CityLLM: A framework for natural-language querying of semantic 3D city models

Rabindra Lamsal[1], Sisi Zlatanova[1], Johnson Xuesong Shen[2]

[1]GRID Lab, School of Built Environment, UNSW Sydney, NSW 2052, Australia - (r.lamsal,s.zlatanova)@unsw.edu.au
[2]School of Civil and Environmental Engineering, UNSW Sydney, NSW 2052, Australia - x.shen@unsw.edu.au

Keywords: CityGML, CityJSON, Conversational Framework, Large Language Model, Cross-database Querying

Abstract

Semantic 3D city models provide rich geometric and semantic information, but remain challenging for non-experts and interdisciplinary researchers to access and query due to their complex structures and specialized data formats. To address this issue, we present CityLLM, a framework for natural-language querying of semantic 3D city models alongside complementary urban datasets. The framework combines spatial and graph databases within an LLM-based workflow that supports iterative query refinement and cross-database chaining. We evaluate CityLLM on a CityJSON dataset of Rotterdam (853 LoD2 buildings) using GPT-OSS, Gemini 3.1, and GPT-5.4, along with selected variants, across multiple metrics: *answer correctness*, *visualization correctness*, *query success*, and *retry attempts*. A total of 54 natural-language queries are curated across four scenarios: spatial, graph, cross-database, and conversational. Results show strong overall performance, with answer correctness ranging from 85.2% to 100%, visualization correctness from 92.9% to 100%, a 100% query success rate, and fewer than three retries across all 54 queries. Overall, the findings suggest that CityLLM provides a lightweight and extensible approach for conversational access to semantic 3D city data.

1. Introduction

Semantic 3D city models are a core component of urban digital twins and smart-city applications. CityGML (Gröger and Plümer, 2012), an OGC standard, provides a rich XML-based exchange format for representing 3D city models, capturing geometry, semantics, topology, and multiple Levels of Detail (LoDs) for urban objects such as buildings, transportation networks, water bodies, and land use (Kolbe, 2009). CityJSON (Ledoux et al., 2019), an alternative encoding of the same data model, offers a more compact, developer-friendly JSON representation suited to web-based and lightweight workflows. Together, these standards have enabled the creation and dissemination of large-scale semantic 3D city datasets[1] that are increasingly used in both research and practical urban applications.

In practice, however, these models remain difficult to access and query. Their hierarchical structure and spatial characteristics demand specialized knowledge of data schemas, spatial query languages, or dedicated GIS software; a barrier that grows when a city model must be queried alongside other urban data sources, such as street networks or amenities, that follow entirely different formats. Effective exploration therefore remains challenging for non-expert and interdisciplinary users alike, especially when a task requires drawing on multiple sources rather than the city model alone. As urban digital twins grow in scale and heterogeneity, more intuitive and accessible ways of interacting with this data are needed.

Recent advances in large language models (LLMs) have opened new possibilities for natural-language interaction with complex data (Achiam et al., 2023). LLMs have shown strong potential for translating natural-language questions into structured queries, coordinating external tools, and supporting multi-step reasoning. These capabilities are particularly relevant for geospatial applications, where users often need to express complex spatial questions or combine information from multiple data sources. For semantic 3D city models, LLM-based interfaces can lower this barrier by letting users ask questions in plain language instead of writing format-native queries (e.g., XQuery, SQL, Cypher). However, this is not straightforward to realize: these datasets combine geometric, semantic, and spatial-relationship information, demanding schema awareness and reliable query generation, and failures can arise when schema context is incomplete, when reasoning spans multiple steps, or when a query must integrate heterogeneous sources. Existing approaches have shown strong potential in this direction (Liu and Wang, 2025; Kanna and Kolbe, 2025b,a), but less attention has been given to lightweight, extensible workflows that coordinate complementary data representations of semantic 3D city models within a single conversational pipeline.

To address this, we present *CityLLM*, a framework for natural-language querying of semantic 3D city models. The framework combines an LLM-based agent with two complementary data backends: CityJSON-derived data are stored in a lightweight spatial database for semantic and spatial queries, while street networks and amenities from OpenStreetMap are represented in a graph database for graph-based analysis. The agent interprets user queries, selects the appropriate backend(s), generates and executes queries, and returns responses that can be visualized in an interactive map. The workflow also supports iterative query refinement and multi-step query chaining across heterogeneous data sources.

The remainder of this paper is organized as follows. Section 2 reviews the relevant literature, Section 3 describes the proposed framework, Section 4 discusses the evaluation setup, Section 5 presents the results and discussion, and Section 6 concludes the work.

2. Literature Review

Recent advances in LLMs (Achiam et al., 2023) are pushing the boundaries of geospatial research. Although the integration of LLMs with geospatial data is still in its early stages (Kanna and

[1] https://3d.bk.tudelft.nl/opendata/opencities/

Kolbe, 2025a), researchers are increasingly embedding LLMs into diverse AI workflows, ranging from natural-language interaction (Pan et al., 2026) to disaster geolocalization (Yin et al., 2025) and automated geospatial modeling (Liang et al., 2025), among other emerging applications. In fact, recent works have begun to envision urban foundation models that place LLM-based agents at the core of city analytics. For example, (Zhang et al., 2024) outlined a taxonomy for urban data and discussed how foundation models can act as central agents for multimodal urban tasks, and (Xu et al., 2023) proposed an Urban Generative Intelligence platform that trained a specialized model "CityGPT" with agents to simulate and reason about complex city environments. Such developments highlight the momentum toward integrating LLMs as core components of urban digital twins and decision-support systems. Concurrently, other research highlights the role of generative AI in smart cities and the metaverse; for instance, (Lifelo et al., 2024) emphasized that advanced NLP capabilities of LLMs enable interactive conversational experiences within urban digital twins, enhancing user engagement in smart city applications.

One of the most immediate applications of LLMs in the geospatial domain is question answering (QA). For instance, (Feng et al., 2023) introduced GeoQAMap (Geographic Question Answering), an LLM-based system that translates natural-language queries into SPARQL queries over the Wikidata knowledge base (Wikidata, 2025) and visualizes the results on interactive maps. Similarly, (Deng et al., 2025) developed Zaha, a QA system integrated with *The World Avatar* knowledge graph (Akroyd et al., 2021), which enables intuitive natural-language queries for smart city and urban planning data by unifying information from diverse sources. In a related context, (Santhanavanich and Coors, 2025) presented OGC-AI, an LLM-based interface that allows users to interact with OGC web services in natural-language. These approaches show how conversational AI can lower the barrier for non-experts to access and analyze complex geospatial data through natural-language.

Particularly concerning semantic 3D city models, their rich but complex schema poses challenges for direct interaction by users. One of the notable works in making CityGML more interpretable was by (Hansson, 2024), where CityGML data was represented as a virtual knowledge graph using its ontology and R2RML (Consortium et al., 2012) mappings. Furthermore, in the context of mapping, (Kanna and Kolbe, 2025b) proposed a system to automatically augment CityGML data with missing real-world attributes. Their approach involved extracting information from external sources (e.g., OpenStreetMap) and mapping it to corresponding CityGML features. First, the CityGML data is imported into a spatial relational database using 3DCityDB (3dcitydb Contributors, 2025), and an LLM agent generates and executes SQL insert statements to populate the newly extracted attributes into the CityGML schema.

Limited studies to date have focused on creating full conversational interfaces for semantic 3D city models. One example is KCityChatBot (Liu and Wang, 2025), which allows natural-language dialogue with a CityGML-based knowledge graph. Their method automatically transforms CityGML data into a property graph, representing each city object as a node with edges capturing the nested hierarchies and then uses a multi-agent LLM framework to handle user queries. In this pipeline, one agent interprets a user's question and generates a corresponding Cypher query, which is executed on a Neo4j graph database, and another agent formulates a natural-language answer based on the query results. Likewise, (Kanna and Kolbe, 2025b) implemented an LLM-based query agent that converts natural-language questions into SQL against a 3DCityDB database. Extending this approach, (Kanna and Kolbe, 2025a) presented an advanced LLM-driven interface for an urban digital twin that combines the semantic city model with time-series sensor data. In their framework, city objects and their dynamic observations are stored in the database, and the LLM acts as an orchestrator that can iteratively refine its queries. For example, given a user prompt, the LLM generates an SQL query (or calls an external tool) to fetch information from either the static CityGML data or the linked sensor data, and then progressively improves the response through follow-up queries or tool use, if needed. The results are presented in natural-language and can be visualized. This interactive system showcases the potential of combining LLMs with 3D city models and real-time data for conversational query and analysis.

Overall, these studies represent important progress toward conversational access to semantic 3D city and urban digital twin data, while also indicating complementary design directions. KCityChatBot (Liu and Wang, 2025) demonstrates the value of a knowledge-graph-based interface for large-scale CityGML interaction, whereas (Kanna and Kolbe, 2025b) highlights LLM-supported enrichment and SQL-based access to CityGML workflows. The advanced interface in (Kanna and Kolbe, 2025a) further shows the potential of LLM orchestration across static city models and dynamic sensor streams. Building on these efforts, this work focuses on coordinating complementary data representations of semantic 3D city models within a single lightweight architecture, one that decides which data backend a user query needs, chains results across them when a query requires both, supports iterative query repair and exposes the generated queries and resulting visualizations through one conversational interface.

3. Proposed Framework

The proposed framework, CityLLM, enables natural-language interaction with semantic 3D city models by integrating an LLM with spatial and graph databases. Its architecture is illustrated in Figure 1. The framework follows a modular design composed of three main components: (i) data backends, (ii) an LLM-based agent, and (iii) an interactive map-based interface. In this section, we describe each component, including its implementation and the underlying sub-components that together form the CityLLM framework.

3.1 Data Backends

Semantic 3D city models in this study are represented in the CityJSON format. To support spatial querying, a CityJSON dataset is imported into a PostgreSQL/PostGIS database using a compact relational schema (as shown in Figure 2). The schema is intentionally simple, reducing structural complexity and avoiding excessive joins, which facilitates more reliable query generation by the LLM while minimizing output tokens and inference cost.

During preprocessing, the dataset is parsed and mapped to the relational schema. Geometries are stored using native spatial data types to support 3D operations and indexing, while semantic attributes are kept separately. Since a city object can have multiple geometries across different LoDs, object attributes are stored in one table and the corresponding geometries in another, linked via the shared *object_id*. The geometries are

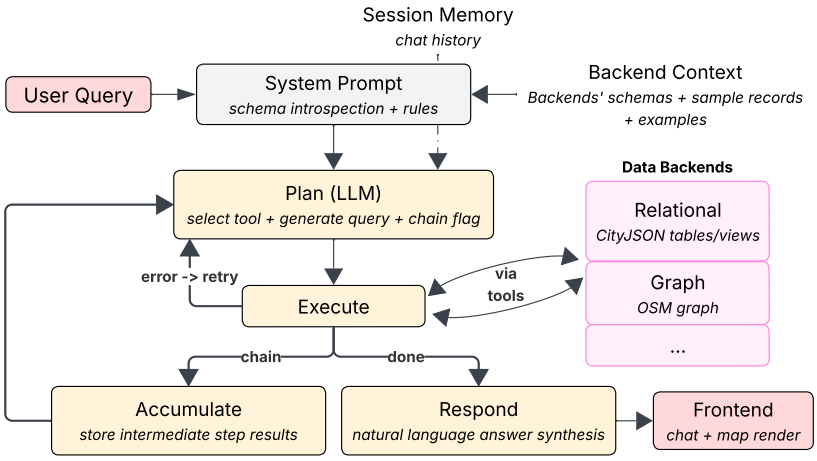


Figure 1. Architecture of the CityLLM framework.

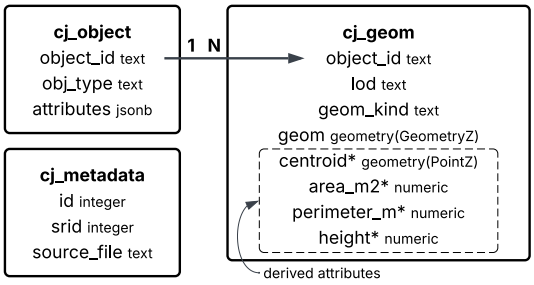


Figure 2. Relational database (PostgreSQL/PostGIS) schema.

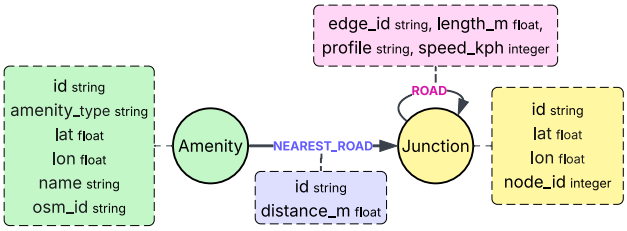


Figure 3. Graph database (Neo4j) schema.

inserted using the dataset's reference coordinate system. In addition, derived properties such as centroid, area, perimeter, and height are computed during preprocessing and stored alongside the geometries. This avoids repeated computation at query time and allows the LLM to directly reference these attributes when answering user queries. With this setup, spatial queries are executed using standard PostGIS functions (e.g., *ST_Intersects*, *ST_Distance*, *ST_DWithin*), supporting tasks such as proximity search, spatial filtering, and geometric analysis. In addition to the base tables, a view is created that combines object attributes with a single geometry per object, selected as the highest available LoD, for use in most LLM queries.

In addition to the semantic 3D city model, the framework supports the integration of external urban datasets to enable complementary analyses; other sources may require different storage models than the relational schema used for CityJSON. To demonstrate this, street-network data derived from OpenStreetMap (OSM) for the geographical extent of the CityJSON dataset are stored in a Neo4j graph database, with road junctions modeled as nodes and street segments as edges (Sun et al., 2026). Amenities (e.g., parks, hospitals, etc.) within the same extent are also retrieved and linked to their nearest junctions. The graph schema is shown in Figure 3. This graph-based representation supports network-oriented analyses such as shortest-path computation, routing, and proximity queries, and later we demonstrate that the LLM can traverse both backends together to answer queries requiring coordinated retrieval.

Extensibility. Neo4j was chosen mainly for its simple property-graph model and easy integration with street-network OSM data, not because it is the only suitable option. More broadly, this second backend is meant to illustrate the framework's extensibility rather than to be exhaustive, and we treat it as a proof of concept; other sources, such as RDF store, sensor data, vector-based retrieval over regulatory documents (e.g., for checking height-regulation compliance), are left as natural extensions.

3.2 LLM Agent

User queries in natural-language are processed by an LLM agent that translates them into executable database queries. The agent is provided with the user query together with contextual information about the available data backends, tools, and their schemas via a system prompt. This prompt defines the operational constraints of the agent, including explicit tool selection, structured output format, and query formulation rules. Given the extensive schema, tool descriptions, and rules, the system prompt in CityLLM exceeds 3,000 tokens, with lengths of 3,764 tokens for Gemini and 3,370 tokens for GPT-5.x.

At a high level, the agent selects among three actions: querying PostGIS (CityJSON data), querying Neo4j (OSM street-network graph with amenities), or returning a *cannot_answer* response for out-of-scope queries. Based on the inferred intent, it generates either SQL (for PostGIS) or Cypher (for Neo4j) queries using the provided schema context. For queries requiring information from multiple data backends, the agent performs *chained execution*, where intermediate results from one query are used to formulate subsequent queries. Table 5 illustrates this for two cross-database examples. In cases where a query execution fails, database error messages are fed back to the LLM, allowing iterative correction and regeneration. This process continues until a valid result is obtained or a predefined

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XII-4-W1-2026-219-2026

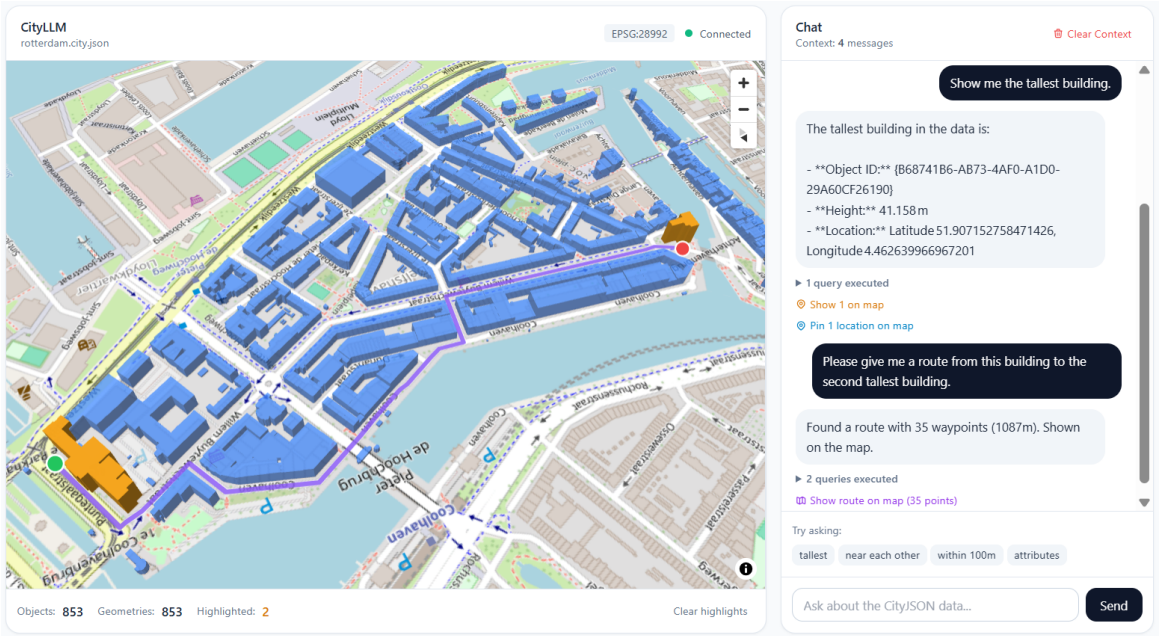


Figure 4. An example conversation with CityLLM.

iteration limit is reached. The final output is then returned as a natural-language response derived from the retrieved data.

3.3 Interactive Map-based Chat Interface

The framework includes an interactive map interface that allows users to explore query results alongside conversational responses. Depending on the query type, different visualization strategies are applied: (i) building geometries retrieved from PostGIS are highlighted, (ii) point-based results (e.g., amenities) are displayed as points, and (iii) routes computed from the Neo4j graph are rendered as lines. Each response generated by the LLM includes object identifiers and/or coordinates, which are rendered by the frontend. The interface supports incremental updates, allowing users to refine queries and immediately see updated results without resetting the session. This integration of conversational querying with direct visual feedback enables intuitive exploration of semantically rich 3D city data. For transparency, the interface also exposes the SQL and/or Cypher queries generated for each user query. Overall, the framework attempts to bridge natural-language interaction and spatial analysis by integrating LLM-based query generation, database execution, and interactive visualization within a unified workflow.

4. Evaluation Setup

The CityLLM framework is implemented in *Python 3.13* using a set of supporting packages. The LLM agent workflow is built with *LangGraph*[2], while the interactive map interface is implemented using *MapLibre GL JS*[3]. The street-network and amenity data was retrieved using *OSMnx* (Boeing, 2025). LLM inference is performed using the following *LangChain integrations*[4]: *langchain_{groq, openai, google_genai}*.

We evaluate CityLLM using the *Rotterdam* dataset available at `https://www.cityjson.org/datasets/`. The dataset contains 853 building objects in LoD2. Following the data preparation workflow described in Section 3.1, the CityJSON dataset was imported into a PostgreSQL/PostGIS database. Additionally, based on the geographical extent (+ *buffer*) of the dataset, a street-network data from OpenStreetMap was stored in a Neo4j graph database. We also extracted amenities (e.g., parks, hospitals) within the same extent and linked each of them to the nearest junction in the Neo4j graph.

Evaluation Queries. Next, we curated a set of 54 natural-language queries. The queries were constructed to test different functional aspects of CityLLM and were grouped into four types: (i) *PostGIS queries* (N=15): Spatial queries operating on the semantic 3D city model stored in the relational database, (ii) *Neo4j queries* (N=12): Graph-based queries involving street-network relationships, (iii) *Cross-database queries* (N=10): Queries requiring information from both databases, and (iv) *Conversational queries* (5 interactions; N=17): Sequential interactions where subsequent queries depend on previous context. Each query was submitted through the CityLLM interface (as shown in Figure 4) and processed as per the workflow illustrated in Figure 1.

LLMs. Three LLMs were used: *GPT-OSS (120B)*, *Gemini 3.1 (Flash Lite)*, and *GPT-5.4*. GPT-OSS was selected as a representative open-source LLM suitable for local deployment, while the other two serve as high-performance proprietary baselines. To further examine failure cases, we additionally evaluate model variants (*GPT-5.4-mini* and *Gemini 3.1 Pro*), as discussed later. For all LLMs, the *iteration limit* for query refinement and *chain limit* for cross database querying was set to 3.

Evaluation Criteria. The evaluation was done as per following criteria: (i) *Answer correctness*: Whether the final response correctly answered the user's query, (ii) *Visualization correctness*: For queries requiring spatial output, whether the correct objects or paths were retrieved and accurately visualized, and (iii) *Query success*: Whether a valid executable database query was generated within the allowed retry limit. Among the 54 queries, 28 required visualizations. All three criteria were checked

[2] https://github.com/langchain-ai/langgraph
[3] https://github.com/maplibre/maplibre-gl-js
[4] https://docs.langchain.com/

Metric	GPT-OSS (120B)	Gemini 3.1	GPT-5.4
Answer correctness (expanded in Table 2)	(54/54) 100%	(51/54) 94.4%	(46/54) 85.2%
Visualization correctness	(28/28) 100%	(26/28) 92.9%	(26/28) 92.9%
Query success	(54/54) 100% (3 retries)	(54/54) 100% (2 retries)	(54/54) 100% (1 retry)

Table 1. Performance comparison across LLMs.

Query Type	N	GPT-OSS (120B)	Gemini 3.1	GPT-5.4
PostGIS	15	15/15 (100%)	14/15 (93.3%)	11/15 (73.3%)
Neo4j	12	12/12 (100%)	12/12 (100.0%)	10/12 (83.3%)
Cross-database	10	10/10 (100%)	10/10 (100%)	10/10 (100%)
Conversational	17	17/17 (100%)	15/17 (88.2%)	15/17 (88.2%)
Overall	**54**	**54/54 (100%)**	**51/54 (94.4%)**	**46/54 (85.2%)**

Table 2. Answer correctness by query type for the three evaluated LLMs.

manually by the authors, comparing each response and visualization against the expected result for that query; partial matches were counted as incorrect.

5. Results and Discussion

The evaluation results are summarized in Tables 1 and 2. Table 1 presents the performance of the three LLMs across the evaluation criteria, while Table 2 provides a breakdown of answer correctness by query type. Across the 54 natural-language queries, GPT-OSS achieved perfect answer correctness. Gemini 3.1 produced incorrect answers in 3 cases, while GPT-5.4 generated 8 incorrect responses. For queries requiring visualization, all LLMs performed strongly: GPT-OSS again achieved perfect accuracy, whereas Gemini 3.1 and GPT-5.4 produced 2 incorrect visualizations. All three models required only a small number of retries across evaluations, indicating that the query refinement mechanism was effective in recovering from initial failures.

Discussion. The evaluation highlights several key observations. The perfect query success rate across all LLMs indicates that the framework design, particularly iterative query refinement and constrained execution, effectively helps ensure the generation of syntactically valid and executable database queries. As a result, performance differences primarily reflect how accurately each LLM interprets user intent and plans the sequence of operations required to retrieve relevant data.

In terms of answer correctness, GPT-OSS achieves perfect performance in this evaluation setting across all query types, suggesting that an open-source LLM can match or even exceed proprietary LLMs under the evaluated setup when guided with structured prompts and detailed schema context. GPT-5.4 shows the lowest overall accuracy (85.2%), suggesting that even strong reasoning-oriented models may struggle with structured data retrieval tasks. Gemini 3.1 performs more consistently (94.4%), although some performance loss is observed in PostGIS and conversational queries. Sample success cases and all failure cases are presented in Tables 3 and 4, respectively. Additionally, two example cases of cross-database queries are provided in Table 5. Similarly, selected visualization outputs produced by the evaluated LLMs are presented in Figure 5.

The query-type breakdown shows that cross-database queries are handled consistently across all models (100% in this evaluation). This suggests that once high-level intent is correctly identified, decomposition into sequential sub-queries is reliably managed by the framework pipeline[5]. In contrast, single-database spatial queries seem to be more sensitive to model-specific reasoning errors. Notably, GPT-5.4 failed on relatively simple aggregation queries such as "*What kinds of city objects are there?*", "*How many objects are there for each type?*", and "*List the attributes available for buildings.*" In these cases, GPT-OSS and Gemini 3.1 correctly applied operations such as `DISTINCT` and `GROUP BY`, whereas GPT-5.4 failed to perform the required aggregation or summarization.

Conversational queries introduce different challenges. While GPT-OSS maintains perfect performance, Gemini 3.1 and GPT-5.4 exhibit reduced accuracy (88.2% for both models), likely due to incomplete or incorrect propagation of context across turns. For example, in the sequence: "How many hospitals are in the area? → Can you list them with their names? → *Which building is closest to the first one?*", GPT-OSS correctly maintained context across all turns. For the final query, it retrieved the coordinates of the first hospital from Neo4j and used them to compute proximity in PostGIS. In contrast, Gemini 3.1 correctly queried Neo4j but propagated the hospital name instead of its coordinates when querying PostGIS, resulting in an empty result set. GPT-5.4, in this case, defaulted to the `cannot_answer` action. Similarly, in the sequence "What is the tallest building? → And the shortest? → How far apart are they? → *Give me a route between them.*", Gemini 3.1 lost context in the final step and generated a route to an unrelated object. In contrast, both GPT-OSS and GPT-5.4 correctly maintained context and produced the expected route between the identified buildings. These failures in maintaining conversational state ultimately affect not only answer correctness but also visualization accuracy.

Failure Case Re-evaluation Using Model Variants. To further examine the observed errors, we re-evaluated failure cases using alternative variants from the same model families. Specifically, failed responses from Gemini 3.1 (Flash Lite) were retested with Gemini 3.1 Pro, and failures from GPT-5.4 were re-evaluated using GPT-5.4-mini. This allows us to assess whether errors are consistent across a model family or specific to a given variant. For Gemini, the Pro variant resolved all previously observed failures. It correctly handled the level-of-detail query by identifying *LoD2* as the only available level, rather than returning a raw value. In the conversational hospital example, this time, it propagated coordinates, thereby performing correct cross-database querying. It also produced the correct route in the routing failure case, although one retry was required.

[5] Table 5 provides two cases of cross-database chaining.

Query Type	Query	GPT-OSS
PostGIS	**Q1**: Give me the 10 largest buildings by footprint area.	Found 10 results. Showing first 5: `[list]`
	Q2: What is the height range of buildings? Like the minimum and maximum.	The height range of buildings in the data is 2.2m (minimum) to 41.158m (maximum).
	Q3: Show me the location with coordinates of the tallest building.	The tallest building in the dataset is located at: `[lat,lon] coordinates`
Neo4j	**Q1**: How many road junctions are in the network?	There are 1731.
	Q2: What types of amenities are available?	Found 22 results. Showing first 5: `[list]`
	Q3: What's the longest single road segment in the network?	The longest single road segment in the network is 202.15 meters long (from junction 12408886628 to junction 4767215171).
	Q4: Find the 5 nearest cafes to the first hospital you can find.	Hospital: Erasmus Medisch Centrum 5 nearest cafés: `[list: Café name | Latitude | Longitude | Distance from hospital]`
Cross-database	**Q1**: How far is the nearest parking from the building with the biggest footprint?	The nearest parking (Q-Park Schiecentrale) is approximately 195 meters from the building with the largest footprint.
	Q2: What's the shortest route between the two tallest buildings?	Found a route with 35 waypoints (1087m). Shown on the map.
Conversational	**sequential interaction**	
	→ What is the tallest building?	The tallest building in the data is the one with ID B687.., which is 41.158 meters high.
	→ And the shortest?	The shortest is 2.2m (object ID C179...).
	→ How far apart are they?	They are approximately 507.36 meters apart.
	→ Give me a route between them.	Found a route with 28 waypoints (691m). Shown on the map.

Table 3. Sample success cases across query types. GPT-OSS responses are shown.

Query Type	Query	GPT-OSS	Gemini 3.1	GPT-5.4
PostGIS	**Q1**: What kinds of city objects are there?	Building	-	*[Failed to return distinct types.]*
	Q2: How many objects are there for each type?	There are 853 objects of type Building.	-	*[Failed to aggregate results]*
	Q3: List me the attributes available for the buildings.	The available building attributes are: bron_geo, TerrainHeight, bron_tex, voll_tex, status.	-	*[Failed to return distinct attributes]*
	Q4: How many levels of detail are available in the geometries?	There are 1.	*[Returned LoD value, not LoD count.]*	-
	Q5: What are the different roof types and how many buildings have each?	No results found for that query.	-	*[executed cannot_answer action]*
Neo4j	**Q1**: What types of amenities are available?	Found 22 results. Showing first 5: `[amenities list]`	-	*[same as above]*
	Q2: How many roads have speed limit available?	There are 2387.	-	*[same as above]*
Conversational	**Sequence 1**: How many hospitals are in the area? → Can you list them with their names? → **Which building is closest to the first one?**	The building with object ID 2283... is the closest to the hospital, at a distance of approximately 497m.	*[executed an SQL with the hospital name in postgres.]*	*[same as above]*
	Sequence 2: What is the tallest building? → And the shortest? → How far apart are they? → **Give me a route between them.**	Found a route with 28 waypoints (691m). Shown on the map.	*[Gave a route to a different object.]*	-
	Sequence 3: Find two nearest cafes to the tallest building. → And the shortest? → Now show me the route to the first cafe. → **Now show me the route to the second cafe.**	Found a route with 54 waypoints (1021m). Shown on the map.	-	*[Could not determine the second cafe coordinates from prior context]*

Table 4. Failure cases. GPT-OSS responses are shown for reference, while errors in Gemini 3.1 and GPT-5.4 are described explicitly. Where '-', a correct response was provided by the LLM.

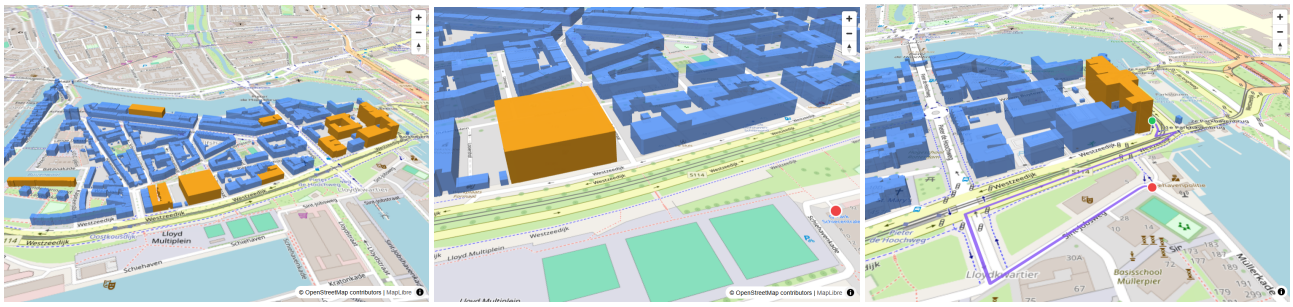
This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XII-4-W1-2026-219-2026

Query	Response	Generated SQL/Cypher

Are there any schools close to the smallest building?	Yes. The nearest school is **Delfshaven**, about **167 m** from the smallest building (height **2.2 m**); other schools are within 300–400 m.	`SELECT object_id, height_m, ST_Y(ST_Transform(ST_SetSRID(centroid,28992),4326)) AS lat, ST_X(ST_Transform(ST_SetSRID(centroid,28992),4326)) AS lon FROM cityllm.cj_llm_preferred WHERE obj_type='Building' AND height_m IS NOT NULL ORDER BY height_m ASC LIMIT 1;`
		
		`MATCH (a:Amenity) WHERE a.amenity_type='school' WITH a, point.distance(point({latitude:a.lat, longitude:a.lon}), point({latitude:51.90770095132175, longitude:4.45532072772442})) AS dist ORDER BY dist LIMIT 5 RETURN a.name AS school_name, a.lat AS lat, a.lon AS lon, dist AS distance_m`

Find and name 5 parks near the center of the city, using the centroid of all buildings as the center point.	Centroid of all buildings: **51.9066269N,4.4543036E.** 5 Nearest parks: **De Schat van Schoonderloo** (113 m), **Kleine Avonturen** (489 m), **Zonne Tuin** (548 m), **Tuschinskipark** (615 m), and one unnamed park (806 m).	`SELECT 'all_buildings_centroid' AS object_id, ST_Y(ST_Centroid(ST_Collect(ST_Transform(centroid,4326)))) AS lat, ST_X(ST_Centroid(ST_Collect(ST_Transform(centroid,4326)))) AS lon FROM cityllm.cj_llm_preferred WHERE obj_type='Building';`
		
		`MATCH (a:Amenity) WHERE a.amenity_type='park' WITH a, point.distance(point({latitude:a.lat, longitude:a.lon}), point({latitude:51.90662691347196, longitude:4.454303621614631})) AS dist ORDER BY dist LIMIT 5 RETURN a.name AS park_name, a.lat AS lat, a.lon AS lon, dist AS distance_m`

Table 5. Example natural-language queries with corresponding LLM (GPT-OSS) responses and generated SQL (PostGIS) and Cypher (Neo4j) queries, illustrating cross-database chaining.



(a) **GPT-OSS**
Query: *Give me the 10 largest buildings by footprint area.*
Response: *Found 10 results.*

(b) **Gemini 3.1**
Query: *How far is the nearest parking from the building with the biggest footprint?*
Response: *The nearest parking to the building with the biggest footprint is Q-Park Schiecentrale, located 194.67 meters away.*

(c) **GPT-5.4**
Query: *Show me a route from the tallest building to the nearest police station.*
Response: *Found a route with 33 waypoints. Shown on the map.*

Figure 5. Representative visualization outputs produced by the evaluated LLMs.

These results suggest that the earlier failures are not inherent to the Gemini family, but are dependent on the specific variant. For GPT-5.4, the mini variant corrected three out of eight failure cases. It successfully handled aggregation-based queries such as identifying object types, counting objects per type, and listing building attributes, by correctly applying `DISTINCT` and `GROUP BY`. However, the remaining failures persisted, with the model often defaulting to the `cannot_answer` action. This indicates that while some errors are variant-dependent, others appear consistent across the GPT-5.4 family.

Implications. The results show that CityLLM can effectively provide natural-language interaction capability over semantic 3D city models. The evaluation confirms that strong performance is achieved when LLMs correctly interpret user intent and operate strictly within a well-defined context (e.g., schema definitions, tool selection, and structured output formats) with availability of retrying and chaining abilities. Evaluation also shows that even advanced models can underperform when these constraints are not followed. Overall, the findings position CityLLM as a lightweight, extensible framework for conversational querying of semantic city data. By mapping CityJSON data into a compact relational schema, exposing executed queries, providing interactive visualizations, and supporting modular integration of additional data sources, the framework enables flexible and interpretable workflows. Importantly, it lowers the barrier to accessing and analyzing complex city data, making it useful for non-expert users and interdisciplinary research as such datasets continue to grow in scale and complexity.

6. Conclusion

This paper presents CityLLM, a framework for natural-language querying of semantic 3D city models, which integrates LLMs with spatial and graph data backends. Evaluated on a CityJSON dataset of Rotterdam (853 LoD2 buildings) using 54 queries across spatial, graph, cross-database, and conversational scenarios, the results show strong performance, with GPT-OSS out-

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XII-4-W1-2026-219-2026

performing Gemini 3.1 and GPT-5.4. The findings highlight that LLMs perform best when operating within a well-defined context, supported by retrying and cross-database chaining.

Several directions remain for future work. Introducing additional data sources and backends, such as RDF store, vector database and real-time data, could allow more sophisticated urban analysis. Another direction is reducing the system prompt size (currently >3k tokens) through dynamic prompt construction, for example by retrieving only relevant contexts via vector search (Lamsal and Zlatanova, 2026). Finally, integrating BIM data (Lamsal et al., 2026) could enable unified querying across city models and building-level information.

Acknowledgements

This work was supported by the Australian Research Council (ARC) grant: ARC ITRP IH210100048, and was done in collaboration with FrontierSI.

References

3dcitydb Contributors, 2025. 3D City Database: The Open Source CityGML Database. `https://github.com/3dcitydb/3dcitydb`.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S. et al., 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Akroyd, J., Mosbach, S., Bhave, A., Kraft, M., 2021. Universal digital twin-a dynamic knowledge graph. *Data-Centric Engineering*, 2, e14.

Boeing, G., 2025. Modeling and analyzing urban networks and amenities with OSMnx. *Geographical Analysis*, 57(4), 567–577.

Consortium, W. W. W. et al., 2012. R2RML: RDB to RDF mapping language.

Deng, X., Tsai, Y.-K., Ganguly, S., Tran, D., Quek, H. Y., Ang, W., Phua, S. Z., Mosbach, S., Akroyd, J., Kraft, M., 2025. Question answering system for smart cities and urban planning with The World Avatar. *IET Smart Cities*, 7(1), e70009.

Feng, Y., Ding, L., Xiao, G., 2023. Geoqamap-geographic question answering with maps leveraging llm and open knowledge base (short paper). *12th International Conference on Geographic Information Science (GIScience 2023)*, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 28–1.

Gröger, G., Plümer, L., 2012. CityGML–Interoperable semantic 3D city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 71, 12–33.

Hansson, F., 2024. Linked geodata: CityGML represented as a virtual knowledge graph. *Student thesis series INES*.

Kanna, K., Kolbe, T. H., 2025a. Advanced user interaction with urban digital twins using large language models. *SPRS Geospatial Week (GSW) 2025*.

Kanna, K., Kolbe, T. H., 2025b. Automatic enrichment of semantic 3d city models using large language models. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 10, 105–112.

Kolbe, T. H., 2009. Representing and exchanging 3d city models with citygml. *3D geo-information sciences*, Springer, 15–31.

Lamsal, R., Zlatanova, S., 2026. Query2property: Semantic retrieval of ifc properties for natural language bim queries. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XI-4-2026, 323–330.

Lamsal, R., Zlatanova, S., Xu, H., Sun, Y., Shen, J. X., 2026. IfcLLM: Natural Language Querying of IFC Models through Complementary Relational and Graph Representations. *arXiv preprint arXiv:2605.13236*.

Ledoux, H., Arroyo Ohori, K., Kumar, K., Dukai, B., Labetski, A., Vitalis, S., 2019. CityJSON: A compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, 4(1), 1–12.

Liang, J., Hou, S., Jiao, H., Qing, Y., Zhao, A., Shen, Z., Xiang, L., Wu, H., 2025. GeoGraphRAG: A graph-based retrieval-augmented generation approach for empowering large language models in automated geospatial modeling. *International Journal of Applied Earth Observation and Geoinformation*, 142, 104712.

Lifelo, Z., Ding, J., Ning, H., Dhelim, S., 2024. Artificial intelligence-enabled metaverse for sustainable smart cities: Technologies, applications, challenges, and future directions. *Electronics*, 13(24), 4874.

Liu, S.-q., Wang, C., 2025. Kcitychatbot: A knowledge graph based chatbot system for large-scale citygml dataset. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 48, 99–105.

Pan, Y., Wang, M., Lu, L., Lamsal, R., Pärn, E., Zlatanova, S., Brilakis, I., 2026. LLM-enabled multi-agent framework for natural language interaction with graph-based digital twins. *Automation in Construction*, 183, 106791.

Santhanavanich, T., Coors, V., 2025. OGC-AI: A Retrieval-Augmented Large Language Model Interface for Open Geospatial Consortium Web Services. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 48, 157–163.

Sun, Y., Shen, X., Zlatanova, S., Barati, K., Linke, J., 2026. Text-based automatic knowledge graph construction for road infrastructure operations management. *Automation in Construction*, 182, 106733.

Wikidata, 2025. Wikidata Query Service. `https://query.wikidata.org/`.

Xu, F., Zhang, J., Gao, C., Feng, J., Li, Y., 2023. Urban generative intelligence (ugi): A foundational platform for agents in embodied city environment. *arXiv preprint arXiv:2312.11813*.

Yin, W., Xue, Y., Liu, Z., Li, H., Werner, M., 2025. LLM-enhanced disaster geolocalization using implicit geoinformation from multimodal data: A case study of Hurricane Harvey. *International Journal of Applied Earth Observation and Geoinformation*, 137, 104423.

Zhang, W., Han, J., Xu, Z., Ni, H., Liu, H., Xiong, H., 2024. Towards urban general intelligence: A review and outlook of urban foundation models. *arXiv preprint arXiv:2402.01749*.