Structured 3D Scene Queries with Graph Databases

Aaron Ray and Luca Carlone

***Abstract*— 3D scene graphs have recently gained popularity in robotics as a scalable means to build rich world representations. However, systems that use 3D scene graphs for downstream applications reason over this graph-structured data in an ad-hoc way. In this abstract, we propose Cypher, an existing graph query language, as an general and flexible query interface for working with 3D scene graph data. We demonstrate that large language models (LLMs) can easily ground natural language questions to this query language, without model finetuning or in-context examples. Finally, we show that even when the LLM prompt contains *no* schema information, an LLM that is allowed to make multiple data queries can achieve non-trivial success rates.**

I. INTRODUCTION

Large foundation models have led to impressive advances in robots reasoning about the world. However, most recent works using LLMs or VLMs for semantic and spatial understanding have focused on small-scale scenes, such as an individual rooms or buildings. Methods that rely on language-aligned embeddings to synthesize information about scene context [1], [2] can be scalable, but they do not directly provide a way of answering questions such as "Which regions have an above-average number of trees?"

We seek to scale robot reasoning to the rich, large-scale 3D scene representations that can be constructed by systems such as Hydra [3] (Figure 1) by using a structured yet flexible graph query interface. One of the limitations for scaling LLM reasoning to larger scenes is the context window size. There is a tension between presenting rich scene descriptions to enable fine-grained reasoning and the need to limit context length. The situation is even more dire if we hope to have the LLM update the scene representation, as the output size limit is often much smaller than the input size. For example, a small serialized scene graph built with Hydra from less than five minutes of data contains approximately thirty million characters, or about seven million tokens. GPT4.1 has about a one million token input limit, and a thirty-two thousand token output limit [4]. These constraints emphasize that updating the scene graph requires an LLM update a representation and not merely to regurgitate a full scene graph.

A structured graph-based query language provides a useful interface for letting an LLM query for information about a scene and update information or structure in the robot's world representation. We seek a flexible and composable interface, providing the LLM with a well-defined way for digesting and emitting information, that is not unduly constrained by a predefined ontology. Existing works [5] have demonstrated the utility of defining a structured interface between an LLM and underlying data structures in perception and planning. In this vein, we propose a system that supports querying large-scale 3D scene representations. It supports both graph-based relational queries and geometric spatial indexing. Our implementation also has the advantage of using an existing graph query language called Cypher [6], enabling the generation of syntactically and semantically correct queries without finetuning or in-context examples.

Fig. 1. *Left*: A kilometer-scale 3D scene graph with 314 objects, 15944 places, and 124 regions. *Right*: Scene graph extent on overhead image.

II. CYPHER FOR 3D SCENE GRAPH QUERIES

Cypher [6] is a query language for specifying query and update operations on graph-structured data. The Cypher data model considers a graph of nodes each with a label (e.g., `Object`) and a set of key-value attribute pairs (e.g., {class: vehicle}). Typed edges may be added between pairs of nodes. Queries written in Cypher are usually executed by a graph database [7] storing the data of interest.

The full specification for Cypher cannot be given here, but the fundamental structure of Cypher queries is straightforward[1]. MATCH clauses retrieve matching nodes, relationships, and attributes from the graph. RETURN clauses specify what information should be returned from the matched values. WHERE clauses filter results based on multiple match clauses.

For example, a simple query for the types of objects in a scene graph could be written as `MATCH (n: Object) RETURN DISTINCT n.class as class, COUNT(*) as count`, which returns the number of distinct semantic classes among `Object` nodes, as well as the count of objects of each class.

We can easily write queries based on the connectivity in the graph, such as finding all scene graph places within five nodes of a particular query place `P32`:

Laboratory for Information and Decision Systems,
77 Massachusetts Avenue, Cambridge, MA 02139.
{aaronray, lcarlone}@mit.edu

[1]A functional prototype of the system discussed in this paper with further examples can be found at `https://github.com/GoldenZephyr/heracles`. A detailed introduction to Cypher can be found at `https://neo4j.com/docs/cypher-manual/current`.

```
MATCH (p: Place {nodeSymbol: P32})
MATCH path=
        (p)-[:CONNECTED *1..5]->(c: Place)
RETURN DISTINCT c.nodeSymbol as ns
```

We can find all objects contained in region `R1` by querying `MATCH (r: Region nodeSymbol: R1)-[:CONTAINS*]->(o: Object) RETURN o`. This query finds all transitive containment relationships, a useful feature to decouple the query logic from idiosyncratic scene graph construction. For example, this query would correctly return objects that are added as children of region nodes, or objects which are children of places which are in turn children of regions, as found in Hydra [3].

Already this is a useful interface for a human programmer to interface with large-scale map representations, but in the next section we will show that Cypher is a convenient target for grounding natural language queries with an LLM.

III. LLM Grounding to Cypher Queries

We load a large-scale 3D scene graph built with Hydra (Figure 1) into a Neo4j graph database and test against a set of 25 evaluation questions (Appendix A) with unambiguous answers in this scene graph. We evaluate an LLM's ability to use Cypher to answer scene graph queries in two settings.

First, we provide the LLM a detailed description of the scene graph schema in the system prompt (Appendix C), which works when our goal is to provide a language query interface to scene graphs built by systems with existing scene graph ontologies such as Hydra [3]. Second, we evaluate the case where the system prompt provides no specific information about the scene graph structure. An LLM must be able to successfully answer queries in this setting if we desire ad-hoc, task-specific scene graph ontologies, or if we expect the LLM to add new kinds of layers or edges to the scene graph. The LLM prompt does not contain any in-context examples of Cypher queries in either setting.

To evaluate the known-schema case, we implement a two-stage process. First, the LLM provides a Cypher query to answer the question. Directly comparing the results of this query to the ground truth solution is difficult, because the result contains extraneous information that is difficult to normalize within the query. The second stage uses another LLM call to turn the Cypher query results into a specified answer formatting. The success rate for these queries is presented in Table I, for several ChatGPT model sizes. ChatGPT4.1 performed well, and most of its incorrect answers were reasonable although technically incorrect. Most of ChatGPT4.1-Nano's incorrect answers were caused by not correctly checking for transitive containment relationships[2].

These queries assumed a specific scene graph ontology, specified ahead of time in the system prompt. However, it may make sense to add new kinds of edges, new kinds of layers, or new node attributes on the fly. Making these changes at runtime requires the LLM to make intermediate queries to the database in order to understand the structure of the information. We evaluate an LLM's ability to do this using the same evaluation questions, but we let the LLM make intermediate Cypher queries (up to 10) before returning its final answer. We test this multi-query setup with both the full-information system prompt used in the previous test, and a less informative system prompt that contains no specific information about the scene graph or database schema (Table I Multi-Query and Multi-Query, No Schema respectively). For these tests, validity of the intermediate graph queries is not tracked.

TABLE I
Query Success Rate

Method	Valid Cypher	Correct
Single Query [ChatGPT-4.1]	24/25	20/25
Single Query [ChatGPT-4.1-Mini]	23/25	17/25
Single Query [ChatGPT-4.1-Nano]	21/25	8/25
Multi-Query [ChatGPT-4.1]	-	24/25
Multi-Query [ChatGPT-4.1-Mini]	-	19/25
Multi-Query [ChatGPT-4.1-Nano]	-	9/25
Multi-Query, No Schema [4.1]	-	10/25
Multi-Query, No Schema [4.1-Mini]	-	9/25
Multi-Query, No Schema [4.1-Nano]	-	3/25

The ability to run multiple queries improves the model's performance in the full-information case, because the model can recover from syntactic errors or resolve ambiguities. Answering the questions is a substantially harder problem without the prior schema knowledge, and Table I shows a corresponding lower success rate. A qualitative analysis of the query sequences shows two areas of difficulty. The first is syntactic ambiguity about property names. For example, asking about "object types" leads to useless queries if the relevant attribute is called "class" instead of "type". However, the failure of the initial query is often enough to prompt better future queries. The second difficulty is semantic ambiguity about the graph connectivity. Many LLM-generated queries assume that objects are directly connected to regions, even though they are not. This kind of misconception does not seem to be remedied by feedback from the query execution. This demonstrates the need for a concise and precise way of describing the semantics of a scene graph, whether it has a rigid ontology like [3], or a more flexible ontology adapted online by an LLM.

IV. Future Work

So far, we have only investigated an LLM's ability to *query* a scene graph. However, Cypher also enables editing nodes, edges and attributes. This is a promising future direction, although evaluation becomes more difficult.

Another challenge is aligning the semantics of the 3D scene graph with the semantics expressed by the query language. For example, locality queries can be run either by bounding box containment or edge connectivity in the graph. The proper choice depends on the (possibly latent) meaning of the edge of the scene graph. A more domain-specific restricted query language may help to align the LLMs expectations of how it can interact with the data.

[2]This failure mode causes a large drop in performance, because regions contain places and places contain objects. Regions are not *directly* connected to objects.

REFERENCES

[1] D. Maggio, Y. Chang, N. Hughes, M. Trang, D. Griffith, C. Dougherty, E. Cristofalo, L. Schmid, and L. Carlone, "Clio: Real-time task-driven open-set 3D scene graphs," *IEEE Robotics and Automation Letters (RA-L)*, vol. 9, no. 10, pp. 8921–8928, 2024, (pdf),(video),(web).

[2] Q. Gu, A. Kuwajerwala, S. Morin, K. M. Jatavallabhula, B. Sen, A. Agarwal, C. Rivera, W. Paul, K. Ellis, R. Chellappa, C. Gan, C. M. de Melo, J. B. Tenenbaum, A. Torralba, F. Shkurti, and L. Paull, "Conceptgraphs: Open-vocabulary 3D scene graphs for perception and planning," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2024.

[3] N. Hughes, Y. Chang, S. Hu, R. Talak, R. Abdulhai, J. Strader, and L. Carlone, "Foundations of spatial perception for robotics: Hierarchical representations and real-time systems," *Intl. J. of Robotics Research*, 2024.

[4] "GPT-4.1," https://platform.openai.com/docs/models/gpt-4.1, accessed: 2025-05-27.

[5] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, "SayPlan: Grounding large language models using 3D scene graphs for scalable task planning," in *Conference on Robot Learning (CoRL)*, 2023, pp. 23–72.

[6] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1433–1445. [Online]. Available: https://doi.org/10.1145/3183713.3190657

[7] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, Feb. 2008. [Online]. Available: https://doi.org/10.1145/1322432.1322433

APPENDIX

A. Evaluation Questions and Answers

The following questions and solutions are used to evaluate the system's ability to answer natural language 3D scene graph questions by grounding to Cypher queries. The particular syntax of solutions and the notion of equality used to evaluate correctness is discussed in Appendix B.

```
Question: "What are the distinct object classes?"
Solution: <tree, fence, vehicle, seating, window,
  sign, pole, door, box, trash, rock, bag>

Question: "How many of each object type are there?"
Solution: {tree: 163, fence: 17, vehicle: 26, seating: 9, window: 1,
  sign: 6, pole: 21, door: 3, box: 4, trash: 1, rock: 62, bag: 1}

Question: "Give me the location of all signs in the scene graph"
Solution: |
         <POINT(-8.284313559532166 -14.206965416035754 1.9744174981370886),
          POINT(-30.409549247386845 -15.735817166261894 1.419422089360481),
          POINT(-29.527192555941067 -11.574765984828655 1.4839779872160692),
          POINT(-29.246166506680574 -8.68991950641979 1.4494154886765913),
          POINT(-29.16052139096144 -7.293464509452262 1.556024234469344),
          POINT(-69.23055948529925 83.1722952524821 0.790333880555062)>

Question: "What object node symbols are within 10 meters of x:-100, y:16, z:0?"
Solution: <O370, O371, O372, O373>

Question: "How many mesh places are within 5 hops of mesh place P1832
  (not including the starting place)."
Solution: 141

Question: "What object types are in Room R91"
Solution: <tree, window, pole>

Question: "How many objects are in room R1?"
Solution: 7

Question: "Which room symbol has the most object types?"
Solution: R96

Question: "Which room symbol has the highest number of objects?"
Solution: R100

Question: "Between tree and box, which is in the fewest rooms?"
Solution: box

Question: "What's the least common object type in Room R100 that is not a rock?"
Solution: vehicle

Question: "What are the empty room symbols (rooms containing no objects) that
  are connected to room R99?"
Solution: <R52, R6, R50>

Question: "List all the room symbols that have a rock in them."
Solution: <R30, R88, R86, R99, R112, R109, R87,
   R8, R11, R12, R121, R122, R65, R2, R16, R81,
   R100, R3, R73, R25, R70, R74, R28, R102>
```

```
Question: "List all the room symbols that have only rocks in them."
Solution: <R88, R109, R8, R11, R12, R121, R65, R73, R25>

Question: "Give me the point coordiantes of the rocks, either in
  Room R88 or Room R100?"
Solution: <POINT(-80.96216372785896 27.755121998403265 4.415851417629198),
           POINT(-0.7888006318588646 -2.903889067318975 0.0882817480941208),
           POINT(-0.825403584722887 -4.263898042210362 0.03717637520381495)>

Question: "Find the distance of the rocks and trees in Room R100 that are
  within 3.0 meters of each other, ordered by shortest distance.
  Return only the distances."
solution: [0.7490734671216159, 1.1143781476916492, 1.2333940296376114,
           1.8810572358450335, 2.297507598063438, 2.940645758149497]

Question: "Give me a list of position coordinates of all the rocks
  and vehicles in Room R100?"
Solution: <POINT(-0.7888006318588646 -2.903889067318975 0.0882817480941208),
           POINT(-0.825403584722887 -4.263898042210362 0.03717637520381495),
           POINT(-2.4370584895468164 2.587475128662892 -0.08760905527767454),
           POINT(-5.885951024601442 -10.49500067755397 -0.2860741552871627),
           POINT(-5.885951024601442 -10.49500067755397 -0.2860741552871627)>

Question: "Give me all the place symbols that have boxes but no trees?"
Solution: <p12343, p12645, p906, p12659>

Question: "Give me all room symbols with an above-average number of trees?"
Solution: <R1,R2,R3,R5,R7,R9,R14,R15,R16,R17,R18,R20,R26,R30,R31,R32,
           R38,R68,R70,R75,R81,R84,R86,R89,R91,R93,R99,R100,R101,R102,R103>

Question: "Give me room symbols with a number of trees that's at least
  one standard deviation above the mean ordered by the most trees?"
Solution: [R100, R20, R2]

Question: "Which object types have more than 30 instances?"
Solution: <tree, rock>

Question: "What is the maxium number of trees in any place?"
Solution: 3
refinement_type: "number"

Question: "What are the object point coordinates inside the bounding box
  with the following coordinates: x1:-0, y1:-0, z1:0, x2:10, y2:10, z2:5?
  Return only the coordinates."
solution: <POINT(0.11896780423469352 6.619487872937831 0.22895595844744182),
           POINT(0.7478225974187459 5.342799944420383 0.49659509568998256),
           POINT(4.855817626186302 1.5613172245024654 0.8235663587714304)>

Question: "What rooms have the same object types as R102?"
Solution: <R2, R70>

Question: "What is the distance between mesh places P906 and P1985?"
Solution: 1.588630981055123
```

B. Evaluating Correctness

Equivalence between two answers is evaluated within the Set, List, Dictionary, Point (SLDP) equality language. In this language, primitives are either a string or a floating point number. A set is denoted with angle brackets, e.g., $< 1, 2, 3 >$. A list is denoted with square brackets, e.g., $[a, b, c]$. A dictionary is denoted with curly braces and colons, as in Python: $\{key1 : val1,\ key2 : val2\}$. A point is denoted as $Point(x\ y\ z)$. Sets, lists, and dictionaries can be arbitrarily composed.

Two strings are equivalent if they are character-wise equal after stripping leading and trailing whitespace and converting to lowercase. Two numbers are equivalent if they are within some tolerance of each other (0.01 in this work). Two lists are equivalent if the i^{th} elements of each list are equivalent. Sets A and B are equivalent if each element of A is equivalent to an element of B and each element of B is equivalent to an element of A. Two dictionaries A and B are equivalent if every key in A is in B, every key in B is in A, and for each key k, A[k] is equivalent to B[k]. Two points are equivalent if they are within some tolerance of each other (here, within .01 in the L_∞ norm).

The solutions to each question are given as an SLDP expression. The LLM answer refinement prompts tell the LLM to return its result as an SLDP expression. An LLM's answer is marked correct if it is equivalent in SLDP to the solution.

C. Prompts

Single-Query System Prompt

```
You are a helpful assistant who is an expert at mapping from natural language
queries to Cypher queries for a Neo4j graph database. You have access to a
database representing a 3D scene graph, which stores spatial information that
robot can use to understand the world. Given a query, your task is to generate a
Cypher query that queries the relevant information from the database.

Labels in Database:
    - Object: a node representing an object in the world.
      Object Properties:
        - nodeSymbol: a unique string identifier
        - class: a string identifying the object's semantic class or type
        - center: the 3D position of the object, as a POINT type
    - MeshPlace: a node representing a 2D segment of space the robot might be able to
                 move to.
        - nodeSymbol: a unique string identifier
        - class: a string identifying the place's semantic class or type
        - center: the 3D position of the mesh place, as a POINT type
    - Place: a node representing a 3D region of free space
        - nodeSymbol: a unique string identifier
        - center: the 3D position of the place, as a POINT type
    - Room: a node representing a room or higher-level region
        - nodeSymbol: a unique string identifier
        - center: the 3D position of the room, as a POINT type

Object, MeshPlace, Place, and Room are all Cypher labels attached to nodes.

Places and Mesh Places represent a higher level of the hierarchy compared to
objects, but lower level than rooms.

There are two kinds of existing edges. First is (a)-[:CONTAINS]->(b),
which connects nodes between different layers and means that b is contained
within a. Nodes in higher levels of the hierarchy may contain nodes in lower
levels of the hierarchy, but nodes in the lower level of the hierarchy will not
contain higher-level nodes. The other kind of edges represent connectivity
within a layer: [:OBJECT_CONNECTED], [:PLACE_CONNECTED],
[:MESH_PLACE_CONNECTED], [:ROOM_CONNECTED].

Note that in the current version of cypher, `distance`
has been replaced by`point.distance`.
```

```
Also, do not use any apoc functions in your queries,
because apoc is not installed so those queries will crash.

Now, generate a cypher query for this natural language query:

< The Natural Language Query >

Use a series of steps to formulate your final answer in a chain of thought
style. Remember the output format must be in JSON and formatted as a Python
dictionary of the form: {"chain of thought": <...>, "cypher": <...>}
```

When we refine the Cypher query result into a final answer in the single-query evaluation, one of the following prompts is used depending on the kind of solution:

Refine to Number

```
You are trying to answer the question: {question}.
You have the following data as an intermediate answer. Please
reformat the following data into a number,
Your response should contain only the number
and no extraneous information.
```

Refine to String

```
You are trying to answer the question: {question}.
You have the following data as an intermediate answer. Please
reformat the following data into a string,
Your response should contain only a *single* word, no quotation,
and no extraneous information.
```

Refine to List

```
You are trying to answer the question: {question}.
You have the following data as an intermediate answer. Please
reformat the following data into a list of the form [element1, element2, .... elementN],
maintaining the order implied by the intermediate data.
The list is denoted by square brackets [ ].
Elements within the list should not have quotations around them.
If you need to represent a POINT in the list, the syntax is POINT(x y z).
Your response should contain only the list and no extraneous information.
```

Refine to Dictionary

```
You are trying to answer the question: {question}.
You have the following data as an intermediate answer. Please
reformat the following data into a
dictionary of the form {{key1: value1, ..., keyN: valueN}}
Keys and values should not have quotations around them.
The dictionary is denoted by curly braces {{ }}.
If you need to represent a POINT, the syntax is POINT(x y z).
Your response should contain only the dictionary and no extraneous information.
```

Refine to Set

```
You are trying to answer the question: {question}.
You have the following data as an intermediate answer. Please
reformat the following data into a set of the form <element1, element2, .... elementN>.
```

```
Elements within the set should not have quotations around them.
The set is denoted by angle brackets < >.
If you need to represent a POINT, the syntax is POINT(x y z).
Your response should contain only the set <element1, ..., elementN>,
and no extraneous information.
```

Schema-Free System Prompt

```
You are a helpful assistant who is an expert at mapping from natural language
 queries to Cypher queries for a Neo4j graph database. You have access to a
 database representing a 3D scene graph, which stores spatial information
 that a robot can use to understand the world. Given a query, your task
 is to generate a Cypher query that retrieves the relevant information
 from the database.

Note that in the current version of cypher, `distance` has been replaced by
`point.distance`. Also, do not use any apoc functions in your queries.

Agent Instructions:

1. Before attempting to generate the final Cypher query, run exploratory queries
to verify available node labels, relationship types, and property keys if
necessary.

2. If the query you generate is invalid, fails to run, or produces an error,
retry by correcting the Cypher based on feedback or known schema rules.

3. You may break down the query into parts to iteratively refine the correct
Cypher expression.

You can call the cypher query tool up to 8 times. If you try to execute more
than 5 queries, your answer will be counted as wrong.

Now, use cypher queries to answer this question:

< The Natural Language Query >

Use a series of steps to formulate your final answer in a chain of thought
style. When you are ready to give your final answer, put it between a beginning
and ending answer tag: <answer> the answer here </answer>.
```

The end of the schema-free prompt contains a description of the question-specific output format, essentially the same as the “Refine to X” prompts above. In the multi-query case, this instruction is part of the initial system prompt rather than an explicit separate step.