Web Semantics: Science, Services and Agents on the World Wide Web 92 (2027) 100890

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Can LLMs enable natural language querying of knowledge graphs? A multi-model evaluation of explanation and generation with interactive refinement

Larissa Pusch [a,*], Alexandre Courtiol [b], Tim Conrad [a]

[a] Zuse Institute Berlin, Department of Visual and Data-Centric Computing, Takustraße 7, Berlin, 14195, Germany
[b] Leibniz Institute for Zoo and Wildlife Research, Department of Evolutionary Genetics, Alfred-Kowalke-Straße 17, Berlin, 10315, Germany

ARTICLE INFO

Dataset link: https://git.zib.de/lpusch/talk2cypher

Keywords:
Large Language Models
Knowledge Graphs
Cypher
Query explanation
Interactive refinement

ABSTRACT

Large Language Models (LLMs) have shown strong performance across language tasks, but remain limited in knowledge-intensive domains due to hallucinations, outdated information, and limited explainability. Text-based retrieval-augmented generation (RAG) helps ground model outputs in external sources but often struggles with multi-hop reasoning. Knowledge Graphs (KGs), in contrast, support precise, explainable querying, yet require specialized query languages that limit accessibility. Using an interactive framework in which users refine generated queries, we evaluate how well current LLMs can bridge this gap for non-expert users by generating and explaining Cypher graph queries and detecting faults across varying levels of structural and semantic complexity. Our multi-model evaluation provides insight into where models succeed and fail, how performance varies across domains, and which query features most affect accuracy. Our core quantitative evaluation is a 90-query benchmark on a synthetic movie KG that measures query explanation quality and fault detection across multiple LLMs, complemented by two smaller real-life query-generation experiments on a Hyena KG and the MaRDI (Mathematical Research Data Initiative) KG. Our findings demonstrate that transparent, auditable KG querying is feasible but not yet reliable without human oversight. The code is available at https://git.zib.de/lpusch/talk2cypher.

1. Introduction

Large Language Models (LLMs) have become integral to question answering (QA) tasks due to their advanced language understanding and generation capabilities. Yet their utility is often limited in knowledge-intensive domains where precision and interpretability are critical. In such settings, LLMs frequently suffer from outdated or hallucinated knowledge and may generate a plausible-sounding but incomplete or incorrect answer [1] without clearly indicating the sources or reasoning steps underlying its conclusion [2].

To address these limitations, LLM-based retrieval -augmented generation (RAG) techniques ground model outputs in external information sources without including these sources in the LLM training process. However, because most RAG systems rely on text-based retrieval [3], they struggle to support multi-hop reasoning - that is, answering questions that require combining information from multiple, indirectly connected sources. For example, determining which researchers cited a paper that implemented a specific algorithm involves chaining through authors, publications, and software entries. Traditional semantic similarity search often fails to capture such indirect relationships, especially when relevant information spans documents or concepts with little lexical overlap [4].

Knowledge Graphs (KGs), structured representations of entities and their relationships, offer an alternative. Unlike unstructured text, KGs encode domain knowledge in a way that supports logical inference, schema evolution, and semantic querying [5]. This makes them well-suited for applications requiring contextualized, verifiable answers, such as natural sciences, biomedicine or mathematics. Search engines have long recognized these advantages: Google's Knowledge Graph rollout in 2012 improved ranking and disambiguation by resolving queries over graph entities instead of raw keywords [5,6]. Knowledge Graphs come in several data models, probably the most prominent being RDF graphs like Wikidata [7] which can be queried using `SPARQL` and labeled property graphs queried with languages such as `Cypher` [5]. In this work we focus on the latter: all three KGs in our evaluation are represented as property graphs in Neo4j and queried with Cypher.

* Corresponding author.
E-mail addresses: pusch@zib.de (L. Pusch), courtiol@izw-berlin.de (A. Courtiol), conrad@zib.de (T. Conrad).

https://doi.org/10.1016/j.websem.2026.100890
Received 6 March 2026; Received in revised form 4 September 2026; Accepted 3 November 2026
Available online 10 November 2026
1570-8268/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Despite recent progress, KG-based retrieval-augmented generation (KG-RAG) pipelines can still be suboptimal. Three design choices are particularly limiting:

(a) **Linearized triples erase graph structure.** Some pipelines flatten a KG or its subgraph into a list of subject - predicate - object triples and pass them to the language model verbatim or in rigid templates [8]. The resulting text is verbose, inflates the context window, and forces the model to reconstruct relational structure that was explicit in the original schema.
(b) **Black-box retrieval hides the reasoning path.** Other systems embed the graph using a graph neural network (GNN) and retrieve nodes or paths by using similarity measures [8]. While the retrieved items and their similarity scores are inspectable, the transformation that maps the user's query to that vector, and thus the full reasoning path, remains implicit and can only be approximated with post-hoc explainability tools. An explicit `SPARQL` or `Cypher` query, by contrast, would expose an auditable, deterministic path from data to answer.
(c) **No user-in-the-loop control.** Many architectures treat the user as a passive consumer: once retrieval and generation start, there is no hook to correct a mistaken entity match, broaden or narrow the scope, or override the model's drafting choices. Lack of such feedback channels reduces transparency and flexibility and weakens trust.

These limitations motivated the design of a system in which queries are explicit and auditable and users can intervene - providing a controlled setting for studying how LLMs perform on query generation, explanation, and refinement.

In our previous work [9], we evaluated LLMs on translating biomedical questions into `Cypher` queries over a biomedical KG. We released a benchmark question set and an open-source interface, but the lack of means to debug, refine or explain queries meant that any failure required a complete regeneration. This highlights a disconnect between user intent and graph-based query execution.

In this paper we close this gap by extending that framework with explanation and amendment capabilities, creating a controlled environment for studying LLM performance on Cypher-related tasks. Users can provide feedback through natural-language amendments, which are used to iteratively update the query. We then use this environment to ask: *How accurately can LLMs explain Cypher queries, how reliably do they detect faults, and how does their query-generation performance vary with query complexity and domain?* We evaluate explanation and fault detection through a 90-query benchmark on a synthetic movie KG, and test generation capabilities via case studies on two heterogeneous real-world KGs: a biological KG from the Ngorongoro Hyena Project and a mathematical research KG from the Mathematical Research Data Initiative (MaRDI) [10].

The Hyena KG was derived from a subset of data collected for the long-term monitoring of a free-ranging population of spotted hyenas from the Ngorongoro Crater in Tanzania (see https://hyena-project.com). Biologists have been studying this population for nearly 30 years, collecting a wide range of data (on demography, behavior, morphology, physiology, diseases and genetics) to address fundamental ecological and evolutionary questions (e.g. *Why do some individuals disperse to other social groups while others do not?* [11]), as well as specific questions related to this particular population (e.g., *How does pastoralism impact these hyenas?* [12]). As the composition of social groups (called clans) is dynamic, and interactions between individuals are strongly influenced by relatedness, much of the work on the hyena data requires tracking the movements of individuals across clans and their affiliations. This means that researchers require in-depth knowledge of the data structure and expertise in database manipulation to answer even the simplest questions. In order to reduce the barriers to entry for new researchers, members of the Ngorongoro Hyena Project have developed a substantial `R` package (*hyenaR*, with over 450 exported functions at the time of writing). While this tool is already helpful, users still need to be proficient in the `R` language, which not all biologists are. Moreover, new projects tend to require additional functions, thereby increasing the development and maintenance burden for the developers. Finally, a tool such as *hyenaR* cannot easily be adapted for the study of other long-term projects by other research groups since much of the coding is tailored to the idiosyncratic structure of the data. A reliable LLM/KG combination could be a more appealing alternative because it represents a generic solution that would largely remove the programming burden while providing an interface accessible to all.

MaRDI's open Knowledge Graph already consists of over 700 million triples, exposing them through public `SPARQL` and `REST` endpoints. This infrastructure is designed to support FAIR principles in mathematics, enabling users to, for example, trace a theorem to its software implementation or link benchmark datasets to all related publications. However, the technical barrier posed by `SPARQL` continues to limit accessibility. Our approach, applied to a subgraph of MaRDI, could remove that barrier without sacrificing the precision and interpretability provided by curated KGs.

To compare many LLMs under identical conditions, we keep the question sets for the MaRDI and Hyena KGs small. This trades depth per KG for breadth across models. As a result, the study is optimized to reveal failure modes and inconsistencies across models, rather than to provide an exhaustive ranking of overall performance. Two models became inaccessible partway through and are therefore scored on a subset of the questions.

We evaluate our system on the three KGs, measuring the faithfulness of natural-language explanations on the synthetic movie KG and the accuracy of generated queries on the MaRDI and Hyena KG. This work contributes toward making semantic graph querying more transparent, interactive, and accessible to non-experts.

In summary, combining the expressiveness of `Cypher` with the accessibility of LLMs, we designed a system that addresses the limitations identified above and provides a controlled setting for evaluations. The key contributions of this work are:

(a) An empirical evaluation of LLM capabilities for Cypher query explanation, fault detection, and generation, using a 90-query benchmark that controls for path length, clause type, and error injection, complemented by two real-world case studies on domain-expert (MaRDI KG and Hyena KG) Knowledge Graphs.
(b) An interactive framework, combining query generation, natural-language explanation, execution, and iterative amendment, that serves as a practical tool for non-expert graph exploration. The framework emphasizes transparent, auditable queries paired with a conversational amendment loop.
(c) Insights into common LLM failure modes when generating and explaining Cypher queries, including performance variability across models and the importance of measuring both accuracy and amendment efficiency (attempts per success).

The novelty of our approach lies in using a transparent, interactive pipeline, where users can inspect, explain, and amend LLM-generated Cypher queries, as a vehicle for systematically evaluating how LLM explanation and generation capabilities scale with query complexity and domain difficulty. In Section 3, we present the architecture and main components of this framework, demonstrating how they work together to enable query generation, execution, transparent explanation, and iterative refinement. To make the framework's capabilities more concrete, we begin with an example interaction that illustrates these functionalities. This also serves to highlight the importance of transparency and control in Knowledge Graph interaction.

1.1. Example interaction

Before describing the internal components, we present an example interaction to give a concrete impression of the framework in action. In this two-turn exchange, a user asks a natural-language question, the LLM generates the corresponding graph (Cypher) query, it gets executed on the graph database and results are returned. The example also shows how the framework handles clarification and refinement when the initial interpretation of a query does not align with the user's intent.

T1 *User:* Which authors does `graphclust` have?

LLM → Cypher

```
MATCH (p:Publication {name:"graphclust"})-[:
    HAS_AUTHOR]->(a:Author)
RETURN a.name;
```

DB → Result *Empty Result because no `Publication` node called *graphclust* exists.*

LLM (short explanation).

(a) Locate the single `Publication` node whose `name` equals "graphclust".
(b) Traverse every outgoing `:HAS_AUTHOR` edge to neighbouring `Author` nodes.
(c) Return the `name` of each such `Author`.

Purpose (one line): List all authors associated with the publication `graphclust`.

Plausibility check: The pattern is sensible: publications "have" authors and the labels/edge name fit the domain. Some schemas flip the edge (`Author` $\xrightarrow{\text{:AUTHORED}}$ `Publication`), but the chosen `Publication`→`Author` direction is equally valid.

T2 *User:* Actually, I meant the *software package*, not the publication.

LLM → Cypher

```
MATCH (s:SoftwarePackage {name:'graphclust'})
    -[:HAS_AUTHOR]->(a:Author)
RETURN a.name;
```

DB → Result *Tabea Rebafka*

This example highlights the framework's core capabilities: translating natural language into structured queries, explaining the resulting logic, and enabling users to refine them through conversational feedback. This allows the system to bridge the gap between natural language and formal query syntax, without requiring users to learn a query language, such as Cypher.

These capabilities are supported by a modular architecture which allows for switching of LLMs or KGs; Section 3 describes it. First, Section 2 positions this work relative to existing research.

2. Related work

Our work builds on three research threads: the generation of structured queries from natural language, the explanation of formal queries to non-expert users, and the interactive refinement of generated queries. We review each and then position our contribution.

2.1. Generating structured queries from natural language

Translating natural-language questions into formal queries is a long-standing research area. The largest body of work concerns text-to-SQL, ranging from early conversational and voice-based interfaces such as EchoQuery [13] to neural approaches for context-dependent query generation [14] and, more recently, LLM-based pipelines, including human-LLM collaborative annotation for adapting text-to-SQL systems to new domains [15].

For Knowledge Graphs, an analogous line of work targets SPARQL and, increasingly, Cypher. On the resource side, SpCQL [16] introduced the first Text-to-Cypher dataset with 10,000 manually annotated question-query pairs over a single Neo4j graph, Text2Cypher [17] and CypherBench [18] provide large-scale corpora of natural-language-Cypher pairs, and ZOGRASCOPE [19] contributes expert-annotated question-query pairs over a property graph with dedicated compositional and length-generalization splits. Auto-Cypher [20] synthesizes training data through an LLM-supervised generation-verification pipeline in which candidate queries are validated by execution. On the methods side, a large body of Knowledge Graph Question Answering (KGQA) work generates SPARQL through fine-tuned models such as ChatKBQA [21] and NLQxform [22], chain-of-thought prompting [23], and semantic parsing or program synthesis [24,25]. Other approaches harden generation structurally: SparqLLM [26], a platform spanning KG construction and result visualization, retrieves SPARQL query templates as context for the LLM to improve query reliability and reduce semantic errors, and GraphQ IR [27] introduces a typed intermediate representation with a natural-language-like surface form that a semantic parser can target and that compiles losslessly into SPARQL, Cypher, and other query languages. For Cypher specifically, Fact Finder [28] incorporates schema and relation descriptions into the prompt, an approach close to our own generation component.

These benchmarks consist of well-formed query-question pairs and these pipelines are evaluated on end-to-end generation accuracy. None contain systematically injected errors, and none evaluate whether models can explain queries or detect faults in them: the abilities targeted by our 90-query benchmark in Section 4.1. Moreover, hardening generation through intermediate representations or templates adds machinery between the user and the query; our framework instead exposes the raw Cypher query and asks whether the LLM itself can make it inspectable.

2.2. Query explanation and guided query construction

A second thread makes formal queries understandable, or avoids free-form generation altogether. For SQL, SQLucid [29], building on STEPS [30], grounds model-generated queries in editable step-by-step natural-language explanations linked to intermediate query results; user edits to the explanation are propagated back into query regeneration. For SPARQL, NLQxform-UI [31] exposes the intermediate artifacts of a BART-based question-to-SPARQL pipeline over the DBLP Knowledge Graph, letting users inspect and manually alter logical forms and matched templates, and, in work concurrent with ours, InteracSPARQL [32] derives structured explanations deterministically from the SPARQL abstract syntax tree before applying LLM-based linguistic refinement. A complementary approach forgoes NLP entirely: guided query builders such as Sparklis [33] and the Wikidata Query Builder [34] let users assemble queries interactively through constrained natural-language elements, guaranteeing well-formed queries by construction at the cost of navigating the schema manually.

Deterministic explainers and guided builders achieve faithfulness or correctness by construction, but presuppose either a language-specific explanation grammar (built for SQL in SQLucid/STEPS and for SPARQL ASTs in InteracSPARQL) or schema navigation by the user. Our work addresses the complementary question these designs sidestep: how reliably can a general-purpose LLM, with no dedicated explanation layer, explain arbitrary Cypher queries and flag faults in them? Quantifying

this (explanation quality, perturbation detection, and false-positive rates across five models) also provides empirical grounding for hybrid designs such as InteracSPARQL, which delegate the linguistic portion of the explanation to an LLM.

2.3. Interactive and automated query refinement

Refining an initial, possibly incorrect query has been studied in both user-driven and automated variants. For SQL, DialSQL [35] detects likely errors in a generated query and resolves them through multiple-choice dialogue with the user, and Zhang et al. [14] generate queries in multi-turn contexts by editing the previous query rather than regenerating it; a similar strategy to the one our Amender adopts for Cypher. SQLucid's explanation-editing loop [29] is likewise a user-driven refinement mechanism. On the automated side, Multi-Agent GraphRAG [36] composes LLM agents for Cypher generation, semantic and syntactic evaluation, named-entity verification against database contents, and feedback aggregation into an iterative, database-grounded correction loop. Validation-based correction of generated SPARQL against endpoint schemas has been proposed for federated bioinformatics KGs [37].

Automated loops treat the user as a passive recipient of the final answer. We argue that for domains where correctness cannot be verified mechanically, such as the expert-formulated ecological questions in Section 5.5, whose intended semantics ("philopatric males", "primary dispersers ", "adopted social cub") are not recoverable from schema or data alone, the human must remain the arbiter. Our framework therefore supports amendment loops driven by free-form natural-language feedback; in two real-world examples we observe how different models respond to such feedback and report efficiency (attempts per correct query) alongside accuracy, without claiming a statistically grounded assessment of the loop itself. In our previous work [9], a failed query could only be regenerated from scratch; the explanation and amendment components evaluated here close that gap.

Our contribution is thus not the idea of coupling generation, explanation, and refinement, as SQLucid demonstrates this coupling for SQL and InteracSPARQL for SPARQL, but a multi-model evaluation of whether current LLMs can fill these roles for Cypher without dedicated rule-based machinery: a controlled benchmark with systematically injected errors measures explanation quality, perturbation detection, and false-positive rates (a combination that, to our knowledge, no existing graph-query benchmark provides) while two real-world case studies examine query generation under interactive, user-driven amendment.

3. Framework architecture

To provide a controlled setting for evaluating LLM performance on Cypher-related tasks, and to make graph-structured knowledge more accessible, we developed an interactive framework that allows users to query a given Knowledge Graph using only natural language (NL). At its core is a modular pipeline that translates natural-language user questions into executable queries (Generator), runs them against the underlying graph database (Executor), and returns the results. The same large language model (LLM) is also used to explain the generated queries (Explainer) and to process follow-up instructions for refinement (Amender). Unlike conventional KG-based retrieval systems, which often treat users as passive recipients of generated answers, our framework emphasizes transparency and user control, which enables a conversational interaction with the underlying structured graph data, without requiring users to write `Cypher` queries (see Fig. 1).

This paper builds on our earlier work introduced in Pusch and Conrad [9] by adding explanation and amendment capabilities. These additions create the infrastructure needed for the evaluation studies that follow: we can now measure not only whether LLMs generate correct queries, but also how well they explain them and how they respond to iterative user feedback.

The next subsection introduces this architecture and describes its core components: query generation, explanation, execution, and refinement.

3.1. Framework main components

Query generation

The query generation component translates a user's natural-language question into an executable `Cypher` query. Its primary goal is to preserve semantic intent while conforming to the graph's structure and constraints.

Inputs and Outputs: The component takes the user's question and the Knowledge Graph schema as input. Its output is a cleaned, executable `Cypher` query string.

Implementation Details: The translation is performed using a schema-aware `LangChain` prompt (Appendix A.2) that constrains vocabulary and structure to valid node and edge types. The LLM (e.g., GPT-4, Claude, or DeepSeek) is wrapped in a `LangChain`-compatible class, ensuring consistent behavior and easy model switching. Post-processing removes markdown wrappers, tags like *thinking*, and other extraneous content, resulting in a syntactically correct query.

Query execution

This component connects to a `Neo4j` instance and executes the `Cypher` query returned by the generator. It is responsible for retrieving structured answers from the Knowledge Graph.

Inputs and Outputs: Its input is the cleaned `Cypher` query along with a database connector. The output is a list of structured records representing the query result.

Implementation Details: Execution is handled using `LangChain`'s `Neo4jGraph` object, which provides an abstraction over query submission and result retrieval. Results are returned as a `Python` data-structure, ready for display or further processing.

Query explanation

To improve transparency and user trust, this component generates human-readable explanations of the `Cypher` query. These explanations help users understand how a query works and why a particular result was produced.

Inputs and Outputs: It takes a `Cypher` query as input and returns a structured explanation, including a step-by-step walkthrough, error flagging and a concise summary.

Implementation Details: The explanation is generated using the same LLM infrastructure as the query generation component but with a different prompt (Appendix A.1). The prompt instructs the model to describe the query's purpose and logic in plain language. Additionally, the model is asked to flag issues such as inverted relationships, invalid node types, or implausible graph patterns. This supports interpretability without requiring users to read `Cypher` directly.

Query amendment

The amendment component enables iterative refinement by allowing users to update an existing query using natural language. Rather than regenerating the query from scratch, the system applies targeted modifications based on user feedback.

Inputs and Outputs: Inputs include the existing `Cypher` query, the user's amendment instruction, and the schema. The output is a revised `Cypher` query string.

Implementation Details: Amendments are handled via a dedicated prompt (Appendix A.3) that asks the LLM to edit the current query in accordance with the new instruction while adhering to schema constraints. The LLM's output is parsed and returned for immediate re-execution or explanation. This component enables conversational query correction, allowing users to adjust filters, entity types, or traversal logic without starting over.

3.2. Summary

We have now described the key components of the framework and how they interact to form an end-to-end pipeline. This allows for the

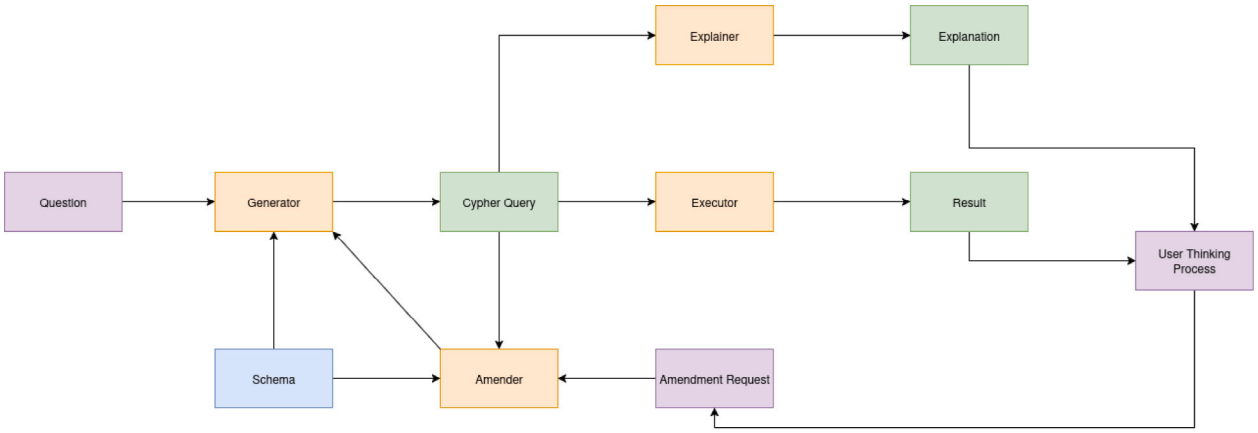


Fig. 1. Architecture diagram; Purple nodes are contributed by the user, orange nodes are modules, the blue one is the graph schema and the green nodes are pipeline outputs.

translation of natural-language questions into executable graph queries, which can be run against a knowledge graph, and the results then be returned to the user. More technically, a large language model generates a `Cypher` query, which is executed via the `Neo4j` driver, and the resulting records are displayed in the user interface. The same model can also provide a step-by-step natural-language explanation of the query or apply an amendment instruction expressed in natural language, without requiring full regeneration.

This process was illustrated in the "graphclust" example above, where a user refines their query to distinguish between a publication and a software package. The interaction demonstrates how the framework supports clarification and allows non-experts to explore graph-structured data without needing to read or write `Cypher`.

With the full framework in place, we first focus on our most detailed quantitative experiment, the explanation component in a controlled benchmark, and then turn to two case studies on real-world KGs.

4. Experiment: How well can LLMs explain `cypher` queries?

To ensure that users can trust and control the query process, they need to understand what a generated `Cypher` query actually does. In this section, we evaluate whether Large Language Models (LLMs) can support that understanding by providing accurate, natural-language explanations and by identifying flawed queries when they occur.

We focus on two main questions:

(a) Can LLMs generate clear and accurate natural language explanations of `Cypher` queries?
(b) Can they detect and flag syntactic or semantic problems in queries?

To isolate these abilities from domain complexity, we use a deliberately simple setting: a synthetic Knowledge Graph about movies. This familiar domain makes it easier to spot errors and ensures that poor performance is not due to obscure content or schema.

For instance, a sample query might be:

```
MATCH (a:Actor)-[:ACTED_IN]->(m:Movie)
WHERE m.release_year > 2000
RETURN a.name, m.title
```

A correct explanation would state that the query retrieves the names of actors and the titles of movies they acted in, limited to movies released after the year 2000. An inaccurate explanation might omit the year constraint. We deliberately introduced errors into these sample queries, such as replacing the relationship name with one that does not make sense in the context, such as `(a:Actor)-[EATS]->(m:Movie)`, or more subtle mistakes, such as flipping the direction of the relationship.

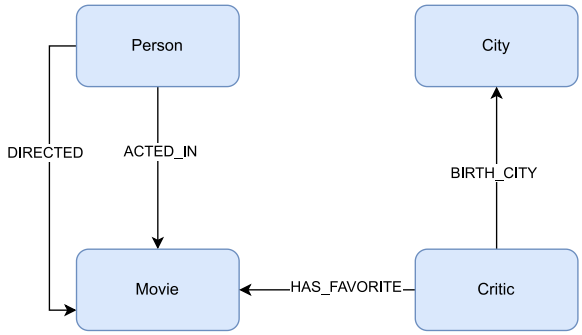


Fig. 2. Schema of the synthetic movie knowledge graph created as basis for the queries in the benchmark dataset. The node types are Person, Movie, Critic and City, the relationships are `DIRECTED`, `ACTED_IN`, `HAS_FAVORITE` and `BIRTH_CITY`.

With this context in mind, we will next describe the benchmark setup, the models tested, and the evaluation results.

4.1. Benchmark dataset

We designed a benchmark of 90 `Cypher` queries (see Appendix C) set in a movie-domain Knowledge Graph (Fig. 2), systematically varying along three dimensions: path lengths, clause types, and error injections. This yields 15 unperturbed and 75 perturbed queries.

As discussed in Section 2.1, existing text-to-Cypher datasets consist of well-formed query–question pairs and lack fine-grained control over query structure; our benchmark was therefore created from scratch to allow perturbation variation and targeted evaluation.

The resulting query set captures a diverse range of structural and semantic patterns. Below, we illustrate each dimension with examples that show how these variations influence query construction and interpretation.

4.1.1. Path length

To vary structural complexity, queries included 1, 2, or 3 hops, that is, traversal paths of increasing length, where each hop corresponds to following a relationship between connected nodes in the graph.

Example:

A 1-hop query might retrieve actors and the movies they acted in:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie) RETURN p.name
    , m.title
```

A 2-hop query could trace from actors to critics via the movie the actor worked in and the critic favored:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)<-[:
    HAS_FAVORITE]-(c:Critic) RETURN p.name, c.name
```

4.1.2. Clause composition

To vary logical and syntactic complexity, we used different Cypher constructs: a baseline MATCH; an OPTIONAL MATCH that retrieves additional data when present; two consecutive MATCH clauses connected by WITH to pass intermediate results; a WHERE filter; and a CASE expression that provides if-else logic.

Example use of CASE:

```
MATCH (p:Person {name: "Alice"})-[:ACTED_IN]->(m:
    Movie)
RETURN p, m,
  CASE WHEN m.release_year < 2010 THEN 'old' ELSE 'new'
    END as movie_age
```

4.1.3. Perturbations

To test whether LLMs would recognize perturbations, we deliberately (i) reversed relationship direction, (ii) introduced implausible relationship labels such as EATS (e.g. `(a:Actor)-[EATS]->(m:Movie)`), (iii) replaced sensible node labels with implausible ones such as Food (e.g. `(a:Actor) -[ACTED_IN]->(m:Food)`), (iv) omitted the node type while giving the variable a misleading name, and (v) introduced clause-specific faults: contradictory WHERE constraints (e.g. a film released both before and after 2020), illogical WHERE constraints such as a release year of −1, or ill-formed value tests such as `movie.title > 2020`.

Example:

Implausible relationship: The relationship EATS is implausible in this context, testing whether the model recognizes semantic inconsistency.

```
MATCH (a:Actor)-[:EATS]->(m:Movie)
RETURN a.name, m.title
```

These variations allow us to assess not just surface-level understanding, but whether LLMs can handle structural, semantic, and logical correctness in Cypher queries.

4.2. LLMs

The LLMs tested were selected to be a good representation of the most commonly used models at the time of the experiment that were also easily accessible via LangChain. This includes two GPT variants, a Claude model, and two DeepSeek models. The 70b DeepSeek model was run locally, all other models were accessed via official APIs. The LLMs and their versions can be found in Table 2.

4.3. Experimental design

We provided the LLM with each query from the benchmark and an instruction prompt (Appendix A.1) to produce a step-by-step explanation and give a brief summary of the query's overall purpose. Note that, unlike in the query-generation examples (Sections 5.4, 5.5), the prompt contained only the query itself, not the graph schema. The LLM was also asked to flag errors, with flipped relationships and ill-formed node or relationship names being specifically mentioned in the prompt. Outputs were then manually evaluated by the first author (L.P.) for explanation correctness of the summary; whether errors, when present, were explicitly identified and no errors were flagged if there were none; and whether the summary omitted the year for queries that specified one. When a query contained an error, the one-sentence summary was counted as correct only if it described the query as written; summaries that described the query's presumable intent (e.g., explaining a reversed relationship as if it pointed in the plausible direction) were counted as incorrect even when the model separately flagged the error, since the one-sentence summary should be reliable for users who read it in isolation. The example given in Fig. 5 illustrates how the step-by-step summary and the one-sentence-summary were checked for completeness. Table 18 reports post-hoc token counts for this experiment. All counts in this paper were computed with a single tokenizer (o200k_base) applied uniformly across models, so they are comparable across models but serve as a proxy for prompt and response length rather than each provider's billable count. Output counts cover the delivered response only: hidden reasoning tokens of reasoning models were not logged by our pipeline, and visible reasoning traces of locally run models were stripped before counting, so output counts for reasoning models constitute lower bounds.

4.4. Statistical analysis

We computed the mean accuracies and associated 95-percent Wilson confidence intervals of the one-sentence summaries, the detection of injected perturbations, and the avoidance of false positives. For one-sentence summaries, a prediction was counted as correct when the summary provided an accurate and complete description of the query. For problem detection, a success was recorded when the model explicitly identified the perturbation in the queries that contained one. For false positives, a success required that the model did not flag any error in queries without perturbations. A flag was any assertion that something was wrong with the query, including cases in which the proposed correction left the query unchanged; criticism of naming conventions in the schema was not counted as a flag. Each of the 90 queries was submitted to each of the five models, so we arrive at the following numbers: one-sentence-summary accuracy is computed over 450 observations (90 per model), perturbation detection over 375 (75 per model), and false-positive avoidance over 75 (15 per model).

We compared the performances for each of these outcomes of each possible pair of LLMs, out of the five LLMs considered, implementing the McNemar's test for paired binary outcomes in Python. For each pair, we ran the test on a 2×2 contingency table of query-level correctness (correct vs. incorrect) and based the test exclusively on the discordant pairs, that is, queries answered correctly by one model and incorrectly by the other. The exact two-sided p-value was obtained from a binomial test on the number of discordant pairs, under the null hypothesis that both models have equal probability of being correct for any query ($p = 0.5$). To account for multiple pairwise comparisons among models, the p-values were adjusted using the Holm step-down procedure to control the family-wise error rate at a nominal level of 0.05. In addition, we recorded the direction of the difference for each pair of models by comparing the counts of queries where only the row model was correct vs. only the column model was correct, so that adjusted p-values could be interpreted together with the sign of the performance difference.

To examine how the accuracy of one sentence summaries varied across multiple dimensions (as described in Sections 4.1.1, 4.1.2, 4.1.3), we used the R package spaMM [38] to fit a series of Generalized Linear Mixed-effects Models (GLMMs) with a binomial link. We generated two alternative response variables representing accuracy: the first

Table 1
Categorization of the amendments used in the MaRDI and Hyena experiments. Each entry gives the number of times the query was correct after the amendment, out of the number of amendments in that category. Amendments addressing several issues are counted in each applicable category, so the counts sum to more than the number of amendments; the *sole correction* column restricts to amendments in which that category was the only correction.

Category	Query correct after amendment				Median words
	Hyena	Mardi	Total	Sole correction	
Direction flip	0/9	8/14	8/23	6/12	15
Schema correction	1/10	–	1/10	1/2	38
Filter/condition adjustment	9/26	7/18	16/44	15/29	18
Structural rewrite	31/123	6/31	37/154	33/120	25
Syntax repair	1/8	1/2	2/10	0/4	16
Output formatting	1/4	0/5	1/9	0/4	19

Table 2
LLMs used in explanation trials in Section 4.

llm	Version
deepseek-r1:70b	0c1615a8ca32
o1-preview-2024-09-12	2024-09-12
o3-mini-2025-01-31	2025-01-31
claude-3-7-sonnet-20250219	20250219
deepseek-reasoner-api	date: 05.03.2025

considered a strict correctness criterion, and the second considered a relaxed criterion that did not treat a missing year in the one-sentence summary as an error if that year appeared in the step-by-step explanation. In both specifications, the model included LLM, hop count, clause type, and perturbation type as fixed-effect predictors, along with a random intercept for the query. For each GLMM, we conducted Tukey-adjusted post-hoc pairwise comparisons for all multi-level predictors using the function `glht()` from the package `multcomp` [39]. To visualize the resulting effects, we computed partial dependence estimates and corresponding confidence intervals using the function `pdep_effects(..., intervals = ''fixefVar'')` and plotted those together with compact letter displays indicating groups not significantly different from each others.

4.5. Results

4.5.1. Accuracy of one-sentence summaries

Effect of the LLM. The accuracy of the LLMs on correct and complete one-sentence explanation summaries (Table 3) exceeded 70% for o1-preview, deepseek-reasoner-api, and o3-mini, with o1-preview performing the best. Deepseek-r1:70b reached 66%, and claude 3.7 sonnet was the lowest at 52%. Trivially wrong statements (e.g., "X is misspelled" followed by a suggestion to change the word to one of identical spelling) and stylistic complaints about node and relationship names were ignored when scoring summary correctness, because they are unlikely to mislead readers. A Holm-corrected pairwise McNemar test (Table 4) showed that deepseek-reasoner-api, o1-preview, and o3-mini performed significantly better than claude 3.7 sonnet on one-sentence summaries. This result was confirmed by a GLMM fit controlling for other sources of variation (Fig. 3a & b). The pattern held irrespective of whether or not year omissions were counted as errors, but deepseek-reasoner-api significantly outperformed deepseek-r1:70b only in the latter case. A detailed breakdown of disagreement counts is provided in Table 15.

Effect of query features. Our query set varied along three axes: clause type, perturbation, and hop count, each potentially affecting explanation performance (Fig. 4). For clause type ((Fig. 4(a))), `CASE` statements stand out as problematic; however, `CASE` queries uniformly include a year and some LLMs struggled to include missing years in the one-sentence summaries, which likely confounds this result. The GLMM results show that clause type no longer significantly influenced the accuracy of one-sentence summary once year mistakes were overlooked (Fig. 3h).

With respect to perturbations, only *contradictory `WHERE` values* and *flip relationship* induce complete failures in some LLMs (Fig. 4(b)), resulting in a significantly worse accuracy of one-sentence summary when either of these two perturbations are present, compared to no perturbation (Fig. 3e & f). More specifically, Claude failed on both for all instances, and o1-preview only failed on all *contradictory `WHERE` values*. Note that the latter comprises only three queries because it applies solely to `WHERE` clauses. For other types of perturbation, no cross-model pattern emerged consistently.

Regarding hop count, 1-hop queries were processed more accurately by LLMs than 2- or 3-hop queries (Fig. 4(c)). The GLMMs shows that, as for clause type, the hop count no longer significantly influenced the accuracy of one-sentence summary once year mistakes were overlooked (Fig. 3d). When `OPTIONAL MATCH` or `WITH` were used to introduce an extra hop ("(+1)" category) to the base hop count, the accuracy surprisingly improved, although such queries were too few to test the resulting differences using a GLMM.

4.5.2. Ability to flag faults in perturbed queries

Effect of the LLM. Among perturbed queries, o1-preview, claude 3.7 sonnet, and deepseek-reasoner-api exceeded 80%, o3-mini scored 77%, and deepseek-r1:70b scored 68% when flagging them as faulty (Table 3). A Holm-corrected pairwise McNemar test (Table 5) showed that deepseek-reasoner-api and o1-preview performed significantly better than deepseek-r1:70b. A detailed breakdown of disagreement counts is provided in Table 16.

Sensitivity to perturbation type in perturbation flagging. Flipped relationships, nonsense node labels and nonsense relationship labels were specifically mentioned in the prompt. All models flagged contradictory `WHERE` clauses, nonsense node- and relationship names, and type mismatches (Fig. 6). Many struggled with the combination of missing node type and misleading node names, where only claude exceeded 50% at 79%. For flipped relationship directions, o1-preview and deepseek-reasoner-api were flawless, whereas the others made occasional to frequent mistakes.

4.5.3. Avoidance of false positives

Effect of LLM. Deepseek-r1:70b flagged a correct query as defective in 47% of cases, claude 3.7 sonnet in 33%, o1-preview in 7%, and o3-mini and deepseek-reasoner-api never did so (Table 3). A Holm-corrected pairwise McNemar test (Table 6) showed no statistically significant differences between the models. Given the small sample (n = 15) and limited power, we interpret differences in this task primarily based on descriptive statistics. A detailed breakdown of disagreement counts is provided in Table 17. We also examined whether the explanations of the 15 unperturbed queries introduced any elements that were not supported by the query, such as non-existing node labels or relationship types. No instance of this occurred. The deviations that did occur were of two kinds: criticism of schema design and assertions of an error accompanied by a correction identical to the original query. Neither introduces information absent from the query.

Table 3
Accuracy with 95% Wilson CI for correct one-sentence summary generation, problem detection, and avoidance of false positives (share of unperturbed queries in which the model correctly flagged no error).

LLM	One-sentence summary				Problem detection				Avoidance of false positives			
	n	Correct	acc.	95% CI	*n*	Correct	acc.	95% CI	*n*	Correct	acc.	95% CI
claude 3.7 sonnet	90	47	0.522	[0.420, 0.622]	75	64	0.853	[0.756, 0.916]	15	10	0.666	[0.417, 0.848]
deepseek-reasoner-api	90	66	0.733	[0.634, 0.814]	75	67	0.893	[0.803, 0.945]	15	15	1.000	[0.796, 1.000]
deepseek-r1:70b	90	60	0.666	[0.564, 0.755]	75	51	0.680	[0.568, 0.775]	15	8	0.533	[0.301, 0.752]
o1-preview	90	69	0.766	[0.669, 0.842]	75	66	0.880	[0.787, 0.936]	15	14	0.933	[0.702, 0.988]
o3-mini	90	64	0.711	[0.610, 0.795]	75	58	0.773	[0.667, 0.853]	15	15	1.000	[0.796, 1.000]

Table 4
Pairwise McNemar tests comparing LLM accuracy of one-sentence summary generated. P-values are Holm corrected. Positive values indicate better performance of the model in the row over the model in the column.

	claude 3.7 sonnet	deepseek-reasoner-api	deepseek-r1:70b	o1-preview	o3-mini
claude 3.7 sonnet	–	**−0.0119**	−0.2061	**−0.0000**	**−0.0121**
deepseek-reasoner-api	**0.0119**	–	1.0000	−1.0000	1.0000
deepseek-r1:70b	0.2061	−1.0000	–	−0.5588	−1.0000
o1-preview	**0.0000**	1.0000	0.5588	–	1.0000
o3-mini	**0.0121**	−1.0000	1.0000	−1.0000	–

Table 5
Pairwise McNemar tests comparing LLMs on error detection capabilities. P-values are Holm corrected. Positive values indicate better performance of the model in the row over the model in the column.

	claude 3.7 sonnet	deepseek-reasoner-api	deepseek-r1:70b	o1-preview	o3-mini
claude 3.7 sonnet	–	−1.0000	0.0504	−1.0000	1.0000
deepseek-reasoner-api	1.0000	–	**0.0003**	1.0000	0.0312
deepseek-r1:70b	−0.0504	**−0.0003**	–	**−0.0005**	−0.1953
o1-preview	1.0000	−1.0000	**0.0005**	–	0.0504
o3-mini	−1.0000	−0.0312	0.1953	−0.0504	–

Table 6
Pairwise McNemar tests comparing LLMs on false positives. P-values are Holm corrected. Positive values indicate better performance (=less false positives) of the model in the row over the model in the column.

	claude 3.7 sonnet	deepseek-reasoner-api	deepseek-r1:70b	o1-preview	o3-mini
claude 3.7 sonnet	–	−0.3750	1.0000	−0.5000	−0.3750
deepseek-reasoner-api	0.3750	–	0.1406	1.0000	–
deepseek-r1:70b	−1.0000	−0.1406	–	−0.2188	−0.1406
o1-preview	0.5000	−1.0000	0.2188	–	−1.0000
o3-mini	0.3750	–	0.1406	1.0000	–

4.6. Discussion

Across the 90 `Cypher` queries in our benchmark, all LLMs produced correct and complete one-sentence summaries for more than half of the cases. Only three models, o1-preview, deepseek-reasoner-api, and o3-mini, exceeded the 70% accuracy mark, which is encouraging but still insufficient for robust real-world use. The gap between the best performer (o1-preview) and the worst (claude) was nearly 20%, highlighting substantial variation in explanation quality across models. These differences were confirmed by the generalized linear mixed model analysis, which showed significant overall accuracy differences across LLMs, with claude 3.7 sonnet performing reliably worse than deepseek-reasoner-api, o1-preview, and o3-mini under both strict and relaxed evaluation criteria, and deepseek-reasoner-api further outperforming deepseek-r1:70b when year omissions are disregarded.

One weakness recurred throughout the evaluation: omission of years from the one-sentence summaries. For deepseek-reasoner-api, more than four out of five incorrect summaries were wrong for this reason; for o1-preview and o3-mini, the omission accounted for more than half of their errors. This suggests that the summary-generation objective may overweight brevity relative to faithfulness. Explicitly requiring the retention of temporal information during instruction tuning could address this issue and improve performance. The mixed-effects analysis reinforces this interpretation: one-hop queries only outperform three-hop queries when year omissions are counted as errors, indicating that the apparent structural difficulty is largely driven by temporal information loss rather than query length alone.

When detecting flawed queries, models varied in how they balanced sensitivity and specificity. Claude, for example, exhibited high sensitivity (identified many injected defects) but low specificity (raised a relatively high number of false alarms) suggesting a trade-off between the two. In contrast, o1-preview and deepseek-reasoner-api combined high defect-detection rates (above 85%) with low false-positive rates (below 10%), showing fewer signs of this trade-off. o3-mini achieved a slightly lower detection rate (around 77%) but did not produce any false positives. These patterns provide a useful basis for choosing models according to the preferred balance between the two metrics.

Across models, there is no simple one-to-one link between strong perturbation detection and low false positive rates. Most systems that rank highly in detection accuracy also avoid many false positives, but claude 3.7 sonnet breaks this pattern: despite being among the strongest at identifying actual perturbations, its false positive rate was only midrange. Deepseek-reasoner-api and o1-preview illustrate the cleaner relationship, pairing high detection accuracy with near-perfect avoidance of false positives, with o3-mini not far behind. In contrast, deepseek-r1:70b performed worse on both fronts. Overall, while good detection often aligned with better false positive avoidance,

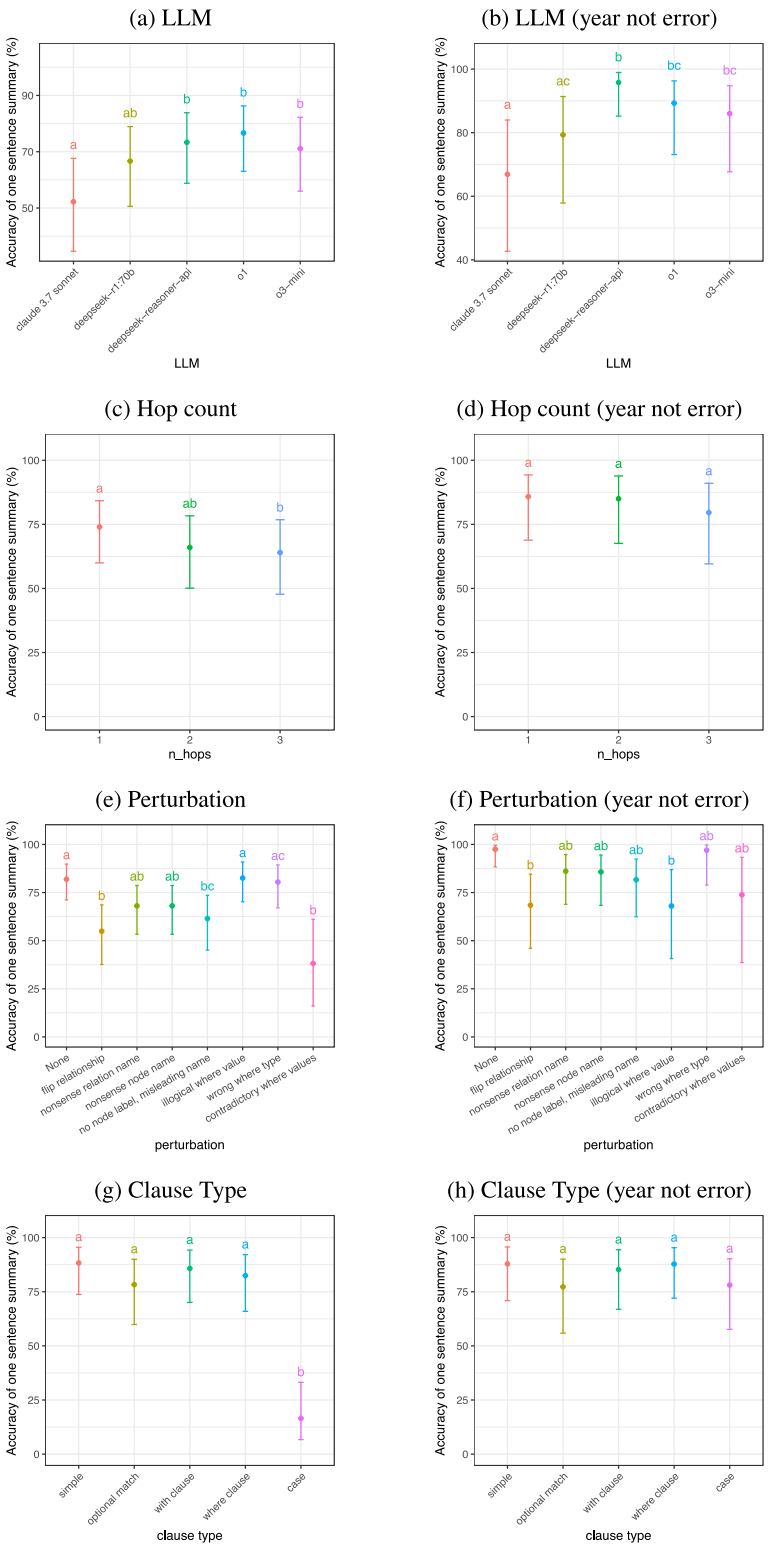
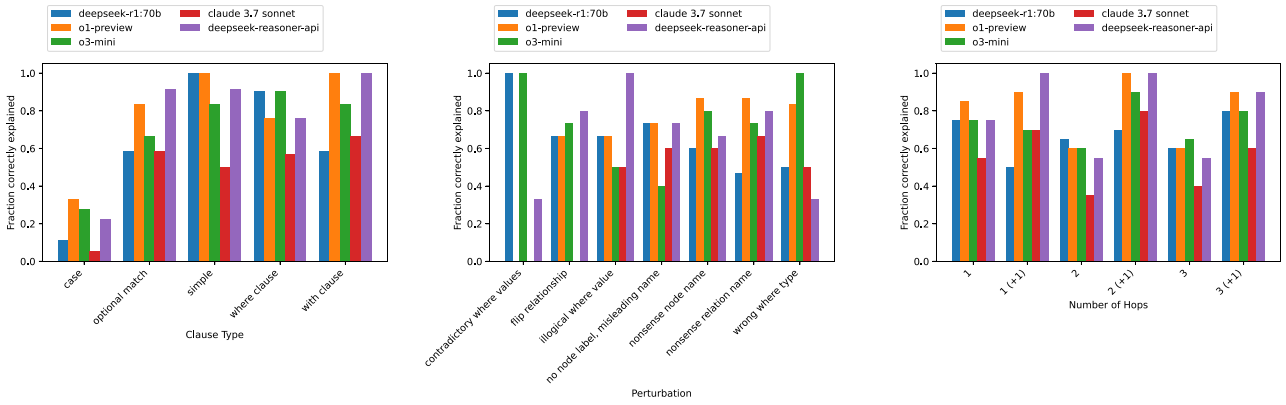


Fig. 3. Effects on explanation accuracy. Left column: year mismatches counted as errors (strict criterion). Right column: year mismatches not counted as errors (relaxed criterion). These figures were constructed using an additive GLM. If items share a letter, they are not significantly different from each other.

the differences among models - especially claude 3.7 sonnet's mix of strong detection and moderate false positive control - show that these abilities do not always improve together. Consistent with these observations, the GLMM shows that under strict evaluation, flipped and contradictory relationships significantly reduce accuracy compared to no perturbation or incorrect WHERE clauses, whereas under the relaxed criterion these differences largely disappear except for the no-perturbation baseline.

Analyzing how query features shape performance reveals a small set of consistent patterns. The apparent weakness on `CASE` clauses is largely a temporality confound: all `CASE` queries contained years, and missing-year omissions account for a substantial share of errors.



(a) By clause (b) By perturbation (c) By number of hops

Fig. 4. How correctness of one-sentence summaries is influenced by query features.

```
Example Query:
MATCH (p:Person name: "Alice")-[:ACTED_IN]->(m:Movie) RETURN p, m
Summary:
1. The query finds a node labeled Person with the property name set to "Alice".
2. It then follows outgoing ACTED_IN relationships from this Person node to find
   connected nodes labeled Movie.
3. Finally, it returns both the Person and the associated Movie nodes.

Overall, this query retrieves the person "Alice" and the movies in which she has acted.
Everything in the query, including the relationship direction and names, is appropriate
and makes sense.
```

Fig. 5. Colored example to illustrate how a generated step-by-step summary a generated one-sentence summary can be checked for completeness.

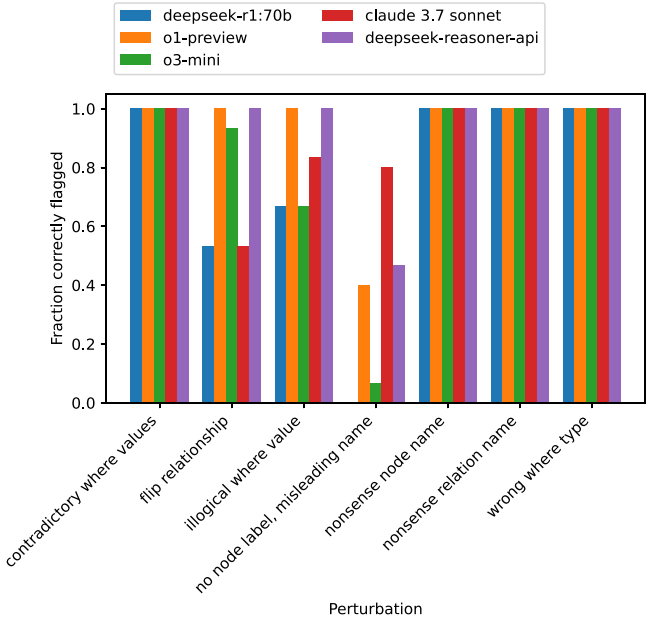


Fig. 6. For each perturbation category (definitions in Section 4.1), the bars show the fraction of queries in which each evaluated LLM correctly signaled that something was wrong.

Notably, the relaxed criterion excused a missing year only when it appeared in the step-by-step explanation, so these omissions occurred in cases where the model had demonstrably registered the constraint. The mixed-effects results align with this explanation: CASE queries perform significantly worse than all other clause complexities only when year omissions are treated as errors, further confirming that temporality rather than clause structure drives this effect. The two perturbations that occasionally produced complete failures, contradictory `WHERE` values and flipped relationship directions, point to specific blind spots (logical consistency checks and small changes with large impact), although flipped relationship directions were specifically mentioned in the prompt (Appendix A.1). Given the small size of the contradictory-`WHERE` set (n = 3), we treat claims here as provisional, especially given that other queries with similar logical consistency checks, such as illogical `WHERE` values, did not show this pattern. Structurally, base 1-hop queries were easier than 2–3 hops, whereas "(+1)" variants created by `OPTIONAL MATCH` or `WITH` were unexpectedly easier than their bases, with the cause being uncertain and meriting follow-up investigation. This finding complicates the structural-difficulty interpretation, since longer paths should not become easier if hop count alone drove performance. These patterns were consistent with the effects estimated in our generalized linear mixed models, which showed significant differences across LLMs, hop counts, perturbation types, and clause complexities.

These observations need to be interpreted in light of certain limitations. The scope of this experiment is intentionally narrow, focusing on 90 handcrafted queries in a movie-style Knowledge Graph. While this controlled setting is well-suited for isolating specific behaviors and testing fundamental capabilities, real-world graph queries, especially in domains such as mathematics or biology, can be more complex and embedded in richer schema contexts. Because the explanation prompt provides the query without the graph schema, the perturbation-detection task measures plausibility judgment based on general domain knowledge rather than conformance checking against a specific schema. In domains where multiple plausible modelings coexist, a plausible-looking query can still mismatch the actual schema;

evaluating schema-grounded explanation, in which the schema is supplied alongside the query, is a natural extension. The absence of the schema did not, however, lead models to invent one in unperturbed queries. We further note that several of the injected fault classes (reversed relationships, invalid labels, and omitted node types) could in principle be detected mechanically by checking against the schema. Our aim, however, was to measure whether LLMs themselves can identify such faults, since this determines how far an explanation component can be trusted on its own. In practice, combining static validation with LLM-based explanation is a good architecture choice, and our results indicate which fault classes the LLM layer can currently be relied upon for. Furthermore, the scoring process relied on human judgement, introducing some subjectivity in determining what constitutes a "complete" and correct summary, which is unavoidable given the tool's interactive design. Finally, because LLMs evolve rapidly, the snapshot presented here does not offer any guarantees about future capabilities, and subsequent models may surpass the performance observed in this study. Despite these constraints, the evaluation highlights concrete strengths and weaknesses that can directly inform the explanation framework's future design goals. The following two experiments examine different aspects of the framework, using examples from mathematical (MaRDI) and biological (a population of spotted hyenas) domains to illustrate different LLM's capabilities in handling query generation on diverse Knowledge Graphs.

5. Real-life examples: Question answering on biological and mathematical knowledge graphs

Having evaluated explanation and fault-detection capabilities in Section 4, we now turn to a different LLM skill: generating correct `Cypher` queries from natural-language questions. To assess how this capability varies across models and domains, we conducted two case studies on real-world Knowledge Graphs with distinct characteristics. The first uses a software-centric slice of the MaRDI Knowledge Graph (including CRAN packages, Zenodo software, cited papers, and authors with zbMATH IDs). The second uses a newly constructed Knowledge Graph derived from long-term field data on a free-ranging population of spotted hyenas in the Ngorongoro Crater, Tanzania (see https://hyena-project.com), with questions formulated by project experts. Across both experiments, we tested 14 `LangChain`-compatible LLMs under identical constraints and kept the number of questions deliberately small to enable systematic comparison across many models rather than exhaustive coverage of each graph. Accordingly, the results are not intended as a full benchmark ranking; instead, they should be interpreted as comparative performance under controlled conditions, highlighting failure modes and model-specific difficulties. Our central research question in both cases is whether an LLM can produce a correct query after an initial `Cypher` generation step and at most two user-submitted amendments, and whether particular questions pose more difficulties than others.

5.1. LLMs

We aimed to include a broad selection of `LangChain`-compatible LLMs, including several open source variants, some GPT models and a Claude model. This experiment evaluates different model versions than the explanation experiment discussed earlier. Claude 3.7 sonnet, deepseek-reasoner-api, gpt 5.2, o1, o3, o3-mini and o4-mini were accessed via official APIs, the remaining models were run locally. Claude 3.7 Sonnet and deepseek-reasoner-api, in the version used here, were no longer accessible when a set of additional questions was added. They were therefore evaluated on only a subset of the questions and their accuracies are not strictly comparable with those of the other models.

5.2. Statistical analysis

For each LLM, we computed the mean accuracies and associated 95-percent Wilson confidence intervals of the production of correct `Cypher` queries on the first attempt and within the allowed sequence of one initial try plus up to two amendments. We then calculated the same accuracy measures on a per-question basis. This analysis was performed across ten natural-language questions each and fourteen LLMs, with each interval computed against that row's own number of trials.

5.3. Experimental design

For each question in the benchmark set, we provided the question answering pipeline described in Section 3 with the question, the graph schema and an instruction prompt (Appendices A.2 or A.4). If the generated query was not judged to be correct by the researcher, up to two natural language amendments were supplied for the framework to improve the query, along with an amendment prompt (Appendix A.3). If the query was still not deemed to be correct after this, it was counted as a failure. This workflow was executed for each LLM mentioned in Section 5.1. Table 19 reports post-hoc token counts per attempt for both experiments. Generations that produced no storable output are not counted.

5.4. Real-life example 1: Question answering on the MaRDI knowledge graph

To assess real-life performance on a software-oriented Knowledge Graph, we tested 14 LLMs on ten natural-language questions over a software-centric slice of the MaRDI Knowledge Graph (CRAN packages, Zenodo software, cited papers, and authors with zbMATH IDs). The underlying subgraph connects software packages, publications, and authors through citation and authorship relationships, requiring the models to reason over multi-step links between research outputs and contributors. The question set is intentionally small to support consistent cross-model comparison and to highlight typical error patterns rather than to provide exhaustive coverage. We evaluate whether models can arrive at a correct `Cypher` query after the initial generation and up to two user-submitted amendments, and we analyze which question types lead to systematic difficulties.

For instance, a sample interaction might be:

```
Natural Language Question:
  Which are the ten authors that created the most
  software packages?

Initially Generated Query:

    MATCH (a:Author)-[:HAS_AUTHOR]->(s:
SoftwarePackage)
    RETURN a.name, COUNT(s) AS packageCount ORDER
BY packageCount DESC LIMIT 10


User Amendment Request:
  The has_author relationship is the wrong way around.

Amended Query:

    MATCH (s:SoftwarePackage)-[:HAS_AUTHOR]->(a:
Author)
    RETURN a.name, COUNT(s) AS packageCount ORDER
BY packageCount DESC LIMIT 10
```

Table 7
LLMs used in MaRDI and hyena experiments.

LLM	Number of parameters	Version/hash
claude-3-7-sonnet-20250219	?	20250219
deepseek-r1	70b	0c1615a8ca32
deepseek-reasoner-api	?	date: 21.03.2025
exaone-deep	32b	a9422f9a5071
gemma3	27b	a418f5838eaf
gpt-5.2-2025-12-11	?	2025-12-11
llama3.3	70b	a6eb4748fd29
nemotron	70b	2262f047a28a
o1-2024-12-17	?	2024-12-17
o3-2025-04-16	?	2025-04-16
o3-mini-2025-01-31	?	2025-01-31
o4-mini-2025-04-16	?	2025-04-16
phi4	14b	ac896e5b8b34
qwq	32b	009cb3f08d74

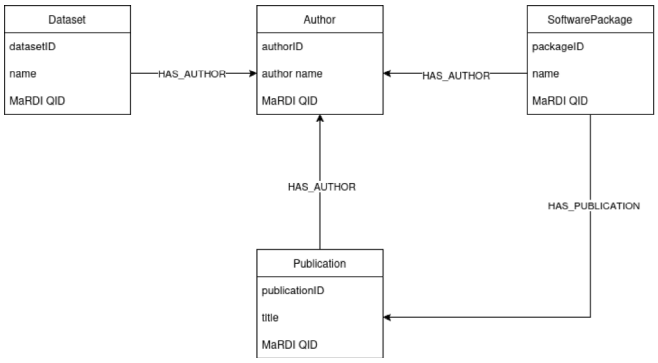


Fig. 7. Data Model for the MaRDI KG subgraph.

5.4.1. Data

The data for this analysis consisted of a subset of the MaRDI Knowledge Graph and a custom set of natural-language questions based on the graph.

Knowledge graph. We created a subgraph of the MaRDI Knowledge Graph [10], restricted to software packages, authors, and publications (Fig. 7). As MaRDI is natively a Wikibase/RDF graph exposed via a public SPARQL endpoint, we extracted the relevant data with SPARQL and loaded it into Neo4j as a labeled property graph. We extracted every CRAN package that cites a paper with a zbMATH de-number [40,41], as well as the authors of those papers. We then added all developers who both maintain CRAN packages and hold a zbMATH author identifier. Finally, the subgraph was supplemented with every Zenodo software whose authors likewise possess a zbMATH author id [42]. The SPARQL queries used for these selections can be found in Appendix B.

Questions. We designed ten benchmark questions (see Appendix D) based on the graph schema of our MaRDI subgraph. Each can be translated into Cypher using various clauses such as DISTINCT, LIMIT, or WHERE. These clauses served only as guidelines: any query that returned the expected result was considered correct, regardless of the constructs used. Although this could in principle allow an incorrect query to match the expected output by coincidence, this is unlikely because the expected results are typically highly specific. An example question, which can be answered using the LIMIT clause, is *"Which are the ten authors that created the most software packages?"*.

5.4.2. Results

We summarized the number of correct answers per model and tracked how many attempts the models needed for their correct answers in Fig. 8. The LLMs o3-mini and GPT 5.2 were the only ones that solved all ten tasks; qwq, gemma3, o4-mini, o1 and o3 followed with nine out of ten successfully solved tasks. Every other model fell between six to eight out of ten (or out of nine, for claude 3.7 sonnet and deepseek-reasoner-api). The majority succeeded on the first try for the majority of their questions; only gemma3 more often required two attempts than one attempt, and a third attempt was seen only for gemma3, GPT 5.2, nemotron, o1 and phi4. If deepseek-reasoner-api, o4-mini and o3 were successful, it was always on the first try. The number of correct answers on the first try and within first tries as well as the accuracy and Wilson 95% CI are listed in Table 8.

Fig. 9 and Table 9 show the per-question accuracy (14 LLMs = maximum score 14; one question only evaluated on 12 LLMs), with the latter also containing total counts and CIs. Four tasks were successfully solved by every LLM (the count-based query, the "simple" query, the intersection query, and the NOT query), while the "shared authors between packages" task proved hardest, with just three correct solutions, and none of them on the first try.

5.5. Real-life example 2: Question answering on a Hyena knowledge graph

To determine whether the performance obtained for the KG used in the previous section remains similar for a real-world Knowledge Graph from a substantially different domain, we created a Knowledge Graph based on data from a long-term study of a free-ranging population of spotted hyenas inhabiting the Ngorongoro Crater in Tanzania (see https://hyena-project.com). We examined the performance of our approach for ten real-world questions about survival, reproduction, kinship and monitoring, which were directly formulated by experts involved in the Ngorongoro Hyena Project. We executed the query generation framework with 14 LangChain-compatible LLMs (see Table 7). As in Section 5.4, we kept the number of questions low to allow systematic comparison across many models rather than exhaustive coverage of the graph. This setup does not identify the best-performing LLMs but highlights which ones fail or struggle under consistent conditions. The main research question we focus on is, again, whether the LLMs are able to produce a correct query after an initial Cypher query generation step and a maximum of two user-submitted amendments to the query. We also analyze the extent to which LLMs have more difficulties with certain questions.

A sample interaction might be:

```
Natural Language Question:
  What is the proportion of cubs that were sired by
  philopatric males?

Initially Generated Query:

    MATCH (cub:Hyena)-[:HAS_FATHER]->(dad:Hyena)
    OPTIONAL MATCH (dad)-[:BIRTH_CLAN]->(bc:Clan)
    OPTIONAL MATCH (dad)-[:CURRENT_CLAN]->(cc:Clan
)
    WITH COUNT(cub) AS totalCubs,
    SUM(CASE WHEN dad.sex = 'male' AND bc = cc THEN
 1 ELSE 0 END) AS philopatCubs
    RETURN philopatCubs * 1.0 / totalCubs AS
proportion


User Amendment Request:
  With how this query is made, it is possible that the
   dad left, had the cub and came back.
  Please check instead that he never changed his clan.

  Amended Query:
```

```
    MATCH (cub:Hyena)-[:HAS_FATHER]->(dad:Hyena)
    OPTIONAL MATCH (dad)-[:BIRTH_CLAN]->(clan:Clan
)<-[:CURRENT_CLAN]-(dad)
    WITH COUNT(cub) AS totalCubs,
    COUNT
        (CASE WHEN dad.sex = 'male' AND clan IS NOT
 NULL
              AND NOT (dad)-[:CHANGED_CLAN]->() THEN
1 END)
        AS philopatricCubs
    RETURN toFloat(philopatricCubs) / totalCubs AS
 proportion
```

5.5.1. Data

This experiment is based on a knowledge graph constructed from long-term ecological observation data and a corresponding set of domain-expert natural-language questions.

Knowledge graph. We created a KG based on real observational data collected and curated by members of the Ngorongoro Hyena Project. To obtain a structurally meaningful yet manageable subset (see Fig. 10), we focused on different kinship relationships between hyenas, such as parents, social mothers (since female spotted hyenas sometimes adopt a cub that is not their genetic offspring [43]) and litter mates (mothers produce between one and three cubs per delivery [44]). Additionally, we integrated information about the birth clan and clans the hyenas

Table 8
First-attempt and within-three-attempts accuracy with 95% Wilson CI on the ten natural language questions for the MaRDI case. n is the number of questions the LLM was evaluated on: Claude 3.7 Sonnet and deepseek-reasoner-api were no longer accessible when the tenth question was added and are therefore scored out of nine. Rows shaded in orange are the ones scored against a reduced n.

LLM	First try				Within 3 tries			
	Correct	n	Accuracy	95% CI	Correct	n	Accuracy	95% CI
claude 3.7 sonnet	6	9	0.6667	[0.354, 0.879]	7	9	0.7778	[0.453, 0.937]
deepseek-r1:70b	5	10	0.5000	[0.237, 0.763]	6	10	0.6000	[0.313, 0.832]
deepseek-reasoner-api	7	9	0.7778	[0.453, 0.937]	7	9	0.7778	[0.453, 0.937]
exaone-deep:32b	6	10	0.6000	[0.313, 0.832]	7	10	0.7000	[0.397, 0.892]
gemma3:27b	3	10	0.3000	[0.108, 0.603]	9	10	0.9000	[0.596, 0.982]
gpt 5.2	9	10	0.9000	[0.596, 0.982]	10	10	1.0000	[0.722, 1.000]
llama3.3:70b	5	10	0.5000	[0.237, 0.763]	8	10	0.8000	[0.490, 0.943]
nemotron:70b	5	10	0.5000	[0.237, 0.763]	8	10	0.8000	[0.490, 0.943]
o1	8	10	0.8000	[0.490, 0.943]	9	10	0.9000	[0.596, 0.982]
o3	9	10	0.9000	[0.596, 0.982]	9	10	0.9000	[0.596, 0.982]
o3-mini	9	10	0.9000	[0.596, 0.982]	10	10	1.0000	[0.722, 1.000]
o4-mini	9	10	0.9000	[0.596, 0.982]	9	10	0.9000	[0.596, 0.982]
phi4:14b	6	10	0.6000	[0.313, 0.832]	7	10	0.7000	[0.397, 0.892]
qwq:32b	8	10	0.8000	[0.490, 0.943]	9	10	0.9000	[0.596, 0.982]

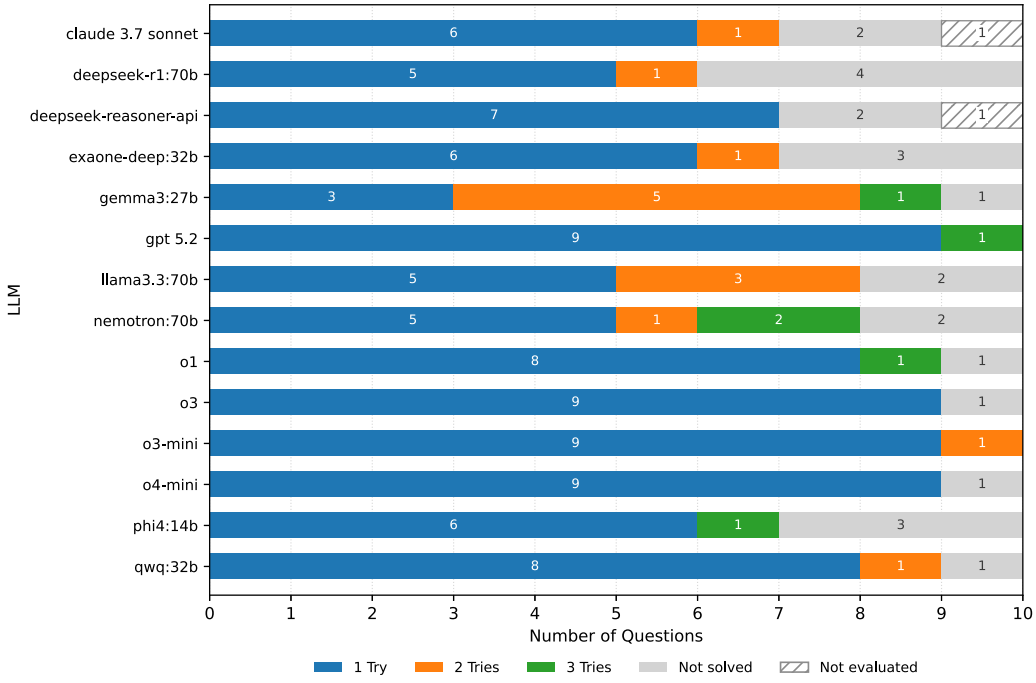


Fig. 8. For each LLM, the colored part of the bar indicates the share of the ten MaRDI questions eventually answered correctly, partitioned by the number of attempts (first, second, or third) needed to reach the correct answer (a maximum of three tries was allowed). The light gray segment counts the questions that the LLM attempted but never answered correctly, and the hatched segment the questions it could not be evaluated on: Claude 3.7 Sonnet and deepseek-reasoner-api in the previously used version were no longer accessible when the tenth question was added.

Table 9
First-attempt and within-three-attempts accuracy for each natural language question in the MaRDI case with 95% Wilson CI. n is the number of LLMs evaluated on the question; the last question was added after two LLMs had become inaccessible and is therefore scored out of 12. Rows shaded in orange are the ones scored against a reduced *n*.

Question	First try				Within 3 tries			
	Correct	*n*	Accuracy	95% CI	Correct	*n*	Accuracy	95% CI
Get me all authors and, if present, any datasets they are tied to.	9	14	0.642857	[0.388, 0.837]	12	14	0.857143	[0.601, 0.960]
Give me all people that created at least 5 datasets.	9	14	0.642857	[0.388, 0.837]	12	14	0.857143	[0.601, 0.960]
How many datasets did Rob Hyndman create?	13	14	0.928571	[0.685, 0.987]	14	14	1.000000	[0.785, 1.000]
Which are the ten authors that created the most software packages?	8	14	0.571429	[0.326, 0.786]	9	14	0.642857	[0.388, 0.837]
Which authors authored publications that contain the substring "Pareto"?	7	14	0.500000	[0.268, 0.732]	13	14	0.928571	[0.685, 0.987]
Which authors does the software package graphclust have?	14	14	1.000000	[0.785, 1.000]	14	14	1.000000	[0.785, 1.000]
Which authors worked on both the dataset "Bitcoin Dataset with Missing Values" and "Rideshare Dataset without Missing Values"?	14	14	1.000000	[0.785, 1.000]	14	14	1.000000	[0.785, 1.000]
Which software packages have no authors?	12	14	0.857143	[0.601, 0.960]	14	14	1.000000	[0.785, 1.000]
Which software packages share the same authors?	0	14	0.000000	[0.000, 0.215]	3	14	0.214286	[0.076, 0.476]
Which authors are credited on a software package's publication without being authors of the package itself?	9	12	0.750000	[0.468, 0.911]	10	12	0.833333	[0.552, 0.953]

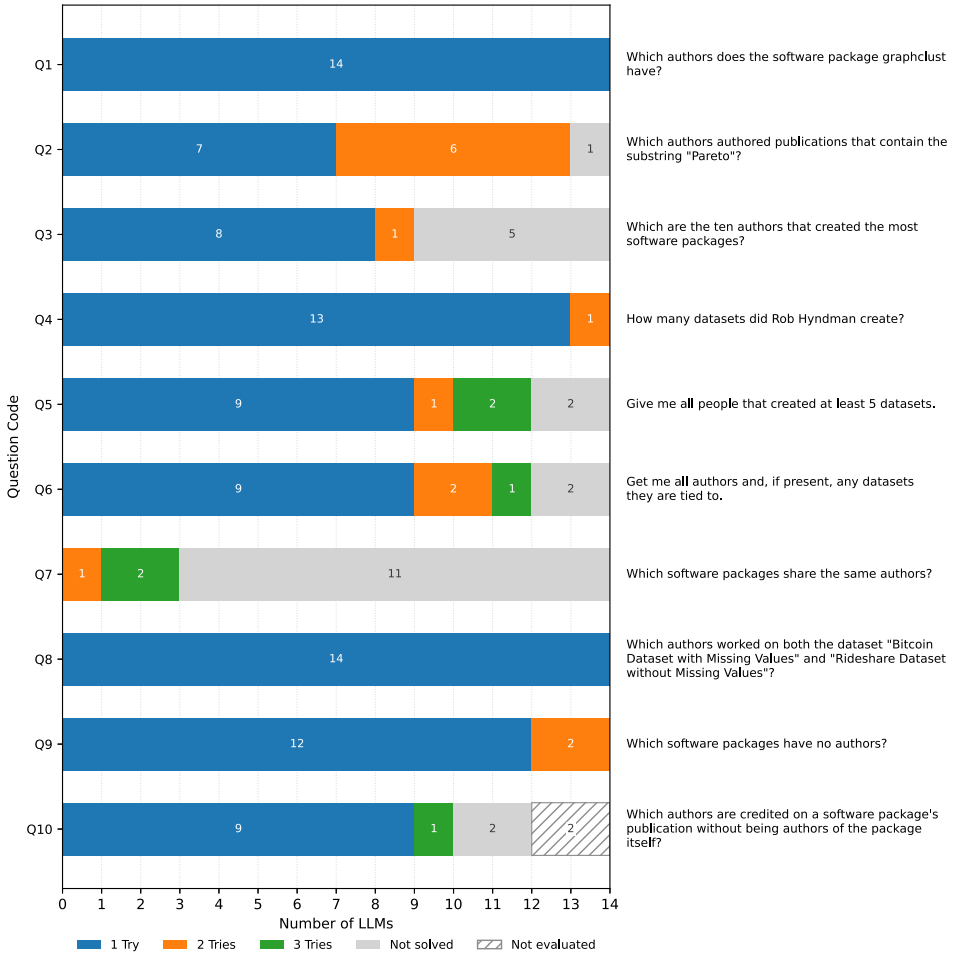


Fig. 9. For each question, the colored part of the bar denotes the number of the 14 evaluated LLMs that returned the correct answer, which is colored according to the number of amendment tries needed to solve it. The light gray segment counts the LLMs that never reached a correct answer, and the hatched segment the LLMs that could not be evaluated on that question: the tenth question was added after Claude 3.7 Sonnet and deepseek-reasoner-api in the current version had become inaccessible, so it was answered by only 12 LLMs.

belonged to at different points in time (most males change social group upon reaching sexual maturity [45]), as well as sightings (where and when a given hyena was observed in the field).

Questions. We received ten realistic questions from experts to be answered on the hyena graph (Appendix E). The ten questions focus on outcomes influenced by the biology of spotted hyenas and the efforts of

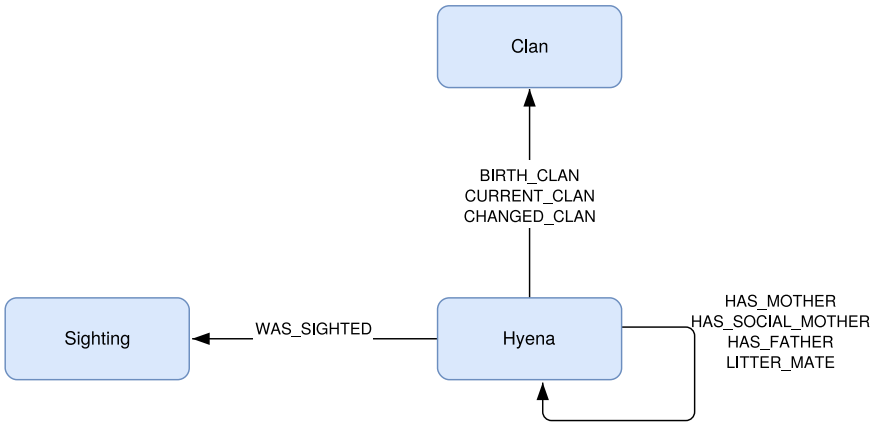


Fig. 10. Data model for the Hyena KG.

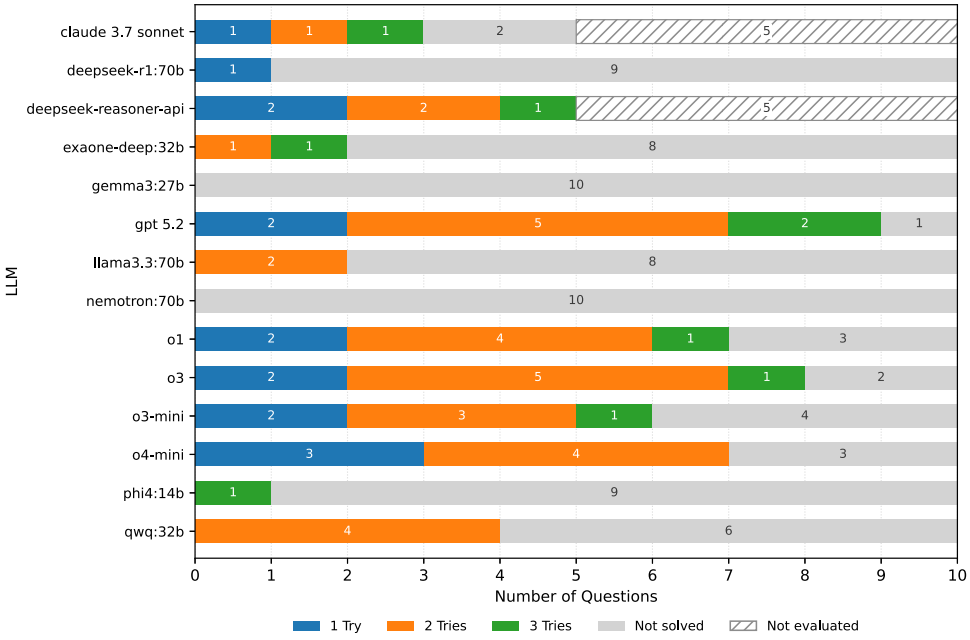


Fig. 11. For each LLM, the colored part of the bar indicates the share of the ten hyena questions eventually answered correctly, partitioned by the number of attempts (first, second, or third) needed to reach the correct answer (a maximum of three tries was allowed). The light gray segment counts the questions that the LLM attempted but never answered correctly, and the hatched segment the questions it could not be evaluated on: Claude 3.7 Sonnet and deepseek-reasoner-api in the previously used version were no longer accessible when the new five questions were added.

field biologists to sample them. The first five questions focus on the contribution of different types of males to paternity. These types include males from non-monitored peripheral clans (clan X), philopatric males breeding in their birth clans, primary dispersers who joined a new clan after leaving their birth clan, and males siring cubs outside their own clan. The answers to these questions are key to understanding the factors that contribute to male dispersal in spotted hyenas, a behavior believed to be the result of their access to females [11,45]. These answers can also be used by biologists to quantify gene flow over time and space, which shapes local adaptation and the risk of inbreeding depression. Next, we addressed the topic of female reproduction with three questions: one concerning their yearly reproductive output, one concerning their contribution to the number of adult females within the clan, and one concerning the discrepancy in birth dates between genetic offspring and offspring that mothers with young cubs may adopt. An additional question referred to individual survival. Since deaths are rarely observed, but rather deduced when individuals have not been observed for a while, we compiled a list of individuals who had not been observed for a year after a specific date. Finally, to summarize the population survey, we asked how many different individuals had been monitored in each clan over the years.

The hyena experts wrote the questions before the graph existed and thus were not familiar with our LLM/KG approach, thereby avoiding bias introduced by knowledge of the graph schema and mimicking a real-world trial of our framework. They selected them because they are representative of the kind of questions they routinely have to answer as part of larger analyses. In particular, the set of questions calls for outputs of different types typically needed by these researchers: ratios (proportions and rates), differences, numbers of individuals and identities of individuals meeting specific criteria. The amendments to the `Cypher` queries generated by the LLMs, if necessary, were performed by the first author (L.P.) in accordance with detailed explanations of how the questions were understood by the biologists. Correctness was judged by the first author on the basis of the experts' explanations of each question and the resulting understanding of its intended meaning; since the ground truth here was the experts' intent rather than a predefined query, no formal reference queries exist for this experiment, and the scoring is correspondingly harder to reproduce than the result-based MaRDI scoring.

5.5.2. Results

Performance varied widely. None of the LLMs answered all questions correctly (deepseek-reasoner-api answered all of the first set of

Table 10
First-attempt and within-three-attempts accuracy with 95% Wilson CI on ten natural language questions in the hyena case. n is the number of questions the LLM was evaluated on: Claude 3.7 Sonnet and deepseek-reasoner-api were no longer accessible when the five additional questions were added and are therefore scored out of five. Rows shaded in orange are the ones scored against a reduced *n*.

LLM	First try				Within 3 tries			
	Correct	*n*	Accuracy	95% CI	Correct	*n*	Accuracy	95% CI
claude 3.7 sonnet	1	5	0.2000	[0.036, 0.624]	3	5	0.6000	[0.231, 0.882]
deepseek-r1:70b	1	10	0.1000	[0.018, 0.404]	1	10	0.1000	[0.018, 0.404]
deepseek-reasoner-api	2	5	0.4000	[0.118, 0.769]	5	5	1.0000	[0.566, 1.000]
exaone-deep:32b	0	10	0.0000	[0.000, 0.278]	2	10	0.2000	[0.057, 0.510]
gemma3:27b	0	10	0.0000	[0.000, 0.278]	0	10	0.0000	[0.000, 0.278]
gpt 5.2	2	10	0.2000	[0.057, 0.510]	9	10	0.9000	[0.596, 0.982]
llama3.3:70b	0	10	0.0000	[0.000, 0.278]	2	10	0.2000	[0.057, 0.510]
nemotron:70b	0	10	0.0000	[0.000, 0.278]	0	10	0.0000	[0.000, 0.278]
o1	2	10	0.2000	[0.057, 0.510]	7	10	0.7000	[0.397, 0.892]
o3	2	10	0.2000	[0.057, 0.510]	8	10	0.8000	[0.490, 0.943]
o3-mini	2	10	0.2000	[0.057, 0.510]	6	10	0.6000	[0.313, 0.832]
o4-mini	3	10	0.3000	[0.108, 0.603]	7	10	0.7000	[0.397, 0.892]
phi4:14b	0	10	0.0000	[0.000, 0.278]	1	10	0.1000	[0.018, 0.404]
qwq:32b	0	10	0.0000	[0.000, 0.278]	4	10	0.4000	[0.168, 0.687]

Table 11
First-attempt and within-three-attempts accuracy for each natural language question in the hyena case with 95% Wilson CI. n is the number of LLMs evaluated on the question; the five additional questions were added after two LLMs had become inaccessible and are therefore scored out of 12. Rows shaded in orange are the ones scored against a reduced *n*.

Question	First try				Within 3 tries			
	Correct	*n*	Accuracy	95% CI	Correct	*n*	Accuracy	95% CI
What is the proportion of cubs that were sired by males born in clan X?	7	14	0.500000	[0.268, 0.732]	8	14	0.571429	[0.326, 0.786]
What is the proportion of cubs that were sired by philopatric males?	1	14	0.071429	[0.013, 0.315]	7	14	0.500000	[0.268, 0.732]
What is the proportion of cubs that were sired by primary dispersers?	0	14	0.000000	[0.000, 0.215]	5	14	0.357143	[0.163, 0.612]
What is the proportion of cubs with known fathers that were sired by males that were members of clans other than the clan of the female at conception?	0	14	0.000000	[0.000, 0.215]	5	14	0.357143	[0.163, 0.612]
What is the proportion of litters for which at least one cub has one father born in clan X?	4	14	0.285714	[0.117, 0.546]	5	14	0.357143	[0.163, 0.612]
Calculate the mean difference in birth dates between adopted social cubs and the genetic cub(s) of the surrogate mother.	0	12	0.000000	[0.000, 0.243]	4	12	0.333333	[0.138, 0.609]
Give me a list of individuals who were not sighted for more than one year on the 15.06.2024.	1	12	0.083333	[0.015, 0.354]	7	12	0.583333	[0.320, 0.807]
Give me the number of female recruits (females turning adult, i.e. becoming 2 years old) per year and clan.	0	12	0.000000	[0.000, 0.243]	2	12	0.166667	[0.047, 0.448]
What is the annual reproductive rate (number of offspring produced) of females?	0	12	0.000000	[0.000, 0.243]	2	12	0.166667	[0.047, 0.448]
What is the number of individuals who were monitored so far in each clan?	2	12	0.166667	[0.047, 0.448]	10	12	0.833333	[0.552, 0.953]

questions, but could not be tested on the second set). GPT 5.2 had the most correct answers with nine, o3 followed with eight and then came o1 and o4-mini with seven. Among the models run via Ollama, qwq was in the lead with four correct answers, while the remaining Ollama models solved between zero and two. A 0/10 outcome implies a low upper confidence limit of about 28 percent, so the poor performance of these models is unlikely to be an artifact of sample size. Efficiency, measured as the number of attempts required per correct answer, was low across models. Most correct answers required a second or third attempt: o4-mini had the highest number of first-attempt successes, three of seven, with five other models reaching two. Two thirds of the models with any correct answer needed three attempts on at least one question (see Fig. 11). The number of correct answers on the first try and within first tries as well as the mean accuracy and Wilson 95% CI are listed in Table 10. No single question was solved by every model, which is not surprising given the low number of correct answers for a lot of the models (see Fig. 12 and Table 11).

5.6. Amendment categorization

To characterize the amendments, we categorized each of the 209 amendments (post hoc, by the first author) using the following types: *direction flip* (reversing a relationship's direction), *schema correction* (replacing a wrong node label, relationship type, or property with the correct one), *filter/condition adjustment* (adding, removing, or changing a `WHERE` clause, aggregation, or `LIMIT`), *structural rewrite* (changing the query's logic or pattern, e.g. how a quantity is computed or how entities are grouped), *syntax repair* (correcting invalid Cypher), and *output formatting* (instructing the model to return only the query without additional text). Amendments addressing several issues were assigned all applicable categories. Table 1 reports, per category and real-life example, how often the query was correct after the amendment, together with the median amendment length. Filter/condition adjustments led to a correct query in 16 of 44 cases and direction flips in 8 of 23, whereas structural rewrites did so in 37 of 154. Direction flips were, however, effective only in the MaRDI experiment (8 of 14);

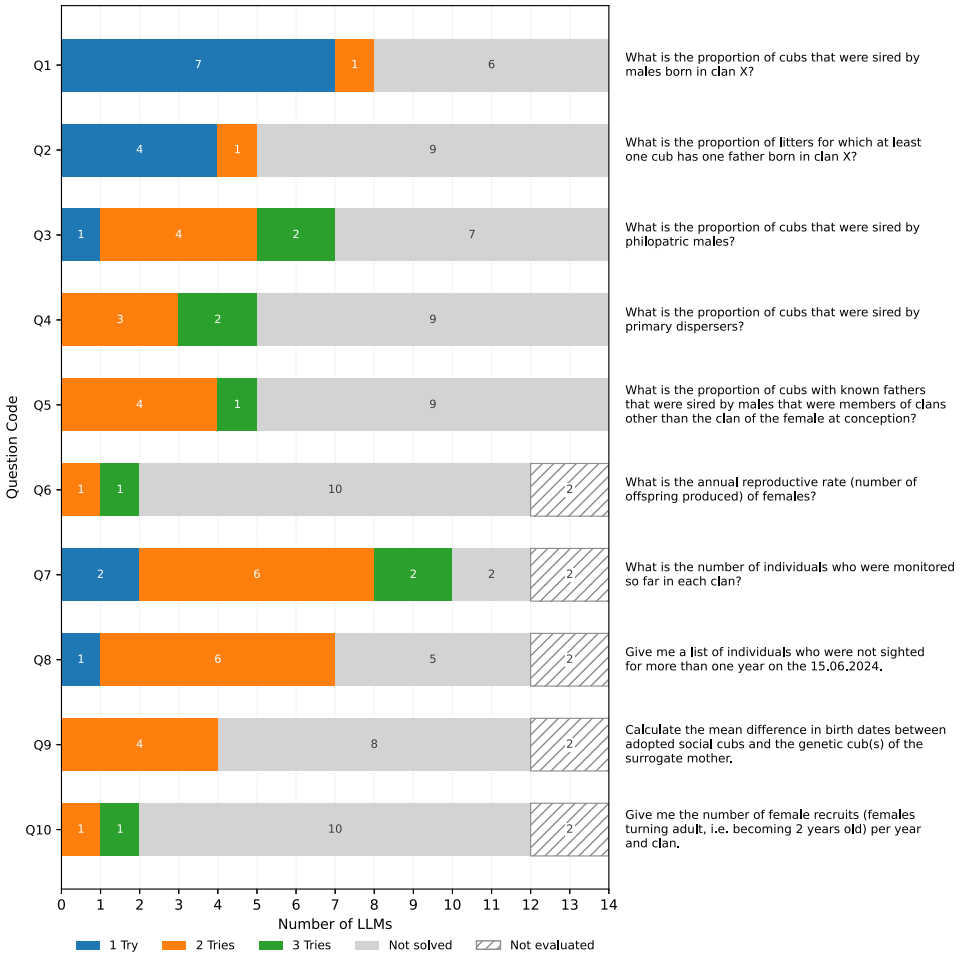


Fig. 12. For each question, the colored part of the bar denotes the number of the 14 evaluated LLMs that returned the correct answer, which is colored according to the number of amendment tries needed to solve it. The light gray segment counts the LLMs that never reached a correct answer, and the hatched segment the LLMs that could not be evaluated on that question: the new five questions were added after Claude 3.7 Sonnet and deepseek-reasoner-api in the current version had become inaccessible, so they were answered by only 12 LLMs.

none of the nine Hyena direction flips yielded a correct query, and only one of them was the sole correction the amendment made. Schema corrections (1 of 2), syntax repairs (0 out of 4), and output-formatting instructions (0 out of 4) rarely or never sufficed on their own. Structural rewrites were the most frequent category in both experiments, but they dominated the Hyena amendments (123 of 144), whereas the MaRDI amendments were more evenly distributed across structural rewrites (31), filter/condition adjustments (18), and direction flips (14). Two caveats apply to these counts. They cover only queries that were amended, so queries that were never corrected are not represented and amendments addressing several issues are counted in each applicable category, so the counts do not partition the amendments.

5.7. Discussion

Most large language models (LLMs) included in this experiment performed well on the questions in Appendix D, with two LLMs (o3-mini and GPT 5.2) generating all ten queries correctly and eleven others achieving between seven and nine correct answers. Only one LLM, deepseek-r1:70b, generated only six correct queries. Allowing users to iteratively revise the query was still beneficial: while most correct queries were generated on the first attempt, a non-negligible share required a second or third amendment, especially for the models gemma3:27b, llama3.3:70b and nemotron. Our analysis further revealed which question structures were more difficult than others. For example, matching authors to a specific software package was handled reliably by every model, whereas identifying software packages which shared the same authors remained impossible for eleven of the fourteen LLMs.

However, strong performance on carefully designed questions did not transfer to the second, expert-driven case study in every case. When we applied the same pipeline to the Ngorongoro Hyena Knowledge Graph and ten questions formulated by domain experts, performance gaps widened visibly. GPT 5.2 performed best (9/10 questions answered correctly), o3 reached 8/10, o1, o4-mini and o3-mini between 6 and 7/10; among Ollama models, the best was qwq with 4/10, with the rest between 0 and 2. Had we relied on the MaRDI experiment alone, these differences would have remained hidden.

The amendment categorization in Section 5.6 shows that the two experiments required different kinds of correction. Structural rewrites, which respecify the query's logic rather than adjust a condition or an edge, account for the large majority of the Hyena amendments but under half of the MaRDI ones, and schema corrections occur only in the Hyena experiment.

Two implications follow. First, expert-sourced questions are essential for discriminating among models that appear comparable on synthetic tests, and several Knowledge Graphs should be used to evaluate a method. Second, if a strategy with several amendment attempts is employed, efficiency, meaning how many tries a model needs, should be reported alongside the number of correct answers, because a rising number of retries makes the use of the tool more inconvenient and more expensive. The token counts in Table 19 underline this in cost terms: the generation prompt is fixed per graph, so the total cost of answering a question scales primarily with the number of attempts. Amendment

prompts are in fact shorter than the initial one, but the responses to them are usually longer, in many cases roughly twice as long, and since output tokens are priced well above input tokens, each retry costs at least as much as the initial attempt. These totals cover only attempts that produced logged output (i.e. not failures because of syntax errors), so the models that fail most often appear cheapest and cost should be read per answered question. Third, the kind of amendment matters as well as their number: reporting which corrections an example required helps explain why some Knowledge Graphs are harder than others.

In short, realistic questions and an additional Knowledge Graph revealed limitations of current LLMs when used in this pipeline, and future evaluations should not ignore the realism of questions and potential differences between domains.

6. Conclusion

We evaluated how well current LLMs explain, generate, and detect faults in `Cypher` queries, using an interactive framework that translates natural-language questions into auditable `Cypher`, explains the resulting queries, and lets users iteratively amend them. The framework served both as the evaluation platform and as a demonstration that transparent, user-controlled KG querying is feasible without requiring users to learn a query language. Across three studies, LLMs performed these roles with varying skill: explaining and sanity-checking queries (Section 4), generating workable queries on a structured mathematical KG (Section 5.4), and struggling unevenly on expert-written questions in a complex biological KG (Section 5.5). The answer to our titular question is conditional: LLMs can enable natural language querying of knowledge graphs, and explanation interfaces with human refinement loops can improve reliability, though performance on a given domain and query complexity cannot be assumed without explicit evaluation. On explanation quality, several models produced correct and complete one-sentence summaries for most movie-graph queries, yet a recurring failure was dropping explicit years - evidence that brevity pressures can harm faithfulness. These results form the most detailed and statistically grounded part of our evaluation. Concerning query generation, many models solved the MaRDI tasks on the first attempt, while the Hyena study exposed wider gaps: only the strongest API models reached high accuracy, while most locally run models solved at most two of ten, underscoring why realistic, expert-sourced questions and cross-domain evaluation matter. These two studies function as targeted case evaluations, illustrating how model behavior shifts with schema regularity, data realism, and question style. For practitioners, our findings suggest that LLM-powered KG querying could benefit from including explanation interfaces and iterative amendment loops, particularly for domains with irregular schemas or complex entity relationships. Future work should pursue human-centered and evaluation-oriented directions: refining explanation prompts to reliably retain temporal and numeric constraints; designing stronger interactive tooling, such as diff views for query edits and side-by-side result previews to shorten the path from first draft to trusted query; expanding evaluation to larger, more diverse real-world KGs to test generalizability; and comparing targeted amendment against regeneration from scratch with amendments formulated in a way such that they can be appended to the question as additional instructions. These evaluations remain bounded: 90 controlled queries and two smaller real-world question sets cannot guarantee performance across all KGs or query types. Furthermore, rapid LLM evolution means today's capability gaps may narrow unpredictably. Taken together, our results show that natural language access to knowledge graphs can be achieved, and can be improved, when transparency and user control are engineered in from the start — not treated as afterthoughts.

CRediT authorship contribution statement

Larissa Pusch: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Alexandre Courtiol:** Writing – review & editing, Resources, Formal analysis. **Tim Conrad:** Writing – review & editing, Supervision, Funding acquisition, Conceptualization.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work the author(s) used ChatGPT and Claude in order to improve readability, flow and language. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the published article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We want to thank Oliver Höner & Eve Davidian for providing us access to the hyena dataset and for formulating expert-level questions. We thank all other members of the Ngorongoro Hyena Project which also collected data used here (Philemon Naman, Bettina Wachter & Arjun Dheer). The authors also thank Leonie F. Walter for answering the hyena data questions independently using the R package `hyenaR` (without relying on LLM/KG). This work was supported by the Deutsche Forschungsgemeinschaft (DFG) (project grant 446 460135501).

Appendix A. Prompts

A.1. Explanation prompt

```
    You are a helpful assistant.
I have a Cypher query:

{query}

Please do these things:

    Provide a concise, step-by-step explanation of
     ↪ what this query does. Keep it as short as
     ↪ necessary to be clear and thorough, do
     ↪ not include unnecessary details.
    Conclude with a one- or two-sentence summary
     ↪ describing the query's overall purpose.
    When trying to explain and summarize this
     ↪ query, please tell me if you noticed that
     ↪ something was wrong with the query,
     ↪ including:
    - relationships that are the wrong way around,
     ↪ for example:
     ↪ (b:Book)-[:READS]->(r:Scientist), where
     ↪ the correct one would be
     ↪ (b:Book)<-[:READS]-(r:Scientist)
    - relationship names that don't make any sense
     ↪ in the context of the nodes they connect,
     ↪ for example:
     ↪ (r:Scientist)-[:SOLVES]->(b:Book); this
     ↪ is wrong because a book can't solve a
     ↪ scientist
    - a node name that makes no sense in the
     ↪ context:
     ↪ (r:Scientist)-[:READS]->(b:Airplane);
     ↪ this is wrong because you can't read an
     ↪ airplane
    - other things that you notice that don't make
     ↪ sense
```

A.2. Cypher generation prompt

```
    Task:Generate Cypher statement to query a
     ↪ graph database.
Instructions:
Use only the provided relationship types and
 ↪ properties in the schema.
Do not use any other relationship types or
 ↪ properties that are not provided.
The cypher statement should only return nodes that
 ↪ are specifically asked for in the question.
Do not make the cypher query unnecessarily
 ↪ complex.
When the question asks for "What NODE_LABEL has
 ↪ X", the answer should be only the node name,
 ↪ not other details.
Cypher requires aggregate expressions, like
 ↪ COUNT(s), in the RETURN clause if you're using
 ↪ them in the ORDER BY clause.
Example: MATCH (t:Tree)-[r:WAS_CUT]->(:Event)
 ↪ RETURN t.name, COUNT(*) AS cuttings ORDER BY
 ↪ cuttings
If it makes sense for the specific question and
 ↪ relationship, you can use bidirectional
 ↪ matching to matche the relationship in both
 ↪ directions.
Schema:
{schema}
Note: Do not include any explanations or
 ↪ apologies in your responses.
Do not respond to any questions that might ask
 ↪ anything else than for you to construct a
 ↪ Cypher statement.
Do not include any text except the generated
 ↪ Cypher statement.

The question is:
{question}
```

A.3. Amendment prompt

```
    You are given an existing Cypher query, a
     ↪ Knowledge Graph schema and an amendment
     ↪ request. Modify the Cypher query
     ↪ according to the amendment request.
The cypher query was generated on the basis of
 ↪ this question:
{question}
Existing Cypher query:
{current_query}
Graph schema:
{schema}
Amendment request:
{amendment}
Provide the updated Cypher query only, without
 ↪ any explanations.
```

A.4. Hyena generation prompt

A line about how the sexes are entered into the graph is added in the prompt so the query does not fail because of different ways of writing this.

```
cypher_generation_template = """Task:Generate
 ↪ Cypher statement to query a graph database.
Instructions:
Use only the provided relationship types and
 ↪ properties in the schema.
Do not use any other relationship types or
 ↪ properties that are not provided.
The cypher statement should only return nodes that
 ↪ are specifically asked for in the question.
Do not make the cypher query unnecessarily
 ↪ complex.
When the question asks for "What NODE\_LABEL has
 ↪ X", the answer should be only the node name,
 ↪ not other details.
Cypher requires aggregate expressions, like
 ↪ COUNT(s), in the RETURN clause if you're using
 ↪ them in the ORDER BY clause.
Example: MATCH (t:Tree)-[r:WAS_CUT]->(:Event)
 ↪ RETURN t.name, COUNT(*) AS cuttings ORDER BY
 ↪ cuttings
If it makes sense for the specific question and
 ↪ relationship, you can use bidirectional
 ↪ matching to matche the relationship in both
 ↪ directions.
The sexes in the graph are "male" and "female".
Schema:
{schema}
Note: Do not include any explanations or
 ↪ apologies in your responses.
Do not respond to any questions that might ask
 ↪ anything else than for you to construct a
 ↪ Cypher statement.
Do not include any text except the generated
 ↪ Cypher statement.

The question is:
{question}"""
```

Appendix B. SPARQL

```
# get cran packages that cite papers that have a de nr
(an identification number from zbMATH)
SELECT ?item ?itemLabel ?target ?targetLabel ?de_nr ?author
?authorLabel ?author_id
WHERE
{
 ?item wdt:P31 wd:Q57080.
 ?item wdt:P223 ?target.
 ?target wdt:P1451 ?de_nr.
 ?target wdt:P16 ?author.
 ?author wdt:P676 ?author_id
 SERVICE wikibase:label
 { bd:serviceParam wikibase:language "en". }
}

#get cran packages that have authors that have
a zbMATH author id
SELECT ?item ?itemLabel ?target ?targetLabel ?author_id
WHERE
{
 ?item wdt:P31 wd:Q57080.
 ?item wdt:P16 ?target.
 ?target wdt:P676 ?author_id
 SERVICE wikibase:label
 { bd:serviceParam wikibase:language "en". }
}
```

Table 12
MaRDI experiment questions, question types and example Cypher queries.

Question	Type	Reference Cypher query
Which authors does the software package graphclust have?	simple	`MATCH (sp:SoftwarePackage {name:"graphclust"})-[:HAS_AUTHOR] ->(a:Author) RETURN a;`
Which authors authored publications that contain the substring "Pareto"?	where	`MATCH (p:Publication) WHERE p.title CONTAINS "Pareto" MATCH (p)-[:HAS_AUTHOR]->(a:Author) RETURN DISTINCT a.name AS authorName, a.authorId AS authorId`
Which are the ten authors that created the most software packages?	top n	`MATCH (a:Author)<-[:HAS_AUTHOR]-(s:SoftwarePackage) RETURN a.name AS authorName, a.authorId AS authorId, COUNT(s) AS packageCount ORDER BY packageCount DESC LIMIT 10`
How many datasets did Rob Hyndman create?	counts	`MATCH (a:Author { name: "Rob Hyndman" })<-[:HAS_AUTHOR]-(d:Dataset) RETURN count(d) AS numberOfDatasets`
Give me all people that created at least 5 datasets.	distinct	`MATCH (p:Author)<-[:HAS_AUTHOR]-(d:Dataset) WITH p, COUNT(d) AS numberOfDatasets WHERE numberOfDatasets >= 5 RETURN p.name AS authorName, numberOfDatasets ORDER BY numberOfDatasets DESC`
Get me all authors and, if present, any datasets they are tied to.	optional match	`MATCH (a:Author) OPTIONAL MATCH (a)<-[:HAS_AUTHOR] -(d:Dataset) RETURN a.name AS authorName, a.authorId AS authorId, COLLECT(d) AS datasets`
Which software packages share the same authors?	shared	`MATCH (p:SoftwarePackage)-[:HAS_AUTHOR]->(a:Author) WITH p, COLLECT(DISTINCT a.authorId) AS authorIds WITH p, apoc.coll.sort(authorIds) AS sortedAuthorIds WITH sortedAuthorIds, COLLECT(p) AS packages WHERE SIZE(packages) > 1 RETURN sortedAuthorIds AS authorGroup, packages`
Which authors worked on both the dataset "Bitcoin Dataset with Missing Values" and "Rideshare Dataset without Missing Values"?	intersection	`MATCH (a:Author)<-[:HAS_AUTHOR]-(d1:Dataset), (a)<-[:HAS_AUTHOR]-(d2:Dataset) WHERE d1.name = "Bitcoin Dataset with Missing Values" AND d2.name = "Rideshare Dataset without Missing Values" RETURN a.name AS authorName, a.authorId AS authorId`
Which software packages have no authors?	not	`MATCH (p:SoftwarePackage) WHERE NOT (p)-[:HAS_AUTHOR] ->(:Author) RETURN p.packageId AS packageId, p.name AS packageName`
Which authors are credited on a software package's publication without being authors of the package itself?	multi-hop	`MATCH (sp:SoftwarePackage)-[:HAS_PUBLICATION]->(p:Publication)-[:HAS_AUTHOR] ->(a:Author) WHERE NOT (sp)-[:HAS_AUTHOR]->(a) RETURN DISTINCT sp.name AS packageName, a.name AS authorName, a.authorId AS authorId ORDER BY packageName, authorName`

```
#get zenodo packages that have authors with
a zbMATH author id
SELECT ?item ?itemLabel ?target ?targetLabel ?author_id
WHERE
{
 ?item wdt:P227 ?b.
 ?item wdt:P16 ?target.
 ?target wdt:P676 ?author_id
 SERVICE wikibase:label
 { bd:serviceParam wikibase:language "en". }
}
```

Appendix C. Explanation queries

- **Query:** `Cypher` query to be explained
- **N Hops:** Number of relationships between nodes in query
- **Clause type:** Types of clauses included in the query
- **Perturbation:** Type of error introduced into the query

The table can be found in the supplementary file longtable.pdf.

Appendix D. MaRDI experiment questions

- **Question:** Input Question
- **Type:** Query features that are included in the reference query
- **Reference `Cypher` Query:** `Cypher` query that outputs the desired results, which the output from the generated query can be compared to (see Table 12).

Appendix E. Hyena experiment questions

- **Question:** Input Question
- **N Hops:** Number of relationships between nodes in the path based on the question
- **Includes Calculation:** If the question includes a calculation, such as a proportion
- **Includes Complicated Niche Concept:** If the question includes a concept which only an expert in the field would understand
- **Disambiguation Notes:** Disambiguation notes are a mix of notes given by the expert and notes by the author (see Table 13).

Appendix F. Detailed results

F.1. Decoding parameters

Temperature was the only decoding parameter we set explicitly; the values per model are listed in Table 14. The OpenAI reasoning models accept only a temperature of 1. All other parameters, including top-p, maximum output tokens, and the reasoning-effort settings of the OpenAI reasoning models, were left at the defaults of the respective LangChain integration and provider API at the time of the experiments. Models marked (local) were served locally and accessed through an OpenAI-compatible Ollama endpoint. The exact configuration is part of the released code (tag v2.0-paper).

Table 13
Hyena experiment questions.

Question	N Hops	Includes calculation	Includes complicated niche concept	Disambiguation notes
What is the proportion of cubs that were sired by males born in clan X?	2	True	False	X-males are males born in clan X; only known fathers!
What is the proportion of litters for which at least one cub has one father born in clan X?	3	True	True	Multi-cub litters can have different fathers
What is the proportion of cubs that were sired by philopatric males?	3	True	True	I define philopatric as having never left
What is the proportion of cubs that were sired by primary dispersers?	3	True	True	I define primary disperser as hyenas that are in their first new clan
What is the proportion of cubs with known fathers that were sired by males that were members of clans other than the clan of the female at conception?	6	True	False	–
What is the annual reproductive rate (number of offspring produced) of females?	1	True	False	–
What is the number of individuals who were monitored so far in each clan?	2	True	False	–
Give me a list of individuals who were not sighted for more than one year on the 15.06.2024.	1	True	False	–
Calculate the mean difference in birth dates between adopted social cubs and the genetic cub(s) of the surrogate mother.	2	True	True	–
Give me the number of female recruits (females turning adult, i.e. becoming 2 years old) per year and clan.	2	True	True	–

Table 14
Explicitly set decoding parameters per model.

Model	Temperature
claude 3.7 sonnet	0
deepseek-reasoner-api	0
deepseek-r1:70b (local)	0
exaone-deep:32b (local)	0
gemma3:27b (local)	0
gpt 5.2	0
llama3.3:70b (local)	0
nemotron:70b (local)	0
o1	1
o1-preview	1
o3	1
o3-mini	1
o4-mini	1
phi4:14b (local)	0
qwq:32b (local)	0

Table 15
Total number of disagreements between the models in the explanation trial when generating one-sentence-summaries, further split up into the ones where only the row model or only the column model was correct.

row_model	col_model	row_only	col_only	total_diff
claude 3.7 sonnet	deepseek-reasoner-api	7	26	33
claude 3.7 sonnet	deepseek-r1:70b	9	22	31
claude 3.7 sonnet	o1-preview	1	23	24
claude 3.7 sonnet	o3-mini	5	22	27
deepseek-reasoner-api	deepseek-r1:70b	15	9	24
deepseek-reasoner-api	o1-preview	7	10	17
deepseek-reasoner-api	o3-mini	12	10	22
deepseek-r1:70b	o1-preview	7	16	23
deepseek-r1:70b	o3-mini	9	13	22
o1-preview	o3-mini	13	8	21

F.2. Explanation trials

The disagreements between models for the correct one-sentence summary are reported in Table 15. Disagreements related to correct perturbation detection appear in Table 16, and disagreements for identifying false positives are summarized in Table 17.

Appendix G. Token counts per attempt

Tables 18 and 19 report token counts for all three experiments, computed post hoc from the released model outputs. All counts were obtained with a single tokenizer (`o200k_base`) applied uniformly to every model, so they are comparable across models but serve as a proxy for prompt and response length rather than each provider's

Table 16
Total number of disagreements between the models in the additional trial, further split up into the ones where only the row model or only the column model was correct.

row_model	col_model	row_only	col_only	total_diff
claude 3.7 sonnet	deepseek-reasoner-api	7	10	17
claude 3.7 sonnet	deepseek-r1:70b	17	4	21
claude 3.7 sonnet	o1-preview	6	8	14
claude 3.7 sonnet	o3-mini	14	8	22
deepseek-reasoner-api	deepseek-r1:70b	16	0	16
deepseek-reasoner-api	o1-preview	3	2	5
deepseek-reasoner-api	o3-mini	9	0	9
deepseek-r1:70b	o1-preview	0	15	15
deepseek-r1:70b	o3-mini	1	8	9
o1-preview	o3-mini	8	0	8

Table 17
Total number of disagreements between the models in the false-positive experiment, split into cases where only the row model or only the column model was correct.

row_model	col_model	row_only	col_only	total_diff
claude 3.7 sonnet	deepseek-reasoner-api	0	5	5
claude 3.7 sonnet	deepseek-r1:70b	3	1	4
claude 3.7 sonnet	o1-preview	0	4	4
claude 3.7 sonnet	o3-mini	0	5	5
deepseek-reasoner-api	deepseek-r1:70b	7	0	7
deepseek-reasoner-api	o1-preview	1	0	1
deepseek-reasoner-api	o3-mini	0	0	0
deepseek-r1:70b	o1-preview	0	6	6
deepseek-r1:70b	o3-mini	0	7	7
o1-preview	o3-mini	0	1	1

Table 18
Token counts for the explanation experiment: number of queries (n) and median input and output tokens per model.

LLM	n	med. in	med. out
claude 3.7 sonnet	90	303	240
deepseek-r1:70b	90	303	206
deepseek-reasoner-api	90	303	255
o1-preview	90	303	507.5
o3-mini	90	303	203.5

Table 19
Token counts per attempt for the MaRDI and Hyena query-generation experiments. For each attempt, n is the number of questions on which the model made that attempt and in/out are the mean input and output tokens; the last column is the total output tokens over all three attempts. Where n is given as r/a, the model made a attempts but only r outputs were recorded (the remainder failed repeatedly because of syntax errors and were not logged) and the output mean and total cover those r items. Shaded rows are models that were no longer accessible when the additional questions were added; they were evaluated on the original questions only, so their n is bounded by that smaller question set and their total is not comparable with the other rows. This is more severe for the hyena experiments, where 5 questions were added, while only 1 was added for MaRDI.

LLM	Attempt 1			Attempt 2			Attempt 3			Total out
	n	In	Out	n	In	Out	n	In	Out	
MaRDI KG										
claude 3.7 sonnet	9	355.4	34.9	3	250.3	53	2	268	56.5	586
deepseek-reasoner-api	9	355.4	31.1	2	245	55.5	2	307	63.5	518
deepseek-r1:70b	9/10	356	37.3	4	244.2	62.5	3	285.3	52	742
exaone-deep:32b	8/10	356	34.4	2	256.5	38	1	259	243	594
gemma3:27b	10	356	33.7	7	248.4	42.4	2	269	46	726
gpt 5.2	10	356	33.8	1	258	158	1	368	110	606
llama3.3:70b	10	356	33.2	5	240.6	59.2	2	320.5	59	746
nemotron:70b	10	356	36.1	5	248	34.8	4	246	45.5	717
o1	10	356	35.8	2	254.5	41.5	2	251	46.5	534
o3	10	356	35.8	1	265	140	1	360	59	557
o3-mini	10	356	34.9	1	264	70	–	–	–	419
o4-mini	10	356	36.5	1	277	116	1	324	74	555
phi4:14b	10	356	41.3	4	266.8	67.8	4	281.5	70.5	966
qwq:32b	10	356	37.6	2	259	14.5	1	202	57	462
Hyena KG										
claude 3.7 sonnet	5	435.6	120	4	425.8	221.2	3	528.7	244	2217
deepseek-reasoner-api	5	435.6	109.4	3	404.7	157	1	394	93	1111
deepseek-r1:70b	6/10	436	82.5	5	393	120	5	431.8	125.6	1723
exaone-deep:32b	3/10	436	49.7	3	340	69	2	335	45.5	447
gemma3:27b	8/10	436	64	8	371.1	103.4	8	400	105.6	2184
gpt 5.2	10	436	89.9	8	390.4	159.4	3	446.7	167.3	2676
llama3.3:70b	8/10	436	96.2	8	409.9	126	6	460.7	151.5	2687
nemotron:70b	4/10	436	143.5	4	443	119.2	4	413.5	126.2	1556
o1	10	436	81.3	8	383.8	145.8	4	472.5	142.2	2548
o3	9/10	436	89.6	7	383.4	141.9	2	394.5	131	2061
o3-mini	10	436	85.8	8	382.8	103.6	5	407.6	100.6	2190
o4-mini	10	436	71.6	7	372.7	123	3	420.7	101.3	1881
phi4:14b	8/10	436	119.8	8	422.1	126	8	426.5	159.5	3242
qwq:32b	7/10	436	71.4	6/7	364.3	96.2	2	427.5	131.5	1340

billable count. Input counts cover the full prompt; output counts cover the delivered response only. Hidden reasoning tokens of the reasoning models were not logged by our pipeline, and visible reasoning traces (`<think>` blocks) of the locally run models were stripped before counting, so output counts for reasoning models constitute lower bounds on the tokens actually generated. Generations that produced no storable output are not counted.

G.1. Explanation experiment

Table 18 lists the counts for the 90 explanation queries. The input comprises the explanation prompt including the query; as the prompt does not contain the graph schema, input length is nearly constant across queries and identical across models.

G.2. Real-life examples

Table 19 breaks the counts down per attempt for the MaRDI and Hyena real-life examples. Attempt-1 inputs contain the generation prompt with the graph schema and the question; attempt-2 and attempt-3 inputs contain the amendment prompt with the schema, the question, the current query, and the amendment text. The models Claude 3.7 Sonnet and the deepseek-reasoner-api were not available anymore when the additional questions were added and therefore are not comparable to the other rows.

Data availability

The code and experimental data are available at https://git.zib.de/lpusch/talk2cypher (tag `v2.0-paper` corresponds to this article). The repository contains the interactive query framework (generation, execution, explanation, and amendment components), the prompts used in all experiments, the 90-query explanation benchmark, and the MaRDI subgraph used in our evaluation. It further provides the raw experimental results: the model outputs from the explanation experiment, and the generated queries, token counts, amendment requests, amendment classifications and amended queries from the MaRDI and Hyena experiments. As the Hyena observational data are maintained by the Ngorongoro Hyena Project and cannot be redistributed here, the repository includes instructions for obtaining access to the subset used in this paper.

References

[1] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, et al., A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, ACM Trans. Inf. Syst. 43 (2) (2025) 1–55.

[2] A.B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, et al., Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI, Inf. Fusion 58 (2020) 82–115.

[3] S. Zhao, Y. Yang, Z. Wang, Z. He, L.K. Qiu, L. Qiu, Retrieval augmented generation (rag) and beyond: A comprehensive survey on how to make your llms use external data more wisely, 2024, arXiv preprint arXiv:2409.14924.

[4] H. Liu, Z. Wang, X. Chen, Z. Li, F. Xiong, Q. Yu, W. Zhang, Hoprag: Multi-hop reasoning for logic-aware retrieval-augmented generation, 2025, arXiv preprint arXiv:2502.12442.
[5] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G.D. Melo, C. Gutierrez, S. Kirrane, J.E.L. Gayo, R. Navigli, S. Neumaier, et al., Knowledge graphs, ACM Comput. Surv. (Csur) 54 (4) (2021) 1–37.
[6] A. Singhal, Introducing the knowledge graph: Things, not strings, 2012, https://blog.google/products/search/introducing-knowledge-graph-things-not/. (Accessed 28 May 2025).
[7] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM 57 (10) (2014) 78–85.
[8] Z. Zhu, T. Huang, K. Wang, J. Ye, X. Chen, S. Luo, Graph-based approaches and functionalities in retrieval-augmented generation: A comprehensive survey, 2025, arXiv preprint arXiv:2504.10499.
[9] L. Pusch, T.O. Conrad, Combining LLMs and knowledge graphs to reduce hallucinations in question answering, 2024, arXiv preprint arXiv:2409.04181.
[10] M. Schubotz, E. Ferrer, J. Stegmüller, D. Mietchen, O. Teschke, L. Pusch, T.O. Conrad, Bravo MaRDI: a wikibase powered knowledge graph on mathematics, 2023, arXiv preprint arXiv:2309.11484.
[11] E. Davidian, A. Courtiol, B. Wachter, H. Hofer, O.P. Höner, Why do some males choose to breed at home when most other males disperse? Sci. Adv. 2 (3) (2016) e1501236, http://dx.doi.org/10.1126/sciadv.1501236.
[12] A. Dheer, E. Davidian, A. Courtiol, L.D. Bailey, J. Wauters, P. Naman, V. Shayo, O.P. Höner, Diurnal pastoralism does not reduce juvenile recruitment nor elevate allostatic load in spotted hyenas, J. Anim. Ecol. 91 (11) (2022) 2289–2300, http://dx.doi.org/10.1111/1365-2656.13812.
[13] G. Lyons, V. Tran, C. Binnig, U. Cetintemel, T. Kraska, Making the case for query-by-voice with echoquery, in: Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 2129–2132.
[14] R. Zhang, T. Yu, H. Er, S. Shim, E. Xue, X.V. Lin, T. Shi, C. Xiong, R. Socher, D. Radev, Editing-based SQL query generation for cross-domain context-dependent questions, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP, 2019, pp. 5338–5349.
[15] Y. Tian, D. Lee, F. Wu, T. Mai, K. Qian, S. Sahai, T. Zhang, Y. Li, Text-to-SQL domain adaptation via human-LLM collaborative data annotation, in: Proceedings of the 30th International Conference on Intelligent User Interfaces, 2025, pp. 1398–1425.
[16] A. Guo, X. Li, G. Xiao, Z. Tan, X. Zhao, Spcql: A semantic parsing dataset for converting natural language into cypher, in: Proceedings of the 31st ACM International Conference on Information & Knowledge Management, 2022, pp. 3973–3977.
[17] M.G. Ozsoy, L. Messallem, J. Besga, G. Minneci, Text2cypher: Bridging natural language and graph databases, 2024, arXiv preprint arXiv:2412.10064.
[18] Y. Feng, S. Papicchio, S. Rahman, CypherBench: Towards precise retrieval over full-scale modern knowledge graphs in the LLM era, 2024, arXiv preprint arXiv:2412.18702.
[19] F. Cazzaro, J. Kleindienst, S. Márquez Gomez, A.J. Quattoni, ZOGRASCOPE: A new benchmark for semantic parsing over property graphs, in: Findings of the Association for Computational Linguistics: EMNLP 2025, Association for Computational Linguistics, 2025, pp. 4239–4246.
[20] A. Tiwari, S.K.R. Malay, V. Yadav, M. Hashemi, S.T. Madhusudhan, Auto-cypher: Improving LLMs on cypher generation via LLM-supervised generation-verification framework, 2024, arXiv preprint arXiv:2412.12612.
[21] H. Luo, E. Haihong, Z. Tang, S. Peng, Y. Guo, W. Zhang, C. Ma, G. Dong, M. Song, W. Lin, et al., Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models, in: Findings of the Association for Computational Linguistics: ACL 2024, 2024, pp. 2039–2056.
[22] R. Wang, Z. Zhang, L. Rossetto, F. Ruosch, A. Bernstein, Nlqxform: A language model-based question to SPARQL transformer, 2023, arXiv preprint arXiv:2311.07588.
[23] H.M. Zahera, M. Ali, M.A. Sherif, D. Moussallem, A.-C.N. Ngomo, Generating SPARQL from natural language using chain-of-thoughts prompting, in: SEMANTICS, 2024, pp. 353–368.
[24] L. Perez-Beltrachini, P. Jain, E. Monti, M. Lapata, Semantic parsing for conversational question answering over knowledge graphs, in: Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, 2023, pp. 2507–2522.
[25] D. Agarwal, R. Das, S. Khosla, R. Gangadharaiah, Bring your own KG: Self-supervised program synthesis for zero-shot KGQA, in: Findings of the Association for Computational Linguistics: NAACL 2024, 2024, pp. 896–919.
[26] M. Arazzi, D. Ligari, S. Nicolazzo, A. Nocera, Augmented knowledge graph querying leveraging LLMs, in: 2025 International Joint Conference on Neural Networks, IJCNN, IEEE, 2025, pp. 1–8.
[27] L. Nie, S. Cao, J. Shi, J. Sun, Q. Tian, L. Hou, J. Li, J. Zhai, GraphQ IR: Unifying the semantic parsing of graph query languages with one intermediate representation, in: Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, 2022, pp. 5848–5865.
[28] D. Steinigen, R. Teucher, T.H. Ruland, M. Rudat, N. Flores-Herr, P. Fischer, N. Milosevic, C. Schymura, A. Ziletti, Fact finder-enhancing domain expertise of large language models by incorporating knowledge graphs, in: Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 3: System Demonstrations), 2026, pp. 101–110.
[29] Y. Tian, J.K. Kummerfeld, T.J.-J. Li, T. Zhang, Sqlucid: Grounding natural language database queries with interactive explanations, in: Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology, 2024, pp. 1–20.
[30] Y. Tian, Z. Zhang, Z. Ning, T.J.-J. Li, J.K. Kummerfeld, T. Zhang, Interactive text-to-sql generation via editable step-by-step explanations, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 2023, pp. 16149–16166.
[31] R. Wang, Z. Zhang, L. Rossetto, F. Ruosch, A. Bernstein, NLQxform-UI: An interactive and intuitive scholarly question answering system, in: Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2025, pp. 3990–3993.
[32] X. Jian, Z. Dong, M.T. Özsu, InteracSPARQL: An interactive system for SPARQL query refinement using natural language explanations, in: Findings of the Association for Computational Linguistics: ACL 2026, 2026, pp. 29599–29624.
[33] S. Ferré, Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language, Semant. Web 8 (3) (2016) 405–418.
[34] Wikimedia Foundation, Wikidata query builder, 2026, https://query.wikidata.org/querybuilder/. (Accessed 07 July 2026).
[35] I. Gür, S. Yavuz, Y. Su, X. Yan, Dialsql: Dialogue based structured query generation, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2018, pp. 1339–1349.
[36] A. Gusarov, A. Volkova, V. Khrulkov, A. Kuznetsov, E. Maslov, I. Oseledets, Multi-agent GraphRAG: A text-to-cypher framework for labeled property graphs, 2025, arXiv preprint arXiv:2511.08274.
[37] V. Emonet, J. Bolleman, S. Duvaud, T.M. de Farias, A.C. Sima, Llm-based sparql query generation from natural language over federated knowledge graphs, 2024, arXiv preprint arXiv:2410.06062.
[38] F. Rousset, J.-B. Ferdy, Testing environmental and genetic effects in the presence of spatial autocorrelation, Ecography 37 (8) (2014) 781–790.
[39] T. Hothorn, F. Bretz, P. Westfall, Simultaneous inference in general parametric models, Biom. J. 50 (3) (2008) 346–363.
[40] K. Hornik, The comprehensive R archive network, Wiley Interdiscip. Rev.: Comput. Stat. 4 (4) (2012) 394–398.
[41] M. Petrera, D. Trautwein, I. Beckenbach, D. Ehsani, F. Müller, O. Teschke, B. Gipp, M. Schubotz, zbMATH Open: API solutions and research challenges, 2021, arXiv preprint arXiv:2106.04664.
[42] I. Peters, P. Kraker, E. Lex, C. Gumpenberger, J.I. Gorraiz, Zenodo in the spotlight of traditional and new metrics, Front. Res. Metr. Anal. 2 (2017) 13.
[43] M.L. East, O.P. Höner, B. Wachter, K. Wilhelm, T. Burke, H. Hofer, Maternal effects on offspring social status in spotted hyenas, Behav. Ecol. 20 (3) (2009) 478–483, http://dx.doi.org/10.1093/beheco/arp020.
[44] L.G. Frank, S.E. Glickman, P. Licht, Fatal sibling aggression, precocial development, and androgens in neonatal spotted hyenas, Science 252 (5006) (1991) 702–704, http://dx.doi.org/10.1126/science.2024122.
[45] O.P. Höner, B. Wachter, M.L. East, W. Streich, K. Wilhelm, T. Burke, H. Hofer, Female mate-choice drives the evolution of male-biased dispersal in a social mammal, Nature 448 (7155) (2007) 798–801, http://dx.doi.org/10.1038/nature06040.