POLITECNICO DI TORINO
Repository ISTITUZIONALE

CypherBench: Towards Precise Retrieval over Full-scale Modern Knowledge Graphs in the LLM Era

Original
CypherBench: Towards Precise Retrieval over Full-scale Modern Knowledge Graphs in the LLM Era / Feng, Y., Papicchio, S., Rahman, S.. - 1: Long Papers:(2025), pp. 8934-8958. (Association for Computational Linguistics Vienna (AT) July 27 - August 1, 2025) [10.18653/v1/2025.acl-long.438].

Availability:
This version is available at: 11583/3002590 since: 2025-08-28T09:38:56Z

Publisher:
Association for Computational Linguistics

Published
DOI:10.18653/v1/2025.acl-long.438

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

28 November 2026

CypherBench: Towards Precise Retrieval over Full-scale Modern Knowledge Graphs in the LLM Era

Yanlin Feng[α] **Simone Papicchio**[*βγ] **Sajjadur Rahman**[†δ]
[α]Megagon Labs [β]Politecnico di Torino [γ]EURECOM [δ]Adobe
yanlin@megagon.ai simone.papicchio@polito.it sajjadurr@adobe.com

Abstract

Retrieval from graph data is crucial for augmenting large language models (LLM) with both open-domain knowledge and private enterprise data, and it is also a key component in the recent GraphRAG system (Edge et al., 2024). Despite decades of research on knowledge graphs and knowledge base question answering, leading LLM frameworks (*e.g.*, Langchain and LlamaIndex) have only minimal support for retrieval from modern encyclopedic knowledge graphs like Wikidata. In this paper, we analyze the root cause and suggest that modern RDF knowledge graphs (*e.g.*, Wikidata, Freebase) are less efficient for LLMs due to overly large schemas that far exceed the typical LLM context window, use of resource identifiers, overlapping relation types and lack of normalization. As a solution, we propose *property graph views* on top of the underlying RDF graph that can be efficiently queried by LLMs using *Cypher*. We instantiated this idea on Wikidata and introduced CypherBench, the first benchmark with 11 large-scale, multi-domain property graphs with 7.8 million entities and over 10,000 questions. To achieve this, we tackled several key challenges, including developing an RDF-to-property graph conversion engine, creating a systematic pipeline for text-to-Cypher task generation, and designing new evaluation metrics.

1 Introduction

Graphs, as a natural modality for modeling entity-relation data, have been widely used for storing both large-scale encyclopedic knowledge and domain-specific enterprise data. Compared to raw textual documents, graphs enable efficient processing of complex multi-hop aggregation queries (*e.g.*, *What is the average height of point guards who have played for the Toronto Raptors?*), where the answer might depend on information spread across thousands of documents or even the entire corpus. Graphs also provide a more compact representation of knowledge. For example, Wikidata (Vrandečić and Krötzsch, 2014) contains on average 4.6 times the entities covered by Wikipedia across the domains we experimented with. These advantages has motivated decades of research in knowledge graphs and knowledge base question answering (KBQA) (Berant et al., 2013; Moon et al., 2019; Dubey et al., 2019; Gu et al., 2021; Cao et al., 2022), as well as the recent proposal of GraphRAG (Edge et al., 2024; Peng et al., 2024).

However, retrieval[1] from modern encyclopedic knowledge graphs (Vrandečić and Krötzsch, 2014; Bollacker et al., 2008; Suchanek et al., 2007; Lehmann et al., 2015), which are predominantly based on RDF, has proven challenging even with the use of LLMs, unlike the success achieved with relational databases using text-to-SQL. Previous studies in KBQA (detailed in Appendix A) typically focused on simplified settings for evaluating algorithmic improvements, using either smaller subgraphs (Cao et al., 2022; Baek et al., 2023; Emonet et al., 2024), simple queries without aggregation or grouping (Berant et al., 2013; Dubey et al., 2019; Nie et al., 2022; Baek et al., 2023), or assuming the entity identifiers are provided (Furrer et al., 2020; Kovriguina et al., 2023; Banerjee et al., 2022), which limits their practical application in real-world scenarios. As a result, leading LLM frameworks, including LangChain and LlamaIndex, have only minimal support for retrieval from

[*]The work began during Simone Papicchio's internship at Megagon Labs. As part of one subtask of his overall internship goal, he implemented an initial version of the benchmark that involved SQL-inspired template design, query categorization, and validation of the generated benchmark. The work has since further evolved to broaden and bolster the template generation process and redefining query categories while introducing new evaluation metrics.

[†]Work done while at Megagon Labs.

[1]Graph retrieval can be considered as a broader task than KBQA, as it is not only essential for question answering but also for other tasks such as fact checking (Kim et al., 2023).

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 8934–8958
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

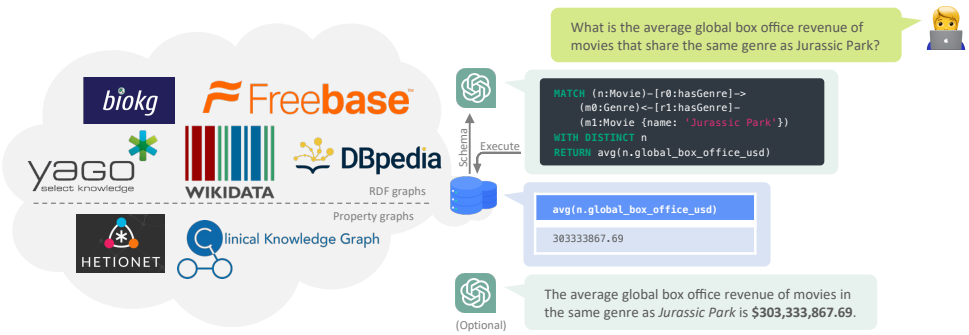


Figure 1: An illustration of Cypher as a unified interface for RDF and property graphs. A graph retrieval for RAG workflow involves: 1) text-to-Cypher translation, 2) query execution, and optionally, 3) final response generation.

modern RDF knowledge graphs as of the writing of this paper. Instead, they have prioritized retrieval from *property graphs*, which are typically domain-specific with smaller schemas (Santos et al., 2022; Himmelstein et al., 2017), using text-to-Cypher generation.

We analyze the root cause of why retrieval from RDF knowledge graphs is challenging in section 2 and propose transforming them into multiple smaller property graphs. These property graphs function as domain-specific *views* (analogous to views in relational databases) that can be efficiently queried by LLMs. This approach not only simplifies retrieval from modern RDF knowledge graphs, but also enables the use of Cypher as a unified query language for both RDF graphs and property graph databases like Neo4j that are widely used in enterprise.

As a proof of concept, we introduced **CypherBench**, a collection of 11 property graphs transformed from Wikidata (detailed in section 3). Each graph contains the *complete* set of entities and relations from Wikidata that conform to a domain-specific schema. Together, these graphs include a total of 7 million entities, covering roughly 25% of Wikipedia and 6% of Wikidata. In addition, we constructed over 10,000 natural language questions that spans 12 types of graph matching patterns (detailed in section 4). Notably, we include *global queries* which have been largely overlooked by prior KBQA benchmarks. The benchmark presents significant challenges, with `gpt-4o` achieving 60.18% execution accuracy, and no LLMs with <10B parameters surpassing 20%.

In summary, this paper makes the following main contributions:

- A novel methodology to enable efficient and accurate text-to-Cypher retrieval over modern RDF knowledge graphs.
- A collection of 11 large-scale property graphs with 7 million entities, serving as groundwork for future graph retrieval research and as a high-quality locally-deployable knowledge source (with a knowledge cutoff of April 2024) for augmenting LLMs.
- An RDF-to-property-graph transformation engine for Wikidata that creates the aforementioned graphs. It handles triple transformation, datatype conversion, and unit standardization to produce clean, schema-enforced property graphs as output.
- A text-to-Cypher / KBQA benchmark with over 10,000 instances spanning 12 types of graph patterns, covering global queries, multi-hop queries, temporal queries and aggregation queries.
- An automatic text-to-Cypher task generation pipeline that creates the aforementioned benchmark. It can be used to generate (question, Cypher) pairs for any Neo4j graph database endpoint.
- A set of related tools, including graph deployment Docker, evaluation scripts, and graph visualization tools.

2 Knowledge Graph Modeling in the LLM Era

2.1 Preliminaries: Knowledge Graphs, RDF and Property Graphs

The Resource Description Framework (RDF) and property graph are two approaches to modeling and querying knowledge graphs. We begin with an abstract definition of a knowledge graph and then

discuss how it is implemented in RDF and property graphs.

In its most basic form, a knowledge graph is a list of relations (also called triples) in the format (subject entity, relation type, object entity), such as ("LeBron James", `playsFor`, "LA Lakers"). [2] Additionally, entities are often assigned entity types (e.g., `Person`). Entities and relations can also be associated with literal values as properties (*e.g.*, `receivesAward.year`).

The most popular public knowledge graphs to date (*e.g.*, Wikidata, Freebase, and DBpedia) are predominantly based on the RDF and queried using SPARQL. In RDF, entities are stored and accessed using Internationalized Resource Identifiers (IRIs). Entity properties, including entity names, are stored as relations, with the subject being the entity IRI and the object being a literal value. To store relation properties, RDF uses a process called reification, which creates a copy of the relation as a special entity[3] and links it to the relation property using an additional relation.

Property graph databases have gained significant popularity in industry in recent years, with Neo4j being the most popular graph database management system today[4]. Unlike RDF, the property graph model treats entities and relations as objects, each of which can be assigned types and have associated properties. In property graphs, entities are often accessed directly using their names.

2.2 Why is retrieval over modern KG hard?

Retrieval over modern encyclopedic knowledge graphs, which are predominantly RDF graphs (shown in Figure 1), poses significant challenges due to several factors.

Overly large schemas. Modern encyclopedic knowledge graphs aim to cover entities and relations across all domains within a single graph, resulting in an extremely large schema that far exceeds the context window sizes of typical LLMs. For instance, Wikidata currently includes over 4 million entity types and 12,000 relation types. Furthermore, RDF graphs allow entities of arbitrary type to serve as subjects or objects for the same relation types, which further increases the number of unique relation schemas.

Use of resource identifiers. SPARQL queries require identifiers for entities, entity types, relation types which must be obtained via external linkers (Sakor et al., 2020; Feng et al., 2023). This also makes SPARQL queries less readable. For instance, consider the SPARQL and Cypher queries for the question "$Q4$. What are the names of taxa that feed on Synsphyronus lathrius?":

SPARQL

```
SELECT ?name WHERE {
  item wdt:P31/wdt:P279* wd:Q16521.
  ?item wdt:P1034 wd:Q10687580.
  ?item rdfs:label ?name. FILTER(LANG(?itemLabel) = "en")
}
```

Cypher

```
MATCH (n:Taxon)-[r0:feedsOn]->(m0:Taxon {name:
      'Synsphyronus lathrius'})
RETURN n.name
```

Overlapping relation types. Wikidata contains semantically overlapping relation types that are created for domain-specific usage. For instance, there are at least six relation types to indicate the starting time of an entity: `start time (P580)`, `inception (P571)`, `date of official opening (P1619)`, `date of first performance (P1191)`, `publication date (P577)`, `service entry (P729)`. This leads to considerable confusion when selecting the correct relation type to use during retrieval.

Lack of normalization. RDF does not enforce type constraints and standardized units on values. As a result, literal values in Wikidata often appear with different units (*e.g.*, centimeters and feet for heights) and sometimes incorrect types, which leads to incorrect results when computing aggregation over these values.

2.3 Hasn't KBQA already solved KG retrieval?

KBQA requires graph retrieval to answer questions. However, most existing studies focused on simplified settings to evaluate algorithmic improvements. For example, a common simplification made by recent work is assuming that the entity and relation identifiers are already provided (Furrer et al., 2020; Kovriguina et al., 2023; Banerjee et al., 2022; LUO et al., 2024; Yu et al., 2022; Jiang et al., 2023), which reduces the task to retrieval over a small local subgraph. Moreover, many studies use custom-designed intermediate logical forms that lack support for certain graph querying functionalities (*e.g.*, relation properties querying, grouping,

[2] Note that relation types are also called properties in Wikidata or predicates in KBQA literature. In this paper, we use "properties" specifically to denote entity and relation attributes in property graphs.

[3] Specifically, the statement node in Wikidata and the CVT node in Freebase.

[4] According to https://db-engines.com/en/ranking/graph+dbms

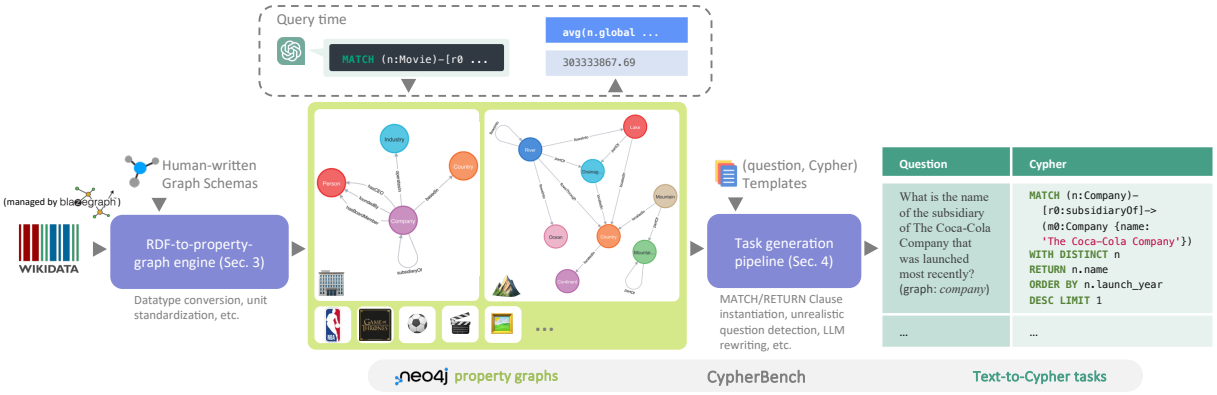


Figure 2: CypherBench construction process: Wikidata is transformed into schema-enforced property graphs, which enables efficient and accurate text-to-Cypher querying. These property graphs are then used to generate text-to-Cypher tasks.

variable-length path matching) (Berant et al., 2013; Berant and Liang, 2014; Gu et al., 2021; Yih et al., 2015; Cao et al., 2022; Nie et al., 2022; Li et al., 2023). As a result, the majority of existing KBQA approaches (see Appendix A for a complete categorization) struggle with some or all of the following types of queries: 1) queries involving relation properties, such as time-sensitive queries; 2) global queries that do not contain any named entities; and 3) queries requiring complex aggregations over a large number of entities.

2.4 Our Proposal: Property Graphs and Cypher as a Unified Interface

To address the aforementioned challenges, we propose transforming the RDF graph into multiple domain-specific *property graphs* and using Cypher to query them (see Figure 2 for an illustration). We chose Cypher, the query language for Neo4j, because of its widespread adoption in open-source projects (including LLM frameworks and GraphRAG). These property graphs function as (materialized) *views* on top of the original RDF graph that can be queried efficiently by LLM. Each property graph view contains all data that conforms to its respective schema and can be updated when the underlying RDF data changes. These views operate independently, may overlap, and may be created on-demand, offering substantial flexibility.

This design enables scaling to a large number of domains without introducing an overly complex schema and eliminates ambiguity from overlapping relation types by contextualizing them within specific domains. Our RDF-to-property-graph transformation layer manages datatype conversion and unit standardization, producing schema-enforced, strictly typed data to ensure the result correctness for aggregation queries. This transformation is also efficient and takes only a few seconds for small graphs (fewer than 10,000 entities), as it is is achieved by executing SPARQL queries and aggregating the results, rather than processing the entire RDF dump.

In the following sections, we instantiated this idea on Wikidata, the largest knowledge graph today. We demonstrate that a direct prompting baseline using `gpt-4o`, without the use of external linkers or retrievers, achieves reasonable performance.

3 Transforming RDF to Property Graphs

In this section, we introduce our approach for transforming RDF, specifically Wikidata, into property graphs as the initial step in building CypherBench (Figure 2). We selected Wikidata because it is the largest and most actively maintained knowledge graph, comprising 114 million entities and having received 270 million edits from over 42,000 active editors in the past 12 months[5].

3.1 Domain-specific Schema Curation

We begin by selecting a domain and manually curating the property graph schema, along with the mapping from the Wikidata schema to this schema. This process typically involves identifying the entity and relation types and their properties relevant to the domain, followed by exploring Wikidata to find the corresponding Wikidata identifiers (QIDs for entity types and PIDs for relation types and properties). A sample entry is shown below:

[5] https://stats.wikimedia.org/#/wikidata.org

Sample Relation Schema with Wikidata Mappings

```
{
  "label": "receivesAward",
  "wd_source": "P166",
  "subj_label": "Movie",
  "obj_label": "Award",
  "properties": [
    {
      "label": "year",
      "wd_source": "P585",
      "datatype": "int"
    }
  ]
}
```

Each property is assigned a datatype. Properties that represent quantities are also given a unit, which is indicated in the property label (e.g., `runtime_minute`) to inform LLMs during graph retrieval.

The authors created all 11 property graph schemas from scratch, with an average time investment of approximately 4 hours per graph. The schemas for all graphs are listed in Table 8.

3.2 Automatic RDF-to-property-graph Transformation

Next, our RDF-to-property-graph engine issues SPARQL queries to Wikidata, using the identifiers from the curated schema to fetch all entities and relations that conform to the schema. For example, the following SPARQL query (simplified for illustration) fetches all Wikidata `award received (P166)` relations where the subject is an instance of `Film (Q11424)` and the object is an instance of `Award (Q618779)`. These relations are then converted into `receivesAward` relations in the target property graph.

Wikidata SPARQL

```
SELECT DISTINCT ?subj ?obj ?statement
WHERE {
  ?subj wdt:P31 wd:Q11424. ?obj wdt:P31 wd:Q618779.
  ?subj p:P166 ?statement. ?statement ps:P166 ?obj.
}
```

The conversion engine further supports datatype conversion, date conversion, unit standardization, rank filtering, and selective entity fetching. Due to space constraints, we only discuss unit standardization and rank filtering and leave the remaining details to Appendix B.6.

Unit standardization. The engine enforces standardized units (*e.g.*, centimeters) on property values that represent quantities by converting all values that can be converted. This eliminates the need for unit conversion during retrieval and ensures accuracy of aggregation queries.

Rank filtering. Wikidata uses rank (`wikibase:rank`) to indicate the reliability or recency of a relation, which can be one of three values: preferred, normal, or deprecated. For time-sensitive relations (*e.g.*, the president of the United States), the currently valid entries are typically marked as preferred, while previously valid entries are marked as normal with additional properties indicating the relevant time period. For time-sensitive relations with time qualifiers in the target schema, we fetch all Wikidata relations with non-deprecated ranks, along with their starting and ending times. For other types of relations, we fetch only the highest-ranked available relation.

The results are aggregated into the final property graph and stored in a DBMS-independent JSON format and ultimately deployed using a custom Neo4j Docker image.

4 Constructing Questions

With 11 large-scale property graphs as a testbed, the next step is to construct questions that require graph retrieval to be answered. We adopt text-to-Cypher as the primary graph retrieval approach and develop a benchmark consisting of (question, Cypher) pairs, but also record execution results of the Cypher queries as answers so that it can serve as a generic KBQA benchmark for evaluating non-Cypher-based approaches.

A text-to-Cypher task can be formulated as follows: given the graph schema and a natural language question as input, output an executable Cypher query that returns the desired answer[6].

Our Text-to-Cypher task generation pipeline involves two main steps: 1) generating initial (question, Cypher) pairs with diverse graph patterns using templates, and 2) rewriting the questions to sound more natural using a LLM.

4.1 Preliminaries: Cypher Query Structure

A Cypher query typically begins with a `MATCH` clause, which identifies the subgraphs that match the specified graph pattern. Following this, the remaining Cypher clauses perform various transformations—such as filtering, ranking, or aggregation—to generate the desired result. For simplicity, we refer to all clauses that follow the `MATCH` clause as the `RETURN` clause, which may include `WHERE` , `WITH`, `ORDER BY` and `RETURN` clauses.

[6]We require the Cypher query to produce the final answer on its own, thus eliminating the need for an additional answer generation step with an LLM as in a standard RAG pipeline.

Pattern/Template	Sample Question	Cypher Query
(Basic MATCH)	$Q1$. What are the names of terrorist attacks that occurred before March 13th, 1997? (*terrorist attack*)	`MATCH (n:TerroristAttack) WITH DISTINCT n WHERE n.date < date('1997-03-13') RETURN n.name`
Optional Match (Special MATCH)	$Q10$. Provide the names of all aircraft models manufactured by ATR, along with the number of flight accidents each has been involved in. (*flight accident*)	`MATCH (n:AircraftModel)-[r1:manufacturedBy]-> (m1:AircraftManufacturer {name: 'ATR'}) OPTIONAL MATCH (n:AircraftModel)<- [r0:involves]-(m0:FlightAccident) WITH n, count(DISTINCT m0) AS num RETURN n.name, num`
AGGREGATE (RETURN Template)	$Q18$. What is the average longest lifespan of taxa that feed on Leporidae? (*biology*)	`MATCH (n:Taxon)-[r0:feedsOn]->(m0:Taxon {name: 'Leporidae'}) WITH DISTINCT n RETURN avg(n.longest_lifespan_years)`

Table 1: A basic MATCH pattern, a special MATCH pattern, and a RETURN template, along with sample questions. The nodes in purple denote the answer entities. Square nodes denote all entities of a particular type, while circular nodes represent named entities. Nodes and edges with dashed lines are optional. The complete list of patterns are provided in Table 10 and Table 11.

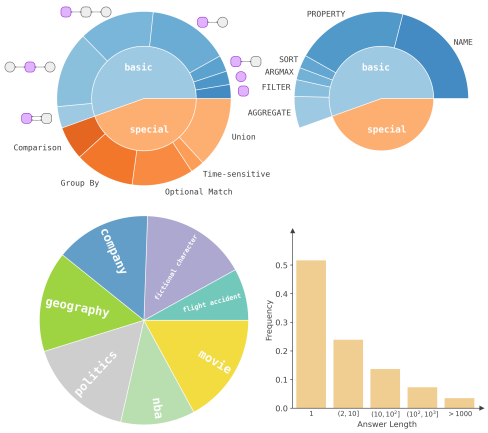


Figure 3: Distribution of graph matching patterns, RETURN templates, domains, and answer lengths (number of rows in the answer) in the CypherBench test set.

4.2 Graph Pattern Design

At the core of graph retrieval is the task of locating the subgraph relevant to the query, which is a fundamental feature of all mainstream graph database query languages (*e.g.*, MATCH clauses in Cypher, WHERE clauses in SPARQL, etc.). To ensure a balanced distribution across various graph matching patterns, we adopt a template-based generation approach rather than crowd-sourcing (as shown in Table 5, even crowd-sourced multi-hop QA benchmarks like HotpotQA tend to be biased toward only a few types of graph matching patterns).

Graph patterns can be categorized based on the isomorphism structure of an undirected graph (see Table 1 for sample patterns and Table 10 for the complete notations). As shown in Table 10, we define seven basic graph patterns, covering all possible isomorphism structures with a single answer node and up to two edges. Additionally, we design five special graph patterns that cover comparison, grouping, optional matching, time-sensitive queries, and union.

We compare the graph patterns covered by representative benchmark in Table 5. A notable observation is that most existing KBQA benchmarks overlook *global queries*, which we define as queries without any specific named entities. These can range from simple listing queries like "$Q13$. List the names of all teams" to more complex ones like "$Q7$. What are the unique countries of citizenship of individuals who both wrote and acted in the same movie?" . The answers to these global queries typically depend on a large number of documents and cannot be easily handled by standard RAG approaches.

4.3 Text-to-Cypher Task Generation

4.3.1 MATCH Clause Instantiation

For each graph pattern, we create multiple Cypher MATCH clause templates by enumerating all possible edge directions, with each MATCH template paired with a human-written question template. For example, one of the (question, Cypher) template for pattern is ("`MATCH (n)-[r0]->(m0<name>)`", "`${n_LABEL} that ${r0_LABEL} ${m0_name}`"). Next, the template is instantiated by sampling entity types for node variables (`n`, `m0`), relation types for edge variables (`r0`), and entity names for named nodes (`m0`). We accomplish this by executing a special Cypher query on a sampled subgraph, which ensures that the instantiated MATCH clause returns non-empty results.

4.3.2 `RETURN` Clause Instantiation

Each instantiated `MATCH` clause in the special categories is paired with its dedicated `RETURN` clause template, while those in the basic categories are paired with one of the six templates (showns in Table 11) that covers basic property fetching (`NAME`, `PROPERTY`), ranking (`SORT`), filtering (`WHERE`) and aggregation (`AGGREGATE`, `ARGMAX`). The `RETURN` clause template is instantiated by sampling properties, ranking orders, comparison operators (*e.g.*, $\leq$, $\neq$) and aggregate functions (*e.g.*, `min`, `avg`, `count`). We take the datatype into account when sampling operators and aggregate functions to ensure the question is realistic. For instance, we do not allow $\leq$ on string properties or `avg` on dates. Similar to `MATCH` clauses, each `RETURN` clause also has a textual template that is instantiated and combined with the one for the `MATCH` clause to form the complete question.

One subtle design choice we made is to ensure the Cypher always returns literal values, such as entity names, instead of node objects. This allows the benchmark to be used to evaluate non-Cypher graph retrieval methods in the future.

The questions are split into training and test sets by domain, with 4 graphs allocated for training and 7 graphs for testing (Table 7). We remove Cypher queries that produce more than 10^5 records or take more than 30 seconds to execute. The distribution of the test set across four dimensions, as shown in Figure 3, demonstrates that the benchmark is both diverse and balanced.

4.3.3 Detecting Semantically Unrealistic Questions

Blind sampling often results in semantically unrealistic or uninteresting questions, such as "Who is married to someone married to Rhaenyra Targaryen?" This issue has been noted in previous KBQA studies (Su et al., 2016), where it was addressed using heuristics that might incorrectly exclude realistic questions. In this work, we take a more systematic approach by modeling the cardinality and participation characteristics of relationships:

Cardinality. A directional relationship[7] can have one of four cardinalities: one-to-one, one-to-many, many-to-one, or many-to-many. For example, `hasSpouse` represents a one-to-one relationship. Cardinality is used to detect unrealistic groupings—groupings that would always result in a single member per group (*e.g.*, "For each character, return the number of fathers."), as well as unnecessary consecutive inverse relations (as in the `hasSpouse` example).

Participation. The participation of the subject or object in a relationship describes whether its entity instances are always associated with that relationship. Participation can either be total (*e.g.*, `Movie` in `directedBy`, assuming every movie has a director) or partial (*e.g.*, `Movie` in `receivesAward`, as not all movies receive awards). We use participation to detect redundant conditions (*e.g.*, "List all movies directed by someone").

Entailment. A relationship can imply another (e.g., `hasFather` implies `hasParent`). This information is also used to detect redundant conditions.

The characteristics of cardinality, participation, and entailment are manually determined rather than derived from the data, due to the presence of missing data. They are also not enforced as constraints on the data for the same reason.

4.4 Question Rewriting and Verification

We employ LLMs to rewrite the template-generated questions into more natural-sounding questions and to diversify their phrasing. However, we intentionally preserve entity names and string values to avoid introducing the additional complexity of entity linking. This design choice allows us to evaluate LLMs directly through prompting without introducing external linkers or retrievers, leaving the task of entity linking for future work. Finally, the authors inspect all instances in the test set to ensure they are correct.

5 Evaluation Metrics

5.1 Execution Accuracy (EX)

Execution accuracy, the standard metric in text-to-SQL evaluation, measures whether the results returned by the predicted query match those returned by the ground truth query:

$$\text{EX}(q, \hat{q}) = \mathbb{1}_{V=\hat{V}}(V, \hat{V})$$

where V and $\hat{V}$ are the execution results of the ground-truth and predicted Cypher. The final dataset-level metric is obtained by averaging across all instances.

[7] A relationship here can be considered as an edge in the graph schema—a (subject entity type, relation type, object entity type) triplet.

5.2 Provenance Subgraph Jaccard Similarity (PSJS)

As discussed in subsection 4.2, the core task of graph retrieval is to locate the relevant subgraph using the `MATCH` clause. While a LLM might generate the correct `MATCH` clause, it can make subtle mistakes such as returning node objects instead of entity names, or including an extra column that was not requested by the question. In other cases, the `MATCH` clause might be partially correct, either missing or including a few extra entities. All these scenarios would result in zero execution accuracy.

We propose Provenance Subgraph Jaccard Similarity (PSJS) as an isolated measure of the subgraph matching performance. We define the *provenance subgraph* as the subgraph matched by the `MATCH` clause, which can be obtained by pairing the `MATCH` clause with `RETURN *`. For example, the provenance subgraph for "$Q18$. What is the average longest lifespan of taxa that feed on Leporidae?" would include the entity Leporidae and all taxa that feed on Leporidae. PSJS is then calculated as the Jaccard similarity — a standard metric for comparing two sets — between the provenance subgraph of the predicted Cypher and that of the ground truth Cypher:

$$\text{PSJS}(q, \hat{q}) = \frac{G \cap \hat{G}}{G \cup \hat{G}}$$

where G and $\hat{G}$ are the provenance subgraphs of the ground-truth and predicted Cypher.

As another example, a predicted Cypher query that satisfies only one condition in a `UNION` query would receive an execution accuracy of 0 and a PSJS score equal to the fraction of correctly retrieved nodes.

6 Experiments

Due to space constraints, we provide the experiment details in Appendix B.8.

6.1 Main Results

As shown in Table 2, the best-performing model `claude3.5-sonnet` achieves an execution accuracy of 61.58% and a PSJS of 80.85%, with `gpt-4o` performing slightly worse. The highest-performing open-source model reaches only 41.87% execution accuracy, while smaller models in the <10B parameter range achieve less than 20% execution accuracy. These results highlight the difficulty of CypherBench.

Model	EX (%)	PSJS (%)	Exec. (%)
Open-source LLMs (<10B)			
`llama3.2-3b`	11.20	17.33	86.46
`llama3.1-8b`	18.82	30.98	90.67
`gemma2-9b`	18.61	30.67	68.57
Open-source LLMs (10-100B)			
`mixtral-8x7b`	19.21	37.01	59.33
`qwen2.5-72b`	41.87	56.39	86.84
`llama3.1-70b`	38.84	54.79	92.25
Proprietary LLMs			
`yi-large`	33.82	47.21	83.52
`gemini1.5-flash-001`	25.26	41.46	83.65
`gemini1.5-pro-001`	39.95	57.70	86.03
`gpt-4o-mini-20240718`	31.43	45.91	87.39
`gpt-4o-20240806`	60.18	76.87	94.93
`claude3.5-sonnet-20240620`	**61.58**	**80.85**	**96.34**

Table 2: Zero-shot execution accuracy (EX), provenance subgraph jaccard similarity (PSJS) and executable percentage (Exec.) on the CypherBench test set.

Furthermore, the low PSJS scores across most models indicate that the challenges are not merely due to basic formatting errors (*e.g.*, including an extra column or duplicate entries) but stem from fundamental graph matching issues. In addition, smaller models within the same family perform significantly worse (as seen in the `gpt-`, `llama-`, and `gemini-` series), highlighting the benchmark's effectiveness in differentiating LLM capabilities.

6.2 Performance Across Graph Matching Patterns

Next, we analyze the performance breakdown across various dimensions[8]. Figure 5 shows the execution accuracy and PSJS scores of these models across various graph matching patterns. Among the basic categories, all models exhibit similar trends—achieving near-perfect accuracy on pattern ● while performing worst on pattern ●—●. The PSJS chart, which evaluates graph matching alone, shows a consistent gradual decline in performance as the graph patterns include more relations.

Comparing the EX and PSJS charts provides insight into whether errors are caused by graph matching. For example, all models achieve near-perfect PSJS scores but low EX on pattern ●. Upon manual inspection, we identified that most errors for this pattern result from incorrect deduplication—merging distinct entities that have the same name.

Within the special categories, models display varying weaknesses across different patterns. For

[8]For this analysis, we focus on `gpt-4o`, `claude3.5-sonnet`, `qwen2.5-72b`, and `llama3.1-70b`, which represent the top 2 performing proprietary LLMs and the top 2 open-source LLMs.

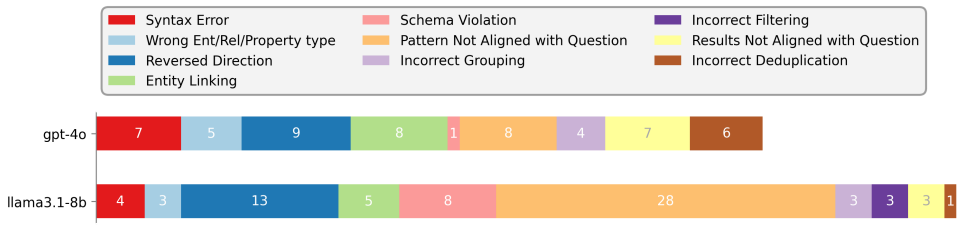


Figure 4: Distribution of errors made by `gpt-4o` and `llama3.1-8b` on 50 randomly sampled incorrect predictions. Note that a model might make multiple errors on one instance.

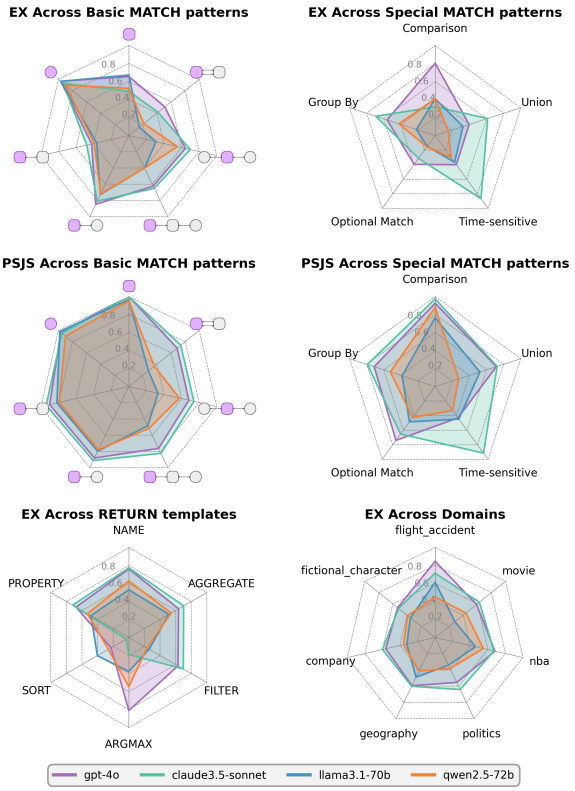


Figure 5: Performance across basic and special `MATCH` patterns, `RETURN` templates and domains.

instance, `gpt-4o` struggles with time-sensitive questions, whereas `claude3.5-sonnet` performs poorly on comparison questions.

6.3 Performance Across `RETURN` Templates

The models also demonstrate different weaknesses depending on the template. Interestingly, `claude3.5-sonnet` achieves near-zero accuracy on `SORT` questions. Upon closer inspection, we observed that it frequently includes the variable used to rank the entities as an extra column, even though the questions only request the entity names, thus resulting in zero execution accuracy (however, PSJS is 1.0 in most of these cases since it is designed to be independent of the `RETURN` clause).

6.4 Error Analysis

We further conduct an error analysis to investigate the types of errors made by LLMs. Specifically, we focus on `gpt-4o` and `llama3.1-8b` to examine whether smaller models behave differently from larger ones. For each model, we randomly sample 50 task instances where they make incorrect predictions and annotate the errors observed in each instance. The error category taxonomy is developed during the annotation process (Appendix B.5 shows the definitions of each error category with examples). The distribution of error categories is shown in Figure 4.

Both models exhibit diverse errors spanning 10 distinct categories. Some error categories that frequently occur in both models include `Reversed Direction` (where the model reverses the direction of a relation), `Entity Linking` (where the entity name does not correspond to the intended entity in the database), and `Pattern Not Aligned with Question` (where the `MATCH` pattern conforms to the schema but does not align with the question's intent). Compared to `gpt-4o`, `llama3.1-8b` makes a significantly higher proportion of schema violation errors, indicating its inferior schema following capabilities.

7 Conclusion

Since its inception, Wikidata has received over 2 billion edits by users worldwide and continues to be actively maintained by over 42,000 editors in the past year, making it one of the most comprehensive knowledge sources available today. This study offers a viable pathway for integrating full-scale modern knowledge graphs like Wikidata with LLMs. The techniques we proposed, along with the numerous design choices made throughout this study, are all centered around accomplishing this goal. We believe our work offers new research opportunities in the areas of knowledge graphs and large-scale graph retrieval.

8 Limitations

One of the limitations of CypherBench arises from its template-based generation approach. Despite our substantial efforts to make the templates as comprehensive as possible, they still fail to cover certain graph querying features, such as variable-length path matching. Additionally, due to limited resources, the training set has not undergone human verification and might contain annotation errors. Furthermore, some features of our RDF-to-property-graph conversion engine are currently tailored for Wikidata. Extending it to support other RDF knowledge graphs, such as DBpedia, would require additional effort.

References

Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. 2019. Rdf and property graphs interoperability: Status and issues. *AMW*, 2369:1–11.

Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. 2020. Mapping rdf databases to property graph databases. *IEEE Access*, 8:86091–86110.

Jinheon Baek, Alham Fikri Aji, Jens Lehmann, and Sung Ju Hwang. 2023. Direct fact retrieval from knowledge graphs without entity linking. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10038–10055, Toronto, Canada. Association for Computational Linguistics.

Debayan Banerjee, Pranav Ajit Nair, Jivat Neet Kaur, Ricardo Usbeck, and Chris Biemann. 2022. Modern baselines for sparql semantic parsing. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2260–2265.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Baltimore, Maryland. Association for Computational Linguistics.

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

Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2022. KQA pro: A dataset with explicit compositional programs for complex question answering over knowledge base. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6101–6119, Dublin, Ireland. Association for Computational Linguistics.

Harrison Chase. 2022. Langchain. `https://github.com/langchain-ai/langchain`. Version released on 2022-10-17.

Yongrui Chen, Huiying Li, Yuncheng Hua, and Guilin Qi. Formal query building with query structure prediction for complex question answering over knowledge base.

Keyuan Cheng, Gang Lin, Haoyang Fei, Lu Yu, Muhammad Asif Ali, Lijie Hu, Di Wang, et al. 2024. Multi-hop question answering under temporal knowledge editing. *arXiv preprint arXiv:2404.00492*.

Philipp Christmann, Rishiraj Saha Roy, and Gerhard Weikum. 2022a. Beyond ned: fast and effective search space reduction for complex question answering over knowledge bases. In *Proceedings of the fifteenth ACM international conference on web search and data mining*, pages 172–180.

Philipp Christmann, Rishiraj Saha Roy, and Gerhard Weikum. 2022b. Conversational question answering on heterogeneous sources. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 144–154.

Philipp Christmann, Rishiraj Saha Roy, and Gerhard Weikum. 2023. Explainable conversational question answering over heterogeneous sources via iterative graph neural networks. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 643–653.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over Freebase with multi-column convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 260–269, Beijing, China. Association for Computational Linguistics.

Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. 2019. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019, Proceedings, Part II 18*, pages 69–78. Springer.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A

graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Vincent Emonet, Jerven Bolleman, Severine Duvaud, Tarcisio Mendes de Farias, and Ana Claudia Sima. 2024. Llm-based sparql query generation from natural language over federated knowledge graphs. *arXiv preprint arXiv:2410.06062*.

Yanlin Feng, Xinyue Chen, Bill Yuchen Lin, Peifeng Wang, Jun Yan, and Xiang Ren. 2020. Scalable multi-hop relational reasoning for knowledge-aware question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1295–1309, Online. Association for Computational Linguistics.

Yanlin Feng, Adithya Pratapa, and David Mortensen. 2023. Calibrated seq2seq models for efficient and generalizable ultra-fine entity typing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 15550–15560, Singapore. Association for Computational Linguistics.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv preprint arXiv:2007.08970*.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond iid: three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, pages 3477–3488.

Aibo Guo, Xinyi Li, Guanchen Xiao, Zhen Tan, and Xiang Zhao. 2022. Spcql: A semantic parsing dataset for converting natural language into cypher. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 3973–3977.

Olaf Hartig. 2014. Reconciliation of rdf* and property graphs. *arXiv preprint arXiv:1409.3288*.

Daniel Scott Himmelstein, Antoine Lizee, Christine Hessler, Leo Brueggeman, Sabrina L Chen, Dexter Hadley, Ari Green, Pouya Khankhanian, and Sergio E Baranzini. 2017. Systematic integration of biomedical knowledge prioritizes drugs for repurposing. *Elife*, 6:e26726.

Seyed Amir Hosseini Beghaeiraveri, Jose Emilio Labra Gayo, Andra Waagmeester, Ammar Ammar, Carolina Gonzalez, Denise Slenter, Sabah Ul-Hasan, Egon Willighagen, Fiona McNeill, and Alasdair JG Gray. 2023. Wikidata subsetting: Approaches, tools, and evaluation. *Semantic Web*, pages 1–27.

Filip Ilievski, Daniel Garijo, Hans Chalupsky, Naren Teja Divvala, Yixiang Yao, Craig Rogers, Rongpeng Li, Jun Liu, Amandeep Singh, Daniel Schwabe, et al. 2020. Kgtk: a toolkit for large knowledge graph manipulation and analysis. In *The Semantic Web–ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part II 19*, pages 278–293. Springer.

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

Jiho Kim, Sungjin Park, Yeonsu Kwon, Yohan Jo, James Thorne, and Edward Choi. 2023. FactKG: Fact verification via reasoning on knowledge graphs. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16190–16206, Toronto, Canada. Association for Computational Linguistics.

Liubov Kovriguina, Roman Teucher, Daniil Radyush, Dmitry Mouromtsev, N Keshan, S Neumaier, AL Gentile, and S Vahdati. 2023. Sparqlgen: One-shot prompt-based approach for sparql query generation. In *SEMANTiCS (Posters & Demos)*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Jaebok Lee and Hyeonjeong Shin. 2024. Sparkle: Enhancing sparql generation with direct kg integration in decoding. *arXiv preprint arXiv:2407.01626*.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. 2015. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024a. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

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

Zhuoyang Li, Liran Deng, Hui Liu, Qiaoqiao Liu, and Junzhao Du. 2024b. Unioqa: A unified framework for knowledge graph question answering with large language models. *arXiv preprint arXiv:2406.02110*.

Yuanyuan Liang, Keren Tan, Tingyu Xie, Wenbiao Tao, Siyuan Wang, Yunshi Lan, and Weining Qian. 2024.

Aligning large language models to a domain-specific graph database. *arXiv preprint arXiv:2402.16567*.

Shicheng Liu, Sina Semnani, Harold Triedman, Jialiang Xu, Isaac Dan Zhao, and Monica Lam. 2024. SPINACH: SPARQL-based information navigation for challenging real-world questions. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15977–16001, Miami, Florida, USA. Association for Computational Linguistics.

LlamaIndex Team. 2023. Introducing the property graph index: A powerful new way to build knowledge graphs with llms. Accessed: 2024-10-24.

LINHAO LUO, Yuan-Fang Li, Reza Haf, and Shirui Pan. 2024. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations*.

Kaixin Ma, Hao Cheng, Xiaodong Liu, Eric Nyberg, and Jianfeng Gao. 2022. Open domain question answering with a unified knowledge interface. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1605–1620, Dublin, Ireland. Association for Computational Linguistics.

Shota Matsumoto, Ryota Yamanaka, and Hirokazu Chiba. 2018. Mapping rdf graphs to property graphs. *arXiv preprint arXiv:1812.01801*.

Seungwhan Moon, Pararth Shah, Anuj Kumar, and Rajen Subba. 2019. OpenDialKG: Explainable conversational reasoning with attention-based walks over knowledge graphs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 845–854, Florence, Italy. Association for Computational Linguistics.

Lunyiu Nie, Shulin Cao, Jiaxin Shi, Jiuding Sun, Qi Tian, Lei Hou, Juanzi Li, and Jidong Zhai. 2022. 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*, pages 5848–5865, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Barlas Oguz, Xilun Chen, Vladimir Karpukhin, Stan Peshterliev, Dmytro Okhonko, Michael Schlichtkrull, Sonal Gupta, Yashar Mehdad, and Scott Yih. 2022. UniK-QA: Unified representations of structured and unstructured knowledge for open-domain question answering. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1535–1546, Seattle, United States. Association for Computational Linguistics.

Makbule Gulcin Ozsoy, Leila Messallem, and Jon Besga. 2024. Introducing the neo4j text2cypher (2024) dataset. Published on the Neo4j Developer Blog.

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

Ahmad Sakor, Kuldeep Singh, Anery Patel, and Maria-Esther Vidal. 2020. Falcon 2.0: An entity and relation linking tool over wikidata. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 3141–3148.

Alberto Santos, Ana R Colaço, Annelaura B Nielsen, Lili Niu, Maximilian Strauss, Philipp E Geyer, Fabian Coscia, Nicolai J Wewer Albrechtsen, Filip Mundt, Lars Juhl Jensen, et al. 2022. A knowledge graph to interpret clinical proteomics data. *Nature biotechnology*, 40(5):692–702.

Alexander Schätzle, Martin Przyjaciel-Zablocki, Thorsten Berberich, and Georg Lausen. 2016. S2x: graph-parallel querying of rdf with graphx. In *Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops, Big-O (Q) and DMAH, Waikoloa, HI, USA, August 31–September 4, 2015, Revised Selected Papers 1*, pages 155–168. Springer.

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gür, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for QA evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 562–572, Austin, Texas. Association for Computational Linguistics.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.

Dominik Tomaszuk. 2016. Rdf data in property graph model. In *Research Conference on Metadata and Semantics Research*, pages 104–115. Springer.

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

Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. 2024. Stark: Benchmarking llm retrieval on textual and relational knowledge bases. *arXiv preprint arXiv:2404.13207*.

Silei Xu, Shicheng Liu, Theo Culhane, Elizaveta Pertseva, Meng-Hsi Wu, Sina Semnani, and Monica Lam. 2023. Fine-tuned LLMs know more, hallucinate less

with few-shot sequence-to-sequence semantic parsing over Wikidata. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5778–5791, Singapore. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. QA-GNN: Reasoning with language models and knowledge graphs for question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546, Online. Association for Computational Linguistics.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China. Association for Computational Linguistics.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Yang Wang, Zhiguo Wang, and Bing Xiang. 2022. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases. In *The Eleventh International Conference on Learning Representations*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. 2022. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5773–5784, Dublin, Ireland. Association for Computational Linguistics.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Ziije Zhong, Linqing Zhong, Zhaoze Sun, Qingyun Jin, Zengchang Qin, and Xiaofan Zhang. 2024. Synthet2c: Generating synthetic data for fine-tuning large language models on the text2cypher task. *arXiv preprint arXiv:2406.10710*.

Yuhang Zhou, Yu He, Siyu Tian, Yuchen Ni, Zhangyue Yin, Xiang Liu, Chuanjun Ji, Sen Liu, Xipeng Qiu, Guangnan Ye, et al. 2024. $ R^3$-NL2GQL: A Model Coordination and Knowledge Graph Alignment Approach for NL2GQL. *arXiv preprint arxiv:2311.01862*.

A Related Work

A.1 KBQA and Graph Retrieval Methods

Our work is related to knowledge base question answering (KBQA) as CypherBench serves as a benchmark for evaluating KBQA and graph retrieval methods.

We categorize existing KBQA and graph retreival methods into two types (see Table 3): approximate retrieval methods, which identify top relevant elements based on some notion of relevance to the question, and precise retrieval methods, which retrieve exactly what the question specifies by executing a formal language query.

The most common approximate retrieval method involves retrieving the k-hop neighborhood of the entities mentioned in the question (Dong et al., 2015; Oguz et al., 2022; Christmann et al., 2022a,b, 2023; Cheng et al., 2024; Zhang et al., 2022; Yasunaga et al., 2021; Feng et al., 2020; LUO et al., 2024; Jiang et al., 2023). Another line of work verbalizes entities or relations into text and uses embedding-based methods to retrieve the top-k most relevant elements (Wu et al., 2024; Baek et al., 2023; Ma et al., 2022; Yu et al., 2022). The fundamental limitation of these methods is their inability to handle questions involving a large number of entities (*i.e.*, global queries or complex aggregation queries), as they require processing all the retrieved information during the answer generation step. Additionally, these methods usually rely on expensive in-memory operations or require embedding the entire graph, which limits their feasibility when applied to full-scale modern knowledge graphs which typically contain billions of triples.

Precise retrieval methods translate the question into a formal language query that fetches exactly what the question asks for. However, most approaches in this category are based on custom logical forms, which are either transpiled into actual database queries or executed by a custom engine (Gu et al., 2021; Berant et al., 2013; Berant and Liang, 2014; Talmor and Berant, 2018; Yih et al., 2015; Su et al., 2016; Chen et al.; Cao et al., 2022; Nie et al., 2022; Li et al., 2023). These custom logical forms are easier to generate for pre-LLM models due to their simpler syntax, but often lack support for certain graph querying features like grouping and variable-length path matching. The ones that are executed by custom engines also face limitations in scalability and real-world applicability compared to standard database query languages. For example, the recently proposed KoPL (Cao et al., 2022) queries are executed by loading the entire graph into memory and processing it in-memory, which makes it impractical to handle graphs of a size comparable to Wikidata. While some recent works use LLMs to directly generate graph database queries (*e.g.*, SPARQL or Cypher) (Kovriguina et al., 2023; Banerjee et al., 2022; Emonet et al., 2024; Lee and Shin, 2024; Li et al., 2024b), they often make simplifications such as assuming that identifiers are provided or working with smaller graphs. Notably, the recently introduced SPINACH (Liu et al., 2024) operates over full Wikidata using an agentic workflow.

A.2 Text-to-Query and KBQA Benchmarks

CypherBench takes the form of a text-to-query benchmark, consisting of databases along with (question, database query) pairs. KBQA benchmarks represent a specific type of text-to-query benchmarks, where the databases are knowledge graphs, and the queries are graph database queries. In Table 4, we compare CypherBench with current representative text-to-query benchmarks.

Looking at the Schema Size and Data Size columns provides insights into the complexity of the databases in existing text-to-query benchmarks, both in terms of their schemas and stored data. Most existing KBQA benchmarks (Berant et al., 2013; Moon et al., 2019; Dubey et al., 2019; Gu et al., 2021; Cao et al., 2022) are predominantly based on text-to-SPARQL over RDF knowledge graphs. However, as discussed in section 2, the massive schema of RDF knowledge graphs poses significant challenges when using these benchmarks to evaluate LLMs in zero-shot settings. In contrast, the graphs in CypherBench have a schema size comparable to those in text-to-SQL benchmarks, while still encompassing up to 7 million entities.

In recent years, several benchmarks focusing on text-to-Cypher or text-to-nGQL[9] have been proposed (Nie et al., 2022; Guo et al., 2022; Zhou et al., 2024; Liang et al., 2024; Zhong et al., 2024). MetaQA-Cypher (Nie et al., 2022) and SpCQL (Guo et al., 2022) are the earliest efforts to develop text-to-Cypher benchmarks. MetaQA-Cypher is adapted from MetaQA (Zhang et al., 2018), a KBQA dataset built on a movie knowledge graph, with Cypher queries annotated using rule-based

[9] nGQL is the query language for NebulaGraph, a property graph database.

Graph Retrieval Method	Papers / Open-source Projects
Approximate Retrieval	
Entity linking + k-hop neighbourhood	LlamaIndex (LlamaIndex Team, 2023), MCCNN (Dong et al., 2015), UniK-QA (Oguz et al., 2022), CLOCQ (Christmann et al., 2022a), Convinse (Christmann et al., 2022b), Explaignn (Christmann et al., 2023), Temple-MQA (Cheng et al., 2024), Subgraph Retriever (Zhang et al., 2022), QA-GNN (Yasunaga et al., 2021), MHGRN (Feng et al., 2020), RoG (LUO et al., 2024), UniKGQA (Jiang et al., 2023)
Top-k entities / relations / pseudo-doc	STaRK (Wu et al., 2024), DiFaR (Baek et al., 2023), UDT-QA (Ma et al., 2022), DecAF (Yu et al., 2022)
Precise Retrieval	
Text-to-SPARQL through intermediate logical form (*e.g.*, λ-DCS, S-expression, etc.)	S-expression (Gu et al., 2021), λ-DCS (Berant et al., 2013; Berant and Liang, 2014), ComplexWebQ (Talmor and Berant, 2018), Staged Query Graph (Yih et al., 2015), Graph Query (Su et al., 2016), Abstract Query Graph (Chen et al.), KoPL (Cao et al., 2022), GraphQ IR (Nie et al., 2022), KB-BINDER (Li et al., 2023)
Text-to-SPARQL	Langchain (Chase, 2022), sparqlgen (Kovriguina et al., 2023), T5-sparql (Banerjee et al., 2022), sparql-llm (Emonet et al., 2024), SPARKLE (Lee and Shin, 2024), WikiSP (Xu et al., 2023), SPINACH (Liu et al., 2024)
Text-to-Cypher (Our focus)	LlamaIndex (LlamaIndex Team, 2023), Langchain (Chase, 2022), UniOQA (Li et al., 2024b)

Table 3: Graph retrieval methods adopted by existing research papers and open-source projects.

Benchmark	Data Source	#graph/db	Avg. Schema Size (per graph/db)	Data Size	LLM Efficient?
Text-to-SQL / relational data					
Spider (Yu et al., 2018)	Wikipedia, etc.	200	5.1 tables, 27.6 columns	400k rows	✓
BIRD-SQL (Li et al., 2024a)	Kaggle, etc.	95	7.3 tables, 54.2 columns	52M rows	✓
Text-to-SPARQL / RDF graphs					
LC-Quad 2.0 (Dubey et al., 2019)	Wikidata	1	12k relation types	114M entities	×
GrailQA (Gu et al., 2021)	Freebase	1	37k relation types	45M entities	×
KQA Pro (Cao et al., 2022)	FB15k-237	1	0.8k relation types	16k entities	×
Text-to-nGQL / property graphs					
R^3-NL2GQL† (Zhou et al., 2024)	OpenKG	3	5.3 relation types, 13 properties	46k entities	✓
Fin/Medi-GQL† (Liang et al., 2024)	OpenKG	2	13 relation types, 38 properties	713k entities	✓
Text-to-Cypher / property graphs					
MetaQA-Cypher (Nie et al., 2022)	OMDb	1	5 relation types, 5 properties	43k entities	✓
SpCQL† (Guo et al., 2022)	OwnThink	1	480k relation types, 1 property	16M entities	×
Neo4j Text2Cypher (2024)	neo4j-graph-examples	-	-	-	✓
CypherBench (ours)	Wikidata	11	7.5 relation types, 18.7 properties	7.8M entities	✓

Table 4: Comparison of representative text-to-query benchmarks. Benchmarks marked by † are non-English (R^3-NL2GQL, FinGQL, MediGQL and spQCL are in Chinese). The column "LLM Efficient?" refers to whether the database schema from the benchmark can fit in the typical context window of LLMs. Existing KBQA benchmarks pose challenges for evaluation in zero-shot settings with LLMs due to the massive schema of RDF knowledge graphs.

methods. SpCQL is based on OwnThink[10], a Chinese encyclopedic knowledge graph. The questions in SpCQL were collected from online forums and annotated with Cypher queries by database professionals. Zhou et al. (2024) constructed a text-to-nGQL dataset over three domain knowledge graphs through a combination of human curation and LLM generation. Liang et al. (2024) instead employed a templated generation approach using eight human-curated templates. However, these benchmarks are restricted to a small number of domains (with the exception of SpCQL), and their questions lack diversity, covering only a limited number of the graph matching patterns in CypherBench (as shown in Table 5).

A parallel effort to create a large-scale text-to-Cypher benchmark is the Neo4j Text2Cypher (2024) dataset[11](Ozsoy et al., 2024). This dataset was developed by cleaning and combining 25 public datasets from Neo4j internal projects, Hugging Face, and academic papers. Compared to previous benchmarks, it is significantly more diverse in terms of domains and question types. However, 49% of the questions in the dataset are not linked

[10]OwnThink is provided in a plain triple format, lacking the notion of entity types as well as entity and relation properties. When stored in Neo4j, it uses a single entity type, `ENTITY`, and a single relation type, `Relationship`, with the actual relation type stored in the `name` property of the relations. Consequently, OwnThink resembles RDF graphs more closely than property graphs.

[11]https://huggingface.co/datasets/neo4j/text2cypher-2024v1

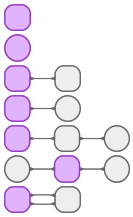
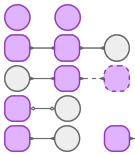
	Natural Questions *(single-hop Text QA)*	HotpotQA *(multi-hop Text QA)*	KQA Pro *(latest text-to-SPARQL)*	MetaQA-Cypher *(latest text-to-Cypher)*
	✓		✓	✓
	✓	✓	✓	✓
		✓	✓	
		✓	✓	
			✓	

Table 5: Graph matching patterns covered by previous question answering datasets. We also include Natural Questions (Kwiatkowski et al., 2019) and HotpotQA (Yang et al., 2018) as representative knowledge-intensive single-hop and multi-hop text QA datasets here. We determined the patterns based on the question curation approach described in the paper and 100 randomly sampled questions.

to any actual graphs (many of these questions are synthetically generated and do not have a corresponding database). Instead, the dataset only provides a textual description of the graph schema for text-to-Cypher generation. This makes it impossible to execute the Cypher queries to evaluate execution-based metrics like EX and PSJS. The remaining 51% of the questions is based on demo graphs from the neo4j-graph-examples repositories[12]. These demo graphs are typically smaller in size and do not comprehensively cover all domain entities, unlike CypherBench, which provides full coverage of domain entities.

A.3 GraphRAG

Recently, Microsoft introduced GraphRAG (Edge et al., 2024) to address corpus-level summarization queries (*e.g.*, "What are the main themes in the dataset?"), which are similar to the global queries explored in this work and cannot be handled by standard top-k embedding-based retrieval methods. At a high level, GraphRAG leverages a centralized knowledge graph to index textual documents, enabling it to handle queries that rely on a large volume of documents. The GraphRAG system has two main stages: knowledge graph construction during indexing time and graph retrieval during query time (Peng et al., 2024). It is worth noting that the original GraphRAG system in (Edge et al., 2024) uses a graph structure slightly different from a typical knowledge graph, where the nodes include entities and communities at multiple levels, with retrieval performed by fetching all communities at a specific level.

Subsequently, LlamaIndex, the leading open-source LLM framework for RAG workflows, introduced the Property Graph Index (LlamaIndex Team, 2023) for general-purpose question answering. It constructs a Neo4j property graph from textual documents using LLMs during indexing time, and conducts graph retrieval via text-to-Cypher during query time. Our work provides the first comprehensive text-to-Cypher benchmark for evaluating graph retrieval, a critical component in GraphRAG.

A.4 Mapping RDF to Property Graphs

Several studies from the semantic web community have explored methods for transforming RDF graphs into property graphs (Hartig, 2014; Tomaszuk, 2016; Schätzle et al., 2016; Angles et al., 2020; Matsumoto et al., 2018)[13]. However, many of these methods require processing the entire RDF dump, which can be computationally expensive for full-size modern RDF graphs like Wikidata. For example, (Hartig, 2014) proposed a two-step process for RDF*, an RDF extension: first, RDF triples are mapped directly to edges in the property graph, and then edges that represent entity properties are transformed into node properties. An exception is (Matsumoto et al., 2018) [14], which adopts an approach similar to ours by transforming RDF into property graphs through executing SPARQL queries over

[12] https://github.com/neo4j-graph-examples

[13] For a more detailed survey, we refer readers to (Angles et al., 2019).

[14] https://github.com/g2glab/g2g

RDF. However, their method lacks key functionalities described in section 3 which are essential for ensuring the output quality.

A.5 Knowledge Graph Subsetting

There are also a few tools developed to extract domain-specific subgraphs of Wikidata or general RDF knowledge graphs to tackle the scalability challenges of modern knowledge graphs[15]. For example, KGTK (Ilievski et al., 2020), WDumper[16], and WDSub[17] are a few such tools (Hosseini Beghaeiraveri et al., 2023). However, these tools also operate by processing the entire RDF dump and produce RDF as output.

B Technical Details

B.1 Graph Matching Patterns and RETURN Templates

The complete list of graph matching patterns is shown in Table 10 and the complete list of `RETURN` templates is shown in Table 11.

B.2 Question Rewriting Prompt

The prompt used to rewrite template-generated questions into more natural-sounding ones is presented in Table 12.

B.3 Text-to-Cypher Prompt

The text-to-Cypher prompt for evaluating LLMs is shown in Table 13.

B.4 Schema Fetching in Neo4j

In a practical text-to-Cypher scenario where only the Neo4j database endpoint (host and port) is provided, the graph schema must be retrieved from the database by executing certain Cypher queries. Neo4j provides built-in procedures such as `db.schema.visualization` and `apoc.meta.data` for this purpose. However, we observed that both methods yield inaccurate results when applied to large graphs: `db.schema.visualization` may return non-existent relationships, while `apoc.meta.data` can miss certain relationships. To address this issue, we use the following queries to retrieve the schemas:

Cypher for fetching entity property schemas

```
MATCH (n)
UNWIND labels(n) AS label
WITH label, keys(n) AS propertyKeys, n
UNWIND propertyKeys AS property
WITH DISTINCT label, property,
     apoc.meta.cypher.type(n[property]) as type
WITH label AS label, apoc.coll.sortMaps(collect({
     property:property, type:type}), 'property') AS
          properties
RETURN label, properties ORDER BY label
```

Cypher for fetching relation property schemas

```
MATCH ()-[r]-()
WITH type(r) AS type, keys(r) AS propertyKeys, r
UNWIND propertyKeys AS property
WITH DISTINCT type, property,
     apoc.meta.cypher.type(r[property]) AS propType
WITH type, apoc.coll.sortMaps(collect({
     property:property, type:propType}), 'property') AS
          properties
RETURN type, properties ORDER BY type
```

Cypher for fetching relation schemas

```
MATCH (n)-[r]->(m)
UNWIND labels(n) AS start UNWIND labels(m) AS end
RETURN DISTINCT start, type(r) as type, end
ORDER BY type, start, end
```

The results are then aggregated and serialized into the format shown in Table 13. While these queries are less efficient than the built-in procedures (approximately 15 times slower than `apoc.meta.data`), they produce complete and accurate schemas deterministically.

B.5 Text-to-Cypher Error Taxonomy

Table 14 shows the detailed definitions of each error category with examples.

B.6 RDF-to-Property-Graph Conversion Engine Details

Datatype conversion. The engine enforces type constraints on property values by converting them into one of the following types and discarding those that cannot be converted: `str`, `int`, `float`, `date`, `list[str]`.

Date conversion. Wikidata stores precision for time values ranging from seconds up to centuries or more (*e.g.*, a historical event might be recorded with century-level precision like 18th century). For date properties, we retrieve the precision using the predicate `wikibase:timePrecision` and keep only those with a precision at least as fine as a date, which are ultimately mapped to Neo4j's `Date` values.

Selective entity fetching. The engine supports fetching only entities linked to certain relations to avoid out-of-memory issues for very broad entity types (e.g., people or organizations). For instance, in the movie graph, instead of fetching all instances of `Person` from Wikidata, the engine limits the

[15] Note that this differs from subgraph retrieval in approximate graph retrieval methods, where a smaller subgraph is extracted for each question.

[16] https://github.com/bennofs/wdumper

[17] https://github.com/weso/wdsub

Graph	Ent.	Rel.	Ent. Types	Rel. Types	Properties	Wikipedia
art	1.1M	1.3M	6	8	16	64.9k
biology	3.7M	7.5M	4	5	8	447.7k
company	581.3k	299.6k	4	6	14	166.3k
fictional character	28.9k	40.5k	4	11	12	8.5k
flight accident	1.7k	2.2k	5	5	25	1.6k
geography	773.5k	903.8k	8	12	19	73.1k
movie	459.4k	1.9M	7	9	21	262.2k
nba	4.3k	19.0k	7	7	27	4.3k
politics	885.2k	1.5M	6	11	25	414.2k
soccer	275.2k	1.1M	6	5	26	206.6k
terrorist attack	1.6k	1.5k	5	4	13	1.3k
Total	7.8M	14.7M	62	83	206	1.7M

Table 6: Statistics of the graphs. *Wikipedia* refers to the number of English Wikipedia articles linked from the entities (roughly the number of entities with Wikipedia articles).

fetch to only those connected to a `Movie` through relations like `directedBy` or `hasCastMember`.

The SPARQL queries issued by the engine are executed against a local Wikidata endpoint loaded with the April 2024 Wikidata dump, allowing us to bypass the time limit of the public endpoint, and the results are aggregated into the final property graph. The transformation time ranges from seconds to hours, depending on the graph size. The graph statistics are shown in Table 6. The property graph is stored in a DBMS-independent JSON format and ultimately deployed using a custom Neo4j Docker container that initializes the data from the JSON file upon startup.

B.7 EX Implementation

Cypher returns results in a tabular format, thus allowing us to borrow the execution accuracy implementation from the text-to-SQL literature. In this work, we adapt the execution accuracy implementation[18] from the Spider (Yu et al., 2018) leaderboard, which considers two tables as identical if one can be transformed into the other through row and column permutations. One difference between Cypher and SQL is that Cypher supports objects (*e.g.*, lists and maps) in query results. We serialize these objects to enable direct comparisons. Note that execution accuracy can also be applied to non-Cypher-based graph retrieval approaches in future research, as long as the approach returns results in the tabular format.

B.8 Experiment Details

We deployed the graphs using a custom Neo4j Docker image on a local server with 1TB memory. Since the Neo4j community edition does not support multiple databases, we ran a separate Docker container for each graph.

To evaluate the zero-shot text-to-Cypher performance of state-of-the-art LLMs, we run a variety of popular LLMs of different sizes on the CypherBench test set[19]. For each task instance, the model was prompted with the question, the graph schema, and a brief instruction. The complete prompt is shown in Table 13. The open-source models and `yi-large` were run using the Fireworks AI API, `gemini1.5` and `claude3.5-sonnet` were run on Google Cloud Vertex AI, while `gpt-` models were run using OpenAI's API. The cost per run is $5.5 for `gpt-4o` and $0.3 for `gpt-4o-mini`.

Finally, the predicted Cypher queries were executed on Neo4j using 8-thread parallelization with a 120-second timeout (4x the maximum execution time of the ground-truth Cypher) to compute the metrics.

C Additional CypherBench Statistics

The graph statistics are shown in Table 6. The question data split is shown in Table 7. The schemas of the 11 property graphs are shown in Table 8 and Table 9.

Split	Graphs	#Questions
Train	*art, biology, soccer, terrorist attack*	8817
Test	*company, fictional character, flight accident, geography, movie, nba, politics*	2488

Table 7: Statistics of the data splits.

[18] https://github.com/taoyds/test-suite-sql-eval

[19] The training set was not used in this study.

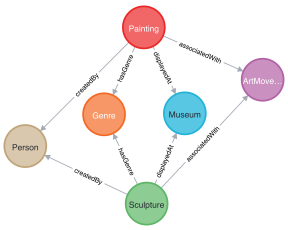
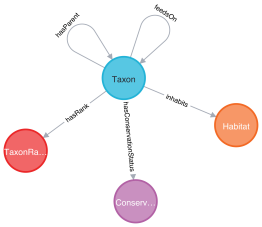
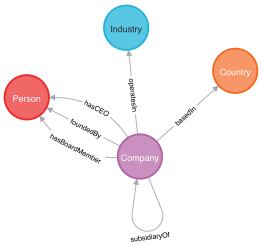
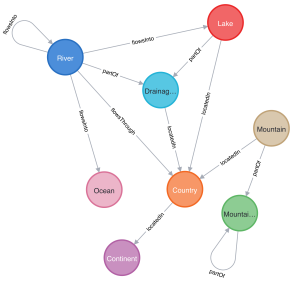
Name	Graph Schema	Entity/Relation Properties
art		Painting name STR creation_year INT country_of_origin STR ArtMovement start_year INT name STR end_year INT Sculpture name STR creation_year INT country_of_origin STR Person place_of_birth STR name STR gender STR date_of_death DATE date_of_birth DATE Museum name STR Genre name STR
biology		Taxon taxon_name STR name STR longest_lifespan_years FLOAT diel_cycle STR avg_gestation_period_days FLOAT ConservationStatus name STR TaxonRank name STR Habitat name STR
company		Company name STR launch_year INT Country name STR Person place_of_birth STR name STR gender STR date_of_death DATE date_of_birth DATE country_of_citizenship LIST[STR] Industry name STR hasBoardMember start_year INT end_year INT hasCEO start_year INT end_year INT
fictional character		Organization name STR Location name STR Character occupation LIST[STR] name STR gender STR creator STR country_of_citizenship LIST[STR] birth_name STR FictionalUniverse name STR inception_year INT creator STR
flight accident		FlightAccident number_of_survivors INT number_of_injuries INT number_of_deaths INT name STR location STR flight_number STR date DATE Airport name STR location STR icao_code STR iata_code STR country STR AircraftModel wingspan_metre FLOAT service_entry DATE range_km FLOAT name STR length_metre FLOAT height_metre FLOAT first_flight DATE AircraftManufacturer name STR launch_year INT country STR Operator name STR launch_year INT country STR
geography		River name STR length_km FLOAT discharge_m3_s FLOAT Lake vertical_depth_m FLOAT name STR area_km2 FLOAT Ocean name STR max_vertical_depth_m FLOAT avg_vertical_depth_m FLOAT area_km2 FLOAT Country name STR capital STR area_km2 FLOAT Continent name STR DrainageBasin name STR area_km2 FLOAT Mountain name STR elevation_m FLOAT MountainRange name STR

Table 8: Schemas of the 11 graphs in the benchmark. The color of the property boxes indicates whether they are entity properties (*e.g.*, name STR) or relation properties (*e.g.*, start_year INT).

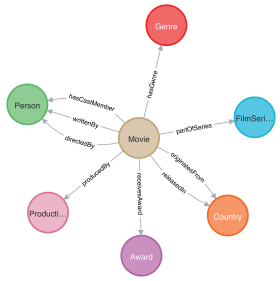
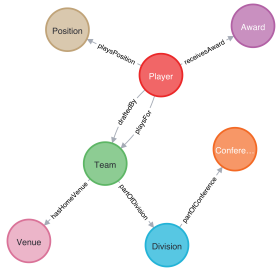
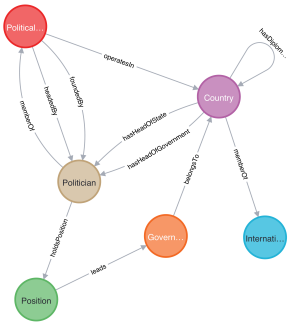
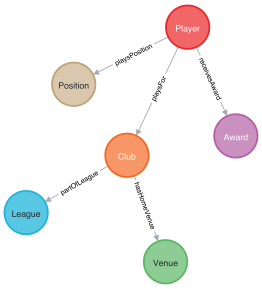
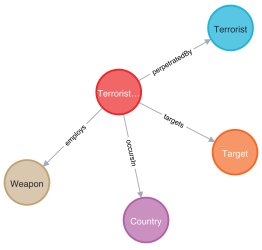
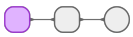
Name	Graph Schema	Entity/Relation Properties
movie		Movie runtime_minute FLOAT original_language LIST[STR] name STR global_box_office_usd FLOAT filming_location LIST[STR] Person place_of_birth STR name STR gender STR date_of_death DATE date_of_birth DATE country_of_citizenship LIST[STR] Genre name STR Country name STR FilmSeries name STR ProductionCompany name STR country STR Award name STR hasCastMember character_role STR receivesAward year INT winners LIST[STR] releasedIn date DATE
nba		Player schools_attended LIST[STR] place_of_birth STR nicknames LIST[STR] name STR mass_kg FLOAT height_cm FLOAT handedness STR gender STR date_of_death DATE date_of_birth DATE country_of_citizenship LIST[STR] Team owners LIST[STR] name STR inception_year INT head_coach STR Venue name STR Division name STR Conference name STR Position name STR Award name STR draftedBy year INT hasHomeVenue start_year INT end_year INT playsFor start_year INT sport_number INT end_year INT receivesAward year INT
politics		GovernmentOrganization name STR Country official_language LIST[STR] name STR founding_date DATE PoliticalParty name STR founding_date DATE Politician schools_attended LIST[STR] place_of_death STR place_of_birth STR name STR gender STR date_of_death DATE date_of_birth DATE country_of_citizenship LIST[STR] Position name STR InternationalOrganization name STR founding_year INT hasHeadOfGovernment start_year INT end_year INT hasHeadOfState start_year INT end_year INT headedBy start_year INT end_year INT holdsPosition start_year INT end_year INT
soccer		Club owners LIST[STR] name STR inception_year INT head_coach STR country STR Venue name STR League name STR Player schools_attended LIST[STR] place_of_birth STR nicknames LIST[STR] name STR mass_kg FLOAT height_cm FLOAT gender STR footedness STR date_of_death DATE date_of_birth DATE country_of_citizenship LIST[STR] Position name STR Award name STR hasHomeVenue start_year INT end_year INT playsFor start_year INT sport_number INT end_year INT receivesAward year INT
terrorist attack		TerroristAttack number_of_injuries INT number_of_deaths INT name STR locations LIST[STR] date DATE Weapon name STR Country name STR Terrorist place_of_birth STR name STR gender STR date_of_birth DATE country_of_citizenship LIST[STR] Target name STR

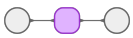
Table 9: (Continued) Schemas of the 11 graphs in the benchmark.

Graph Pattern	Sample Question	Cypher Query
Basic Categories		
	$Q1$. What are the names of terrorist attacks that occurred before March 13th, 1997? (*terrorist attack*)	`MATCH (n:TerroristAttack) WITH DISTINCT n` `WHERE n.date < date('1997-03-13') RETURN n.name`
	$Q2$. What is the discharge rate in cubic meters per second of the Guamués River? (*geography*)	`MATCH (n:River {name: 'Guamués River'}) WITH DISTINCT n` `RETURN n.discharge_m3_s`

	$Q3$. List the players who have received an award, from tallest to shortest. (*soccer*)	`MATCH (n:Player)-[r0:receivesAward]->(m0:Award)`
		`WITH DISTINCT n RETURN n.name`
		`ORDER BY n.height_cm DESC`

	$Q4$. What are the names of taxa that feed on Synsphyronus lathrius? (*biology*)	`MATCH (n:Taxon)-[r0:feedsOn]->(m0:Taxon {name:`
		`'Synsphyronus lathrius'})`
		`WITH DISTINCT n RETURN n.name`

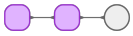
	$Q5$. What are the names of companies that operate in the same industries as Bardel Entertainment? (*company*)	`MATCH (n:Company)-[r0:operatesIn]->(m0:Industry)<-`
		`[r1:operatesIn]-(m1:Company {name: 'Bardel`
		`Entertainment'})`
		`WITH DISTINCT n RETURN n.name`

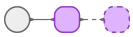
	$Q6$. Who are the point guards who have played for the Toronto Raptors? (*nba*)	`MATCH (n:Player)-[r0:playsFor]->(m0:Team {name: 'Toronto`
		`Raptors'}),(n:Player)-`
		`[r1:playsPosition]->(m1:Position {name: 'point`
		`guard'})`
		`WITH DISTINCT n RETURN n.name`

	$Q7$. What are the unique countries of citizenship of individuals who both wrote and acted in the same movie? (*movie*)	`MATCH (n:Person)<-[r0:writtenBy]-(m0:Movie),`
		`(n:Person)<-[r1:hasCastMember]-(m0:Movie)`
		`WITH DISTINCT n`
		`UNWIND n.country_of_citizenship AS prop`
		`RETURN DISTINCT prop`

Special Categories		


Comparison
	$Q8$. Which painting was created later, Edward George Villiers Stanley, 17th Earl of Derby or Tulip Field in Holland? (*art*)	`MATCH (n:Painting {name: 'Edward George Villiers Stanley,`
		`17th Earl of Derby'}), (m0:Painting {name: 'Tulip`
		`Field in Holland'})`
		`RETURN CASE`
		`WHEN n.creation_year > m0.creation_year THEN n.name ELSE`
		`m0.name`
		`END AS answer`


Group By
	$Q9$. What are the names of the mothers whose children were killed by Cersei Lannister, and how many children did Cersei kill for each of these mothers? (*fictional character*)	`MATCH (n:Character)<-[r0:hasMother]-(m0:Character)-`
		`[r1:killedBy]->(m1:Character {name: 'Cersei`
		`Lannister'})`
		`WITH n, count(DISTINCT m0) AS num`
		`RETURN n.name, num`


Optional Match
	$Q10$. Provide the names of all aircraft models manufactured by ATR, along with the number of flight accidents each has been involved in. (*flight accident*)	`MATCH (n:AircraftModel)-[r1:manufacturedBy]->`
		`(m1:AircraftManufacturer {name: 'ATR'})`
		`OPTIONAL MATCH (n:AircraftModel)<-[r0:involves]-`
		`(m0:FlightAccident)`
		`WITH n, count(DISTINCT m0) AS num`
		`RETURN n.name, num`


Time-sensitive
	$Q11$. Who was the CEO of Mercedes-AMG in the year 1999? (*company*)	`MATCH (n:Person)<-[r0:hasCEO]-(m0:Company {name:`
		`'Mercedes-AMG'})`
		`WHERE r0.start_year <= 1999 AND (r0.end_year >= 1999 OR`
		`r0.end_year IS NULL)`
		`WITH DISTINCT n RETURN n.name`


Union
	$Q12$. Who are the politicians who have either led the Law and Justice party or served as the head of state of Poland at any time? (*politics*)	`CALL {`
		`MATCH (n:Politician)<-[r0:headedBy]-(m0:PoliticalParty`
		`{name: 'Law and Justice'}) RETURN n, m0 AS m`
		`UNION`
		`MATCH (n:Politician)<-[r1:hasHeadOfState]-(m1:Country`
		`{name: 'Poland'}) RETURN n, m1 AS m`
		`}`
		`WITH DISTINCT n RETURN n.name`

Table 10: Sample questions with various graph matching patterns from the benchmark. The nodes in purple denote the answer entities. Square nodes () denote all entities of a particular type, while circular nodes () represent named entities. Nodes and edges with dashed lines () are optional. Edges with diamond arrowheads () indicate relations with time sensitivity constraints.

Return Clause Category	Sample Question	Cypher Query
NAME	$Q13$. List the names of all teams. (*nba*)	`MATCH (n:Team) WITH DISTINCT n RETURN n.name`
PROPERTY	$Q14$. When did the Falcon 50 have its first flight? (*flight accident*)	`MATCH (n:AircraftModel {name: 'Falcon 50'}) WITH DISTINCT n RETURN n.first_flight`
SORT	$Q15$. What are the names of mountains located in Nepal, sorted by elevation from lowest to highest? (*geography*)	`MATCH (n:Mountain)-[r0:locatedIn]->(m0:Country {name: 'Nepal'}) WITH DISTINCT n RETURN n.name ORDER BY n.elevation_m ASC`
ARGMAX	$Q16$. What is the name of the Spider-Verse movie that earned the most at the global box office? (*movie*)	`MATCH (n:Movie)-[r0:partOfSeries]->(m0:FilmSeries {name: 'Spider-Verse'}) WITH DISTINCT n RETURN n.name ORDER BY n.global_box_office_usd DESC LIMIT 1`
FILTER	$Q17$. What are the names of companies in the public relations industry that were launched after 1927? (*company*)	`MATCH (n:Company)-[r0:operatesIn]->(m0:Industry {name: 'public relations'}) WITH DISTINCT n WHERE n.launch_year > 1927 RETURN n.name`
AGGREGATE	$Q18$. What is the average longest lifespan of taxa that feed on Leporidae? (*biology*)	`MATCH (n:Taxon)-[r0:feedsOn]->(m0:Taxon {name: 'Leporidae'}) WITH DISTINCT n RETURN avg(n.longest_lifespan_years)`

Table 11: RETURN clause categories and sample questions from the benchmark. The RETURN clauses shown here apply to any graph pattern in the *basic* categories.

Question Rewriting Prompt

```
Rewrite the given template-generated question in a text-to-Cypher translation task to make it sound more natural:
- Ensure the rewritten question remains semantically equivalent to the original question and the provided Cypher query. Do
      not remove or add any constraints.
  - Pay attention to the direction of the relation pattern (indicated by `->` or `<-`) in the Cypher query.
    For example, `(n:Character)-[r0:hasFather]->(m0:Character)` indicates m0 is the father of n,
    while `(n:Character)<-[r0:hasFather]-(m0:Character)` matches n as the father of m0.
  - Pay attention to the direction of relation in the template-generated question.
    For example `List the names of Character that "Rhaenys Targaryen" hasFather"` means "Rhaenys Targaryen" connects
    to the Character via relation `hasFather`, thus the questions is asking for the father of Rhaenys Targaryen.
    While `List the names of Character that hasFather "Rhaenys Targaryen"` selects the Character that has Rhaenys
         Targaryen as father.
- Ensure the rewritten question is grammatically correct and sounds natural.
- The brackets in the question are parsing hints for the question structure to ensure it is unambiguous. Do not include
      them in the rewritten question.
- For relation types (e.g. hasCastMember), rewrite them to natural language and diversify the expressions. Feel free to
      change from passive to active voice or vice versa.
  - e.g. A hasCastMember B -> B is cast in A, B stars in A, A features B, etc.
- For entity types (e.g. Person, FlightAccident) and properties (e.g. watershed_area_km2) rewrite them to natural language
      and diversify the expressions.
  - e.g. Person -> individual; human; passenger; etc.
  - e.g. TaxonRank -> taxonomic rank; etc.
  - e.g. FlightAccident -> aviation accident; plane crash; etc.
  - e.g. watershed_area_km2 -> size of the watershed in square kilometers; area covered by the watershed in km^2; etc.
- For multi-hop patterns, you can simplify it if the same meaning is preserved.
  - e.g. "teams that belong to a division that belongs to Western Conference" -> "teams in the Western Conference"
  - e.g. "the father of the mother of the person" -> "the person's maternal grandfather"
  - e.g. "the children of the father of the person" -> "the person's paternal siblings"
- For quoted names and string values, remove the quotes but ensure the same text is preserved.
- For numerical values and dates, diversify the expressions, but ensure the same value is preserved.
  - e.g. 1990-07-04 -> July 4th, 1990; 4 July 1990; 07/04/1990 (US format); 4th of July, 1990; etc.
  - e.g. 2000 -> two thousand; 2000; 2,000; 2k; etc.
- For operators (e.g. >, >=, IN, NOT IN, etc.), rewrite them to natural language and diversify the expressions but ensure
      the meaning is preserved.
  - e.g. NOT 'France' IN n.country_of_citizenship -> "is not a citizen of France"
- For "ascending" and "descending", rewrite them to natural language and diversify the expressions.
  - e.g. "the years in descending order" -> "the years from the most recent to the oldest"
- Output only the rewritten question, without any additional explanation.

=== Example ===
Cypher: MATCH (n:Taxon)<-[r0:hasParent]-(m0:Taxon)-[r1:hasConservationStatus]->(m1:ConservationStatus {name: 'Near
      Threatened'}) WITH DISTINCT n RETURN n.name
question: List the names of Taxon that [some Taxon that hasConservationStatus "Near Threatened"] hasParent
rewritten_question: What are the names of parents of taxa with a conservation status of Near Threatened?

Cypher: MATCH (n:Character)<-[r0:hasFather]-(m0:Character),(n:Character)<-[r1:killedBy]-(m0:Character) WITH DISTINCT n
      RETURN n.name
question: List the names of Character that a Character [hasFather and killedBy].
rewritten_question: List the names of fathers who killed their children.

=== Your task ===
Cypher: MATCH (n:Continent) WITH DISTINCT n RETURN n.name
question: List the names of Continent
rewritten_question:
```

Table 12: A sample question rewriting prompt.

Text-to-Cypher Prompt

```
Translate the question to Cypher query based on the schema of a Neo4j knowledge graph.
- Output the Cypher query in a single line, without any additional output or explanation. Do not wrap the query with any
    formatting like ```.
- Perform graph pattern matching in the `MATCH` clause if possible.
- Avoid listing the same entity multiple times in the results. However, if multiple distinct entities share the same name,
    their names should be repeated as separate entries.
- Do not return node objects. Instead, return entity names or properties.

Graph Schema:
{
  "name": "company",
  "entities": [
    {
      "label": "Company",
      "properties": {
        "launch_year": "int", "name": "str"
      }
    },
    {
      "label": "Country",
      "properties": {
        "name": "str"
      }
    },
    {
      "label": "Industry",
      "properties": {
        "name": "str"
      }
    },
    {
      "label": "Person",
      "properties": {
        "country_of_citizenship": "list[str]", "date_of_birth": "date", "date_of_death": "date", "gender": "str", "name":
            "str", "place_of_birth": "str"
      }
    }
  ],
  "relations": [
    {
      "label": "basedIn",
      "subj_label": "Company",
      "obj_label": "Country",
      "properties": {}
    },
    {
      "label": "foundedBy",
      "subj_label": "Company",
      "obj_label": "Person",
      "properties": {}
    },
    {
      "label": "hasBoardMember",
      "subj_label": "Company",
      "obj_label": "Person",
      "properties": {
        "end_year": "int", "start_year": "int"
      }
    },
    {
      "label": "hasCEO",
      "subj_label": "Company",
      "obj_label": "Person",
      "properties": {
        "end_year": "int", "start_year": "int"
      }
    },
    {
      "label": "operatesIn",
      "subj_label": "Company",
      "obj_label": "Industry",
      "properties": {}
    },
    {
      "label": "subsidiaryOf",
      "subj_label": "Company",
      "obj_label": "Company",
      "properties": {}
    }
  ]
}

Question: Provide the names of individuals who have served as board members for companies based in Russia, along with the
    count of such companies for each person.
Cypher:
```

Table 13: A sample text-to-Cypher prompt used in experiments.

Error Category	Question	Ground-truth Cypher	Predicted Cypher

Syntax Error
The Cypher is not executable.
	What are the names and head coaches of teams that have used the Toyota Coliseum as their home venue at some point? (*nba*)	`MATCH (n:Team)-[r0:hasHomeVenue]->(m0:Venue {name: 'Toyota Coliseum'})`
		`WITH DISTINCT n`
		`RETURN n.name, n.head_coach`
			`MATCH (t:Team)-[:hasHomeVenue {start_year: start, end_year: end}]->(v:Venue {name: 'Toyota Coliseum'})`
			`RETURN DISTINCT t.name, t.head_coach`

Wrong Ent/Rel/Property type
The Cypher uses a valid but not intended entity/relation/property type.
	What is the name of the movie from the same country where The Fighter was released, which has the highest global box office earnings? (*movie*)	`MATCH (n:Movie)-[r0:originatesFrom]->(m0:Country)<-[r1:releasedIn]-(m1:Movie {name: 'The Fighter'})`
		`WITH DISTINCT n RETURN n.name`
		`ORDER BY n.global_box_office_usd DESC LIMIT 1`
			`MATCH (m:Movie)-[:releasedIn]->(c:Country)<-[:releasedIn]-(f:Movie {name: 'The Fighter'})`
			`RETURN m.name`
			`ORDER BY m.global_box_office_usd DESC LIMIT 1`

Reversed Direction
A relation is used in the opposite direction.
	What is the name of the subsidiary of The Coca-Cola Company that was launched most recently? (*company*)	`MATCH (n:Company)-[r0:subsidiaryOf]->(m0:Company {name: 'The Coca-Cola Company'})`
		`WITH DISTINCT n RETURN n.name`
		`ORDER BY n.launch_year DESC LIMIT 1`
			`MATCH (parent:Company {name: 'The Coca-Cola Company'})-[:subsidiaryOf]->(subsidiary:Company)`
			`RETURN subsidiary.name ORDER BY subsidiary.launch_year DESC LIMIT 1`

Entity Linking
An entity name in the Cypher does not match the name of the intended entity in the database.
	Which ocean, the Atlantic or the Arctic, has a greater maximum vertical depth? (*geography*)	`MATCH (n:Ocean {name: 'Atlantic Ocean'}),(m0:Ocean {name: 'Arctic Ocean'})`
		`RETURN CASE WHEN n.max_vertical_depth_m > m0.max_vertical_depth_m THEN n.name ELSE m0.name END AS answer`
			`MATCH (o:Ocean) WHERE o.name IN ['Atlantic', 'Arctic']`
			`RETURN o.name`
			`ORDER BY o.max_vertical_depth_m DESC LIMIT 1`

Schema violation
The MATCH pattern does not conform to the graph schema (aside from reversed relations). For example, the Cypher contains a hallucinated entity/relation/property type, or a relation that is invalid between its subject and object.
	What are the names of film series that include movies originating from the Czech Republic? (*movie*)	`MATCH (n:FilmSeries)<-[r0:partOfSeries]-(m0:Movie)-[r1:originatesFrom]->(m1:Country {name: 'Czech Republic'})`
		`WITH DISTINCT n`
		`RETURN n.name`
			`MATCH (m:Movie)-[:originatesFrom]->(c:Country {name: 'Czech Republic'})-[:partOfSeries]->(fs:FilmSeries)`
			`RETURN DISTINCT fs.name`

Pattern Not Aligned with Question
The MATCH pattern conforms to the graph schema, but does not align with the question.
	What are the names of aviation accidents that either departed from Aeroparque Jorge Newbery or were headed to Ingeniero Aeronáutico Ambrosio L.V. Taravella International Airport? (*flight accident*)	`CALL { MATCH (n:FlightAccident)-[r0:departsFrom]->(m0:Airport {name: 'Aeroparque Jorge Newbery'}) RETURN n, m0 as m`
		`UNION MATCH (n:FlightAccident)-[r1:destinedFor]->(m1:Airport {name: 'Ingeniero Aeronáutico Ambrosio L.V. Taravella International Airport'})`
		`RETURN n, m1 as m }`
		`WITH DISTINCT n RETURN n.name`
			`MATCH (fa:FlightAccident)-[:departsFrom|:destinedFor]->(a:Airport)`
			`WHERE a.name IN ['Aeroparque Jorge Newbery', 'Ingeniero Aeronáutico Ambrosio L.V. Taravella International Airport']`
			`RETURN DISTINCT fa.name`

Incorrect Grouping
The grouping operation in the Cypher is incorrect.
	Provide the names of all lakes that are part of the Congo basin, along with the number of countries each lake is located in. (*geography*)	`MATCH (n:Lake)-[r1:partOf]->(m1:DrainageBasin {name: 'Congo basin'})`
		`OPTIONAL MATCH (n:Lake)-[r0:locatedIn]->(m0:Country)`
		`WITH n, count(DISTINCT m0) AS num`
		`RETURN n.name, num`
			`MATCH (l:Lake)-[:partOf]->(d:DrainageBasin {name: 'Congo'}),(l)-[:locatedIn]->(c:Country)`
			`RETURN l.name AS lakeName, count(DISTINCT c) AS numberOfCountries`

Incorrect Filtering
The filtering operation in the Cypher is incorrect.
	What are the names of lakes situated in countries through which the Natara River flows, with an area of less than 390,000 square kilometers? (*geography*)	`MATCH (n:Lake)-[r0:locatedIn]->(m0:Country)<-[r1:flowsThrough]-(m1:River {name: 'Natara'})`
		`WITH DISTINCT n`
		`WHERE n.area_km2 < 390000`
		`RETURN n.name`
			`MATCH (r:River {name: 'Natara'})-[:flowsThrough]->(c:Country),(c)-[:locatedIn]->(co:Continent),(r)-[:flowsInto]->(l:Lake)`
			`WHERE c.area_km2 < 390000`
			`RETURN DISTINCT l.name`

Results Not Aligned with Question
The RETURN clause does not align with the question, resulting in outputs that do not match the intended results of the query.
	What are the unique countries of citizenship for characters who have a father? (*fictional character*)	`MATCH (n:Character)-[r0:hasFather]->(m0:Character)`
		`WITH DISTINCT n UNWIND n.country_of_citizenship AS prop`
		`RETURN DISTINCT prop`
			`MATCH (c:Character)-[:hasFather]->(:Character)`
			`RETURN DISTINCT c.country_of_citizenship AS countries`

Incorrect Deduplication
The Cypher execution results contain the same set of entries as the ground truth, but include entries that should be deduplicated, or missing entries that should be duplicated.
	What are the names of mountains that are part of the Andes mountain range? (*geography*)	`MATCH (n:Mountain)-[r0:partOf]->(m0:MountainRange {name: 'Andes'})`
		`WITH DISTINCT n`
		`RETURN n.name`
			`MATCH (m:Mountain)-[:partOf]->(mr:MountainRange {name: 'Andes'})`
			`RETURN DISTINCT m.name`

Table 14: Definitions and examples for the 10 text-to-Cypher error categories.