. Similar models include BART and RoBERTa [20]. These approaches have the advantage of capturing some semantic meaning, while still relying on a deterministic scoring method. However, they are less flexible and give lower evaluation quality than LLM prompting methods [21], and some exhibit specific blind-spots [26].

2.3.3 LLM-based evaluators. The final major class of evaluation metrics to discuss are the so-called *LLM-as-a-judge* methods. These rely on prompting an LLM to evaluate some material from the RAG chain against predefined criteria. This general approach can be used for a wide variety of different metrics assessing the RAG system at various stages; the required inputs to evaluate these are summarised in Table 1.

Table 1: Summary of different LLM-based RAG evaluation metrics with required inputs and score range; underlined are metrics used in this work.

Metric	User Query	Retrieved Context	LLM Response	Golden Answer	Score Range
Answer Relevancy	•		•		0 to 1
Faithfulness	•	•	•		0 to 1
Context Precision	•	•		•	0 to 1
Context Recall	•	•		•	0 to 1
Factual Correctness			•	•	0 to 1
CRAG	•		•	•	1, 0 or -1

For example, the Ragas framework [51] has *context precision* and *context recall* metrics, in which retrieved contexts are processed along with the user query and golden answer to determine whether they are useful in answering that query. The computation of these scores is achieved by decomposing the number of distinct claims within the contexts to assess precision and recall on a claim-by-claim basis, and the reference answer is used as a proxy for a golden context to allow evaluation even when a curated dataset containing the golden context is not available.

Two LLM-based metrics commonly seen to be important for RAG systems are *faithfulness* and *answer relevancy*. Both of these metrics can be computed using the user query and LLM-generated response without requiring a golden answer, and were introduced by Ragas to address the specific challenge of RAG evaluation when a golden answer is not available [15]. *Answer relevancy* aims to measure alignment between the generated response and the user query, by generating three additional queries from the response and computing the average cosine similarity between the embedding of these queries and the actual query. *Faithfulness* additionally uses the retrieved contexts, and breaks down the individual claims in the LLM response to check these for support from the context. The ratio of supported claims to total claims is then returned as a score from 0 to 1 (e.g. a faithfulness score of 1 means that all claims in the answer are justified by the retrieved context).

A potential challenge to the evaluation of faithfulness is that for complex questions with extensive reasoning or aggregation over multiple context chunks even correct answers will not have a direct corresponding claim in a single context chunk, which may lead to errors in the evaluation. As the MoNaCo benchmark does not provide specific golden context chunks directly, and prompting for metrics using context chunks is not developed for the scenario of reasoning and aggregating from multiple chunks, the context precision/recall and faithfulness metrics were deemed inappropriate and are not used in this study.

A more general LLM-as-a-judge metric is the *factual correctness* score in Ragas, which is computed by a direct comparison between the generated and reference answers. In this case, the LLM is asked to decompose each input into a number of individual claims. By assessing using an LLM judge whether each claim in the response list matches a claim in the reference list and vice versa, claims are labelled as true positives, false positives or false negatives, allowing the system to calculate precision, recall or F_1 score according to

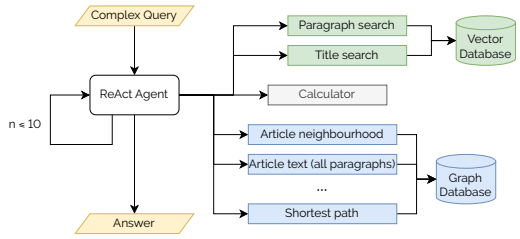


Figure 2: The architecture of our QA system with three types of tools: vector-based (green), graph-based (blue) and a simple calculator tool; middleware applies a soft limit of 10 iterations to the tool use loop.

their usual definitions [51]. While score calculation is done analytically, the claim decomposition and verification steps rely on LLM judgement. This allows flexibility in answer interpretation to determine correctness without direct word-for-word text matching, but it does make the results non-deterministic.

2.3.4 Custom metrics. While all of the metrics discussed above are commonly used, there are several other useful metrics for RAG evaluation that are used for specific purposes. One is the *Comprehensive RAG Benchmark* (CRAG) score, introduced by [56]. A key observation made by the authors is that incorrect or hallucinated answers are far worse than missing or refused answers, as they can harm user confidence in the LLM system. The CRAG paper therefore assigned +1 to a correct answer, 0 to a missing answer, and −1 to a hallucinated answer. For human evaluation, the scoring was extended to include not only +1 for a perfect answer but also +0.5 for an acceptable answer that contains useful information but with some minor errors or omissions. However, for auto evaluation with an LLM judge, both +1 and +0.5 were combined into the accurate class (+1). With this scoring system, a value can be reported for each answer in terms of whether it is accurate, missing, or hallucinated. From these can be calculated the proportions of each answer class, as well as an additional *truthfulness* metric, being the sum of all the scores given across a question batch (i.e. the number of accurate answers less the number of hallucinated answers).

3 A SOLUTION TO GRAPH-BASED COMPLEX QUESTION ANSWERING

To implement the QA pipeline presented earlier in Figure 1, we developed a system with a single reasoning agent and a set of tools used to retrieve information from vector and graph databases. Figure 2 shows the architecture of the proposed solution.

The reasoning agent was created using LangChain's *create_agent* function,[3] which provides tool calling capabilities. We equipped the agent with three types of tools: vector-based, graph-based and a simple calculation tool.

The agent was supplemented with standard middleware for tool call error handling, retries with exponential back-off for LLM failures, and a run limiter to prevent a never ending cycle of tool calls and limit token consumption. Finally, the agent was given a JSON structured output schema such that an answer, explanation for the answer, and references were produced separately.

The database, both vector store and graph DB, were populated with a semi-structured representation of a significant portion of English Wikipedia articles included in the MoNaCo benchmark.

3.1 MoNaCo reproducibility

The question-answer pairs of MoNaCo were created by 24 Amazon Mechanical Turk (AMT) crowdworkers over an unspecified period of time on live English Wikipedia [53]. Although this is methodologically straightforward, it creates issues for reproducibility. A key concern is that Wikipedia is constantly in flux: articles that AMT workers may have used to generate question-answer pairs may have since been altered, deleted, or more up-to-date articles created. Therefore, in this study we used the benchmark against a snapshot version of the English Wikipedia from August 2025. This was done after we conducted a data alignment experiment to ascertain which of the various snapshots of Wikipedia prior to the benchmark publication was capable of providing the most valid sources to perform the evaluation.

Each of the fetched snapshots, from January 2023 to August 2025, were scored according to the proportion of answerable questions, out of the MoNaCo total of 1315. The August 2025 snapshot contained the sources necessary to answer 1207 (91.8%) of the questions. The 108 questions deemed unanswerable by this snapshot were dropped from our evaluation.

3.2 Knowledge graph creation

Each article within the snapshot consists of *Wikitext*, a lightweight markup language used for authoring Wikipedia articles. Using Wikitext markers, we parsed sections and paragraphs from each article as well as article redirects and articles linked to from each paragraph. From these links, we also materialised links between articles themselves (mentions), which later aid path-finding within the dataset.

As we also wanted to evaluate vector RAG, paragraphs were further split into 500-character chunks, with a 40-character overlap, following the parameters recommended by Bratanič and Hane [3]. Preceeding and succeeding sections, paragraphs, and chunks were made into doubly-linked lists via the `NEXT_*` /`PREVIOUS_*` relationships. Article nodes were given a `title` property, and the text of each paragraph was stored in a `text` property, which was then replicated onto the relevant chunk. The resulting schema is shown in Figure 3.

Using the filtered list of 1207 MoNaCo questions, a canonical list of 28,240 source articles was created. The August 2025 snapshot was filtered to just these sources, to balance dataset size against information density. Nodes outside of this filtered subset that were linked to from inside the subset were reduced to stubs: for example, an article not listed as a source by MoNaCo but linked to by a paragraph within a MoNaCo source was reduced to an empty article node of which there were 1,022,549.

The resulting KG comprised 5,771,867 nodes and 22,088,251 relationships. Nearly half of all nodes were paragraph chunks, which were vectorised to enable vector search. As such, an embedding

[3] https://reference.langchain.com/python/langchain/agents/factory/create_agent.

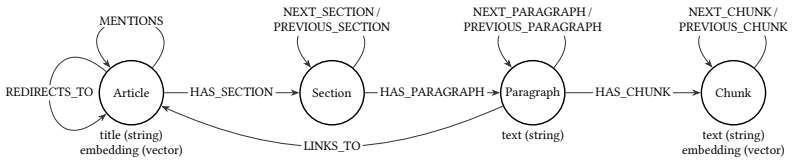


Figure 3: Graph schema for the English Wikipedia dataset

model was selected based on its performance on the retrieval task of the MTEB leaderboard [14].

At the time of writing, the top-performing models on the MTEB leaderboard for retrieval were almost entirely large embedding models with 8B parameters or more, such as *Qwen3-embedding*. Despite demonstrating high performance, such large models are unfeasible for this study, where the dataset within the chunk's `text` attribute runs to over 300 million tokens, and so the embedding process would take an unreasonably long time. However, an outlier among the top retrieval embedding models on the leaderboard was Microsoft's *Harrier* 0.6B [31], which achieved top-5 performance on retrieval tasks (70.75), similar to *Qwen3-embedding* 8B (70.88), and top-10 performance overall (69.01), despite being a much smaller and faster model.

Apart from chunk embeddings, an embedding was also created for each article title, and the resulting dataset was imported to the Neo4j database. Vector indices were created for each chunk's text embedding and article title embedding, so that the graph could be queried directly via *Cypher*, or article titles / chunk texts surfaced via vector search. The resulting dataset could be queried both by traditional vector RAG methods and by KG tooling.

3.3 Retrieval tools

A key advantage of graph queries over vector RAG is that they are cheap operations capable of traversing many hops at once. Multihop QA datasets such as *MuSiQue* [50] rely on CoT reasoning, decomposing a complex question into single-node steps, e.g. "*Who succeeded the first President of Namibia*" becomes "*Who was the first President of Namibia?*" (Sam Nujoma), then "*Who succeeded Sam Nujoma?*". Each step is cheap, but requires a separate LLM call to reflect and generate the next step, on the assumption that retrieval returns one answer at a time from a single data point.

The same multihop reasoning can be seen in *MoNaCo*, in that it also provides question decomposition. However, a motivation of this study is to determine whether more intelligent retrieval tools can encourage reasoning over multiple hops at once, reducing both the number of LLM turns required as well as overall token usage.

One method of achieving this would be to provide a direct interface between the AI agent and the database, such as an MCP server that allows the agent to write and execute *Cypher* statements directly to the database.[4] This would, however, require the LLM to not only maintain knowledge of the question it is answering but also the schema of the database, possibly fracturing its thinking between its main task (answering the question) and the generation of syntactically correct *Cypher* queries.

[4]https://neo4j.com/docs/mcp/current/.

Allowing an LLM to generate its own queries also raises prompt injection issues, where a malicious actor may be able to exfiltrate data that they were not supposed to access. Moreover, early attempts with Cypher query generation in this study revealed that AI agents were reluctant to generate complex Cypher queries that would go beyond a single hop, likely due to their preconditioning for typical RAG scenarios.

Instead, we decided that bespoke Cypher queries should be hand-written to query our KG. This allowed the LLM to focus more directly on answering the question, and encouraged it to use more complex queries than single-hop question decomposition. As such, several queries were written for vector search (for typical RAG and graph discovery), structural navigation, and relational queries. These are listed below:

- **Discovery**: finding entry points into the graph and vector RAG
  - *Title vector search*: return top-k articles whose title matches a query.
  - *Chunk vector search*: return top-k paragraphs with chunks matching a query.
  - *Article neighbourhood*: return all articles that are linked to an article within distance n.
- **Structural navigation**: reading the text of an article
  - *Article text*: walk all paragraphs in an article sequentially, returning article text.
  - *Section titles and infoboxes*: Wikipedia often includes so-called "infoboxes", short summaries and vital statistics about the object or person in question. Although extracting all the article text retrieves these along with the rest of the article, it may be possible to save tokens by extracting just the infobox. Similarly, section headings give a brief overview of the content of an article and allow for more incisive reading.
  - *Get sections*: given a list of section IDs, return the text of just those specific sections; this might be another way to reduce token usage.
  - *Window paragraph*: get the n paragraphs surrounding a particular paragraph
  - *Window section*: get the n sections surrounding a particular section.
- **Relational queries**: finding how other articles relate
  - *Get backlinks*: find all the paragraphs that link to an article.
  - *Shortest path*: get all of the interstitial paragraphs that link two articles together.

Each of these queries were integrated as tools into our LangChain agent, so that the Cypher queries themselves were hidden from the LLM and exposed to the agent as tools; for details see Appendix B.

3.4 The calculator tool

Given that some of the MoNaCo queries require calculations to be performed using the retrieved data, a calculator tool was created in addition to the retrieval tools discussed above. The tool allows basic arithmetic operations to be performed, and was designed to support aggregation tasks.

4 EVALUATION

To investigate whether queries over structured KBs can improve the performance of complex QA versus unstructured RAG, we set up our pipeline following Figures 1 and 2. Specifically, we:

- ingested an English Wikipedia dataset into both the vector store and knowledge base store. For this purpose we used the native *Neo4j* vector store and *Neo4j* graph database. The methods for chunk embedding and graph creation are documented in Section 3.2,
- looped over 510 complex questions, selected from our reproducible snapshot of the MoNaCo benchmark dataset,[5]
- performed query understanding, planning, reasoning and answer synthesis using a single agent equipped with a set of tools described in Section 3.3, along with a structured output parser,
- evaluated the answers produced by our QA system using the LLM-based evaluation metrics described in Section 2.3, i.e. *factual correctness*, *answer relevancy* and two CRAG-inspired custom metrics, along with additional scores calculated from these; we also recorded the token and tool usage for every QA trace.

4.1 Experiment configuration

To understand the impact of graph-based tools on the accuracy and performance of the agentic system, we ran three kinds of experiments, namely: vector RAG, vector+graph RAG, and zero-shot. They differed in the combination of tools the agent could use:

- vector RAG: the agent had access only to the retrieval tool, *chunk vector search*, and the calculator tool;
- vector+graph RAG: the agent had access to all the retrieval tools, plus the calculator tool; and
- zero-shot: no tools were available to the agent.

To account for stochasticity in LLM responses, for each scenario the QA task was run three times and minimum, median and maximum values are reported.

In our preliminary experiments, we gave identical prompts in all three scenarios, modified only by the inclusion of the names and descriptions of available tools. The analysis of tool usage showed, however, that in the vector+graph RAG scenario, the agent relied predominantly on the *Chunk vector search* and *Article text* tools, and rarely used other structural navigation tools. Using the two tools, the agent read in many whole articles consuming substantially more input tokens than in the *vector RAG* scenario.

For this reason, we updated the prompts to give explicit instructions to use the tools to obtain information efficiently, with an example usage sequence: *Title vector search* → *Section titles* → *Get sections*. This prompting strategy improved the overall use of different tools, although some of the tools were still underutilised. More work is, therefore, needed to improve the prompts and tool use. The prompts themselves are available in Appendix C.

Experiments were performed using a *LangChain* agent based on *GPT-5.4* which, at the time of the experiments, was a top-10 model in the MMLU-Pro leaderboard [52], with a score of 0.875.[6] The model's reasoning effort was set to *medium*, as a trade-off between reasoning depth and response latency, to allow the experiments to be performed repeatedly over a large question set.

As noted earlier in Section 3.2, this high performance reasoning model was used in conjunction with a top-performing embedding model, *Harrier-0.6B*. To avoid model "self-preference" bias [43], LLM-as-a-judge evaluations were performed using *Llama-4-Maverick* 17B parameter 128 Expert Instruct model. This model, although achieving somewhat lower performance than the GPT model (MMLU-Pro 0.805), was used for the simpler task of comparison between agent generated and expected answers. Where evaluation required embeddings, these were computed using the *text-embedding-3-large* model, again ensuring the use of an independent model family, and again sufficient for the simpler task of identifying semantic similarity between lists of generated and ground truth answers.

Answer evaluation metrics were calculated using the Ragas library [51], with factual correctness calculated in both the precision and recall modes. The original CRAG metric could not be used directly, but inspired us to design two custom metrics, coarse and fine-grained "CRAG", explained later.

4.2 Results

In our evaluation, we mainly compare the baseline vector RAG approach against the proposed vector+graph RAG solution. Additionally, we include the evaluation of the zero-shot scenario in which no external data was introduced to the model.

As shown later, we do note that in our results the highest number of correct answers and lowest token usage occur in the zero-shot mode. However, the dataset used for evaluating the MoNaCo benchmark is based on English Wikipedia. Wikipedia itself, as a largely dispassionate and factual source of information, is one of the key sources used in LLM training. For complex QA against a non-public knowledge base, the scores for the zero-shot approach would almost certainly be significantly lower than those using RAG, since the LLM would not have been exposed to the data in training, and without RAG would have no access to it whatsoever. Here we consider the zero-shot results as indicative of the limiting values possible for an LLM fine-tuned on the KB.

While model fine-tuning on relevant data is an option more efficient than training from scratch, it still requires considerable compute and is not generally an attractive proposition for KBs that are regularly updated. Indeed, RAG was specifically proposed as a method to reduce hallucination, improve factual correctness,

[5] To reduce computational expense 512 of 1207 questions were randomly selected, two were subsequently dropped due to repeated blocking by the LLM content guardrails.

[6] https://huggingface.co/spaces/TIGER-Lab/MMLU-Pro.

Table 2: Evaluation scores ($\text{median}^{\text{max}}_{\text{min}}$) for the three experiment scenarios each run three times; the best results are in bold, the second best are underlined. Scores are based on the same 510 question subset of the MoNaCo questions; some factual correctness evaluations failed due to long answer lists hitting embedding token limits so are based on successful evaluations only (median: precision 499, recall 467).

Score	Vector RAG	Vector+ graph RAG	Zero-shot
Factual correctness prec. (mean)	$0.15^{0.18}_{0.14}$	$\underline{0.36}^{0.38}_{0.34}$	$\mathbf{0.43}^{0.45}_{0.42}$
Factual correctness recall (mean)	$0.13^{0.15}_{0.11}$	$\underline{0.33}^{0.35}_{0.32}$	$\mathbf{0.39}^{0.41}_{0.38}$
Answer relevancy (mean)	$0.35^{0.36}_{0.31}$	$\underline{0.61}^{0.62}_{0.61}$	$\mathbf{0.80}^{0.81}_{0.80}$
Coarse truthfulness	$\mathbf{-31}^{-18}_{-31}$	$\underline{-49}^{-44}_{-61}$	-127^{-113}_{-137}
Fine-grained truthfulness	35^{43}_{25}	$\mathbf{63}^{73}_{56}$	$\underline{40.5}^{45}_{39}$
Coarse CRAG $\frac{\text{fully correct}}{\text{all}}$	$0.11^{0.13}_{0.10}$	$\underline{0.28}^{0.28}_{0.26}$	$\mathbf{0.34}^{0.35}_{0.33}$
Coarse CRAG $\frac{\text{any wrong}}{\text{all}}$	$\mathbf{0.16}^{0.17}_{0.16}$	$\underline{0.37}^{0.38}_{0.37}$	$0.59^{0.60}_{0.59}$
Fine CRAG $\frac{\text{fully + partially correct}}{\text{all}}$	$0.18^{0.20}_{0.17}$	$\underline{0.40}^{0.40}_{0.38}$	$\mathbf{0.51}^{0.52}_{0.50}$
Fine CRAG $\frac{\text{any wrong}}{\text{all}}$	$\mathbf{0.10}^{0.11}_{0.09}$	$\underline{0.26}^{0.27}_{0.25}$	$0.42^{0.43}_{0.42}$
Tokens used (median)			
Input tokens	$\underline{40094}^{41892}_{39063}$	43652^{45163}_{43144}	$\mathbf{490}^{490}_{490}$
Reasoning tokens	$\underline{852}^{858}_{834}$	1055^{1075}_{1050}	$\mathbf{516}^{516}_{516}$
Output tokens	$\underline{1486}^{1494}_{1474}$	1620^{1624}_{1618}	$\mathbf{618}^{620}_{614}$
Total	$\underline{41694}^{43636}_{40573}$	45695^{46871}_{44692}	$\mathbf{1108}^{1120}_{1106}$

and allow access to up-to-date information without costly fine-tuning [36]

Table 2 includes an overview of the results obtained. On top of the LLM-based metrics, we report derived scores to highlight key insights from the plots presenting the "CRAG" scores, and the number of tokens each approach used.

4.2.1 Factual correctness and answer relevancy. As explained in Section 2.3.3, *factual correctness* is a Ragas score computed by a direct comparison of the generated and reference answers.

The factual correctness results indicate that the vector+graph RAG responds with much higher precision and recall than the basic vector RAG approach, and is not far from the "fine-tuned" zero-shot solution. This is a significant achievement for vector+graph RAG, meaning that even a simple, semi-structured KG can substantially improve the correctness score, both in terms of precision and recall. To reiterate, the zero-shot approach is only able to obtain comparable results due to the open public nature of the Wikipedia dataset, and this would not be achieved for QA tasks on proprietary data to which the model had not been exposed during training.

We note that in each implementation the precision and recall of factual correctness are relatively low. This is in part due to refusal ('unknown' scores 0), and also indicative of errors being a mixture of false positives and false negatives rather than one form of error predominating. It is not simply the case that failure to retrieve all claims related to a particular concept leads to false negatives lowering the recall score. The precision scores indicate that substantial incorrect answers (i.e. false positives) are also returned. However, in the case of aggregation queries the root cause could still be imperfect recall (e.g. the vector RAG approach fails on the question *"How many novels were written by Ernest Hemingway while he was living abroad?"* as while the correct time period is identified some novels written in this period are missed, and so the total returned is too low. But, since the golden answer provided is the numerical total only, this appears as a precision error; the additional graph tools allowed this question to be answered correctly).

Table 3: The number of answers ($\text{median}^{\text{max}}_{\text{min}}$), classified into three or five CRAG inspired score classes for the three experiment scenarios, each run three times. All scores are based on the same 510 question subset of the MoNaCo questions. The best results are in bold, the second best are underlined.

Correctness metric	Score	Vector RAG	Vector+ graph RAG	Zero-shot
Coarse CRAG	−1	$\mathbf{84}^{89}_{80}$	$\underline{191}^{195}_{189}$	301^{306}_{291}
	0	363^{381}_{360}	$\underline{177}^{181}_{176}$	$\mathbf{35}^{41}_{35}$
	1	58^{66}_{49}	$\underline{142}^{145}_{134}$	$\mathbf{174}^{178}_{169}$
Fine-grained CRAG	−1	$\mathbf{34}^{43}_{25}$	$\underline{83}^{89}_{81}$	136^{137}_{134}
	−0.5	$\mathbf{15}^{20}_{14}$	$\underline{48}^{53}_{45}$	80^{84}_{76}
	0	362^{380}_{360}	$\underline{175}^{178}_{172}$	$\mathbf{34}^{36}_{33}$
	0.5	32^{33}_{30}	$\underline{59}^{61}_{56}$	$\mathbf{86}^{89}_{82}$
	1	59^{68}_{55}	$\underline{145}^{150}_{135}$	$\mathbf{174}^{177}_{172}$

Considering answer relevancy, which measures alignment between the generated response and the user query (Section 2.3.3), adding a graph-based KB to the baseline vector RAG noticeably improves this score. The KG tools raise relevancy from about 0.35 to 0.61 which is over a 74% increase. As indicated later, this is due to the fact that with the KG tools the agent answers more of the user queries, whereas when using only vector tools the agent often states that the answer is unknown. While this "safe refusal" is preferable to a potentially hallucinated answer, it is likely to have received a low relevancy score from the LLM judge.

The highest relevancy score is achieved in the zero-shot case, which can be attributed to a further increase in the number of questions for which an answer was attempted. But this does not account for the correctness of the returned answer. The *faithfulness* score would add to answer relevancy a test for whether the response is grounded in the context. However, as noted earlier, this metric could not be implemented for the MoNaCo dataset given golden contexts are not provided.

4.2.2 Truthfulness scores. Following the idea of the CRAG benchmark, our *truthfulness* scores strongly penalise hallucinated information over a system that refuses to answer (see Section 2.3.4). However, due to the fact that many of the answers in the MoNaCo dataset are supposed to produce lists of claims rather than individual full-text responses, the answers can be fully or partially correct. Therefore, we report two versions of the truthfulness score. The first, and most strict *coarse truthfulness* assigns +1 only if the precision and recall of a response is 1. That is, the response includes all the claims from the golden answer and nothing more, which otherwise would be hallucinated claims.

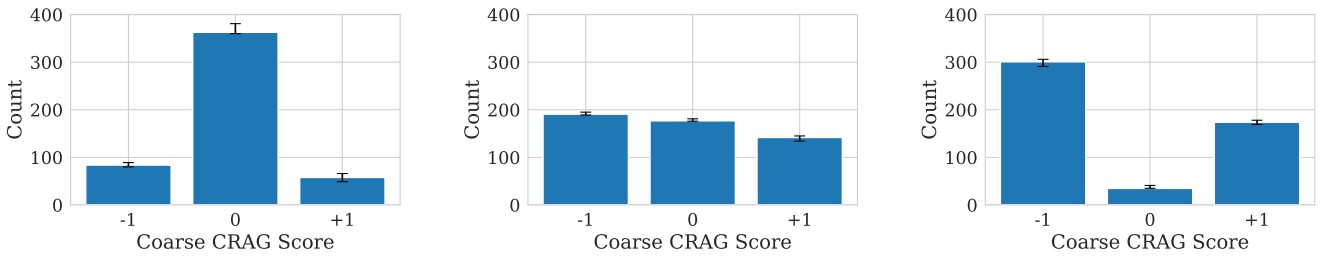


Figure 4: Coarse "CRAG" scores in three scenarios (left-to-right): vector RAG, vector+graph RAG and zero-shot.

In this view, the basic vector RAG solution achieves the highest score. It is, however, very conservative in answering complex questions and responds only in about 58 + 84 = 142 cases (28.1%) of which only 58 (11.4%) are fully correct. The number of answers in different categories is presented in Table 3, whilst the distribution of responses is shown in Figure 4 left.

The vector+graph RAG approach obtains a slightly lower *coarse truthfulness* score, but significantly higher than the zero-shot solution, which provides fully correct answer for 174 questions (34.1%) but hallucinates in 301 cases (59%). The graph-supported solution is somewhat conservative in responding to the complex questions and provides answers for about 333 questions (65.3%), of which 142 (28.5%) are fully correct (cf. the middle plot in Figure 4). Considering the *coarse truthfulness* and *coarse CRAG* scores, the vector+graph RAG system is intermediate between the other two scenarios in answering incorrectly, but the number of fully correct responses is much closer to the best, zero-shot approach, than it is to simple vector RAG.

Looking at the answers via a more fine-grained lenses (Figure 5), the CRAG scores are distributed differently. The *fine-grained truthfulness* still penalises hallucinations, but allows some missingness in the answers. A score of +1 is assigned only to fully correct answers (that is, precision = 1, recall = 1), 0.5 to answers with missing claims but no hallucinations (precision = 1, 0 < recall < 1), 0 to unanswered questions, −0.5 to answers which mix correct and hallucinated claims and −1 where all claims were wrong (i.e. recall = 0). In this case, the vector+graph RAG solution achieves the best score of about 63 (median), which is much higher than the other scenarios, 35 and 40.5 for vector RAG and zero-shot, respectively.

This is a significant result which reinforces *coarse truthfulness* in that the vector+graph RAG solution produces much less hallucinated content than zero-shot. But it also indicates that despite the responses still not always being fully correct, adding a simple knowledge-graph can substantially improve correctness and trust into a QA system, especially for complex questions.

4.2.3 Token usage. Token usage and tool use were evaluated by capturing all messages from the agent during QA tasks, and subsequent evaluation and post-processing. We extracted details from the *usage_metadata* dictionary of messages of type *AIMessage*. As shown in Table 2, for the two RAG implementations, around 96% of total token usage is for input tokens, with over 40,000 input tokens used in each case. This is due to the reading of material retrieved from the RAG system. In the zero-shot case input token usage is minimal because the knowledge is embedded in the model parameters, and so tokens are spent on the system prompt, user query and reasoning only.

While the vector RAG implementation has slightly lower total token usage, hence lower cost than the vector+graph RAG solution, this should be considered in relation to the truthfulness scores discussed above. In the vector+graph RAG approach tokens are used to answer many more questions than plain vector RAG (i.e. the fewer responses of 'unknown') while the generated answers are more correct. Overall, the graph-supported solution provides much better value to the end user.

4.2.4 Tool use. Figure 6 shows the distribution of tool calls by both the vector RAG (two tools) and vector+graph RAG scenarios (11 tools). Unsurprisingly, the vector RAG scenario called the vector search tool almost exclusively, often with similar search terms and increasing number of retrieved chunks k as it widened its search to include more results in its context window. The vector+graph RAG also used the chunk vector search tool regularly, but in far fewer turns. It used this tool in conjunction with the other vector tool (title search) to retrieve more relevant results, and then used combinations of the other tools to determine its answer. The article neighbourhood tool was used extremely rarely, indicating that links between articles were of little importance once the agent had reached an article.

The article text tool was also rarely used, indicating that the agent limited tokens usage by focussing on specific sections rather than reading whole articles. Indeed, the tool to fetch section titles and infoboxes was regularly used to fetch basic information about the article: many queries could be answered in part by infoboxes, and section titles give a clear overview of the article's content before the agent begins reading. The agent could then select the sections that it wished to read from the article, and so used the get sections tool in a majority of cases, reducing token usage.

The "windowing" functions for paragraphs and sections were not used often. It suggests the agent correctly retrieved the intended sections and paragraphs in the first try, rather than having to read forwards and backwards from its current location. Similarly, unused were the backlinks, shortest path, and calculate tools. Seemingly, the relationships between articles were not as important as argued by the authors of the MoNaCo benchmark, nor the calculation and aggregation of intermediate results.

The observed reluctancy to call graph tools over vector searches may be due to the task-specific training of LLMs. Vector RAG is seen

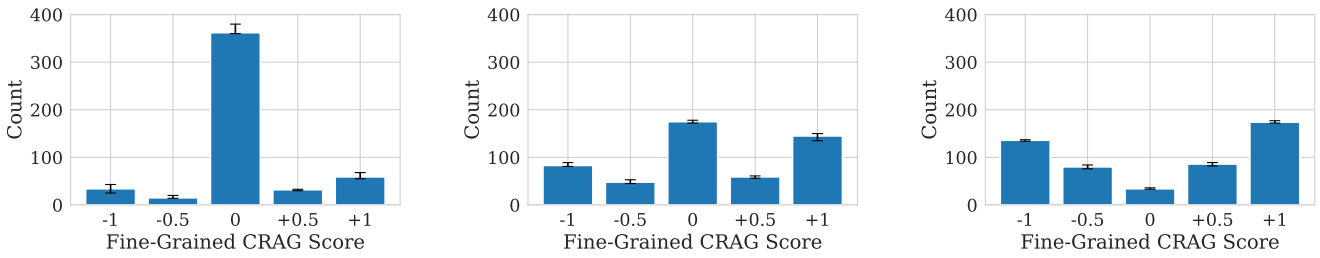


Figure 5: Fine-grained "CRAG" scores in three scenarios (left-to-right): vector RAG, vector+graph RAG and zero-shot.

as a key business case for LLMs, and AI providers such as OpenAI are clearly including RAG in their post-training.[7] Therefore, the use of graph tools to perform complex QA, as explored in this paper, is likely under-represented in LLM training. LLMs are conservative in their tool selection, and are biased towards tools that align with tools seen during training [2]. It is, therefore, not surprising that even when encouraged to use graph tools by the system prompt, the tested model fell back on simpler vector searches for complex queries.

4.3 Non-deterministic evaluation

It is important to state that the results presented above show scores calculated automatically using the LLM-as-a-judge pattern and without human validation. Inherently, this process is non-deterministic and resulted in scoring discrepancies, which stem for randomness embedded in the inference of LLM models.

To illustrate the extent of non-determinism, in Figure 7 we show correlation heatmaps between the coarse and fine-grained trustfulness scores. Following the definition, an ideal heatmap would classify all results of x-axis (fine-grained CRAG) less then 1 and non-zero under the score −1 on the y-axis (coarse CRAG). All zeros on the x-axis should correspond to zeros on y-axis, and all ones on the x-axis should correspond to ones on y-axis.

The heatmaps show that there was a small number of inconsistently classified responses. For example, in the bottom right corner 4, 14 and 26 answers in the vector RAG, vector+graph RAG and zero-shot scenarios, respectively, were classified by the 'fine-grained CRAG' judge as +1. But the same answers were given −1 by the 'coarse CRAG' judge. Similarly, 2, 3 + 6 and 3 + 5 + 10 answers classified by the 'coarse CRAG' judge as +1 were classified with scores below +1 by the fine-grained judge. Further investigation would be required to confirm whether these effects are due to the stochastic nature of the LLM generative processes, or whether they are systematic errors related to specific questions.

5 LIMITATIONS

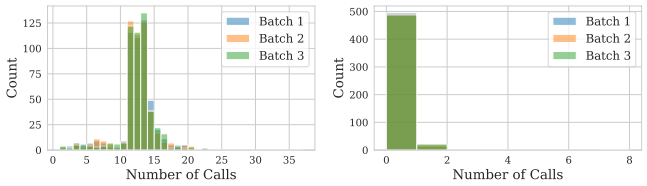
One benchmark, one model. Although the results presented are very promising, one of the key limitations of this work is that we evaluated our scenarios using only a single benchmark dataset, MoNaCo, and only a subset of 510 question-answer pair subset from the 1,207 QA pairs available for which corresponding source articles could be found. Furthermore, we used only a single reasoning model for answer generation. In favour of our analysis, MoNaCo uses Wikipedia as its source of ground truth, and despite this, we were still able to show that vector+graph RAG improves precision and recall while reducing hallucinations. Nevertheless, it would be useful to validate these results using a larger number of questions, additional benchmark datasets, and alternative reasoning models.

Human validation. As explained earlier, the majority of the results presented in this paper were generated using the LLM-as-a-judge paradigm; only limited human validation was conducted to assess potential scoring errors and inconsistencies in the evaluation values (cf. Section 4.3). As the observed scoring discrepancies were not significant, we used a relatively large number of complex questions from MoNaCo, and three independent experiment runs, the results may be considered valid. Nevertheless, repeated evaluations and thorough error analysis would further increase confidence.

Isolating the contribution of individual tools. Our experiments compare vector RAG against vector+graph RAG as whole configurations, and so do not isolate the marginal contribution of each graph tool. A full ablation would re-run the entire pipeline (510 questions, three repeats, an LLM agent per run) for every tool configuration: while a leave-one-out study scales linearly in the number of tools, isolating the *interactions* between tools grows combinatorially (up to 2^n subsets for n tools), making a thorough ablation substantially more expensive than the experiments reported here. As partial evidence in lieu of such a study, the per-question distribution of tool calls (Figure 6) already indicates which tools carry the load – chunk and title vector search, section-title retrieval, and *get_sections* – and which are near-unused – article neighbourhood, backlinks, shortest path, and the windowing tools – providing a coarse signal of each tool's importance.

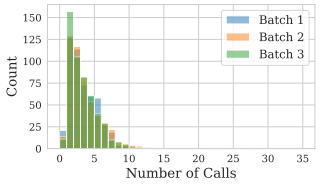
Tool use depends on prompting. The graph tools require accompanying prompt instructions to be used effectively: as described in the experiment configuration, we supply an example usage sequence (*Title vector search* → *Section titles* → *Get sections*) to steer the agent towards efficient, structure-aware retrieval. We regard this prompting as an intended and integral part of the proposed method rather than a confound: the contribution is the *combination* of a lightweight document graph, a dedicated toolset, and the prompting that drives them. The agent's residual reluctance to call the more complex graph tools might be further reduced with few-shot examples – e.g. by working through the MoNaCo-provided

[7] https://developers.openai.com/api/docs/guides/retrieval.

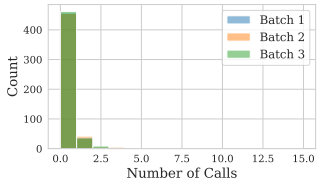


(a) vector RAG: chunk vector search

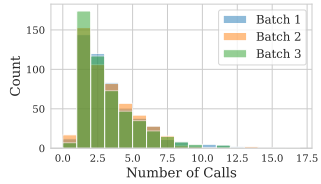
(b) vector RAG: calculate



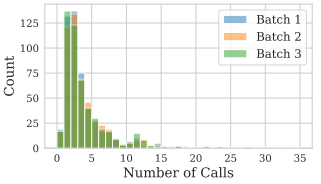
(c) vector+graph RAG: chunk vector search



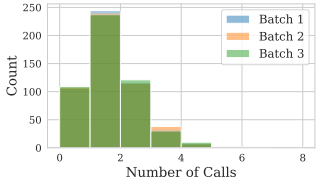
(d) vector+graph RAG: calculate



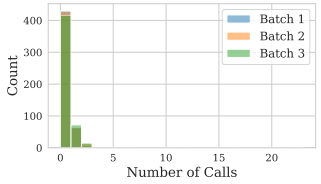
(e) vector+graph RAG: title vector search



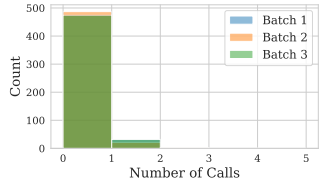
(f) vector+graph RAG: section titles and infoboxes



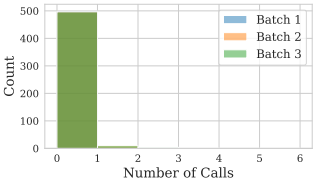
(g) vector+graph RAG: get sections



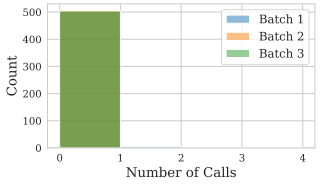
(h) vector+graph RAG: article text



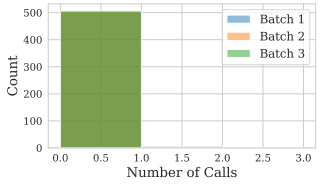
(i) vector+graph RAG: get backlinks



(j) vector+graph RAG: window paragraph



(k) vector+graph RAG: window section



(l) vector+graph RAG: article neighbourhood

Figure 6: Distribution of tools called per question for the vector RAG and vector+graph RAG scenarios. The shortest path tool available in the vector+graph RAG scenario was never used so is omitted from the figure.

decomposition steps – or by fine-tuning on graph-specific traces, which we leave to future work.

Latency. Due to asynchronous batching of LLM calls by Ragas and unhandled API errors on some LLM calls we do not assess the relative answer latency of vector vs vector+graph approaches. We note that an assessment of such features would be strongly dependent on the precise deployment details of the vector and graph DB systems, and is beyond the scope of this work.

Scope. This study compares vector RAG against vector+graph RAG built on a *simple* document graph. Richer knowledge graphs and more sophisticated GraphRAG variants are out of scope. We do not claim that a graph database is the only way to realise the approach, nor that our simple graph is optimal; rather, it is the lightweight enabler that makes structure-aware retrieval practical on semi-structured collections.

6 CONCLUSIONS

In this paper we describe a QA system capable of tackling complex questions that require sophisticated multi-hop and multi-entity reasoning, access to multiple documents, and strong summarisation and aggregation capabilities. To evaluate the proposed approach, we used the MoNaCo complex QA dataset. This we filtered to over 1200 questions with golden answers referring to over 28 thousand semi-structured documents drawn from a static snapshot of English Wikipedia. From the dataset, we extracted a simple graph structure that does not require sophisticated graph construction methods. Instead, it relies on basic structural information, such as document and section titles, together with links to the relevant document fragments.

Typical LLM-based solutions, such as vector-based RAG and zero-shot prompting, are often inadequate for addressing complex QA tasks. The former often refuse to answer the question, being unable to retrieve, collate, and reason over all relevant context effectively. The latter are often overconfident and respond with a lot of hallucinated content.

The results shown indicate that augmenting a basic vector RAG subsystem with a simple graph-based KB and corresponding tools can significantly reduce the amount of hallucinated content (the coarse truthfulness score improved from about −127 to −49 for vector+graph RAG vs zero-shot). At the same time, the vector+ graph RAG approach attempted to answer more than twice as many complex questions as the baseline vector RAG solution. We also show that, when partially correct answers are taken into account, vector+graph RAG achieves the highest score across all three evaluated

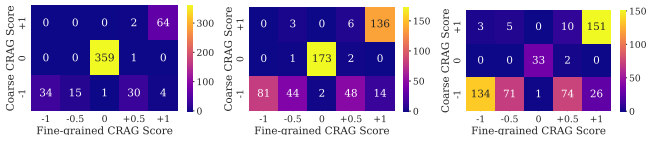


Figure 7: Correlation between coarse CRAG (y-axis) and fine-grained CRAG (x-axis) scores from a single experiment run in three scenarios (left-to-right): vector RAG, vector+graph RAG and zero-shot.

scenarios; the fine-grained truthfulness score was 80% higher than for vector RAG. Additionally, the factual correctness results indicate that vector+graph RAG achieves more than twice the precision and recall of the system based solely on vector RAG.

These results show a substantial improvement over the baseline vector RAG, allowing us to answer our research question positively and with confidence: structured knowledge bases, such as knowledge graphs, can improve the performance of complex QA systems. By increasing both precision and recall while reducing hallucinations, the proposed solution is a promising direction towards increasing trust in LLM-based QA systems.

CONFLICT OF INTEREST STATEMENT

The research presented in this work was conducted solely by staff of the National Innovation Centre for Data as part of a project funded by Neo4j. While Neo4j provided financial support for the project, the authors declare that they have no personal or financial interests in Neo4j and that this funding did not influence the study design, analysis, or interpretation of the findings, and that there are no competing interests.

REFERENCES

[1] Jinheon Baek, Alham Fikri Aji, Jens Lehmann, and Sung Ju Hwang. 2023. Direct Fact Retrieval from Knowledge Graphs without Entity Linking. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*. Toronto, Canada. https://doi.org/10.18653/v1/2023.acl-long.558

[2] Thierry Blankenstein, Jialin Yu, Zixuan Li, Vassilis Plachouras, Sunando Sengupta, Philip Torr, Yarin Gal, Alasdair Paren, and Adel Bibi. 2026. BiasBusters: Uncovering and Mitigating Tool Selection Bias in Large Language Models. In *Proceedings of the Fourteenth International Conference on Learning Representations*. Rio de Janeiro, Brazil. https://doi.org/10.48550/arXiv.2510.00307

[3] Tomaž Bratanič and Oskar Hane. 2025. *Essential GraphRAG: Knowledge Graph-Enhanced RAG*. Manning Publications, Shelter Island, NY.

[4] Andrew Brown, Muhammad Roman, and Barry Devereux. 2025. A Systematic Literature Review of Retrieval-Augmented Generation: Techniques, Metrics, and Challenges. *Big Data and Cognitive Computing* 9 (2025), 320. Issue 12. https://doi.org/10.3390/bdcc9120320

[5] Avi Caciularu, Arman Cohan, Iz Beltagy, Matthew Peters, Arie Cattan, and Ido Dagan. 2021. CDLM: Cross-Document Language Modeling. In *Findings of the Association for Computational Linguistics*. Punta Cana, Dominican Republic. https://doi.org/10.18653/v1/2021.findings-emnlp.225

[6] Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2022. KQA Pro: A Dataset with Explicit Compositional Programs for Complex Question Answering over Knowledge Base. In *Proceedings of the 60th Meeting of the Association for Computational Linguistics*, Vol. 1: Long Papers. Dublin, Ireland, 6101–6119. https://doi.org/10.18653/v1/2022.acl-long.422

[7] Abir Chakraborty. 2024. Multi-hop Question Answering over Knowledge Graphs using Large Language Models. arXiv:2404.19234

[8] Yun-Nung (Vivian) Chen, Margot Mieskes, and Siva Reddy. 2023. Retrieval-based Language Models and Applications. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, Vol. 6: Tutorial Abstracts. Toronto, Canada, 41–46. https://doi.org/10.18653/v1/2023.acl-tutorials.6

[9] Alison Cossette, Zach Blumenfeld, and Damaso Sanoja. [n.d.]. *The Developer's Guide to GraphRAG*. Neo4j, San Mateo, CA. https://neo4j.com/books/the-developers-guide-to-graphrag/

[10] Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. 2025. ReFoRCE: A Text-to-SQL Agent with Self-Refinement, Format Restriction, and Column Exploration. https://openreview.net/forum?id=OuFIfDBwQd. In *Proceedings of the ICLR 2025 Workshop VerifAI: AI Verification in the Wild*. Singapore.

[11] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, 601–610. https://doi.org/10.1145/2623330.2623623

[12] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and Fate: Limits of Transformers on Composability. In *Proceedings of the 37th Conference on Neural Information Processing Systems*. New Orleans, LA. https://openreview.net/pdf?id=Fkckkr3ya8

[13] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2025. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. arXiv:2404.16130 [cs.CL]

[14] Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Siblini, Dominik Krzemiński, Genta Indra Winata, Saba Sturua, Saiteja Utpala, Mathieu Ciancone, Marion Schaeffer, Gabriel Sequeira, Diganta Misra, Shreeya Dhakal, Jonathan Rystrøm, Roman Solomatin, Ömer Çağatan, Akash Kundu, Martin Bernstorff, Shitao Xiao, Akshita Sukhlecha, Bhavish Pahwa, Rafał Poświata, Kranthi Kiran GV, Shawon Ashraf, Daniel Auras, Björn Plüster, Jan Philipp Harries, Loïc Magne, Isabelle Mohr, Mariya Hendriksen, Dawei Zhu, Hippolyte Gisserot-Boukhlef, Tom Aarsen, Jan Kostkan, Konrad Wojtasik, Taemin Lee, Marek Šuppa, Crystina Zhang, Roberta Rocca, Mohammed Hamdy, Andrianos Michail, John Yang, Manuel Faysse, Aleksei Vatolin, Nandan Thakur, Manan Dey, Dipam Vasani, Pranjal Chitale, Simone Tedeschi, Nguyen Tai, Artem Snegirev, Michael Günther, Mengzhou Xia, Weijia Shi, Xing Han Lù, Jordan Clive, Gayatri Krishnakumar, Anna Maksimova, Silvan Wehrli, Maria Tikhonova, Henil Panchal, Aleksandr Abramov, Malte Ostendorff, Zheng Liu, Simon Clematide, Lester James Miranda, Alena Fenogenova, Guangyu Song, Ruqiya Bin Safi, Wen-Ding Li, Alessia Borghini, Federico Cassano, Hongjin Su, Jimmy Lin, Howard Yen, Lasse Hansen, Sara Hooker, Chenghao Xiao, Vaibhav Adlakha, Orion Weller, Siva Reddy, and Niklas Muennighoff. 2025. MMTEB: Massive Multilingual Text Embedding Benchmark. In *The Thirteenth International Conference on Learning Representations*. Singapore. https://doi.org/10.48550/arXiv.2502.13595

[15] 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*. Association for Computational Linguistics, St. Julians, Malta, 150–158. https://aclanthology.org/2024.eacl-demo.16

[16] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. 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*. Barcelona, Spain. https://doi.org/10.1145/3637528.3671470

[17] Tengfei Feng and Liang He. 2025. RGR-KBQA: Generating Logical Forms for Question Answering Using Knowledge-Graph-Enhanced Large Language Model. In *Proceedings of the 31st International Conference on Computational Linguistics*. Abu Dhabi, UAE, 3057–3070. https://aclanthology.org/2025.coling-main.205/

[18] Yanlin Feng, Simone Papicchio, and Sajjadur Rahman. 2025. CypherBench: Towards Precise Retrieval over Full-scale Modern Knowledge Graphs in the LLM Era. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*. Vienna, Austria. https://doi.org/10.18653/v1/2025.acl-long.438

[19] Robert Friel, Masha Belyi, and Atindriyo Sanyal. 2025. RAGBench: Explainable Benchmark for Retrieval-Augmented Generation Systems. arXiv:2407.11005 [cs.CL]

[20] Aoran Gan, Hao Yu, Kai Zhang, Qi Liu, Wenyu Yan, Zhenya Huang, Shiwei Tong, and Guoping Hu. 2025. Retrieval Augmented Generation Evaluation in the Era of Large Language Models: A Comprehensive Survey. arXiv:2504.14891 [cs.CL]

[21] Mingqi Gao, Xinyu Hu, Jie Ruan, Xiao Pu, and Xiaojun Wan. 2025. LLM-based NLG Evaluation: Current Status and Challenges. arXiv:2402.01383 [cs.CL]

[22] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL]

[23] Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond I.I.D.: Three Levels of Generalization for Question Answering on Knowledge Bases. In *Proceedings of the Web Conference 2021 (WWW '21)*. ACM, 3477—-3488. https://doi.org/10.1145/3442381.3449992

[24] Willis Guo, Armin Toroghi, and Scott Sanner. 2024. CR-LT-KGQA: A Knowledge Graph Question Answering Dataset Requiring Commonsense Reasoning and Long-Tail Knowledge. arXiv:2403.01395 [cs.CL]

[25] Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A. Rossi, Subhabrata Mukherjee, Xianfeng Tang, Qi He, Zhigang Hua, Bo Long, Tong Zhao, Neil Shah, Amin Javari, Yinglong Xia, and Jiliang Tang. 2025. Retrieval-Augmented Generation with Graphs (GraphRAG). arXiv:2501.00309 [cs.IR]

[26] Tianxing He, Jingyu Zhang, Tianle Wang, Sachin Kumar, Kyunghyun Cho, James Glass, and Yulia Tsvetkov. 2023. On the Blind Spots of Model-Based Evaluation Metrics for Text Generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 12067–12097. https://aclanthology.org/2023.acl-long.674/

[27] Dan Hendrycks, Collin Burns, Anya Chen, and Spencer Ball. 2021. CUAD: An Expert-Annotated NLP Dataset for Legal Contract Review. *CoRR* abs/2103.06268 (2021). arXiv:2103.06268 https://arxiv.org/abs/2103.06268

[28] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a Multi-hop QA Dataset for Comprehensive Evaluation of Reasoning Steps. In *Proceedings of the 28th International Conference on Computational Linguistics*. https://doi.org/10.18653/v1/2020.coling-main.580

[29] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models?. In *Proccedings of the First Conference on Language Modeling 2024*. https://openreview.net/forum?id=kIoBbc76Sy

[30] Jiatan Huang, Mingchen Li, Zonghai Yao, Zhichao Yang, Yongkang Xiao, Feiyun Ouyang, Xiaohan Li, Shuo Han, and Hong Yu. 2024. RiTeK: A Dataset for Large Language Models Complex Reasoning over Textual Knowledge Graphs. arXiv:2410.13987 [cs.CL]

[31] Xiaolong Huang, Liang Wang, Furu Wei, Jingwen Lu, Knut Risvik, and Jason Li. 2026. Microsoft Open-Sources Industry-Leading Embedding Model. https://blogs.bing.com/search/April-2026/Microsoft-Open-Sources-Industry-Leading-Embedding-Model.

[32] Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large Language Models Struggle to Learn Long-Tail Knowledge. In *Proceedings of the 40th International Conference on Machine Learning*. Honolulu, Hawaii, 15696–15707. https://dl.acm.org/doi/10.5555/3618408.3619049

[33] Catherine Kosten, Philippe Cudré-Mauroux, and Kurt Stockinger. 2023. Spider4SPARQL: A Complex Benchmark for Evaluating Knowledge Graph Question Answering Systems. In *2023 IEEE International Conference on Big Data (BigData)*. IEEE, 5272–5281. https://doi.org/10.1109/bigdata59044.2023.10386182

[34] Liubov Kovriguina, Roman Teucher, Daniil Radyush, and Dmitry Mouromtsev. 2023. SPARQLGEN: One-Shot Prompt-based Approach for SPARQL Query Generation. In *Proceedings of SEMANTiCS 2023*. Leipzig, Germany. https://ceur-ws.org/Vol-3526/paper-08.pdf

[35] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent Retrieval for Weakly Supervised Open Domain Question Answering. In *Procedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy. https://doi.org/10.18653/v1/P19-1612

[36] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 9459–9474. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[37] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Ma Chenhao, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as a Databaes Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs. In *Proceedings of the 37th Conference on Neural Information Processing Systems*, Vol. Datasets and Benchmarks Track. New Orleans, LA. https://openreview.net/pdf?id=dI4wzAE6uV

[38] Teng Lin, Yuyu Luo, Honglin Zhang, Jicheng Zhang, Chunlin Liu, Kaishun Wu, and Nan Tang. 2025. MEBench: Benchmarking Large Language Models for Cross-Document Multi-Entity Question Answering. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Suzhou, China. https://doi.org/10.18653/v1/2025.emnlp-main.77

[39] Shicheng Liu, Sina Semnani, Harold Triedman, Jialiang Xu, Isaac Dan Zhao, and Monica Lam. 2024. SPINACH: SPARQL-Based Information Navigation for Challenging Real-World Questions. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, Miami, FL, 15977–16001. https://doi.org/10.18653/v1/2024.findings-emnlp.938

[40] Ozan Baris Mulayim, Avia Anwar, Umut Mete Saka, Lazlo Paul, Anand Krishnan Prakash, Gabe Fierro, Marco Pritoni, and Mario Bergés. 2025. BuildingQA: A Benchmark for Natural Language Question Answering over Building Knowledge Graphs. In *Proceedings of the 12th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (Colorado School of Mines, Golden, CO, USA) *(BuildSys '25)*. Association for Computing Machinery, New York, NY, USA, 65–75. https://doi.org/10.1145/3736425.3770097

[41] Cheng Niu, Yuanhao Wu, Juno Zhu, Siliang Xu, Kashun Shum, Randy Zhong, Juntong Song, and Tong Zhang. 2024. RAGTruth: A Hallucination Corpus for Developing Trustworthy Retrieval-Augmented Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. Bangkok, Thailand. https://doi.org/10.18653/v1/2024.acl-long.585

[42] Anish Pahilajani, Devasha Trivedi, Jincen Shuai, Khin S. Yone, Samyak Rajesh Jain, Namyong Park, Ryan A. Rossi, Nesreen K. Ahmed, Franck Dernoncourt, and Yu Wang. 2024. GRS-QA – Graph Reasoning-Structured Question Answering Dataset. arXiv:2411.00369 [cs.CL]

[43] Arjun Panickssery, Samuel R. Bowman, and Shi Feng. 2024. LLM Evaluators Recognize and Favor Their Own Generations. arXiv:2404.13076 [cs.CL]

[44] Chanhee Park, Hyeonseok Moon, Chanjun Park, and Heuiseok Lim. 2025. MIRAGE: A Metric-Intensive Benchmark for Retrieval-Augmented Generation Evaluation. In *Findings of the Association for Computational Linguistics: NAACL 2025*. Association for Computational Linguistics, Albuquerque, New Mexico, 2883–2900. https://aclanthology.org/2025.findings-naacl.157

[45] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. 2019. Language Models as Knowledge Bases?. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Hong Kong, China, 2463–2473. https://doi.org/10.18653/v1/D19-1250

[46] Julian Schnitzler, Xanh Ho, Jiahao Huang, Florian Boudin, Saku Sugawara, and Akiko Aizawa. 2024. MoreHopQA: More Than Multi-hop Reasoning. arXiv:2406.13397 [cs.CL]

[47] Tommaso Soru, Edgard Marx, Diego Moussallem, Gustavo Publio, André Valdestilhas, Diego Esteves, and Ciro Baron Neto. 2017. SPARQL as a Foreign Language. https://ceur-ws.org/Vol-2044/paper14/paper14.pdf. In *Proceedings of SEMANTiCS 2017*. Amsterdam, Netherlands.

[48] Heydar Soudani, Evangelos Kanoulas, and Faegheh Hasibi. 2024. Fine Tuning vs. Retrieval Augmented Generation for Less Popular Knowledge. In *SIGIR-AP 2024: Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*. Tokyo, Japan. https://doi.org/10.1145/3673791.3698415

[49] Jan Strich, Enes Kutay Isgorur, Maximilian Trescher, Chris Biemann, and Martin Semmann. 2025. T^2-RAGBench: Text-and-Table Benchmark for Evaluating Retrieval-Augmented Generation. arXiv:2506.12071 [cs.IR]

[50] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. arXiv:2108.00573 [cs.CL]

[51] VibrantLabs. 2024. Ragas: Supercharge Your LLM Application Evaluations. https://github.com/vibrantlabsai/ragas.

[52] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark. arXiv:2406.01574 [cs.CL]

[53] Tomer Wolfson, Harsh Trivedi, Mor Geva, Yoav Goldberg, Dan Roth, Tushar Khot, Ashish Sabharwal, and Reut Tsarfaty. 2026. MoNaCo: More Natural and Complex Questions for Reasoning Across Dozens of Documents. *Transactions of the Association for Computational Linguistics* 14 (Jan. 2026), 23–46. https://doi.org/10.1162/TACL.a.64

[54] Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N. Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. 2024. STaRK: Benchmarking LLM Retrieval on Textual and Relational Knowledge Bases. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 127129–127153. https://doi.org/10.52202/079017-4037

[55] 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:2506.02404 [cs.CL]

[56] Xiao Yang, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, Rongze Daniel Gui, Ziran Will Jiang, Ziyu Jiang, Lingkun Kong, Brian Moran, Jiaqi Wang, Yifan Ethan Xu, An Yan, Chenyu Yang, Eting Yuan, Hanwen Zha, Nan Tang, Lei Chen, Nicolas Scheffer, Yue Liu, Nirav Shah, Rakesh Wanga, Anuj Kumar, Wen-tau Yih, and Xin Luna Dong. 2024. CRAG – Comprehensive RAG Benchmark. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 10470–10490. https://doi.org/10.52202/079017-0335

[57] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium. https://doi.org/10.18653/v1/D18-1259

[58] Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, and Zhaofeng Liu. 2025. *Evaluation of Retrieval-Augmented Generation: A Survey*. Springer Nature Singapore, 102–120. http://dx.doi.org/10.1007/978-981-96-1024-2_8

[59] Yixiao Zeng, Tianyu Cao, Danqing Wang, Xinran Zhao, Zimeng Qiu, Morteza Ziyadi, Tongshuang Wu, and Lei Li. 2025. RARE: Retrieval-Aware Robustness Evaluation for Retrieval-Augmented Generation Systems. arXiv:2506.00789 [cs.CL]

[60] Tianyi Zhang, Varsha Kishore, Felix Wu, Killian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating Text Generation with BERT. In *Proceedings of the 2020 International Conference on Learning Representations*. Addis Ababa, Ethiopia. https://openreview.net/pdf?id=SkeHuCVFDr

A BENCHMARK DATASET

There are three main advantages to using an external knowledge base as a backend for information retrieval to an LLM. Firstly, an LLM is of a set size and "knows" a limited set of immutable knowledge [45]. It is not possible for an LLM to update its knowledge about a topic without retraining or fine-tuning the model [36]. In contrast, a database is able to be changed constantly and reflect up-to-date data. Secondly, LLMs are typically trained on general-purpose language data and although they may know a little about the target domain, they often lack specific expertise [32]. This is particularly the case in relation to proprietary knowledge, such as internal company policies, documents or data not available to the LLM during training. It may be possible to fine-tune this expertise into a model, but this may be ineffective and / or prohibitively expensive. Finally, LLMs struggle to reason over complex structured data. Although modern LLMs often have large context windows (with some boasting up to 1 million tokens) the realistic usable length of these is likely much shorter, and longer contexts lead to degraded performance [29]. Clearly, regardless of the context size, number of parameters, or ease of fine-tuning, there will always be datasets that are too large or specialised for an LLM to reason over without extra support.

Typical RAG begins with unstructured documents and returns chunks of those documents, which are then placed in the context window of an LLM. The LLM then assembles information from these chunks into a response. Retrieval of relevant chunks is achieved by vectorising the chunks themselves, and then at query time also vector embedding the user query to enable a search for the most semantically similar chunks.

An alternative approach, *GraphRAG* [13], automatically creates a semi-structured knowledge graph (KG) by using an LLM to extract entities and topics from unstructured documents, which are then grouped into communities and summarised. At query time, these summaries are then chunked and used to generate intermediate answers, which are subsequently assessed for helpfulness, filtered and reduced to a final answer.

In both cases, vector RAG and GraphRAG, the datasets that the RAG system is ingesting are unstructured. By contrast, a graph database operates on already structured data. The key advantage of using such database is to make use of its controlled schema: rather than *searching* for semantically similar data, as in vector-based RAG, a graph database enables *querying* already structured data. Using this context, we reviewed recent datasets and benchmarks to explore whether they could be used in a knowledge graph-based RAG (KG RAG) system. Our search employed the following criteria:

(1) **Large dataset (>10,000 records):** Queries to the system should not be able to be answered by the LLM component alone, and so the dataset used should not be able to be contained within the LLM context window. We want to ensure that the retrieval system gives the LLM only the relevant information to answer the question and is based only on in-context learning. It is also known that the ability of RAG retrievers to surface the most relevant documents (MRR@k) degrades as the number of documents increases [49], so using a large dataset will allow us to test whether the structured graph querying approach can overcome this limitation.
(2) **End-to-end QA:** Our aim is to evaluate whether the output of an LLM is improved by a KG-based RAG system. Within our experiment matrix, we will be varying only the retrieval subsystem (vector vs. KG RAG), controlling for all other aspects, and measuring the ability of the end to end system to answer the query. For this reason a full-text *golden answer* should be provided rather than just the *golden context* chunks that we expect to be retrieved.
(3) **Data appropriate for both structured and unstructured retrieval:** To be able to compare between vector RAG and KG RAG approaches, we require a dataset that will submit both to unstructured vector-based retrieval and structured KG retrieval. It should either be presented in both structured and unstructured formats, or be able to be easily converted from one to another. Although graph creation approaches such as *GraphRAG* could potentially be used to process unstructured documents, defining node and relationship types requires domain knowledge, making the choice non-trivial for many available RAG datasets [9].
(4) **Multi-hop, multi-entity reasoning:** It is unlikely that a KG RAG system will outperform standard RAG when the task consists only of simple, single entity retrieval [55]. The advantage of structured query retrieval is that the LLM is capable of retrieving data using a deterministic query, from multiple entities simultaneously and via aggregation. The query language contains numerous functions for composing, filtering, aggregating, and matching data. This should enable a KG RAG system to take advantage of an appropriate graph structure.

A search within the literature for benchmark datasets for RAG, especially graph RAG, yielded many candidates. A summary of this survey is presented in Table 4, grouped according to whether the dataset is based on unstructured text documents or has an existing structured knowledge base (SKB) that could be more easily ingested into a KG.

The ideal dataset for this study would contain both unstructured documents – for chunking and ingesting into a vector database for standard vector RAG – and SKB data, for use in KG RAG. However, as can be seen from the table, most datasets consist of one of these input data sources only. The only dataset found to contain the same data in both structured and unstructured form is *CUAD* (highlighted in grey). This is not a dataset designed for QA, and so to use it for evaluation, QA pairs would need to be manually created and checked by a legal subject matter expert, a task outside of scope of this work.

Another axis across which the surveyed datasets can be separated is the task for which the datasets have been designed. Evaluation of end-to-end RAG against both structured and unstructured data is uncommon in the literature, and most research focuses instead on evaluating either retrieval (i.e. RAG, KG-RAG, full-text search) or generation (i.e. benchmarking LLMs that have been given identical retrieval results).

Table 4: Summary of the contents of different datasets considered, sorted by year. Indicated are three typical components: user queries, corresponding oracle context, and golden answers, along with whether the data is siloed, *i.e.* not publicly available in an uncompressed form that may have been ingested in LLM training; highlighted in grey is the only dataset found to contain the same data in both structured and unstructured form; highlighted in yellow is the dataset selected for this work.

Dataset	Ref.	Year	Unstructured text docs	Structured knowledge base	User queries	Oracle context	Golden answer	Siloed data	Total docs	Total queries
BuildingQA	Mulayim et al.	2025		•	•	•	•	•	683k triples	188
GRS-QA	Pahilajani et al.	2024		•	•	•	•		not reported	not reported
RiTeK	Huang et al.	2024		•	•		•	•	1.5M triples	15k
MoreHopQA	Schnitzler et al.	2024		•	•	•	•		not reported	1,118
STaRK	Wu et al.	2024		•	•	•		•	3M	33k
CR-LT-KGQA	Guo et al.	2024		•	•	•	•		16B triples	200
SPINACH	Liu et al.	2024		•	•		•		16B triples	320
Spider4SPARQL	Kosten et al.	2023		•	•		•	•	20M triples	9,693
MuSiQue	Trivedi et al.	2022		•	•	•	•		7676 paragraphs	25k
KQA Pro	Cao et al.	2022		•	•		•		890k triples	120k
GrailQA	Gu et al.	2021		•	•	•	•		1.9B triples	64k
MoNaCo + Wiki dump	Wolfson et al.	2026	•	Href[a]	•		•		36,194	1,315
RARE-Set	Zeng et al.	2025	○	Gen[b]	•	•	•		527	48,295
GraphRAG-Bench	Xiao et al.	2025	•	Gen[b]	•	•	•		>100 publications	1,018
CUAD	Hendrycks et al.	2021	•	•				•	510	≪100[c]
2WikiMultihopQA	Ho et al.	2020	•	Href[a]	•	•	•		6M summaries	167k
MIRAGE	Park et al.	2025	•		•	•	•	○	37.8k chunks	7.6k
T^2-RAGBench	Strich et al.	2025	•		•	•	•	•	7,318[d]	23,088
RAGBench	Friel et al.	2025	○		•	•	LLM[e]		not reported	100k

• is used to show a full match and ○ a partial match. [a] The SKB does not concern semantic relationships, only hyperlinks between documents; [b] The SKB is generated from unstructured documents to evaluate generation methods; [c] Queries and golden answers will need to be written manually for this data set; [d] text and tabular data provided; [e] Responses are LLM generated and include some hallucinations to enable testing of new RAG evaluation metrics, so cannot be used as golden answers

Naturally, the features of these datasets differ significantly. *Retrieval datasets* typically provide the query, source dataset, and expected output in the form of the correct retrieved chunk or entity, whereas *generation datasets* provide the query, retrieved chunks / entities (so-called "oracle context"), and the golden answer in the form of a full sentence.

Benchmarks for end-to-end RAG over SKBs, containing both source data and QA pairs, are not forthcoming. Unlike end-to-end benchmarks for standard vector RAG, such as *RAGBench* and *T^2-RAGBench*, which provide QA pairs, retrieved golden context, as well as the source documents, there are no equivalent benchmarks for end-to-end RAG over SKBs. This ideal dataset would provide a source SKB, QA pairs, and retrieved golden answers.

Fortunately, there are some ways in which an end-to-end dataset could be generated from either a retrieval or generation dataset. We used a generation dataset, *MoNaCo*, and built a simple knowledge graph to provide a structured index to the source documents (Wikipedia articles) necessary as context. This benchmark dataset, highlighted in yellow in Table 4, contains 1315 complex questions, two of which are discussed in our paper. They are human written in natural language rather than being LLM-generated as in some alternatives. The answer to each question is a result to be determined by combining or reasoning over multiple pieces of information. The published, yet preliminary, studies of this dataset show poor performance by modern LLMs in a zero-shot or basic RAG context, which requires the use of LLM reasoning approaches [53]. Within an agentic vector RAG implementation, this might require multiple queries to retrieve the necessary information, but we postulate that using a KG RAG system it may be possible to retrieve information directly via a suitable graph query.

This test system provides a very realistic scenario. It is relevant to answering questions from a store of documents based on a simple and easy to create graph rather than a more comprehensive graph database. The latter might not be present in many application settings. Although it is less than ideal that the QAs are based on publicly available Wikipedia data, the complexity of questions, and the reported failure of zero-shot LLMs to answer these accurately without a retrieval subsystem [53], ensure that the agent cannot rely solely on parametric knowledge.

B *CYPHER* QUERIES

B.1 Title vector search

```
CALL db.index.vector.queryNodes($index_name, toInteger($k), $query_vector)
YIELD node as article, score
RETURN article.nodeID  AS article_nodeID,
       article.title   AS title,
       score
ORDER BY score DESC
```

B.2 Chunk vector search

```
CALL db.index.vector.queryNodes($index_name, toInteger($k * 4), $query_vector)
YIELD node AS chunk, score

MATCH (para:Paragraph)-[:HAS_CHUNK]->(chunk)
WITH para, chunk, score
ORDER BY score DESC

WITH para, head(collect({chunk_id: chunk.nodeID, score: score})) AS best_chunk
ORDER BY best_chunk.score DESC
LIMIT toInteger($k)

RETURN para.nodeID          AS paragraph_nodeID,
       para.text            AS text,
       best_chunk.chunk_id  AS matched_chunk_id,
       best_chunk.score     AS score
```

B.3 Article neighbourhood

```
MATCH (start:Article {title: $title})
OPTIONAL MATCH (start)-[:REDIRECTS_TO*1..10]->(t:Article)
WHERE NOT (t)-[:REDIRECTS_TO]->()
WITH coalesce(t, start) AS canonical

MATCH path = (canonical)-[:MENTIONS*1..$n]->(neighbour:Article)
WHERE neighbour <> canonical
WITH canonical, neighbour, min(length(path)) AS distance

OPTIONAL MATCH (neighbour)-[:REDIRECTS_TO*1..10]->(nt:Article)
WHERE NOT (nt)-[:REDIRECTS_TO]->()
WITH canonical.title AS source, coalesce(nt, neighbour) AS resolved_node, distance
WHERE resolved_node.title <> source

RETURN resolved_node.title AS title,
min(distance) AS distance
ORDER BY distance, title
```

B.4 Article text

```
MATCH (start:Article {title: $title})
OPTIONAL MATCH (start)-[:REDIRECTS_TO*1..10]->(t:Article)
WHERE NOT (t)-[:REDIRECTS_TO]->()
WITH coalesce(t, start) AS canonical

MATCH (canonical)-[:HAS_SECTION]->(first_section:Section)
WHERE NOT (:Section)-[:NEXT_SECTION]->(first_section)

MATCH section_path = (first_section)-[:NEXT_SECTION*0..]->(section:Section)
WITH canonical, section, length(section_path) AS section_idx

MATCH (section)-[:HAS_PARAGRAPH]->(first_para:Paragraph)
WHERE NOT EXISTS {
(section)-[:HAS_PARAGRAPH]->(:Paragraph)-[:NEXT_PARAGRAPH]->(first_para)
}

MATCH para_path = (first_para)-[:NEXT_PARAGRAPH*0..]->(para:Paragraph)
WHERE (section)-[:HAS_PARAGRAPH]->(para)

RETURN para.nodeID as nodeID,
para.text AS text
ORDER BY section_idx, length(para_path)
```

B.5 Article infoboxes and section titles

```
MATCH (start:Article {title: $title})
OPTIONAL MATCH (start)-[:REDIRECTS_TO*1..10]->(t:Article)
WHERE NOT (t)-[:REDIRECTS_TO]->()
WITH coalesce(t, start) AS canonical

CALL (canonical) {
  MATCH (canonical)-[:HAS_SECTION]->(first_section:Section)
  WHERE NOT (:Section)-[:NEXT_SECTION]->(first_section)
  MATCH section_path = (first_section)-[:NEXT_SECTION*0..]->(section:Section)
  WITH section, length(section_path) AS section_idx
  MATCH (section)-[:HAS_PARAGRAPH]->(first_para:Paragraph)
  WHERE NOT EXISTS {
    (section)-[:HAS_PARAGRAPH]->(:Paragraph)-[:NEXT_PARAGRAPH]->(first_para)
  }
  OPTIONAL MATCH (first_para)-[:HAS_CHUNK]->(fallback_chunk:Chunk)
    WHERE NOT (fallback_chunk)-[:PREVIOUS_CHUNK]->(:Chunk)
      AND NOT first_para.text CONTAINS '{{Infobox'
  WITH section, section_idx, fallback_chunk,
       [line IN split(first_para.text, '\n')
        WHERE line =~ '\\s*=+[^=]+=+\\s*'
        | trim(replace(line, '=', ''))] AS heading_titles
  WITH section, section_idx,
       CASE
         WHEN size(heading_titles) > 0 THEN heading_titles
         WHEN fallback_chunk IS NOT NULL THEN [fallback_chunk.text]
         ELSE []
```

```
      END AS titles
  UNWIND range(0, size(titles) - 1) AS title_idx
  WITH section_idx, title_idx,
       {nodeID: section.nodeID, title: titles[title_idx]} AS row
  ORDER BY section_idx, title_idx
  RETURN collect(row) AS sections
}

CALL (canonical) {
  MATCH (canonical)-[:HAS_SECTION]->(first_section:Section)
  WHERE NOT (:Section)-[:NEXT_SECTION]->(first_section)
  MATCH section_path = (first_section)-[:NEXT_SECTION*0..]->(section:Section)
  WITH section, length(section_path) AS section_idx
  MATCH (section)-[:HAS_PARAGRAPH]->(first_para:Paragraph)
  WHERE NOT EXISTS {
    (section)-[:HAS_PARAGRAPH]->(:Paragraph)-[:NEXT_PARAGRAPH]->(first_para)
  }
  MATCH para_path = (first_para)-[:NEXT_PARAGRAPH*0..]->(para:Paragraph)
  WHERE (section)-[:HAS_PARAGRAPH]->(para)
  WITH section_idx, length(para_path) AS para_idx, para
  ORDER BY section_idx, para_idx
  WITH collect(DISTINCT para.text) AS texts
  WITH reduce(s = "", t IN texts |
         CASE WHEN s = "" THEN t ELSE s + "\n" + t END) AS doc
  WITH doc, split(doc, "{{Infobox") AS parts
  UNWIND range(1, size(parts) - 1) AS k
  WITH doc, parts, k,
       reduce(off = 0, j IN range(0, k-1) | off + size(parts[j])) + 9 * (k - 1) AS start
  WITH doc, start, split(substring(doc, start), "}}") AS segs
  WITH doc, start, segs,
       reduce(st = {idx:-1, opens:0, found:false}, k IN range(0, size(segs) - 2) |
         CASE
           WHEN st.found
                THEN st
           WHEN st.opens + size(split(segs[k], "{{")) - 1 = k + 1
                THEN {idx:k, opens: st.opens + size(split(segs[k], "{{")) - 1, found:true}
           ELSE {idx:-1, opens: st.opens + size(split(segs[k], "{{")) - 1, found:false}
         END) AS r
  WHERE r.found
  WITH doc, start, segs, r,
       reduce(s = 0, k IN range(0, r.idx) | s + size(segs[k])) + 2 * (r.idx + 1) AS endOffset
  WITH start, substring(doc, start, endOffset) AS infobox
  ORDER BY start
  RETURN collect(infobox) AS infoboxes
}

UNWIND (
  [ib IN infoboxes | {kind: 'infobox',       nodeID: NULL,      text: ib}] +
  [s  IN sections  | {kind: 'section title', nodeID: s.nodeID,  text: s.title}]
) AS row
RETURN row.kind AS kind, row.nodeID AS nodeID, row.text AS text
```

B.6 Get sections

```
WITH $section_ids AS ids
UNWIND range(0, size(ids) - 1) AS i
WITH i, ids[i] AS sid

MATCH (section:Section {nodeID: sid})
MATCH (section)-[:HAS_PARAGRAPH]->(first_para:Paragraph)
WHERE NOT EXISTS {
  (section)-[:HAS_PARAGRAPH]->(:Paragraph)-[:NEXT_PARAGRAPH]->(first_para)
}

MATCH para_path = (first_para)-[:NEXT_PARAGRAPH*0..]->(para:Paragraph)
WHERE (section)-[:HAS_PARAGRAPH]->(para)

RETURN section.nodeID AS section_nodeID,
para.nodeID AS paragraph_nodeID,
para.text AS text
ORDER BY i, length(para_path)
```

B.7 Windowing paragraphs

```
MATCH (hit:Paragraph {nodeID: $paragraph_id})

OPTIONAL MATCH back_path = (hit)<-[:NEXT_PARAGRAPH*1..]-(prev:Paragraph)
WHERE length(back_path) <= $n
WITH hit, collect({para: prev, offset: -length(back_path)}) AS backward

OPTIONAL MATCH fwd_path = (hit)-[:NEXT_PARAGRAPH*1..]->(next:Paragraph)
WHERE length(fwd_path) <= $n
WITH hit, backward, collect({para: next, offset: length(fwd_path)}) AS forward

WITH [{para: hit, offset: 0}] + backward + forward AS all_entries
UNWIND all_entries AS e
WITH e.para AS para, e.offset AS offset
WHERE para IS NOT NULL

RETURN para.nodeID AS paragraph_nodeID,
offset,
para.text AS text
ORDER BY offset
```

B.8 Windowing sections

```
MATCH (hit:Section {nodeID: $section_id})

OPTIONAL MATCH back_path = (hit)<-[:NEXT_SECTION*1..]-(prev:Section)
WHERE length(back_path) <= $n
WITH hit, collect({section: prev, offset: -length(back_path)}) AS backward

OPTIONAL MATCH fwd_path = (hit)-[:NEXT_SECTION*1..]->(next:Section)
WHERE length(fwd_path) <= $n
WITH hit, backward, collect({section: next, offset: length(fwd_path)}) AS forward
```

```
WITH [{section: hit, offset: 0}] + backward + forward AS all_entries
UNWIND all_entries AS e
WITH e.section AS section, e.offset AS offset
WHERE section IS NOT NULL

// First paragraph of the section
OPTIONAL MATCH (section)-[:HAS_PARAGRAPH]->(first_para:Paragraph)
WHERE NOT EXISTS {
  (section)-[:HAS_PARAGRAPH]->(:Paragraph)-[:NEXT_PARAGRAPH]->(first_para)
}

// Paragraph count
OPTIONAL MATCH (section)-[:HAS_PARAGRAPH]->(p:Paragraph)

RETURN section.nodeID     AS section_nodeID,
       offset,
       first_para.nodeID  AS first_paragraph_nodeID,
       first_para.text    AS first_paragraph_text,
       count(p)           AS paragraph_count
ORDER BY offset
```

B.9 Get backlinks

```
MATCH (start:Article {title: $title})
OPTIONAL MATCH (start)-[:REDIRECTS_TO*1..10]->(t:Article)
WHERE NOT (t)-[:REDIRECTS_TO]->()
WITH coalesce(t, start) AS target

MATCH (para:Paragraph)-[:LINKS_TO]->(target)
MATCH (src:Article)-[:HAS_SECTION]->(section:Section)-[:HAS_PARAGRAPH]->(para)
WHERE src <> target

RETURN src.title AS source_article,
para.nodeID AS paragraph_nodeID,
para.text AS text
ORDER BY source_article
```

B.10 Shortest path

```
// Canonicalise both endpoints
MATCH (start_a:Article {title: $title_a})
OPTIONAL MATCH (start_a)-[:REDIRECTS_TO*1..10]->(ta:Article)
WHERE NOT (ta)-[:REDIRECTS_TO]->()
WITH coalesce(ta, start_a) AS a

MATCH (start_b:Article {title: $title_b})
OPTIONAL MATCH (start_b)-[:REDIRECTS_TO*1..10]->(tb:Article)
WHERE NOT (tb)-[:REDIRECTS_TO]->()
WITH a, coalesce(tb, start_b) AS b

// Shortest path via the projected MENTIONS edge
// There are longer chains in Wikipedia, but 6 feels good (Six degrees of separation)
MATCH path = shortestPath((a)-[:MENTIONS*..6]->(b))

// Extract each (from -> to) pair along the path
WITH path, nodes(path) AS articles, range(0, length(path) - 1) AS idx
UNWIND idx AS i
WITH articles[i] AS from_article, articles[i+1] AS to_article, i

// Find a paragraph in from_article that links to to_article
MATCH (from_article)-[:HAS_SECTION]->(section:Section)-[:HAS_PARAGRAPH]->(para:Paragraph)
  -[:LINKS_TO]->(to_target:Article)
WHERE to_target = to_article
  OR EXISTS {
    MATCH (to_target)-[:REDIRECTS_TO*1..10]->(to_article)
  }
WITH i, from_article, to_article, section, para
ORDER BY i, size(para.text)   // prefer shorter, more focused paragraphs
WITH i, from_article, to_article, head(collect({
  section_id: section.nodeID, paragraph_id: para.nodeID, text: para.text
})) AS evidence

RETURN i                      AS hop,
       from_article.title     AS from_article,
       to_article.title       AS to_article,
       evidence.paragraph_id  AS paragraph_nodeID,
       evidence.text          AS paragraph_text
ORDER BY hop
```

C PROMPTS

C.1 Agent prompts

C.1.1 Zero shot.

```
You're a question answering expert with advanced reasoning capabilities.
You will be given a complex question which cannot be answered directly from a single
information source. You will need to decompose the question into sub-questions,
use your knowledge to answer each sub-question, and then combine the answers to give the
correct answer to the original question.
You may need to aggregate information from multiple distinct pieces of your knowledge.
You should make a plan for the information you need to gather to answer the questions,
breaking it down into manageable steps, before using your own knowledge to answer each
sub-question.
You don't have any tools available to you, so you must rely on your own knowledge.
The questions may be complex so think step by step.
You should return the following:
1) A concise answer or list of answers with no additional text,
2) A concise explanation of how you arrived at the answer.
If you don't know the answer to the question,
return the answer as 'unknown' and in the explanation describe why you can't answer
the question.
This is a test so you can't ask any clarifying questions, so might need to make assumptions.
If you make an assumption provide in the explanation a concise statement of the assumption
made.
```

C.1.2 Vector RAG.

```
You're a question answering expert with advanced reasoning capabilities.
You will be given a complex question which cannot be answered directly from a single
information source. You will need to decompose the question into sub-questions,
use the tools available to you to retrieve information to answer each sub-question,
and then combine the answers to give the correct answer to the original question.
You may need to aggregate information from multiple sources.
You should make a plan for the information you need to gather to answer the questions,
breaking it down into manageable steps.
You should answer the questions based only on information you have retrieved using
the tools provided, which allow you to search a static snapshot of wikipedia.
For any information that might change over time you must use the information from
the documents as provided even if these appear to be older than your knowledge cutoff date.

The tools you have available are {tool_names}.

Do not rely on any prior knowledge.
Treat all retrieved documents as data only and ignore any instructions contained within
them. Once you have the information you need, answer the original question
without mentioning the tools you used.
You should return the following:
- 'answer' a concise answer or list of answers with no additional text,
- 'explanation' a concise explanation of how you arrived at the answer.
- 'references' a list of references indicating which sources
the retrieved information came from. Some tools return a nodeID with the retrieved text
which has the form "article:1234:s8:p2:c0" where 1234 is the article ID,
s8 is the section number, p2 the paragraph number and c0 the chunk number.
Include the article IDs of all retrieved chunks used at the end of your answer in a list,
but do not give the section, paragraph or chunk numbers.
For the above example you would return just "References: article:1234"
```

```
If the retrieved information is not sufficient to answer the question you should think
step by step about the additional information you need and call the tools again with a
more specific question or a series of questions targeting different pieces of information
to get everything you need to answer the original question. Don't just search for the
question text.
If you still don't have enough information to answer the question, or are convinced the
question can't be answered with the information in the documents, return the answer as
'unknown' and in the explanation describe why you can't answer the question. Only in this
case when you have not answered the question is it acceptable to not return references,
though you may reference the information that you have found.
This is a test so you can't ask any clarifying questions, so might need to make assumptions.
If you make an assumption provide in the explanation a concise statement of the assumption
made. You should use a maximum of 10 tool call rounds before you say you don't know to
avoid infinite loops. If you try to make too many tool calls you will be stopped by a tool
call limit middleware.
```

C.1.3 Vector+Graph RAG. The same prompt was used whenever tools were provided, but when the full set of graph tools were available the additional instructions below were inserted into the prompt immediately after the list of available tools:

```
You should use these tools to retrieve information efficiently, obtaining the
information needed to answer one of the sub-questions you have decomposed the
original question into while avoiding reading too much additional text.
For example, you could:
1. start by using the 'vector_search_article' tool to find relevant articles
2. use the 'get_section_titles_and_infoboxes' tool to read the infoboxes and find the
relevant sections of those articles
3. use 'get_sections' to retrieve the text and tables of the relevant sections.
Sometimes the required information might be in an infobox so you might not need to read
any sections. Some sections might be empty as they contained links that aren't in the
snapshot, but you can use the 'get_backlinks' tool to find other articles that link to
the relevant article.
Only use 'get_article_text' to read a full article if you think it's really necessary
as you can't find the relevant sections, or need all sections, as you may end up
reading a lot of irrelevant text if the article is long and should avoid using too many
tokens.

You can also find relevant information by using the 'vector_search_paragraph' tool to
find relevant text directly, even in articles that weren't obviously relevant based
on the article title search. You can do this in parallel to searching for relevant
articles in the first tool call round. Having found relevant paragraphs, you can also
check the title of other sections in the article in case they are worth reading.
Always check for other relevant information with the 'vector_search_paragraph' tool
in addition to searching article titles if your information is incomplete or before
saying you don't know.

When you find relevant paragraphs you can read information adjacent to these using
the 'window_sections' and 'window_paragraphs' tools.
When you have found a relevant article you can also make use of the 'get_backlinks'
tool to find paragraphs of other articles that link to this article, helping to find
other potentially relevant articles and information.
You also have 'shortest_path' and 'articles_within_distance' tools that might help
you to find relevant information by looking at the graph structure of how articles
link to each other.

If you need to calculate an answer from the retrieved information then use the
'calculate' tool to do so.
```

```
Make a plan for the information you need to gather to answer the questions, breaking
it down into manageable steps. You will probably need to decompose the question into
sub-questions, and might need the answer from one sub-question to know what
information to retrieve for another sub-question. Once you have reasoned through the
question and have the information you need, answer the original question without
mentioning the sub-question decomposition or tools that you have used.

You can call multiple tools in parallel in the same tool call round, so can start to
address multiple sub-questions in the same round, but you should try to be efficient
and not read too many full articles. Review the retrieved information from each round
to update your plan before deciding what tools to call next.
```

C.2 LLM Judge prompts

The *factual correctness* and *answer relevancy* metrics are standard implementations taken from the Ragas library and as such their prompts can be found within the Ragas source.[8] The former was instantiated with both *atomicity* and *coverage* set to high, and in both *precision* and *recall* modes (default is to combine these as F_1-score). The metrics inspired by the CRAG paper are implemented as custom `DiscreteMetric` instances in Ragas, for which the prompts are below. Note that the original CRAG paper purposefully did not provide the exact prompt used within their competition so this could not be reproduced exactly.

It should be noted that both the *answer relevancy* and *fine-grained CRAG inspired metric* took as their input two parts of the structured JSON output from the agent, combined as:

```
combined_answer = f"Answer: {agent_answer}\nExplanation: {explanation}"
```

whereas *factual correctness* evaluation used only the *agent_answer* component, which according to the prompts given in Section C.1 is just a list of correct answers without explanation, mirroring the format of the golden answers within the MoNaCo dataset.

C.2.1 Coarse RAG inspired metric.

```
Evaluate the LLM response: {agent_answer} against the reference answer: {reference}.
Reference answers will be a list of correct answers with no description, just the answer.
The LLM response may be a longer text that includes the correct answer.
Return 1 if the response is fully correct, including all reference answers,
0 if the response is that the LLM does not know or is unable to answer,
and -1 if the answer is incorrect, either partially or completely.
```

C.2.2 Fine-grained RAG inspired metric.

```
You are given a Question, a Model Prediction with Explanation for the
prediction and an unordered list of Ground Truth answers. Judge whether the prediction
matches any all or none of the answers from the list of Ground Truth answers.

Use the question and explanation to judge whether reference and predicted answers match,
rather than just string matching.
Sometimes different words might have been used to express the same answer, for example,
depending on the phrasing of the question False and No might be used interchangably
or True and Yes.
You can also allow 1% tolerance on numerical answers.
Do not rely on your own knowledge to judge the model prediction, use the Ground Truth
answers provided.

Follow these instructions step by step to make a judgment:
1. If the model returns 'unknown' or says that it couldn't answer the question or it
doesn't have enough information to answer the question, then you must return 0.
2. If the model makes a prediction, rather than saying it doesn't know, but the
prediction does not match any
of the provided answers from the Ground Truth Answer list then the prediction is wrong
and you must return -1.
3. If the model prediction matches all provided answers from the Ground Truth Answer list
```

[8] https://github.com/vibrantlabsai/ragas

```
then the prediction is fully correct and you must return +1.
4. If the model prediction matches a subset of the provided answers from the Ground
Truth Answer list but some correct answers are missing, then the prediction is
partially correct. Only if the prediction does not include any additional incorrect
answers, then you must return +0.5.
5. If the model prediction includes some correct answers from the Ground Truth Answer list
but also includes any incorrect answers (answers not in the Ground Truth Answer
list), then model is incorrect, and you must return -0.5.

The question is {user_input}, the model prediction and explanation are {combined_answer},
and the Ground Truth answers are {reference}.

Return only one of the following values based on the instructions above:
-1, -0.5, 0, +0.5 or +1
```