Advanced User Interaction with Urban Digital Twins using Large Language Models

Khaoula Kanna [1], Thomas H. Kolbe [1]

[1]Chair of Geoinformatics, Technical University of Munich, 80333 Munich, Germany - (khaoula.kanna, thomas.kolbe)@tum.de

Keywords: Urban Digital Twins, CityGML, SensorThings API, SQL, Large Language Models (LLMs), OpenAI.

Abstract

Large language models (LLMs), such as OpenAI's Generative Pre-trained Transformer (GPT), commonly known as ChatGPT has witnessed a very rapid evolution which has opened the door for new possibilities across various industries and academic fields. These advanced technologies are transforming how we view and interact with data, how we communicate and solve complex problems. In this paper, we present a framework that employs LLMs to interact with an Urban Digital Twin (UDT) of a district. The framework utilizes the semantic richness of CityGML for representing 3D city models and the SensorThings API for managing dynamic sensor data, allowing users to query and visualize geospatial and dynamic information intuitively. Through experiments with different types of queries from stakeholders, varying from city planners, to utility providers, and citizens, we found that LLMs can effectively translate natural language queries into complex geospatial and temporal operations, narrowing the gap between non-expert users and complex urban datasets into a fine margin. The results shed light on the potential of LLMs to support decision-making in smart city applications.

1. Introduction

Smart city projects often require a lot of expertise to analyze diverse datasets for decision-making purposes, especially geospatial data. This can be very time-consuming as well as very expensive. In the last few years, the heavy advancements in Generative AI, particularly Large Language Models (LLMs), have drawn eyes and attention from both industry and academia. LLMs are known for their superior and impressive capabilities in better interpreting and generating natural language. They have successfully been integrated into various applications, such as robotics and medicine (Wu et al., 2024), serving as powerful assistants across different domains. Moreover, and on a specific note, LLMs show great potential for understanding of GIS and related fields [(Singh et al., 2024), (Huang et al., 2024), (Li & Ning, 2023)]. LLMs have facilitated intuitive question answering which made the gap between geospatial data and end-users to almost inexistent by making geospatial information more accessible to non-experts. However, in the context of smart cities, decision-making increasingly relies on Urban Digital Twins (UDTs), which host not only static geospatial data but also dynamic data, such as real-time sensor information. Many studies have focused on LLM integration with GIS data and workflows, however, there is a noticeable gap in research exploring how LLMs can interact with complex datasets including static and dynamic data within UDTs to support user interaction with UDTs and urban decision-making.

Addressing these challenges requires the use of LLMs not only for natural language interaction but also customizing their capabilities to meet the particular needs of different stakeholders who are directly involved in smart city projects. In this paper, we apply the concept of Urban Digital Twins to the EU research project ASCEND, aiming at creating Positive Energy Districts for the use case of the Harthof district in Munich. This project involves various partners, including the City of Munich, the MGS as Partner A, which is the Society for Urban Renewal, the Stadtwerke München as Partner B, which is City Utilities, and local citizens. Each of these stakeholders has distinct interests and information needs. For instance, Partner A is primarily focused on identifying buildings that are less energy efficient and require refurbishment strategies, whereas Partner B is more concerned with monitoring the electricity demand and consumption of each building at specific times to optimize energy distribution. In contrast, citizens are interested in more localized queries, such as: "Show me the building on X Street with the number Y, provide the solar potential of its roof, and calculate how many photovoltaic panels I can install, the expected energy generation, and the associated costs."

To address these varied requirements, we aim to build a flexible framework capable of answering complex queries from different user groups, making it possible for each stakeholder to interact intuitively with the digital twin using natural text. Given the diverse needs and varying levels of expertise among our stakeholders, the key challenge is to develop a unified framework that allows each group to interact with the Urban Digital Twin using natural language, without requiring specialized knowledge of geospatial databases, data models, or query languages. To address this, our research focuses the following hypothesis: ***Integrating LLMs with the open geospatial standards CityGML and SensorThings API in the UDT enables the accurate translation of natural language into structured geospatial and temporal queries, allowing diverse user groups (e.g., city planners, utility providers, and citizens) to receive accurate relevant responses in different languages, including German and English.***

By addressing this hypothesis, this study aims to establish a scientific approach for employing LLMs to enable intuitive interaction with Urban Digital Twins, supporting complex urban data interactions across multiple stakeholder domains. The rest of the paper is structured as follows: Section 2 lists some recent advancements and identifies the research gaps in this area. Section 3 introduces the methodology of the framework. Section 4 introduces the case study of the framework in the district Harthof in Munich, with the results. This section is followed by the evaluation section, where the framework is being evaluated using evaluation metrics. Section 6 discusses the framework result, followed by a conclusion for future work.

2. Related Work

Artificial intelligence (AI) has gradually been integrated into the study of geospatial research in recent years, giving birth to GeoAI. This multidisciplinary area of study analyzes very large and varied geospatial data using AI which ultimately improves our comprehension of urban environments. GeoAI has been utilized in many different fields, such as the generation of 3D city models from aerial data and object detection in satellite imagery ((Biljecki & Ito, 2021), (Kang et al., 2020), (F. Zhang et al., 2023), (Wu & Biljecki, 2021)). GeoAI has also significantly

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-X-G-2025-469-2025

changed 3D city modeling. Researchers have created novel ways to create realistic and intricate 3D representations of urban landscapes using AI approaches. For instance, (Pang & Biljecki, 2022) worked on creating three-dimensional city models using aerial imagery. These advancements have set the stage for more sophisticated AI applications, particularly in the context of urban digital twins and dynamic urban environments. This however should make GeoAI go beyond merely applying AI and machine learning methods to geoscience use cases, and should also integrate spatial, temporal, and place-based aspects into the AI methods (Janowicz, 2023).

In this regard, using LLMs opens up additional GeoAI possibilities. It provides the capacity to handle and comprehend complex geospatial data by fusing natural language with spatial data. Many researchers investigated the use of LLMs with GIS in various ways, even though it's still in the early stages of development and deployment. For example, (Mooney et al., 2023) evaluated ChatGPT’s understanding of GIS concepts by having it take an exam based on GIS textbooks without any prompting or finetuning. They found out that even though GPT showcased a very promising understanding of geospatial concepts, there is still room for improvement. (Li & Ning, 2023) went one step further with their prototype, LLM-Geo, which uses the GPT-4 API to conduct spatial analysis in an autonomous manner. The findings demonstrate how automating spatial analytic operations and improving the usability of GIS technology for non-specialists in the field could revolutionize the GIS industry. Similar to this, (Y. Zhang, Wei, et al., 2024) created the user-friendly framework GeoGPT, which blends LLMs with established GIS community tools to tackle a variety of geographic jobs and GIS analysis independently.

Few other works focused on creating innovative user-interfaces. For instance, (Census GPT, 2023) created a user interface that takes natural language questions, converts them to SQL using GPT-3.5, then queries an SQL database, and finally provides 2D data visualization using tools such as Mapbox and Plotly. ChatGeoGPT (Strong, 2023) is another initiative of using ChatGPT to talk to the Open Street Map and query geographical information. The authors found out that ChatGPT was trained on the Overpass API (which is an API to interact with OSM map data) and can construct queries better than humans do. ChatGeoGPT uses GPT-3’s knowledge to convert natural text to Overpass QL queries and translate API responses back into easy-to-understand, actionable summaries. The authors however pointed out that ChatGPT has limitations for more sophisticated Overpass QL queries.

LLMs still face many challenges and obstacles on its way, distinctly when it comes to discipline-specific knowledge, despite all these promising results and achievements. This limitation often results in hallucinations, where the models produce incorrect or misleading information. Nevertheless, fine-tuning LLMs for GIS-specific tasks enhances their accuracy and reliability. For instance, BB-GeoGPT (Y. Zhang, Wang, et al., 2024) is a fine-tuned LLM for GIS-specific knowledge. BB-GeoGPT was developed on the LLaMA-2-7B model using thousands of GIS datasets. While BB-GeoGPT exhibits promising results, it lags behind more advanced commercial models like GPT-3.5-turbo. LAMP (Balsebre et al., 2024) is another fine-tuned LLM for GIS specific knowledge. It was developed on the LLaMa-2-7B-Chat model. The authors compared the model to famous LLMs such as LLaMa-2-70B, GPT-3.5 and GPT-4, and the results were promising, however, hallucination problems were still frequent as the authors pointed out, and the lagging behind commercial models such as GPT-4 is still prominent.

In general, LLMs can help in making the transition toward a world where users do not need to be experts in specific fields. This can free up time for users to focus on decision-making instead. However, due to the complexity of data sources including static 3D city models and dynamic IoT datastreams, integrating LLMs into complex ecosystems such as UDTs becomes very challenging. To address the heterogeneity of these data sources, open standards developed by the Open Geospatial Consortium (OGC) are important. For instance, SensorThings API (Liang, Khalafbeigi, et al., 2021) is used for managing dynamic IoT data, while CityGML (Gröger et al., 2012) provides a standardized format for representing 3D city models. The OGC standard CityGML is the most widely adopted open standard for the exchange and representation of semantic 3D City Models. CityGML enables both geometrical and semantic representations of urban objects at multiple levels of detail, ranging from windows and building units to entire districts and cities. For managing CityGML datasets, several free and commercial tools are available. One of the most prominent is 3DCityDB (Z. Yao et al., 2018), an open-source software suite designed for spatial relational database management systems (SRDBMS) such as Oracle Spatial and PostGIS. CityGML is widely used by many cities around the world for urban planning, management, and

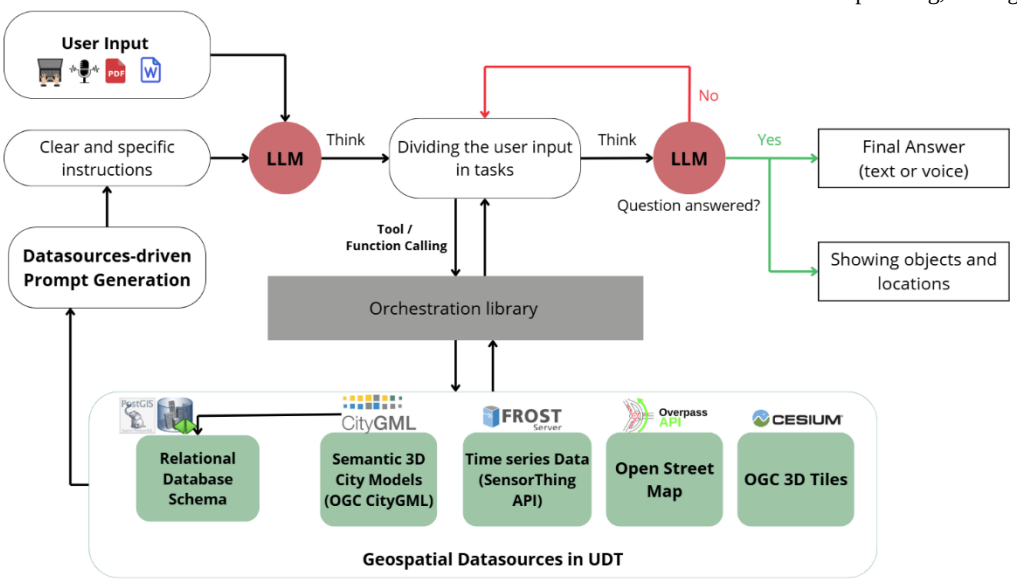

Figure 1. The overall methodology of the framework

analysis. However, its complexity poses a challenge and a difficulty for non-experts or professionals from other domains. It is designed in such a way to fully support all aspects of urban object representation, which renders it somewhat difficult to navigate without specialized knowledge and guidance. While visualization through 3D viewers can make CityGML data more accessible by allowing users to interact with urban objects and view their attributes, complex and nested queries still require the expertise of a GIS professional to extract meaningful insights from the semantic model. This makes it difficult for non-GIS experts to interact with the 3D City model.

UDTs are not only static, they also include dynamic data from IoT sensors or simulations. One widely used way of representing dynamic data is the OGC SensorThings API standard (Liang, Khalafbeigi, et al., 2021). The SensorThings API is a RESTful and lightweight API that uses JSON data encodings for representing sensors and their data in a standardized and unified format. Integrating dynamic data with 3D city models is challenging but has been approached through various methods. For instance, CityGML 3.0 introduces the Dynamizer module, which enables the integration of time-series data with CityGML (Kutzner et al., 2020). Another approach, proposed by (Santhanavanich & Coors, 2021), is CityThings, which integrates CityGML with the SensorThings API. The idea behind CityThings is to include the gml_id, which is the unique identifier of the urban object, in the properties of the Thing entity in the SensorThings API. This allows dynamic data to be queried using the gml_id of the corresponding CityGML feature, establishing a direct link between static 3D models and real-time sensor data. For implementing the SensorThings API in this context, the FROST Server (Fraunhofer IOSB, 2019) is a widely used solution. It provides very efficient and scalable web platform to help manage IoT data streams given the fact that it's an open-source implementation of the SensorThings API. The FROST Server ensures continuous dynamic data retrieval and allows the UDTs to include real-time or simulated dynamic data alongside their static 3D models.

In this paper, we propose a framework for interacting with an UDT of a district. The UDT integrates a CityGML model with dynamic time-series data such as monthly heat demand and hourly electricity consumption per building, managed using the 3DCityDB and the SensorThings API. The LLM determines the suitable action to take and outputs the results in a clear, user-friendly format, all based on the user's query. In our framework, the LLM acts as the 'brain,' while the functions and tools are the 'hands' (Li & Ning, 2023). The results are visualized in a 3D environment using CesiumJS with OGC 3D Tiles (Cozzi et al., 2019), to render and highlight specific objects in the digital twin. The entire framework is built on OGC standards (CityGML, SensorThings API, and 3D Tiles), ensuring scalability and compatibility for diverse applications.

3. Methodology of the research

This section presents the methodology behind our framework, which is designed to enable intuitive interaction with UDTs using LLMs. The system allows users to ask questions, which are interpreted and translated into structured data requests to retrieve geospatial and temporal information. Figure **1** shows the overall methodology of the framework. The framework is designed in three parts: the input that the LLM takes in, the geospatial data pool that the LLM needs to get data from, and the output that the LLM delivers. The input is of two types: the context prompt, which contains predefined instructions and context provided to the LLM to guide its behavior during interactions, and the user prompt. The user prompt can be of any type: text, voice or document. Second, the geospatial data pool is where the UDT data is stored. It includes the 3D city model in CityGML standard, which is stored and managed in the 3DCityDB. Moreover, all the dynamic data is stored in the FROST-Server, which is an implementation of the SensorThings API using the CityThing approach (Santhanavanich & Coors, 2021). Each urban object (apartment, building, street, district, or city) can be modelled as a Thing in the SensorThings API, and its unique identifier (gml_id) will be stored in the properties of the Thing. This creates a direct link between the CityGML and the SensorThings API. Additional contextual information, such as streets, restaurants, and building usage, data is retrieved from OpenStreetMap (OSM) using the Overpass API. The visualization of the 3D city model is enabled through the CesiumJS Web Viewer using the 3D Tiles format. The output that the LLM delivers is a text with the information the user asked for, and a visualization of the highlighted objects in the 3D Viewer.

In order for the LLM to interact with these external geospatial databases, it needs some ‘hands’ for that. In fact, the LLMs naturally function as a closed "black box," meaning their built-in knowledge is static and outdated, and they cannot interact directly with dynamic external systems without additional


```
  "building": {
    "columns": [
      {
      },
      {
        "column_name": "measured_height_unit",
        "data_type": "character varying",
        "description": "Unit of the measured height."
      },
      {
        "column_name": "storeys_above_ground",
        "data_type": "numeric",
        "description": "Number of storeys above ground."
      }
    ],
    "primaryKeys": [
      {
        "column_name": "id"
      }
    ],
    "foreignKeys": [
      {
        "table_name": "building",
        "column_name": "id",
        "foreign_table_name": "cityobject",
        "foreign_column_name": "id"
      },
      {
        "table_name": "building",
        "column_name": "objectclass_id",
        "foreign_table_name": "objectclass",
        "foreign_column_name": "id"
      },
```


Figure 2. Excerpt of the 3DCityDB relational schema, which includes 60 tables. The schema was provided to the LLM as a JSON file containing 758 lines.

components. This limitation in our framework is addressed through the Orchestration Layer, which enables the LLM to access external data sources and APIs, thereby augmenting its capabilities at runtime. In the Orchestration layer, the LLM can connect to external tools using its reasoning ability. For example, LLMs can call functions and use the function outputs in to solve the task. A notable and famous approach for function calling is ReAct (S. Yao et al., 2023), where LLMs invoke a function, analyze the outputs, and reason about the next action. The Retrieval-Augmented Generation or RAG for short (Lewis et al., 2020) is also an approach that enable the LLM to access external documents or databases dynamically, retrieving context-specific information before generating the final response. Common implementations of the Orchestration layer are LangChain and

the OpenAI’s API. Both of them support RAG and Function Calling.

Function	Explanation
run_sql_query(query)	The LLM passes the generated SQL query from the user input and connects to the 3DCityDB to run the query.
query_timeseries (gml_id, observedProperty, phenomenonTime)	The LLM generates the SensorThings API query and send a request to the SensorThings API using the parameters of the function.
show_results (gmlids)	This function is designed to highlight specific objects in the Cesium Viewer.
query_OSM_data (overpass_query)	The LLM can find specific objects in the OSM (restaurants, transports, streets) using the Overpass QL language.
show_marker (latitude, longitude)	Shows a marker on the viewer using coordinates of a place. Usually used after querying from the OSM.
calculate_distance (lat1, long1, lat2, long2)	This function calculates the distance between two objects using their coordinates.

Table 1. The defined functions for the AI Assistant to interact with the UDT.

Our methodology uses the RAG and Function Calling tools in the Orchestration Layer to enable direct interaction between the LLM and the UDT. Specifically, RAG is employed to access the relational database schema of the 3DCityDB, which is storing the semantic 3D city model. To facilitate the LLM’s understanding of the complex schema, the 3DCityDB schema was exported in JSON format, enriched with detailed descriptions for each table and their relationships. Each table in the schema includes its primary key, along with foreign keys that define the relationships between tables. Figure **2** lists an excerpt of the schema of the 3DCityDB and shows the *building* table, which contains detailed information about buildings, their classifications and functions according to the CityGML standard. The *building* table has columns such as *id* (primary key of the building table), *storeys_above_ground* (which represents the number of storeys of the building), and *objectclass_id* (foreign key linking the building table to the objectclass table), among many other columns. By incorporating the column description, the primary key and the foreign keys with the relationships to the other tables, the LLM is able to accurately navigate and query the 3DCityDB, facilitating writing complex geospatial queries. The RAG can also be used if the user input includes a document and shall enable the LLM to go through the document and analyze it to answer the user question.

Indeed, the LLM is able to generate even very complex SQL queries based on the user input and the database schema, but to run the SQL query on the database, the LLM needs to call a specific function. For this reason, Function Calling was used to allow the LLM invoke functions and access additional resources and data. Technically, the LLM analyses the user input and decides which function to call based on the user’s prompt, and call the function. An example would be calling a weather API to get current weather of a specific city. LLMs such as GPT-4 and GPT-3.5 are trained to recognize when a function should be invoked and then generate a JSON object with the necessary arguments for that function call. The functions triggered by this process serve as tools within the AI application, and multiple functions can be defined and utilized in a single request. Table 1 enumerates the functions used in our framework.
For example, if a user asks: “Show me all the buildings in *street_name* that have more than 7 storeys.”, this requires the LLM to write a SQL query that would answer this question according to the 3DCityDB schema. In this case, the gml_ids are stored in the cityobject table, where the buildings have an objectclass_id of 26. This should be joined with the address and address_to_building tables where the *street_name* is stored. The generated SQL query is as follows:

```
SELECT DISTINCT co.gmlid
FROM cityobject co
JOIN building b ON b.id = co.id
JOIN address_to_building ab ON b.id = ab.building_id
JOIN address a ON ab.address_id = a.id
WHERE a.street ='street_name'
        AND b.storeys_above_ground > 7
        AND co.objectclass_id = 26;
```

The LLM then generates this SQL query based on the table and column descriptions and primary/foreign keys information. Afterwards, the LLM calls the *run_sql_query* function and passes the generated SQL query to run it on the 3DCityDB. Given that the 3DCityDB is based on PostgreSQL with PostGIS extension, the LLM can also call PostGIS functions such as ST_3DAREA to calculate areas and ST_VOLUME to calculate volumes of buildings if required. The LLM then analyzes the output and decides on the next action. If the *run_sql_query* function returns an error from the database, the LLM analyzes the error and

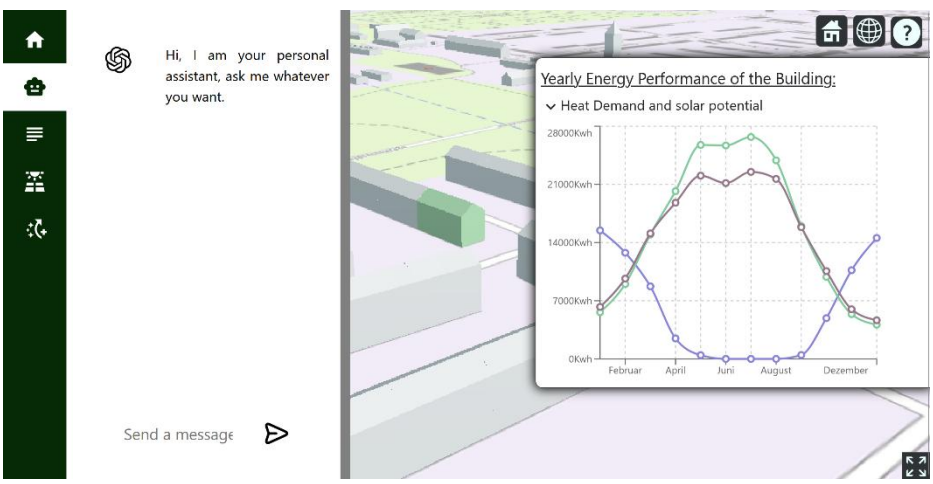

Figure 3. The User Interface of the UDT of Harthof

corrects the SQL query accordingly. If there is no error, the LLM calls the *show_results* function and passes in the building gml_ids returned from the database, then highlights the results using the styling functions of CesiumJS to show the buildings to the user. If the user query includes questions about dynamic data, the LLM uses the gml_ids to generate the SensorThings API query accordingly and call the *query_timeseries* function. Places from OSM can also be fetched using generated Overpass queries by the LLM.

4. Case Study

4.1 Case Study context

The framework was developed within the scope of the European project ASCEND. The project aims to mitigate the effects of climate change through the creation of Positive Clean Energy Districts in Europe. The project area in Munich is the Harthof district. The Harthof district exemplifies many of Munich’s residential areas, with most buildings constructed between the 1940s and 1970s. It covers 56 ha and is home to 11,500 inhabitants. It is a primarily residential area with 119 residential buildings and 5500 residential units. The ASCEND EU-Project aims to develop, test, and implement various initiatives, including technical and digital requirements, building refurbishments, PV panels installations and new constructions and mobility concepts, while measuring the impact of these actions to facilitate replication on other districts and European cities. To achieve these goals and make the Harthof district a positive clean energy district (PCED), it is essential to calculate and monitor various social, energetic, and financial Key Performance Indicators (KPIs) at different levels and with different time resolution. In fact, most KPIs are time-dependent. Examples of KPIs are the monthly heat energy demand of each building, hourly electricity consumption of the apartments within each building, and monthly roof and façade solar irradiation of a building.

4.2 Data preparation

To calculate the energy KPIs in the Harthof district, two categories of data were included: publicly available data and private data. The publicly available data consists of CityGML data in Level of Detail 2 (LoD2), made accessible by the State Office for Digitization and Surveying in Bavaria. The LOD2 downloaded data includes information such as the number of storeys above ground, roof type, measured height, building usage, building type, etc. Complementary data such as the year of construction, number of residential units per building, number of inhabitants per building, and the heating system (gas, oil, district heating) was obtained from the city of Munich in a 2D format and merged with the LOD2 CityGML data. The resulted CityGML dataset was then stored in the 3DCityDB, using the software 3DCityDB Importer/Exporter. Afterwards, the software SimStadt (HFT Stuttgart, 2015), which is an energy urban simulation environment, was used to calculate energy indicators in the Harthof district using the CityGML dataset. Additionally, solar potential analysis was also calculated for roofs and facades of each building using the 3D city model (Willenborg et al., 2018). All the time-series indicators were then stored according to the SensorThings API in the FROST Server using the CityThings approach.

4.3 Implementation of the solution

The implementation of the system consists of two main components: A Cesium Viewer, where the LOD2 model and the calculated KPIs can be visualized, and a chat widget that allows users to interact with the AI Assistant. The user interface is shown in Figure **3**. In the Cesium Viewer, users can explore LOD2 buildings of the Harthof district. Additionally, clicking on a building reveals all its attributes stored in the 3DCityDB, together with the KPIs as time-series data. The time-series data are queried using the building’s gml_id directly from the SensorThings API. To the left, we have added a chat window where users can communicate directly with the AI Assistant.

To implement the AI Assistant, the OpenAI’s Assistant API was used. The Assistant API allows developers to create customized agents that are capable of assisting and automating complex tasks. Moreover, in order for the LLM to interact with external knowledge, the API is currently supporting three types of tools: Code Interpreter, File Search, and Function calling. These tools can be considered as the implementation of the Orchestration Layer. The Code Interpreter tool allows the assistant to execute scripts (e.g. running complex mathematical calculations using Python). The File Search allows developers to retrieve relevant information by searching through documents using the Retrieval Augmented Generation (RAG) framework. And with Function Calling, the LLM can invoke functions if required after analyzing the user’s input.

4.4 Results

To interact with the system, we have tried different queries. Initially, we prompted simple queries, such as *“Show me the buildings in Röblingweg Street.”* This request required a basic SQL query to the 3DCityDB to retrieve and highlight the

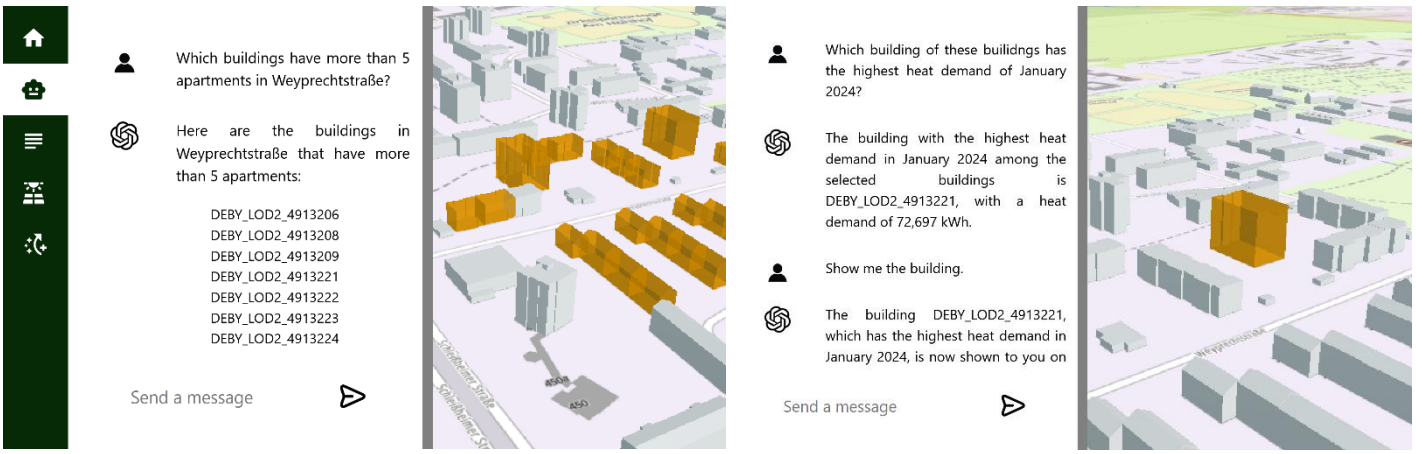

Figure 4. Examples of user questions to the Assistant.

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-X-G-2025-469-2025 | © Author(s) 2025. CC BY 4.0 License.

specified buildings. Next, we asked about specific indicators for individual buildings. For example, *"Which building in Röblingweg Street had the highest heat demand in January 2024?"* and *"What is the total heat demand for all buildings in Röblingweg Street in January 2024?"* The Assistant responded to these queries with a high degree of accuracy within a few seconds. It was able to achieve this by retrieving building IDs from the 3DCityDB and subsequently querying the SensorThings API to access the relevant data. Figure **4** illustrates a query in which the Assistant was asked to highlight the building with the highest heat demand in February 2024 among those on a specific street with more than five apartments.

We also tested more advanced queries that involved joining different tables within the 3DCityDB. Examples of such queries include, *"How much living space do the buildings in Röblingweg street have per resident?"* and *"What is the oldest building in the Röblingweg Street, and what is the distance of this building to the nearest metro station?".* The Assistant was able to find the correct SQL query and fetch the corresponding data. It was also able to generate the correct Overpass QL query to find nearby metro stations and calculate the distance to these stations using the *calculate_distance* function.

Additionally, we tested the system's ability to process documents. We uploaded an energy certificate for a random building and asked the Assistant to locate the building and summarize the information contained in the document (as shown in Figure **5**). In this case, the Assistant used the RAG to navigate the uploaded document, extract the building's address, search for it in the 3DCityDB, and locate it in the 3D viewer.

However, the system faced challenges with complex requests. In this case, the user should bring the request down to smaller steps to avoid such issue. For example, instead of asking: "*Which building in Röblingweg Street with more than 6 apartments has the highest electricity consumption in the last 24h*?", the user could ask first for "*Which buildings in Röblingweg Street have more than 6 apartments?",* then "*Which of these buildings above had the highest electricity consumption in the last 24h*?".

In our experiment, two OpenAI models were employed: GPT-3.5-turbo and GPT-4o. For simple queries and tasks, such as highlighting specific buildings, both models performed comparably. However, as the complexity of the requests increased, GPT-4o consistently outperformed GPT-3.5-turbo on nearly all queries. This can be attributed to GPT-4o's superior knowledge of SQL and its enhanced ability to handle complex tasks. Consequently, GPT-4o was more effective in addressing complex user queries compared to other models. If the SQL query happens to be wrongly generated by the model, we automatically forward the error from the database to the model to analyze it and re-write the SQL query accordingly. For example, the model might have looked for a column name that does not exist. The model gets the error from the database that this column does not exist, which makes the model go back and search in the database schema for the right column.

5. Evaluation of the framework

In this chapter, we evaluate the proposed framework to validate the research hypothesis outlined in the introduction. The goal is to assess whether our framework can accurately interpret complex queries from diverse user groups and produce meaningful outputs using the open standards CityGML, SpatialSQL, and SensorThings API. To evaluate our framework, we have collected 71 queries in both German and English. Examples of these queries and the generated SQL queries can be found here: https://github.com/tum-gis/ai_assistant_queries. As our framework intends to serve as an assistant for the UDT of the Harthof district, these queries were collected from the partners of the project that have different backgrounds and interests. For example, the Partner A (Society for Urban Renewal) is more interested in renovation-related queries. They would like to know which buildings are already renovated, and which buildings need a refurbishment based on the energy consumption of the buildings. The Partner B (the City Utilities) do have questions regarding the buildings attached to the district heating and the grid. Citizens have specific questions about the buildings they live in, such as building energy consumption, energy management, refurbishment strategies, costs, etc. We also collected queries from GIS experts to evaluate the framework's capability with complex geospatial tasks.

As mentioned previously, we have used two models for implementing the framework: GPT-3.5-turbo and GPT-4o. To measure the relevance, accuracy and completeness of the generated outputs, we use three metrics: Precision, Recall and F1-Score. Precision is the proportion of true positive predictions over true positive and false positive predictions made by the model. Recall which is also known as true positive rate, is equal to true positives divided by the sum of true positives and false negatives. Lastly, the F1-Score balances the two metrics to ensure both relevance and completeness. It can be calculated as follows:

$$F1\_Score = 2\,\frac{\text{Precision x Recall}}{\text{Precicion+Recall}} \qquad (1)$$

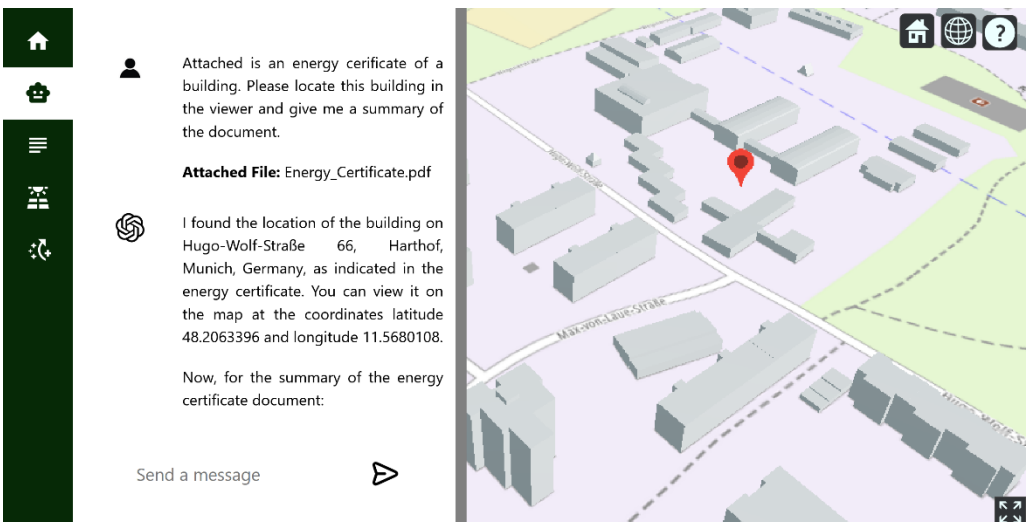

Figure 5. Example of a user request to the Assistant joining in an energy certificate of a random building and asking the Assistant to locate the building and to summarize the important information mentioned in the document.

The Precision, Recall and consequently the F1-Score metrics were calculated **10 times** for each single query. Each query was asked to 10 different GPT-3.5-turbo-based AI Assistants, and to 10 different GPT-4o-based AI Assistants. Then, we calculated the average Precision and Recall for each query and for each LLM.

Query Example	GPT-3.5 F1-Score	GPT-4o F1-Score
Show all buildings in Röblingweg Street	0.94	1
Which building in Parlestaße Street is the oldest?	0.81	1
Find all buildings with more than 5 apartments	0.75	0.92
Which buildings in Weyprechtstraße street are not connected to the district heating?	0.60	0.82
Show buildings in Röblingweg Street with yearly global solar irradiation of more than 8000 kWh	0.42	0.67
Which building in Röblingweg street had the highest heat energy demand in January 2024?	0.82	1
Locate the building named in the energy certificate	0.81	1
Calculate distance of this building X to nearest metro station	0.80	0.92

Table 2. The evaluation results obtained by using our framework based on two large language models (gpt-3.5-turbo and gpt-4o) on 8 example queries out of 71 queries.

Let’s take the query “*Show all buildings in Röblingweg Street*” as an example. This query was asked to 10 different GPT-3.5-turbo-based Assistants. Each time, we divide the true positives (TP) (in this case, all the buildings correctly identified as being in the Röblingweg Street), by the sum of true positives (TP) and false positives (FP) to calculate Precision. A false positive occurs when the model returns buildings that are not actually located in Röblingweg Street but are by mistake included in the result set. Similarly, Recall is calculated by dividing the true positives (TP) by the sum of true positives (TP) and false negatives (FN). A false negative in this scenario would be any building that is located in Röblingweg Street but was not retrieved by the model in the results. For the GPT-3.5-turbo model, while it achieved relatively high precision, it occasionally included a few buildings that were not in Röblingweg Street (false positives), and in some cases, it missed a few relevant buildings (false negatives). This caused minor drops in both Precision and Recall, which resulted in an average F1-Score of 0.94. On the other hand, GPT-4o consistently returned the correct set of buildings in every time. There were no false positives or false negatives, leading to perfect Precision and Recall values of 1.0, and consequently an F1-Score of 1.0.

Table 2 shows the calculated F1-Score for 8 example queries out of 71 queries. From the results, it can be seen that GPT-3.5 performed well on most simple queries, with some errors for the complex queries. On the other hand, GPT-4o performed very well on most queries, even the complex ones. Our framework achieved an overall accuracy rate of 62% with GPT-3.5-turbo and 81% with GPT-4o. The average response time for simple queries was 1.2 seconds, while more complex queries took 3.4 seconds on average.

Based on these results, our framework successfully answered typical queries by different user groups using the semantic and geometric richness of CityGML, its systematic mapping onto a well-documented 3D geodatabase schema, and the well-structured dynamic data from the SensorThings API. This demonstrates the usability and relevance of the system and validates the research hypothesis. However, there were some incorrect outputs and hallucinations generated by the model. This usually happens when the model is faced with complex scenarios and cannot solve the task. The hallucination problem is a common limitation that the user should be aware of. To reduce it, we need to implement additional guidance mechanisms to control the behavior of the LLM in such cases. For example, the LLM could ask for more clarification from the user or provide step-by-step instructions to solve the task.

6. Discussion

In this paper, we demonstrated the potential of integrating Large Language Models with semantic 3D City Models for enabling intuitive interaction with Urban Digital Twins. Our evaluation results showed that the framework could accurately interpret and respond to a variety of stakeholder queries, using the integration of LLMs with the open standards CityGML and SensorThings API. This enabled effective mapping of natural language queries onto structured geospatial and temporal data, therefore supporting our research hypothesis. The system successfully used CityGML’s semantic structure to retrieve static attributes and utilized SensorThings for dynamic data, achieving a significant accuracy boost. The use of these open standards enables the system to directly be applied to others districts and cities when their data is given in the standardized format. However, challenges like inconsistency in outputs and limitations in arithmetic reasoning were noted, especially with GPT-3.5-turbo. Moreover, performance dropped with more complex queries, indicating that while the framework is usable for most stakeholders, it still struggles with highly complex queries.

It is also important to mention other potential challenges of the system. First, there is the issue of uncertainty and unpredictability. The system does not consistently provide the same outputs for identical inputs, particularly when handling complex tasks. This issue is more frequent with GPT-3.5-turbo and occurs less frequently with GPT-4o. Lastly, while commercial models like GPT-4o are among the most advanced LLMs available, their cost and usage fees are significantly high. For this reason, many researchers advocate for the usage of open source LLMs and their fine-tuning for specific use cases. In our case, we believe that fine-tuning an LLM to work efficiently with the 3DCityDB and SensorThings API would be advantageous for the GIS community.

7. Conclusions and future work

In this work, we presented an innovative framework for user interaction with the Urban Digital Twin using Large Language Models. The primary goal of this system is to enable citizens and non-GIS experts to engage with and evaluate complex geospatial data in a very interactive and intuitive way. In this framework, the LLM operates as the "brain," while the functions serve as the "hands". Using the LLM's interpretation capabilities of human language and its querying knowledge with highly structured geospatial data models like CityGML and stored in geospatial databases like 3DCityDB, users can navigate, interact with, and analyze complex geospatial data within seconds.

The prototype and the study results are already usable and encourage further exploration of LLMs in geospatial applications. However, challenges such as uncertainty, hallucination, and cost were also identified. This requires further attention and investigation. In the future, we would like to address these issues by implementing control mechanisms to enhance the user experience and employ open LLMs that run locally on computers to reduce the costs. We also plan to explore and test fine-tuning strategies with local LLMs to interact with 3D models more effectively and intuitively.

Acknowledgements

We thank all our partners in the ASCEND Project that is funded by European Union, especially the city of Munich for their collaboration and for providing the data for this work.

References

Balsebre, P., Huang, W., & Cong, G. (2024). *LAMP: A Language Model on the Map*. http://arxiv.org/abs/2403.09059

Biljecki, F., & Ito, K. (2021). Street view imagery in urban analytics and GIS: A review. *Landscape and Urban Planning*, *215*.

Census GPT. (2023). *caesarHQ/textSQL* [GitHub]. caesarHQ/textSQL. https://github.com/caesarHQ/textSQL

Cozzi, P., Lilley, S., & Getz, G. (2019, January 31). *3D Tiles Specification 1.0*. Open Geospatial Consortium.

Fraunhofer IOSB. (2019). *FROST-Server: An Open-Source SensorThings API Implementation (Version 1.14)*. https://github.com/FraunhoferIOSB/FROST-Server

Gröger, G., Kolbe, T. H., Nagel, C., & Haefele, K.-H. (2012). *OGC City Geography Markup Language (CityGML) Encoding Standard* (pp. 12–019).

HFT Stuttgart. (2015). *SimStadt Documentation*. SimStadt. https://simstadt.hft-stuttgart.de/

Huang, C., Chen, S., Li, Z., Qu, J., Xiao, Y., Liu, J., & Chen, Z. (2024). GeoAgent: To Empower LLMs using Geospatial Tools for Address Standardization. *Findings of the Association for Computational Linguistics ACL 2024*, 6048–6063.

Janowicz, K. (2023). *Philosophical Foundations of GeoAI: Exploring Sustainability, Diversity, and Bias in GeoAI and Spatial Data Science*.

Kang, Y., Zhang, F., Gao, S., Lin, H., & Liu, Y. (2020). A review of urban physical environment sensing using street view imagery in public health studies. *Annals of GIS*, *26*(3), 261–275.

Kutzner, T., Chaturvedi, K., & Kolbe, T. H. (2020). CityGML 3.0: New Functions Open Up New Applications. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, *88*(1), 43–61.

Lewis, P., Perez, E., Piktus, A., Petroni, …, & Kiela, D. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, *33*.

Li, Z., & Ning, H. (2023). Autonomous GIS: The next-generation AI-powered GIS. *International Journal of Digital Earth*, *16*(2), 4668–4686.

Liang, S., Khalafbeigi, T., & Van der Schaaf, H. (2021). *OGC SensorThings API Part 1: Sensing Version 1.1*. https://docs.ogc.org/is/18-088/18-088.html

Mooney, P., Cui, W., Guan, B., & Juhász, L. (2023). Towards Understanding the Geospatial Skills of ChatGPT: Taking a Geographic Information Systems (GIS) Exam. *Proceedings of the 6th ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery*, 85–94.

Pang, H. E., & Biljecki, F. (2022). 3D building reconstruction from single street view images using deep learning. *International Journal of Applied Earth Observation and Geoinformation*, *112*, 102859.

Santhanavanich, T., & Coors, V. (2021). CityThings: An integration of the dynamic sensor data to the 3D city model. *Environment and Planning B: Urban Analytics and City Science*, *48*(3), 417–432.

Singh, S., Fore, M., & Stamoulis, D. (2024). GeoLLM-Engine: A Realistic Environment for Building Geospatial Copilots. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 585–594.

Strong, B. (2023, March 18). ChatGeoPT: Exploring the future of talking to our maps. *Earth Genome*. https://medium.com/earthrisemedia/chatgeopt-exploring-the-future-of-talking-to-our-maps-b1f82903bb05

Willenborg, B., Sindram, M., & Kolbe, T. H. (2018). Applications of 3D city models for a better understanding of the built environment. *Trends in Spatial Analysis and Modelling: Decision-Support and Planning Strategies*, 167–191.

Wu, A. N., & Biljecki, F. (2021). Roofpedia: Automatic mapping of green and solar roofs for an open roofscape registry and evaluation of urban sustainability. *Landscape and Urban Planning*, *214*.

WU, J., Wu, X., & Yang, J. (2024). *Guiding Clinical Reasoning with Large Language Models via Knowledge Seeds*.

Yao, S., Zhao, J., Yu, D., ... & Cao, Y. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*.

Yao, Z., Nagel, C., … & Kolbe, T. H. (2018). 3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. *Open Geospatial Data, Software and Standards*, *3*(1), 5.

Zhang, F., Wegner, J. D., Yang, B., & Liu, Y. (2023). Street-level imagery analytics and applications. *ISPRS Journal of Photogrammetry and Remote Sensing*, *199*, 195–196.

Zhang, Y., Wang, Z., … & Yu, W. (2024). BB-GeoGPT: A framework for learning a large language model for geographic information science. *Information Processing & Management*, *61*(5).

Zhang, Y., Wei, C., He, Z., & Yu, W. (2024). GeoGPT: An assistant for understanding and processing geospatial tasks. *International Journal of Applied Earth Observation and Geoinformation*, *131*.