LLMs' Understanding of Queries on Property Graphs

Thomas CERESA, Lucca HOOGENBOSCH

February 2025

Abstract

Graph-oriented databases are widely used in domains such as social networks, bioinformatics, and fraud detection. To query them effectively, formal models [4] and languages like Cypher (Neo4j) have been developed [9].

Recent work has explored how large language models (LLMs) can understand and manipulate knowledge graphs (KGs), transforming structured data into natural language and vice versa [6], [10]. While progress has been made, challenges remain, especially when LLMs are tasked with property graph queries. Our work promptly investigates LLMs' ability to comprehend and apply graph queries (Cypher or Natural Language). Eventually, our work attempts to use LLMs so as to write queries.

Keywords: LLM, property graph, reasoning

Contents

1 Introduction

1.1 Context

During the 'Ouverture à la Recherche' course at Claude Bernard University Lyon 1, from October to late February, we focused on studying literature related to encoding property graphs for Large Language Models (LLMs) and developing a framework. Our research was conducted within the LIRIS laboratory, under the guidance of Angela Bonifati (HDR) and Andrea Mauri (PhD) (Database group), to whom we warmly acknowledge. We met with Andrea almost every week, either remotely or in the lab, to discuss relevant literature and our progress on the implementation. Initially, our focus was on prompt engineering for LLMs in the context of property graphs. However, as our work progressed, we gradually shifted towards understanding how LLMs interpret queries on the Incidence Representation of graphs.

1.2 Previous Research

1.2.1 Encoding, Q/A Framework

Previous research has pointed out the relevant encoding of graphs that LLM can comprehend more. The works [6], [10] have shown different ways of translating Knowledge graphs (KG) into natural languages (Figure 1). It points out that the performance of LLMs on graph tasks is heavily influenced by graph structure and the complexity of graph encoding.

These papers also describe the first frameworks which rephrase a question q and a graph G. Let f (resp. g, q) be a LMM (resp. encoding of a graph, rephrasing of a question). Let S be the solution of query Q. Therefore, formally, we want:

$$\max_{g,q} E_{G,Q,S\in D}\left[score_f(g(G), q(Q), S)\right]$$

(over the training dataset D)

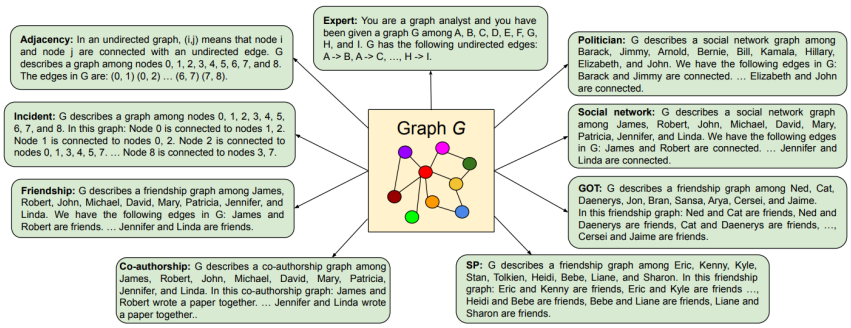

Figure 1: From [6]: Summary of types of encoding

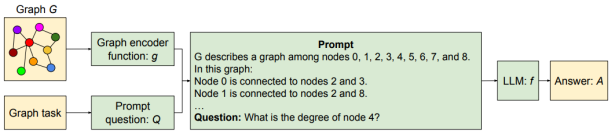

Figure 2: From [6]: inspiration of our framework

1.2.2 Property Graph

Over the years, with the goal of standardizing GQL, released in last April [7], academic research has focused on defining a Property Graph [1], [2] and queries [4].

A property graph is a structure $G = (N, E, \rho, \lambda, \pi)$, where:

- N is a finite set of nodes.
- E is a finite set of edges, such that $N \cap E = \emptyset$.
- $\rho : E \to (N \times N \times \{\Rightarrow, \Leftarrow\})$ is a total function that associates edges with ordered pairs of nodes.
- $\lambda : (N \cup E) \to 2^L$ is a total function that maps nodes and edges to sets of labels, including the empty set.
- $\pi : (N \cup E) \times L \rightharpoonup V$ is a partial function that maps nodes and edges, combined with property names, to property values.

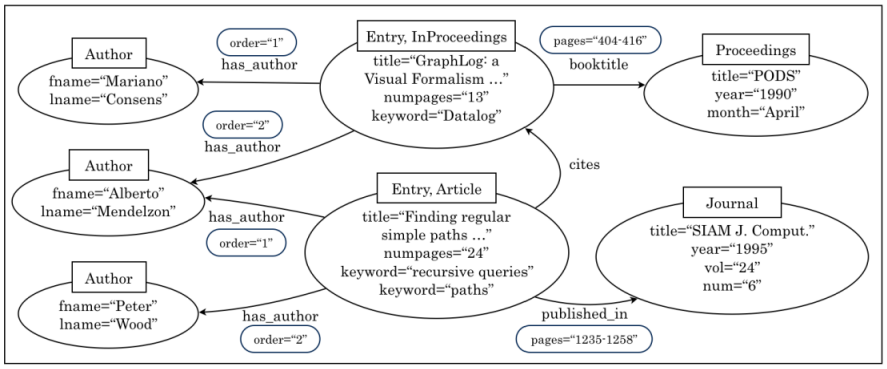

Figure 3: e.g from [1] : there are properties (key-value) in circle, labels of node in rectangle, labels of edge just in text

1.2.3 Queries, Cypher

In graph databases, querying is a fundamental operation that enables the extraction of meaningful information from complex networks of interconnected entities. One of the most expressive and widely used languages for querying graph databases is Cypher, the declarative query language of Neo4j.

Cypher allows users to specify patterns in graphs through a syntax that is both intuitive and powerful. A Cypher query generally consists of several key components:

- **MATCH**: Specifies the pattern to search for in the graph.
- **WHERE**: Adds constraints to the patterns.
- **RETURN**: Defines what should be returned from the query.
- **CREATE**, **MERGE**, **SET**: Commands for modifying the graph.

For example :

```
MATCH (a:Author {fname: "Alberto", lname: "Mendelzon"})-[:has_author]->(e:Entry)
RETURN e.title, e.keyword
```

This query navigates the graph to match the authors and their work.

1.2.4 PGDF Consideration

Additionally, we have explored a new paper [3] on the Property Graph Data Format (PGDF) as a potential standard for representing and exchanging graph data. PGDF defines graphs specifying nodes, edges, properties, and labels in a structured format (highly inspired by CSV syntax). This allows less memory usage, less interchange between graph tools while maintaining data integrity. Its key components include collections of vertices and edges, each described with unique identifiers, property mapsn and headers. Integrating PGDF could enhance interoperability and simplify data migration across graph systems, however team decided to set it aside for project priorities and resource constraints.

2 Experimentation

2.1 Objective

The purpose of this experimental protocol is to evaluate the ability of Large Language Models (LLMs) to reason about property graphs. Five distinct pipelines are compared to assess their performance and capabilities in handling graph-based queries.

2.2 Pipeline Configurations (Figure 5)

1. **Pipeline 1: Plain Text with LLM**
   - The graph is represented in text format.
   - A Cypher query is provided as input.
   - The LLM interprets the query and returns simulated results.
2. **Pipeline 2: Plain Text and Natural Language with LLM**
   - The graph is presented in text format.
   - executes the query of pipeline 3 in Neo4j.
   - Results from Neo4j are analyzed for accuracy.
3. **Pipeline 3: Natural Language with LLM**
   - The graph is represented in text format.
   - Natural language is provided
   - The LLM interprets the query and returns simulated results.
4. **Pipeline Neo4j: Classical Approach with Neo4j**
   - The graph data is loaded directly into Neo4j.
   - Cypher queries are executed without LLM intervention.
   - Results are retrieved directly from Neo4j.

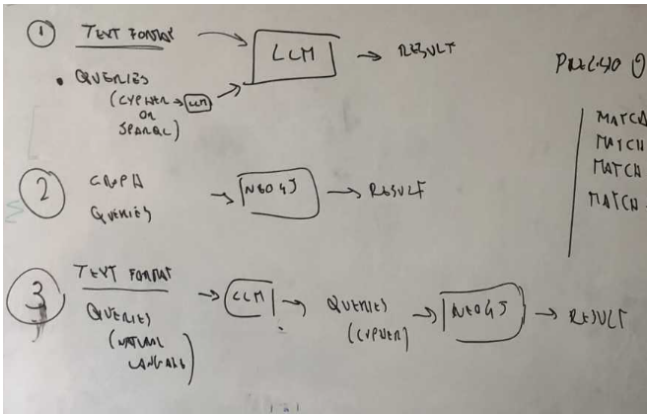

Figure 4: Pipeline : Initial Plan during a meeting

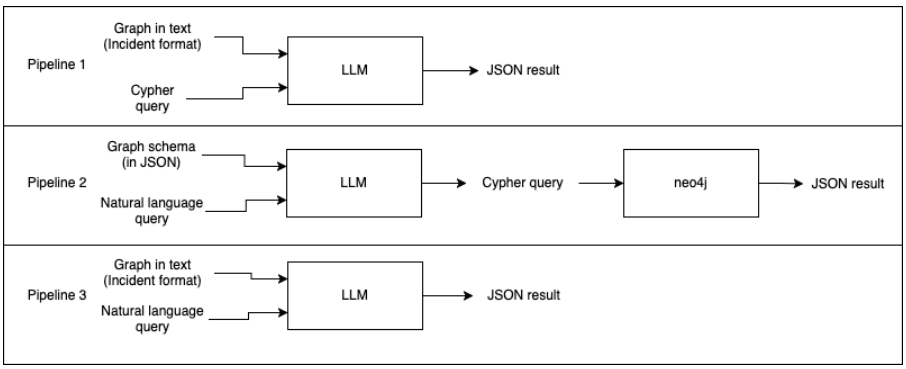

Figure 5: Schema of Pipelines

2.3 Graph Schema

From World Cup 2019 graph [8]


```
[
  {"StartLabels": ["Match"], "RelationshipType": "IN_TOURNAMENT", "EndLabels": ["Tournament"]},
  {"StartLabels": ["Person"], "RelationshipType": "COACH_FOR", "EndLabels": ["Squad"]},
  {"StartLabels": ["Person"], "RelationshipType": "IN_SQUAD", "EndLabels": ["Squad"]},
  {"StartLabels": ["Person"], "RelationshipType": "PLAYED_IN", "EndLabels": ["Match"]},
  {"StartLabels": ["Person"], "RelationshipType": "REPRESENTS", "EndLabels": ["Team"]},
  {"StartLabels": ["Person"], "RelationshipType": "SCORED_GOAL", "EndLabels": ["Match"]},
  {"StartLabels": ["Squad"], "RelationshipType": "FOR", "EndLabels": ["Tournament"]},
  {"StartLabels": ["Team"], "RelationshipType": "NAMED", "EndLabels": ["Squad"]},
  {"StartLabels": ["Team"], "RelationshipType": "PARTICIPATED_IN", "EndLabels": ["Tournament"]},
  {"StartLabels": ["Team"], "RelationshipType": "PLAYED_IN", "EndLabels": ["Match"]}
]
```


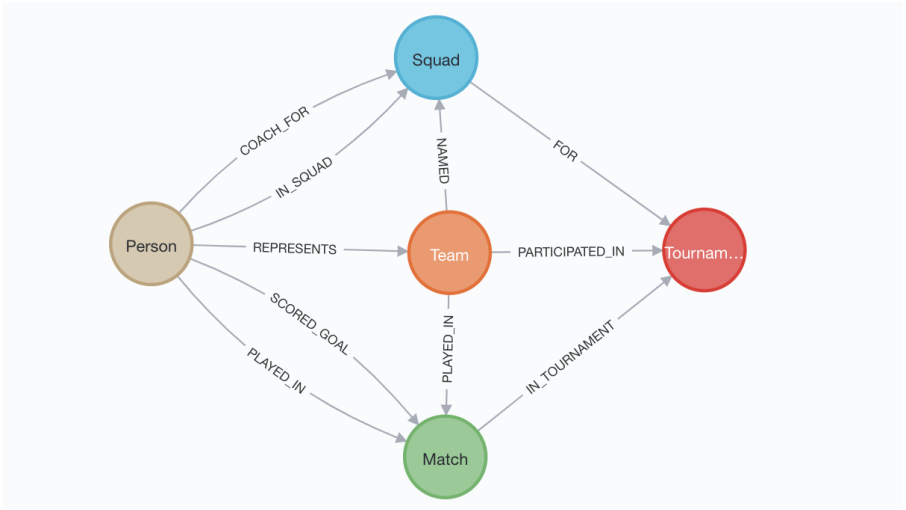

Figure 6: Schema of 2019 WWC graph

2.4 Experimental Questions

The questions are divided into three categories: **Schema Discovery**, **Analytical Queries**, and **Aggregate Queries**. Each question is accompanied by an example Cypher query and its corresponding natural language query that LLMs are expected to generate. (c.f Table 1)

2.5 Evaluation Criteria

- **Accuracy of Results:** Compare LLM-generated results with Neo4j outputs.
- **Linguistic Understanding:** Assess the LLM's ability to generate correct Cypher queries from natural language.

- **Performance:** Measure response times for each pipeline.
- **Qualitative Analysis:** Evaluate clarity and precision of responses for complex queries.
- **Format Enforcement:** LLM responses are strictly required to follow a predefined JSON format. This format includes the Cypher query and results structured as key-value pairs, ensuring consistency and machine-readability.

3 Implementation [5]

You can find the code of the project here :
`https://forge.univ-lyon1.fr/p2105653/ouverture-recherche`

3.1 Overview

Our Python framework [5] uses principaly `networkx`, `ChatOllama` and `langchain`. Pipeline 1, 2, 3 and Neo, were implemented – findable in `main.py`, which executes subprogramms of `libs/pipelines.py`. The loading and the encoding of graph is located in `libs/graph_managment.py` (particularly in functions `create_node_string`, `encode_graph`).
Queries and natural language queries (c.f Table 1) are located in the directory `requests`. Each one is automatically computed by pipelines.
For each pipeline, our framework saves five results in the directory `results`

3.2 Limitation

3.2.1 Number of Tokens

Our framework requires working with graphs that are compatible with NetworkX, but also with manageable sizes. Indeed, Ollama's maximum limit of 2048 tokens (around 4500-5000 characters) forces us to be conscientious of the prompt size, as well as the capacity of memorisation. We are therefore obliged to use subgraphs which must miss information or smaller graphs. You can see on Figure 7 the graph used for testing.

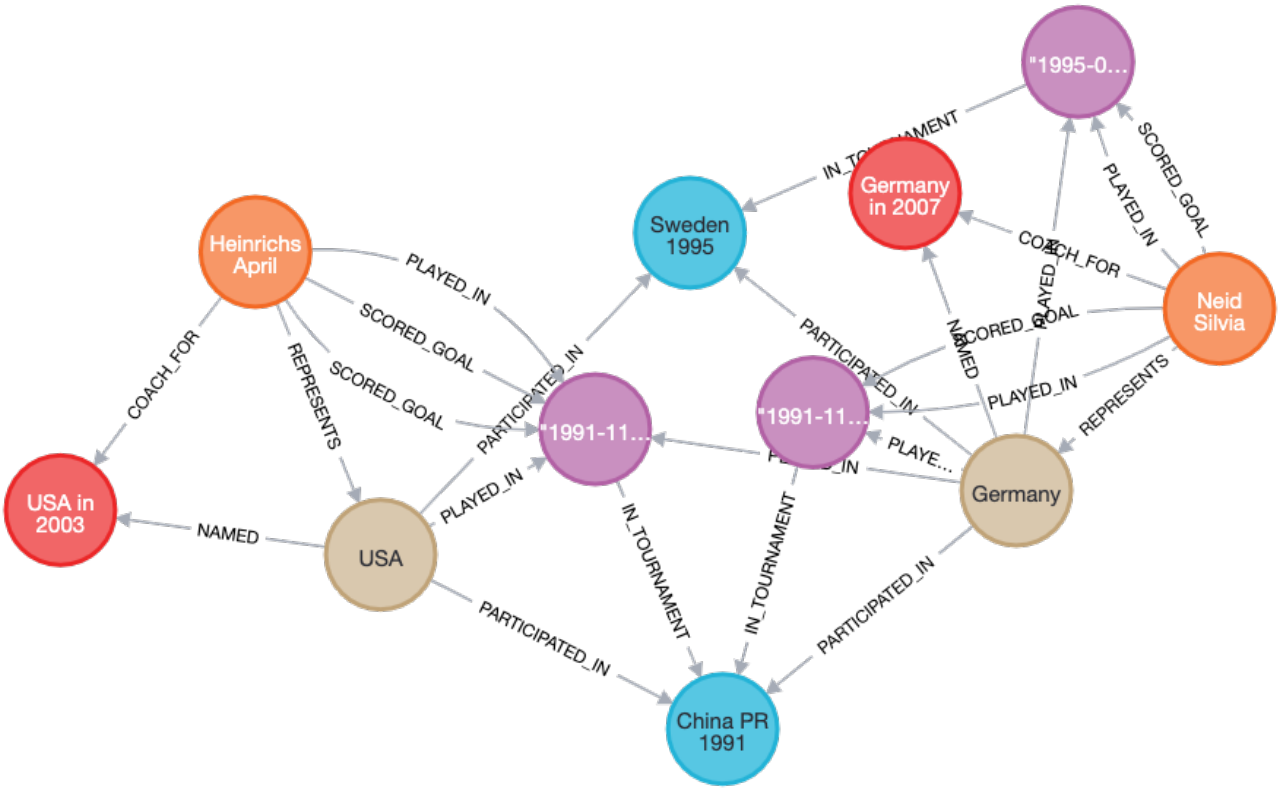

Figure 7: The property graph used by our framework – Nodes = 11, Edges = 64, Properties = 365

3.2.2 Naive Encoding

The team and supervisors, for the sake of time, decided to defer the encoding problem to the future. Indeed, as it stands, our framework uses a rather hybrid encoding. It follows the principles of an incident representation of KGs, yet it remains naive in its representation of properties and labels. To illustrate, our framework computes prompt like this:

> You are a Cypher inference engine. Here is a graph in the incident format : G describes a graph among nodes:
> A [Person, Employee] (age: 29, department: IT), B [Person] (age: 35, department: HR), C [Company] (name: UCBL, location: Lyon).
>
> In this graph
> Node A is connected to nodes B (relationship: colleague, since: 2019), C (relationship: works_for, since: 2020). Node B is connected to node C (relationship: consults, since: 2021).

(with properties between parenthesis and labels between brackets)

4 Results

4.1 First Pipeline

The scores are calculated based on the **mean of good results**. A perfect query gets **5/5**, while missing parts, hallucinations, or incorrect structures result in deductions. Queries that should return empty results but instead hallucinate the information score **0/5**.

As we can see from the results displayed in Table 2, the capacity of the LLM for query inference varies significantly depending on the query. The LLM appears to have difficulty retrieving all the results; it often provides some accurate responses but then either hallucinates or misses others. Additionally, it struggles with obtaining paths and their associated labels. Finally, we can noticeably see that functions like COUNT, AVG, and DISTINCT yield either random results or are not comprehended correctly.

4.2 Pipeline 2 misunderstanding

Results show that pipeline 2 (which attempts to write Cypher queries) totally screws up. Ollama (used by our framework) seems to be the cause of the issue. Indeed, after a few trials with `ChatGPT3.5`, we noticed that the latter can write correct queries. Thus, by prompting the same text that our framework could generate, we collected the answers from `ChatGPT3.5` manually. You can see in Table 3 an assessment based on three tests.

Globally, the problem is that natural language questions can be ambiguous and may not precisely specify the desired representation of the result. Therefore, in its generated queries, `ChatGpt3.5` appears to have the freedom to represent results as it sees fit, particularly for queries 3, 4, and 9. Otherwise, for queries 6, 7 and 10, team would argue those results by supposing that `ChatGpt3.5` can theorically translate in Cypher, but suffers in comprehending questions. Furthermore it would be interesting of encoding the schema (2.3) as well to if it upgrades the LLMs' understanding.

5 Conclusion

Our automatic framework may be seen as a great base for questioning LLMs on PGs' queries, with average results, for future work. Still, this latter has unfortunately suffered from lacking time. Encoding of different aspect – such as Property Graph in text, Schema and natural language questions — shall be more deeply studied. Plus the problem of graph sizes will have to be resolved so as to correctly test our experiment protocol. Lastly, a larger comparison between LLMs would be a lot appreciable.

Category	Cypher Query	Natural Language Query
Schema Discovery		
Q1	`MATCH (n) RETURN DISTINCT labels(n) AS NodeLabels`	What are all the distinct types of nodes in the graph?
Q2	`MATCH ()-[r]->() RETURN DISTINCT type(r) AS RelationshipTypes`	What are all the distinct types of relationships in the graph?
Q3	`MATCH (n) RETURN DISTINCT labels(n) AS NodeLabels, keys(n) AS Properties`	What are the properties associated with each type of node in the graph?
Q4	`MATCH ()-[r]->() RETURN DISTINCT type(r) AS RelationshipTypes, keys(r) AS Properties`	What are the properties associated with each type of relationship in the graph?
Q5	`MATCH (n)-[r]->(m) RETURN labels(n) AS StartLabels, type(r) AS RelationshipType, labels(m) AS EndLabels`	What are the relationships between node types and their connected node types?
Analytical Queries		
Q6	`MATCH (p:Person)-[:SCORED_GOAL]->(m:Match) RETURN m, collect(p) AS Players`	Which players scored in each match?
Q7	`MATCH (t:Team)-[:PARTICIPATED_IN]->(tr:Tournament) RETURN t, collect(tr) AS Tournaments`	What tournaments did each team participate in?
Q8	`MATCH (p:Person)-[:SCORED_GOAL]->(m:Match) WITH p, count(m) AS MatchCount WHERE MatchCount >= 2 RETURN p`	Which players scored in at least two matches?
Q9	`MATCH (t1:Team)-[:PARTICIPATED_IN]->(tr:Tournament)<-[:PARTICIPATED_IN]-(t2:Team) RETURN t1, t2`	Which teams participated in the same tournaments?
Q10	`MATCH (p:Person)-[:REPRESENTS]->(t:Team)-[:PARTICIPATED_IN]->(tr:Tournament) RETURN tr, collect(DISTINCT t) AS Teams, collect(DISTINCT p) AS Players`	Which teams and players were associated with each tournament?
Aggregate Queries		
Q11	`MATCH (p:Person)-[:REPRESENTS]->(t:Team) RETURN t, count(p) AS PlayerCount`	How many players represent each team?
Q12	`MATCH (p:Person)-[:SCORED_GOAL]->(m:Match)<-[:PLAYED_IN]-(t:Team) RETURN t, avg(size((p)-[:SCORED_GOAL]->())) AS AvgGoals ORDER BY AvgGoals DESC`	What is the average number of goals scored by players for each team?

Table 1: List of Cypher Queries and their Natural Language Descriptions

Query	Score	Comments
Query 1	4/5	Occasionally repeats or omits information.
Query 2	3/5	Tends to miss distinct details and sometimes repeats itself.
Query 3	3/5	Generally good, but sometimes mixes or adds extra information.
Query 4	2/5	Completely fabricates properties.
Query 5	1/5	Mislabels start and end points of the path.
Query 6	2/5	Randomly includes or omits properties; not all results are returned.
Query 7	3.5/5	Often misses some information and randomly includes properties.
Query 8	2/5	Frequently miscalculates counts, leading to incorrect results and missing properties.
Query 9	4.5/5	Misses some properties but mostly correct.
Query 10	1.5/5	Produces incorrect properties and results unpredictably.
Query 11	3/5	Occasionally miscounts and provides inconsistent answers.
Query 12	1/5	Averages are completely random.

Table 2: Query Evaluation Results on pipeline 1 ($\mu = 2.54$, $\sigma = 1.14$)

Query	Score	Comments
Query 1	3/3	
Query 2	3/3	
Query 3	1/3	Two others work but with different results, potentially 3/3.
Query 4	1/3	Two others work but with different results, potentially 3/3.
Query 5	3/3	
Query 6	0/3	
Query 7	0/3	
Query 8	2/3	
Query 9	0/3	if result format is important, otherwise 3/3.
Query 10	0/3	
Query 11	3/3	
Query 12	0/3	Compilation error: Can't use aggregate functions inside of aggregate functions.

Table 3: Query Evaluation Results on pipeline 2

References

[1] R. Angles. The property graph database model. In *Proc. Alberto Mendelzon Int. Workshop Found. Data Manage. (AMW)*, volume 2100, pages 1–10, 2018.

[2] R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, K. W. Hare, J. Hidders, V. E. Lee, B. Li, L. Libkin, W. Martens, F. Murlak, J. Perryman, O. Savković, M. Schmidt, J. Sequeda, S. Staworko, and D. Tomaszuk. Pg-keys: Keys for property graphs. In *Proc. Int. Conf. Manage. Data*, pages 2423–2436, New York, NY, USA, 2021. ACM.

[3] Renzo Angles, Sebastián Ferrada, and Ignacio Burgos. The property graph data format (pgdf). *IEEE Access*, 12:123456–123467, 2024. Received 20 September 2024, accepted 11 October 2024, published 24 October 2024, current version 7 November 2024.

[4] Fletcher G. Voigt H. Yakovets N. Bonifati, A. Querying graphs. springer. 2022.

[5] Hoogenbosch.L Ceresa.T. https://forge.univ-lyon1.fr/p2105653/ouverture-recherche, 2025. Accessed: 2025-02-18.

[6] Halcrow J. & Perozzi B. Fatemi, B. Talk like a graph: Encoding graphs for large language models. *arXiv preprint*, arXiv:2310.04560, 2023.

[7] International Organization for Standardization (ISO). Iso/iec 39075:2024 information technology — database languages — gql. ISO/IEC Standard, 2024.

[8] Neo4j. Wwc2019, https://github.com/neo4j-graph-examples/wwc2019, 2019.

[9] Neo4j. The world's leading graph database. 2023.

[10] Wang H. Feng S. Tan Z. Han X. He T. & Tsvetkov Y. Zhang, Y. Can llm graph reasoning generalize beyond pattern memorization? *arXiv preprint*, arXiv:2406.15992, 2024.