NANYANG TECHNOLOGICAL UNIVERSITY

SCSE23-0651: HELPING LANGUAGE MODELS PROCESS SPATIAL DATA USING LANGCHAIN

Gareth Thong Jun Hong U2021083G

Project Supervisor: Ast/P Long Cheng

Examiner: Mr Ong Chin Ann

School of Computer Science and Engineering

Academic Year 2023/24

NANYANG TECHNOLOGICAL UNIVERSITY

SCSE23-0651: HELPING LANGUAGE MODELS PROCESS SPATIAL DATA USING LANGCHAIN

Submitted in Partial Fulfillment of the Requirements

for the Degree of Bachelor of Engineering in Computer Science

of Nanyang Technological University

By

Gareth Thong

School of Computer Science and Engineering

Academic Year 2023/24

Abstract

The advent of ChatGPT and other Large Language Models in recent years has caused a surge in popular interest in Artificial Intelligence and its capabilities. Although Large Language Models may seem capable of an endless variety of tasks, there are still areas it struggles in such as the hallucination problem or in its understanding of non-textual data, like geospatial data. This project seeks to address this issue by exploring methodologies for Large Language Models to interpret and use geospatial data more accurately, and to develop an easily operable workflow that uses PostGIS and LangChain, among other technologies. The workflow will be grounded in theoretical concepts like few-shot learning, which will be elaborated upon in this report. An application user interface incorporating this workflow will be built in Flask, and it is hoped that the results presented in this report will be useful for the future development of software wishing to best utilize Large Language Models for processing geospatial data.

Acknowledgments

I would like to extend my deepest gratitude to Assistant Professor Long Cheng for providing his guidance throughout the course of this project. His insights and advice during my many meetings with him were invaluable in spurring me on to overcome obstacles and complete the project.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Background

Chat Generative Pre-trained Transformer (ChatGPT) is a chatbot developed by OpenAI for natural language processing. As a Large Language Model (LLM) trained on a large corpus of text sourced from the Internet [1], ChatGPT has been widely credited with focusing popular attention on Artificial Intelligence (AI) [2], particularly on its most well-known, basic text generation function based on relationships within its training data [3]. ChatGPT has seen rapid development since its inception as well. Its latest model, GPT-4, utilizes training data as recent as December 2023 and is capable of interpreting images [4]. Many competitors to ChatGPT have emerged since, such as Google's Gemini and purpose-built models like GitHub's CoPilot for writing code [5].

However, ChatGPT still faces limitations in areas outside text generation, such as in solving mathematical questions accurately [6]. One key limitation ChatGPT currently faces are difficulties with discerning and understanding spatial relationships between objects, which might arise as a result of its limited understanding of the world [7]. The performance of LLMs like ChatGPT in responding to questions about spatial information leave much to be desired, especially when faced with a wide variety of geometrical relationships [8]. This lack of comprehension extends to Geographic Information System (GIS) data [9], which may cause such limitations to become apparent when tested with GIS data like a pair of (x,y) coordinates, such as in Figure 1:

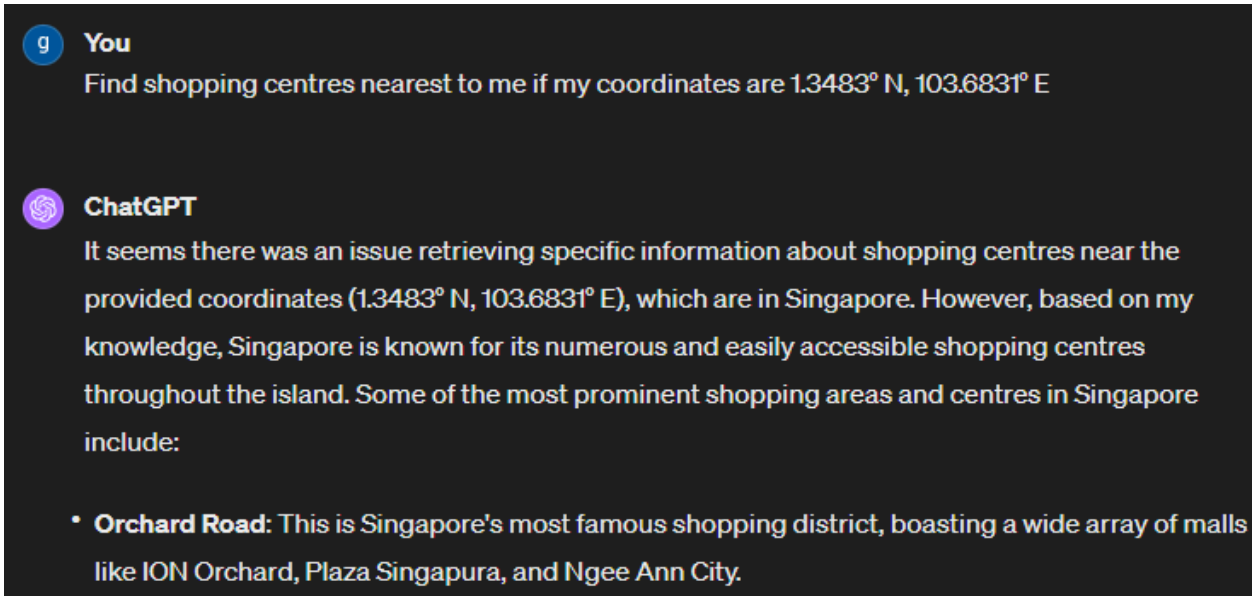

Figure 1: Difficulties in utilizing exact coordinate information

Figure 1 is a screenshot, captured in 2024, of GPT-4 responding to user input providing the coordinates of Nanyang Technological University (NTU). Figure 1 shows that ChatGPT did not utilize the coordinate information well, being unable to respond with specific information about shopping centers nearest to NTU. This may be caused by both a lack of understanding of geographical information, as well as specific spatial datasets for ChatGPT to reference, relying on its generalized training data instead. Given the ubiquity of GIS in a wide range of important applications like urban planning [10], the value of integrating LLMs like ChatGPT with GIS data for enhancing these applications cannot be ignored.

1.2. Objectives

The current objective of this project, therefore, is to develop a workflow for an LLM to accurately interpret natural language input for spatial data processing and retrieval from a spatial database. This workflow would entail accessibility to layman users, who would be able to query geographical information without requiring overly technical knowledge of databases and GIS.

Another goal would be transferability, such that the workflow may accommodate different spatial datasets sourced from diverse geographical locations. Ultimately, it is hoped that this workflow would serve as a framework for future targeted application development for LLMs to understand specific spatial datasets, mitigating the problems explained in the Background of this report.

1.3. Project Scope

This project is focused on enabling meaningful and accurate interaction between an LLM, natural language input, and spatial database as explained previously. A review of the research literature and investigation into possible technologies, like LangChain, PostGIS, and few-shot learning paradigms useful for development will thus be the first step. Subsequently, the focus will be on the integration of these possible technologies and conceptual design behind this interaction workflow, which will be incorporated into an application user interface written in Flask. This report will present a detailed breakdown of these steps, along with demonstrations of the application's functionality through tests on its various features to ensure some form of correctness. The final section of this report will focus on possible improvements to the application beyond the scope of this project.

1.4. Report Organization

Chapter 1	An introduction into the insights and rationale behind this project.
Chapter 2	A review of the existing technologies and associated literature.
Chapter 3	This chapter covers the workflow design and implementation.
Chapter 4	This chapter documents the application testing procedures and results.
Chapter 5	A concluding chapter covering the project challenges faced and future work.

Table 1: Report breakdown

The organization of this report is shown in table 1. This project was carried out under the supervision and mentorship of Assistant Professor Long Cheng.

2. Literature Review

2.1. Initial Investigation

LangChain is a framework for building software with LLM capabilities [11], with one essential LangChain component being its various Python libraries. Initial investigations were carried out with SQLDatabaseChain [12] from the langchain_experimental Python library, a class that allows for building chains of SQL queries constructed by an LLM for database interaction [13] [14]. These investigations were carried out on a sample spatial dataset containing information about New York City downloaded from the “Introduction to PostGIS” workshop, a workshop about the fundamentals of PostGIS [15]. PostGIS is an extension for PostgreSQL that provides support for processing and storing geospatial data. PostGIS [16] essentially enables the usage of a PostgreSQL relational database with spatial data.

During the investigations, SQLDatabaseChain was used together with the OpenAI “gpt-4” model to build spatial data-specific SQL queries by consuming inputs consisting of user questions in natural language. The LLM could also output a result in natural language as shown in Figure 2:

```
> Entering new SQLDatabaseChain chain...
What coffee shop is nearest to the coordinates (37.787162, -122.410725)?
SQLQuery:SELECT coffee_shop FROM locations WHERE ST_Distance(ST_MakePoint(lon, lat), ST_MakePoint(-122.410725, 37.787162)) = (S
ELECT MIN(ST_Distance(ST_MakePoint(lon, lat), ST_MakePoint(-122.410725, 37.787162))) FROM locations);
SQLResult: [('Starbucks',)]
Answer:The nearest coffee shop is Starbucks.
> Finished chain.
press to continue
```

Figure 2: SQLDatabaseChain usage

The response from the LLM providing the name of the nearest coffee shop after constructing its SQL query was promising. It was hypothesized that this was largely due to the technical coordinate information provided in the input to SQLDatabaseChain, as it was discovered that the SQL query generated by the LLM may be inaccurate depending on the amount of

information provided in the user input. This is demonstrated in Figure 3, where technical information such as coordinates was not provided in the natural language question posed by the user. This caused the LLM to respond with a null result, that the ‘Broad St’ station is in no neighborhood.

```
> Entering new SQLDatabaseChain chain...
What neighborhood is the 'Broad St' station in?
SQLQuery:SELECT nghbhd FROM nyc_subway_stations WHERE name = 'Broad St';
SQLResult: [(None,)]
Answer:The 'Broad St' station is in no neighborhood.
> Finished chain.
```

Figure 3: Null result from SQLDatabaseChain

The result in Figure 3 is wrong however, as it would be impossible for a subway station to be situated in no neighborhood at all. This need for technical information in the input is further exemplified in Figure 4, where the input to the SQLDatabaseChain contained extra instructions on constructing the SQL query. After being instructed to use a join on the relevant nyc_neighborhoods and nyc_subway_stations tables, the LLM was able to construct a correct SQL query to determine the different neighborhoods that the ‘Broad St’ station is associated with, which showed that the ‘Broad St’ station was partially contained within a few neighborhoods:

```
> Entering new SQLDatabaseChain chain...
By spatial joining the nyc_neighborhoods and nyc_subway_stations tables and by specifying which table the column names are from
, can you tell me what neighborhood the 'Broad St' station is in?
SQLQuery:SELECT nyc_neighborhoods.name
FROM nyc_neighborhoods
INNER JOIN nyc_subway_stations
ON nyc_neighborhoods.boroname = nyc_subway_stations.borough
WHERE nyc_subway_stations.name = 'Broad St'
LIMIT 5;
SQLResult: [('East Village',), ('West Village',), ('Battery Park',), ('Carnegie Hill',), ('Harlem',)]
Answer:The 'Broad St' station is in the East Village, West Village, Battery Park, Carnegie Hill, and Harlem neighborhoods.
> Finished chain.
```

Figure 4: Improved result from SQLDatabaseChain

The ‘ST_Contains’ function is a function specific to the processing of geospatial data [17]. By being even stricter with the criteria in the input to SQLDatabaseChain and specifying that the

‘ST_Contains’ function was to be used, it can be seen that the geometry of the ‘Broad St’ station is strictly within the ‘Financial District’, as shown in Figure 5:

```
> Entering new SQLDatabaseChain chain...
Using the nyc_neighborhoods and nyc_subway_stations tables and by specifying which table the column names are from, and by usin
g ST_Contains and spatial joins, can you tell me what neighborhood the 'Broad St' station is in?
SQLQuery:SELECT nyc_neighborhoods.name
FROM nyc_neighborhoods
INNER JOIN nyc_subway_stations
ON ST_Contains(nyc_neighborhoods.geom, nyc_subway_stations.geom)
WHERE nyc_subway_stations.name = 'Broad St';
SQLResult: [('Financial District',)]
Answer:The 'Broad St' station is in the Financial District.
> Finished chain.
```

Figure 5: Obtaining a precise result from SQLDatabaseChain

Although the results shown so far might alleviate some of the problems with unreliable spatial data retrieval highlighted in this report’s Introduction, the input in Figure 2, “What coffee shop is nearest to the coordinates (37.787162, -122.418725)?”, consisted of precise coordinate information which might not be reasonably expected from a layman user. Similarly, to obtain a correct and more precise result in Figure 4 and 5 respectively, increasing layers of technical depth in the input to SQLDatabaseChain were required. An interaction workflow built on this trend goes against one of this project’s objectives of accessibility, resulting in the need to explore a more robust process.

2.2. Spatial Data

The Open Geospatial Consortium (OGC) is an organization that sets international standards for handling geospatial data, such as geospatial data encoding standards [18]. An important feature of spatial data stored in a PostGIS-enabled database is the use of the geometry data type, which is an implementation of the OGC Geometry representation for modeling geospatial data [19]. The geometry data type is stored in a separate column in a PostGIS-enabled schema. Values of this data type, geometry values, represent different geometric shapes and provide information about the geometry of each entry in the spatial data schema. For instance, a

geometry value may correspond to a Polygon, in the PostGIS documentation's example as shown in figure 6 [20]:

```
POLYGON ((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
```

Figure 6: Example of a Polygon geometry value

In the representation in figure 6, the first and second elements of the outer tuple correspond to the outer perimeter of the polygon and the inner perimeter of a hole within the polygon. Each 3-digit element of the inner tuple corresponds to the x, y, and z-coordinate of a vertex. The shape of the polygon is determined by joining the vertices together, and different polygons will be represented by vertices with different x, y, and z-coordinates. Besides a Polygon, examples of different geometries that may be represented by the geometry data type include Points, which correspond to a single point in coordinate space, or a LineString, which is formed by joining multiple line segments together. An example of a representation of a Point is depicted in figure 7 taken from the PostGIS documentation [21], where the first and second digits in the tuple represent the x and y-coordinates of a point respectively.

```
POINT (1 2)
```

Figure 7: Example of a Point geometry value

A LineString may be represented as shown in figure 8 taken from the PostGIS documentation [22], where each 2-digit element of the tuple corresponds to the x and y-coordinates of the start or end point of a line segment.

```
LINESTRING (1 2, 3 4, 5 6)
```

Figure 8: Example of a LineString geometry value

Access and operations on geometry values are executed via special functions, of which the full list is listed in the PostGIS documentation [23]. An example of such a function is the ST_X() [24] function, which accepts an argument corresponding to a Point geometry value, and returns the x-coordinate of the point. Similarly, ST_Y() [25] will return the y-coordinate of a point. Other functions may perform more complex operations, like ST_Area() [26], which accepts a geometry value and returns the area of the geometry, or ST_Perimeter() [27], which returns the perimeter. The creation of a robust interaction workflow would therefore incorporate the utilization of these special functions in SQL queries, to enable accurate and efficient retrieval and computation of spatial data from a PostGIS-enabled PostgreSQL database.

2.3. Few Shot Learning

Few-shot learning is a paradigm that involves prompt retrieval from a collection of annotated examples for performance improvement in natural language tasks [28]. This can be achieved by associating relevant annotated in-context examples for specific test inputs [28]. Few-shot prompting could therefore be useful in improving the accuracy of results retrieved by an LLM from a spatial database, even when the natural language prompt contains limited technical or spatial information. The LLM would be able to perform accurate data retrieval even against a complicated relational database structure with unclear schema names [29]. Few-shot prompting could potentially help the LLM better process layman natural language questions requesting for specific geometric information such as “What is the distance between Angel Cafe and Deli?”.

By using geospatial data-specific SQL queries which incorporate the special functions explained in the Spatial Data section in few-shot learning examples, an LLM will be able to reference the most relevant few-shot example to construct syntactically correct geospatial data queries, mitigating the problems encountered during the Initial Investigation section. A few-shot learning example would thus consist of an enhanced SQL query annotated by a natural language geospatial question. The detailed implementation of this concept in the interaction workflow will be explained in the following sections.

3. Design And Implementation

3.1. Spatial Data Pipeline

OpenStreetMap is a project that publishes geographical data for use free-of-charge [30]. To supply the spatial data necessary for a PostGIS-enabled PostgreSQL relational database for application development, the use of a sample OpenStreetMap (OSM) dataset was explored. The OSM data was first downloaded from an endpoint supplied by Geofabrik [31], which is a free download server that hosts OSM data files. These OSM data files are divided by regions and continents. According to Geofabrik, the OSM data stored on their servers is updated daily with the latest data [31].

The downloaded OSM data can be imported into a local database using the osm2pgsql tool [32]. Osm2pgsql allows a significant amount of customisation regarding the type of OSM data to be imported into the database, and automatically creates the database schema during the initial data import. This can be especially useful given the huge diversity of spatial data provided by OSM. The tool was run from the command line using the command in Figure 9:

```
(geoenv) C:\Users\Gareth Thong\FYP\osm2pgsql\osm2pgsql-bin>osm2pgsql -d
osm2pgsqldb -U postgres -W -H localhost -P 5432 -S default.style
osmdata_geofabrik/malaysia-singapore-brunei-latest.osm.pbf
```

Figure 9: Command to run osm2pgsql

In the command, the string “malaysia-singapore-brunei-latest.osm.pbf” states the name of the pbf file to be imported, and contains spatial data of Singapore, Malaysia and Brunei downloaded from Geofabrik.

The “default.style” string refers to the file containing configurations for the conversion of OSM data for use in the database. A snippet from the default.style file is shown in figure 10:

```
node,way   public_transport text     polygon
node,way   railway      text         linear
```

Figure 10: Code snippet from default.style

With reference to the figure 10’s code snippet, node and way correspond to elements of OSM data [33], which are essential concepts used by OSM to model the tangible world. For instance, nodes correspond to unique points on earth with a latitude and longitude, while ways represent an ordered collection of nodes that can be joined to form a line. The last type of element, relations, record the relationship between two or more elements [33]. The tags in the second column from the left provide the real-world context of this relationship, such as a physical railway [33]. The third column from the left indicates the type of column to be created in a particular database schema for storing tag information, such as textual data. Finally, the flag (‘polygon’ or ‘linear’) in the rightmost column specifies the target schema for importing the OSM object into [34]. Using the unedited default.style file ensures that all tags are imported into the database.

The import process when running the osm2pgsql tool is captured in the command line screenshot in figure 11:

```
(geoenv) C:\Users\Gareth Thong\FYP\osm2pgsql\osm2pgsql-bin>osm2pgsql -d osm2pgsqldb -U postgres -W -H localhost -P 5432
-S default.style osmdata_geofabrik/malaysia-singapore-brunei-latest.osm.pbf
osm2pgsql version 1.0.0 (64 bit id space)


!! You are running this on 32bit system, so at most
!! 3GB of RAM can be used. If you encounter unexpected
!! exceptions during import, you should try running in slim
!! mode using parameter -s.
Password:
Allocating memory for sparse node cache
Node-cache: cache=800MB, maxblocks=12800*65536, allocation method=1
Using built-in tag processing pipeline
Using projection SRS 3857 (Spherical Mercator)
Setting up table: planet_osm_point
Setting up table: planet_osm_line
Setting up table: planet_osm_polygon
Setting up table: planet_osm_roads

Reading in file: osmdata_geofabrik/malaysia-singapore-brunei-latest.osm.pbf
Using PBF parser.
Processing: Node(25679k 8559.8k/s) Way(962k 34.36k/s) Relation(0 0.00/s)
(geoenv) C:\Users\Gareth Thong\FYP\osm2pgsql\osm2pgsql-bin>
```

Figure 11: Screenshot of import process using osm2pgsql

With reference to figure 11, the import process sets up 4 target schemas for storing spatial data by default, the planet_osm_point, planet_osm_line, planet_osm_polygon, and planet_osm_roads tables. planet_osm_point stores data with point geometry values derived from the node element, planet_osm_roads and planet_osm_line stores data with line geometry values derived from ways and relations, and planet_osm_polygon stores data with polygon geometry values derived from ways and relations [34].

Upon completion of the import process, the PostGIS-enabled PostgreSQL database can be queried directly to check the results. For example, querying the planet_osm_point schema may yield a table as shown in figure 12:

	sport text	surface text	toll text	tourism text	tower:type text	tunnel text	water text	waterway text	wetland text	width text	wood text	z_order integer	way geometry
444	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…
445	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…
446	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…
447	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…
448	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…
449	[null]	[null]	[null]	attraction	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…
450	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…
451	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…
452	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…
453	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…
454	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	0101000020110…

Figure 12: Result from querying the planet_osm_point schema

In the result table in figure 12, the tags have been imported as individual columns, such as “sport” or “waterway”, while each row corresponds to a geographical feature. Null values are present in most columns for each row as the majority of the tag columns would be irrelevant for describing a particular geographical feature. For instance, in row 449, the text “attraction” is present in the “tourism” column as it denotes the geographical feature as a tourist attraction [35], while the remaining columns would be unsuitable for describing the feature. The “way” column is present in all 4 target schemas as it is an essential column containing the geometry value for each geographical feature. Running the SQL query “SELECT ST_AsText(way) FROM planet_osm_point;” converts the geometry values in the “way” column into a human-readable format [36] and displays them as shown in figure 13:

	st_astext text
1	POINT(11574325.600160467 147797.96458227385)
2	POINT(11563201.176278718 144220.5102294393)
3	POINT(11567040.952870496 144840.5965117653)
4	POINT(11574831.825544849 148589.0585238085)
5	POINT(11560531.412062969 144897.18367040198)
6	POINT(11559744.305339418 144732.3994910612)
7	POINT(11559494.83836055 144807.481560385)
8	POINT(11560630.519805625 145292.6260101525)
9	POINT(11560606.675170695 145256.9277234276)
10	POINT(11556847.14879983 141654.59739481422)
11	POINT(11557019.348920139 141031.1999209907)
12	POINT(11557065.913863136 141635.23416147154)

Figure 13: Human-readable geometry values

With reference to figure 13, the first float value and the second float value in the POINT tuple represent longitude and latitude respectively. For the purposes of further validation, the spatial data can also be loaded into QGIS, an open source Geographic Information System [37], and visualized as shown in figure 14:

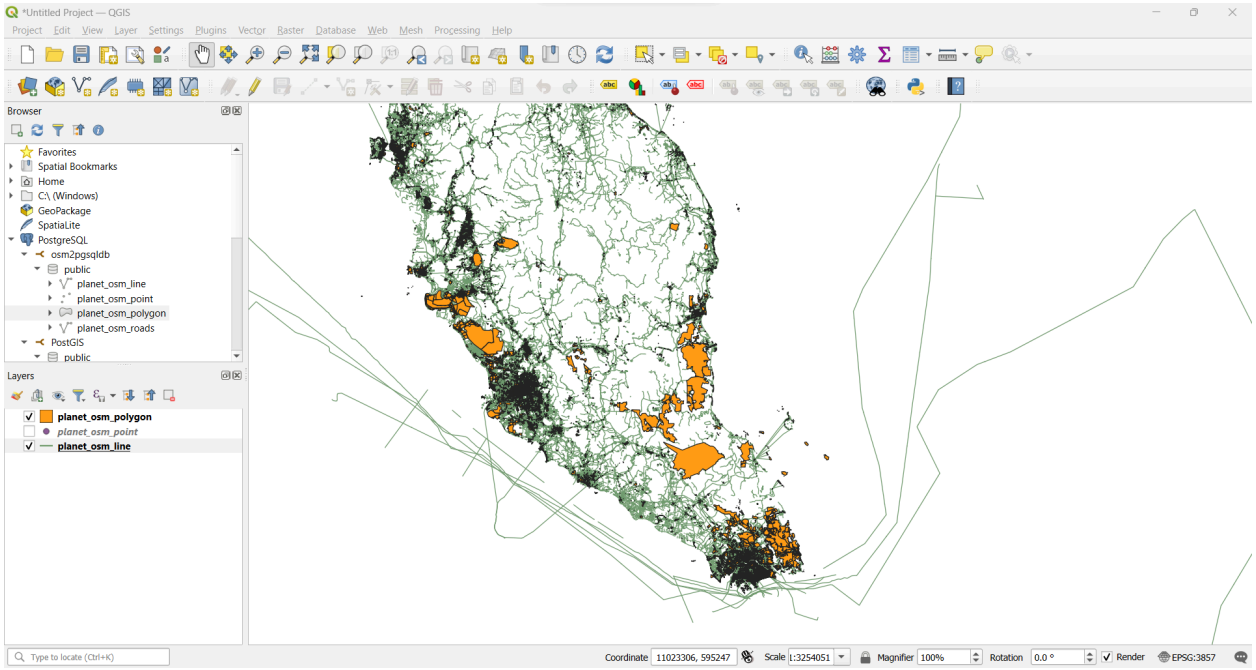

Figure 14: Spatial data visualization in QGIS

Figure 14 depicts a rendering of spatial data from planet_osm_polygon and planet_osm_line. The contours and boundaries of Peninsular Malaysia and Singapore can be seen. Implementation of the SQL Agent can proceed upon confirmation of a successful spatial data import.

3.2. SQL Agent

A LangChain agent is a framework for enabling an LLM to dynamically rationalize an ordered list of steps to take in pursuit of a particular goal [38], which an SQLDatabaseChain would not be capable of. An SQL LangChain agent is more powerful than a SQLDatabaseChain as it would be able to incorporate domain knowledge and provide responses based on specific database schema and data. The development of an SQL agent created specifically for interaction with the imported spatial data of Singapore, Malaysia, and Brunei in a PostgreSQL database will be explored in this section, and would constitute the core functionality of the interaction workflow.

3.2.1. Dependencies and LLM

The various libraries imported for building the core functionality are shown in the table 2:

Library Name	Version
langchain	0.0.353
langchain_community	0.0.10
openai	1.7.0
langchain_openai	0.0.2

Table 2: Libraries used for building core functionality

Functions imported from the langchain and langchain_community libraries help with building the SQL agent, while the openai and langchain_openai libraries assist with making Application Programming Interface (API) calls to the OpenAI “gpt-4” model, which will be used as the LLM for the interaction workflow.

3.2.2. sql_get_similar_examples tool

A key procedure in initializing the SQL agent is through the defining of additional tools that can be placed at the agent's disposal, which would be used as part of the agent's reasoning process. These tools can be appended into a Python list argument, custom_tool_list, which will be expanded upon in later sections.

One important tool that was appended to the custom_tool_list argument was the sql_get_similar_examples tool, which helps the agent perform few-shot learning to understand relationships in spatial data and construct its own SQL queries [39]. A database containing examples of natural language questions annotated by their respective spatial data-specific SQL queries for the agent to reference was thus set up as the first step in creating the sql_get_similar_examples tool. A vital component of this step was ensuring the proper syntactic construction of natural language questions, in order to avoid inconsistent improvements in the SQL agent that may be caused by minor inconsistencies in wording or structure [40]. A list of benchmark geospatial natural language questions retrieved from a study on template questions on geospatial data was thus used as a reference for constructing the natural language questions [41].

This reference list of natural language questions was categorized into different question types based on the type of geospatial relations or quantities that the question was looking for. Table 3 displays these categories and their respective sample questions taken from the list [41]:

Category	Question Type	Reference Question
1	Asking for the attribute of a feature.	Where is Loch Goil located? What is length of River Thames?
2	Asking whether a feature is in a geospatial relation with another feature.	Is Liverpool east of Ireland?

3	Asking for features of a given class that are in a geospatial relation with another feature.	Which counties border county Lincolnshire?” Which hotels in Belfast are at most 2km from George Best Belfast City Airport? Which restaurants are near Big Ben in London? Which rivers cross London in Ontario?
4	Asking for features of a given class that are in a geospatial relation with any features of another class.	Which churches are near castles?
5	Asking for features of a given class that are in a geospatial relation with an unspecified feature of another class which, in turn, is in another geospatial relation with a feature specified explicitly	Which churches are near a castle in Scotland?
6	The questions are similar to the ones in Categories 3 to 5 above, but in addition, the thematic and/or geospatial characteristics of the features that are expected as answers (i.e., the features of the first class mentioned in the question) satisfy some further condition (e.g., numeric).	Which mountains in Scotland have height more than 1000 meters? Which villages in Scotland have a population of less than 500 people? Is there a church in the county of Greater Manchester dedicated to St. Patrick? Which Greek restaurants in London are near Wembley stadium?
7	Questions with quantities and aggregates	Which is the highest mountain in Ireland? Which hotel is the nearest to Old Trafford Stadium in Manchester? Which is the largest lake by area in Great Britain? Which is the largest county of England by population which borders Lincolnshire?

Table 3: Reference list of natural language questions

As these sample questions were specific to a different geographical region, another set of syntactically similar natural language questions, specific to the imported spatial data of Singapore, Malaysia, and Brunei, was constructed instead. These questions, which have been sorted into the same category numbers as in table 3, are listed in table 4:

Question ID	Category	Example Question
1	1	Where is the National Cancer Centre Singapore located?
2	1	What is the length of Bishan Street 13?
3	2	Is the National Cancer Centre Singapore west of Tekka Centre?
4	3	Which restaurants are at most 2km from Ang Mo Kio public library?
5	3	Which temples are next to Bao Gong Temple?
6	4	Which universities are near kindergartens?
7	7	Which is the largest park by area?
8	8	How many buildings are there in 'Tiong Bahru'?

Table 4: Natural language questions constructed for Singapore, Malaysia, and Brunei

To complete the creation of the few-shot examples, the necessary SQL queries for spatial data retrieval were written for each question. The final list of SQL queries for these few-shot examples, and their corresponding Question ID, is presented in table 5. A few-shot example is thus composed of both an SQL query from table 5 and natural language question annotation from table 4 with the same Question ID. For instance, the first few-shot example would consist of the SQL query with Question ID 1 annotated by the example natural language question with the same Question ID 1.

Question ID	SQL Query	Explanation
1	SELECT name, ST_AsText(way) as coordinates FROM planet_osm_polygon AS	This SQL query retrieves the name of an outer geometrical area in which the National Cancer Centre Singapore lies in

	outer_poly WHERE ST_Contains(outer_poly.way, (SELECT way FROM planet_osm_polygon WHERE name = 'National Cancer Centre Singapore' ORDER BY way_area DESC LIMIT 1));	by checking if all points of the geometry of the National Cancer Centre Singapore lie within that outer geometrical area, using data from the planet_osm_polygon schema. The ST_Contains(A, B) [17] function returns True if all points of a geometry value B lie within a geometry value A. ST_AsText(way) [36] returns the geometry values in the way column in a human-readable format.
2	SELECT SUM(ST_Length(way)) AS total_distance FROM (select way from planet_osm_line where name = 'Bishan Street 13') AS linestrings;	The subquery retrieves all LineString [22] geometry values corresponding to Bishan Street 13. Each LineString geometry value thus corresponds to a different segment of this street. ST_Length(way) [42] returns the cartesian length of each LineString geometry value, which is then summed up to obtain the street's length. Note that the schema specified in the subquery is planet_osm_line.
3	SELECT (SELECT ST_X(ST_Centroid(way)) FROM planet_osm_polygon WHERE name = 'National Cancer Centre Singapore' ORDER BY way_area DESC LIMIT 1) < (SELECT ST_X(ST_Centroid(way)) FROM planet_osm_polygon WHERE name = 'Tekka Centre' ORDER BY way_area DESC LIMIT 1) AS isWest;	This query checks if the x-coordinate of the National Cancer Centre Singapore is less than the x-coordinate of Tekka Centre, which would indicate that the National Cancer Centre Singapore is to the west of Tekka Centre. This is done by getting the x-coordinate, using ST_X [24], of the center of mass of National Cancer Centre Singapore's and Tekka Centre's geometry values returned by ST_Centroid(way) [43] and comparing them with the < operator. The schema used is planet_osm_polygon.
4	SELECT poly1.name FROM planet_osm_polygon AS poly1 INNER JOIN planet_osm_polygon AS poly2 ON ST_DWithin(poly1.way, poly2.way, 2000) WHERE poly1.name != 'null' AND poly1.amenity = 'restaurant' AND poly2.name = 'Ang Mo Kio Public Library';	ST_DWithin(poly1.way, poly2.way, 2000) [44] is used to check if the geometry values of a restaurant and Ang Mo Kio Public Library are within 2000 units of distance (2km) from each other. If this condition is True, the query returns the name of the restaurant. The final result is a list of restaurants fulfilling this condition.

5	SELECT poly1.name FROM planet_osm_polygon AS poly1 INNER JOIN planet_osm_polygon AS poly2 ON ST_Touches(poly1.way, poly2.way) WHERE poly1.name != 'null' AND poly2.name = 'Bao Gong Temple';	ST_Touches(poly1.way, poly2.way) [45] retrieves temples with geometry values that have at least a common point with the geometry value of Bao Gong Temple, such that these common points lie only on the boundaries of these geometries. The names of these temples are then returned. An INNER JOIN between two planet_osm_polygon schemas is used here.
6	SELECT poly1.name FROM planet_osm_polygon AS poly1 INNER JOIN planet_osm_polygon AS poly2 ON ST_DWithin(poly1.way, poly2.way, 1000) WHERE poly1.name != 'null' AND “poly1.amenity = 'university' AND poly2.amenity = 'kindergarten';	The function of ST_DWithin(poly1.way, poly2.way, 1000) [44] here is similar to that in the SQL query with question ID 4, but here, the word “near” from the respective annotated natural language question was defined by this report’s author as 1000 distance units (1km). The query then returns the names of universities near kindergartens using this definition. An INNER JOIN between two planet_osm_polygon schemas is used here.
7	SELECT name FROM planet_osm_polygon WHERE name != 'null' AND leisure = 'park' ORDER BY ST_Area(way) DESC LIMIT 1;	To retrieve the largest park by area, the query uses ST_Area(way) [26] to calculate a two-dimensional cartesian area of each park, before sorting them with the largest area at the top. LIMIT 1 ensures that only the name of the park sorted to the top is returned. The schema used is planet_osm_polygon.
8	SELECT building, name, way FROM planet_osm_polygon WHERE building != 'null' AND name LIKE '%Tiong Bahru%';	This query returns the names of buildings which contain the word 'Tiong Bahru'. The schema used is planet_osm_polygon.

Table 5: SQL queries for few-shot examples

Finally, these few shot examples were inserted into the “fewshotexamples” table in the “fewshots” database. Future few shot examples can then also be inserted into the “fewshotexamples” table.

During initialization of the sql_get_similar_examples tool, the data in the “fewshotexamples” table can be retrieved and used to create a Facebook AI Similarity Search (FAISS) vector database to facilitate efficient retrieval of few-shot examples [46] [47], as shown in figure 15.

```
connection = dbsearch_module.create_connection("fewshots")
few_shots = dbsearch_module.select_query("SELECT * from fewshotexamples;", connection)

few_shot_docs = [
    Document(page_content=question, metadata={"sql_query": few_shots[question]})
    for question in few_shots.keys()
]
embeddings = OpenAIEmbeddings(openai_api_key="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")
vector_db = FAISS.from_documents(few_shot_docs, embeddings)
```

Figure 15: FAISS vector database for few-shot examples

This vector database would be converted into a Retriever object [48] and passed into the create_retriever_tool() constructor [49]. The sql_get_similar_examples tool returned from the create_retriever_tool() constructor would then be appended to the custom_tool_list argument, as shown in figure 16.

```
retriever = vector_db.as_retriever()

tool_description = """
This tool will help you understand similar examples to adapt them to the user question.
If possible, invoke the database query as-is directly from the similar examples.
Input to this tool should be the user question.
"""

retriever_tool = create_retriever_tool(
    retriever, name="sql_get_similar_examples", description=tool_description
)
custom_tool_list = [retriever_tool]
return custom_tool_list
```

Figure 16: Appending the sql_get_similar_examples tool to custom_tool_list

During operation, the agent would thus be responsible for using the sql_get_similar_examples tool to retrieve the most relevant few shot example to construct a syntactically correct geospatial SQL query.

3.2.3. search_proper_nouns tool

User-specified natural language questions may contain misspelled proper nouns like names and addresses, which may potentially cause the performance of the agent to deprove. Another important tool to be appended into the custom_tool_list argument, for increasing the robustness of the agent, was thus the search_proper_nouns tool, which enables the agent to verify the spelling of proper nouns and correct them if necessary.

A “propernouns” table containing correctly spelled proper nouns was thus set up for the agent to reference as part of the creation of the search_proper_nouns tool. As the manual insertion of all proper nouns found in the imported spatial data would be extremely tedious, a limited number of proper nouns was therefore selected by the author of this report for insertion into the “propernouns” table instead. No other heuristic for proper noun insertion was used. The list of proper nouns that was inserted into the “propernouns” table is shown in table 6:

Proper Noun ID	Proper Noun
1	Faber Point
2	Tekka Centre
3	Keppel Distripark
4	Parit Setongkat
5	Ang Mo Kio Public Library
6	Regent Secondary School
7	Bao Gong Temple
8	Katong Park

Table 6: List of proper nouns used

Future proper nouns can also be inserted into the “propernouns” table, like the “fewshotexamples” table. Similar to the sql_get_similar_examples tool, the data in the

“propernouns” table can also be retrieved to create a FAISS vector database to facilitate efficient retrieval of proper nouns [46] [47], as shown in figure 17.

```
propernounlist = dbsearch_module.select_query_list("SELECT * from propernouns;", connection)
vector_db = FAISS.from_texts(propernounlist, OpenAIEmbeddings(openai_api_key="sk-Ivbj17kOHhD
```

Figure 17: FAISS vector database for proper nouns

This vector database would be also converted into a Retriever object [48] and passed into the create_retriever_tool() constructor [49]. The search_proper_nouns tool returned from the create_retriever_tool() constructor is shown in figure 18.

```
retriever = vector_db.as_retriever(search_kwargs={"k": 5})
description = """Use to look up values to filter on. Input is an approximate spelling of the proper noun, output is \
valid proper nouns. Use the noun most similar to the search."""
retriever_tool = create_retriever_tool(
retriever,
name="search_proper_nouns",
description=description,
)
return retriever_tool
```

Figure 18: Returning the search_proper_nouns tool

The search_proper_nouns tool could then be appended to the custom_tool_list argument. During operation, the agent would thus be responsible for using the search_proper_nouns tool to retrieve the most relevant proper noun to verify proper noun spellings in user input.

3.2.4. SQL Agent Creation

The custom_tool_list argument could then be passed into the extra_tools field in the create_sql_agent() [50] constructor imported from the langchain library, after appending the necessary tools into the custom_tool_list, as shown in figure 19:

```
agent = create_sql_agent(
llm=llm,
toolkit=toolkit,
verbose=True,
prefix=prefix_template,
agent_type=AgentType.OPENAI_FUNCTIONS,
extra_tools=custom_tool_list,
suffix=custom_suffix,
)

return agent
```

Figure 19: Passing custom_tool_list into create_sql_agent()

In figure 19, the llm and agent_type arguments were used for specifying that the LLM that the agent would be interacting with would be a OpenAI “gpt-4” model. Setting the verbose argument to True ensured that the agent would explain its actions in its output when run. The prefix and suffix arguments contain text strings which serve as instructions to the agent. The gist of these instructions consisted of instructing the agent to use the search_proper_nouns tool before the sql_get_similar_examples tool, and to only perform data retrieval by not generating any SQL queries containing keywords like INSERT and UPDATE.

A connection to the database with the planet_osm_point, planet_osm_line, planet_osm_polygon, and planet_osm_roads tables, osm2pgsqldb, was provided when constructing the toolkit argument in order to grant the agent access to the spatial data of Singapore, Malaysia, and Brunei. This is shown in figure 20:

```
db = SQLDatabase.from_uri("postgresql://postgres:postsuperzax@localhost:5432/osm2pgsqldb")
llm = ChatOpenAI(temperature=0, openai_api_key="sk-Ivbj17kOHhD14Jo2ttXKT3BlbkFJkWj2BvdVX9S
toolkit = SQLDatabaseToolkit(db=db, llm=llm)
```

Figure 20: Establishing a connection to the osm2pgsqldb database

The agent object returned in figure 19 can thus be used for processing natural language geospatial questions, using the reasoning tools and the spatial data from osm2pgsqldb. The operation of the agent will be explored in the next section.

3.3. Application User Interface

The next step of the implementation process was the design of a simple application User Interface (UI) to allow for easy operation of the LangChain SQL agent. An application UI consisting of 3 webpage endpoints, langchainmodel/input, langchainmodel/dataimportview, and langchainmodel/fewshotsview, was built using Flask [51] for this purpose. The web page-based application UI can be run from the command line.

The webpage located at langchainmodel/input, shown in figure 21, serves as an interface for a user to input natural language questions.

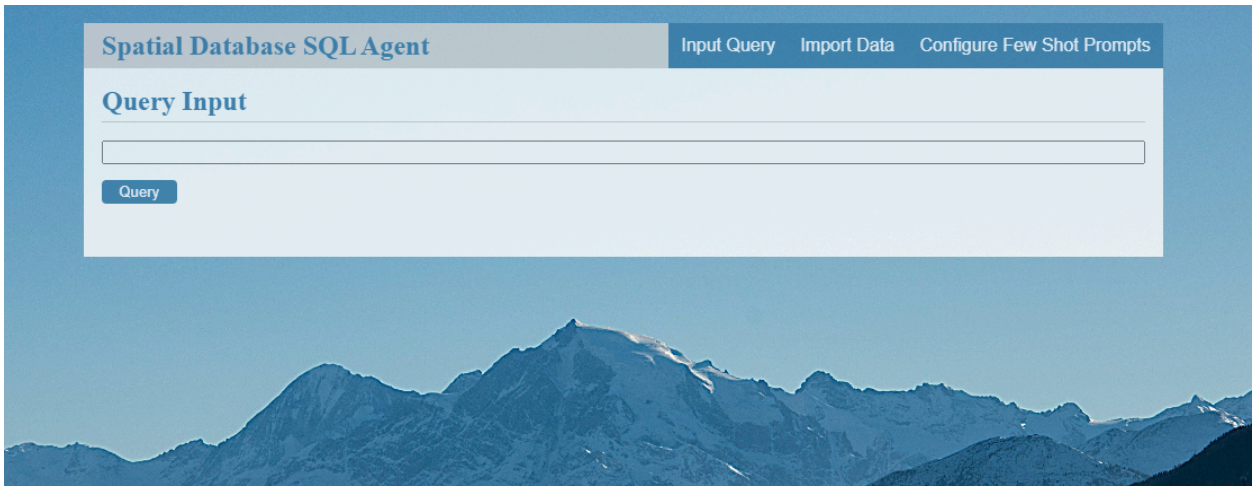

Figure 21: Webpage located at langchainmodel/input

The question string input into the “Query Input” field in figure 21 would be processed by the agent using the command shown in figure 22, after creating the agent object:

```
response = agent_executor.run(query)
```

Figure 22: Command for the agent object to process the question string input

Responses returned from the agent would then be posted to and displayed on the webpage.

By clicking on the “Import Data” button in the navigation bar at the top right of the webpage, the user could be brought to the webpage located at langchainmodel/dataimportview, shown in figure 23:

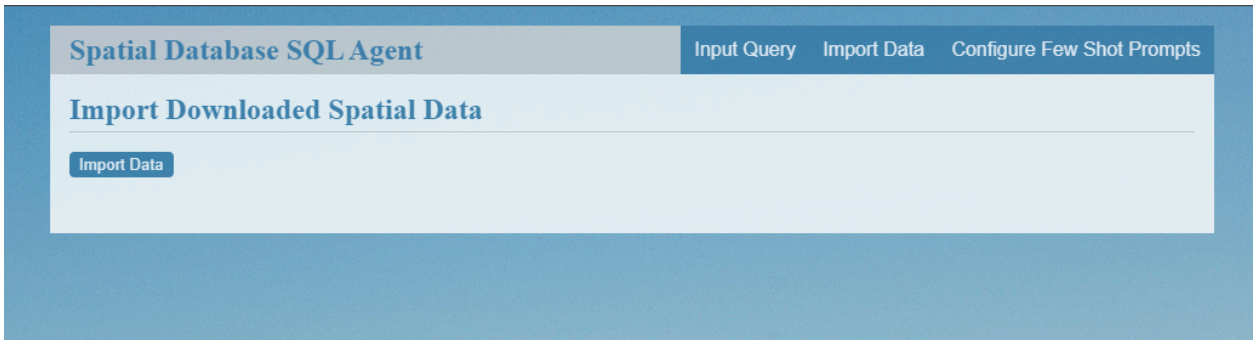

Figure 23: Webpage located at langchainmodel/dataimportview

Clicking on the “Import Data” button below “Import Downloaded Spatial Data” would allow a user to import downloaded spatial data into a database, as explained in the Spatial Data Pipeline section. By replacing the “malaysia-singapore-brunei-latest.osm.pbf” string in the command in figure 9 with the name of another file containing other regions’ spatial data, the “Import Data” button can be used to import spatial data from different geographical locations.

Clicking on the “Configure Few Shot Prompts” button in the navigation bar would bring a user to the webpage at langchainmodel/fewshotsview, an interface allowing a user to insert few-shot examples and proper nouns into the database by clicking on the “Insert” button, as shown in figure 24:

Figure 24: Webpage located at langchainmodel/fewshotsview

An SQL query could be input into the “Few Shot Example Query” field, while the natural language question for the SQL query to be annotated by could be input into the “Few Shot Example Question” field. Proper nouns could be input into the “Proper Noun” field. The user would also be able to view the few-shot natural language questions and proper nouns currently populated in the database. For instance, clicking on the “Display Proper Nouns” button would fetch all the proper nouns stored in the “propernouns” table in the database and post them to the webpage to be displayed as shown in figure 25:

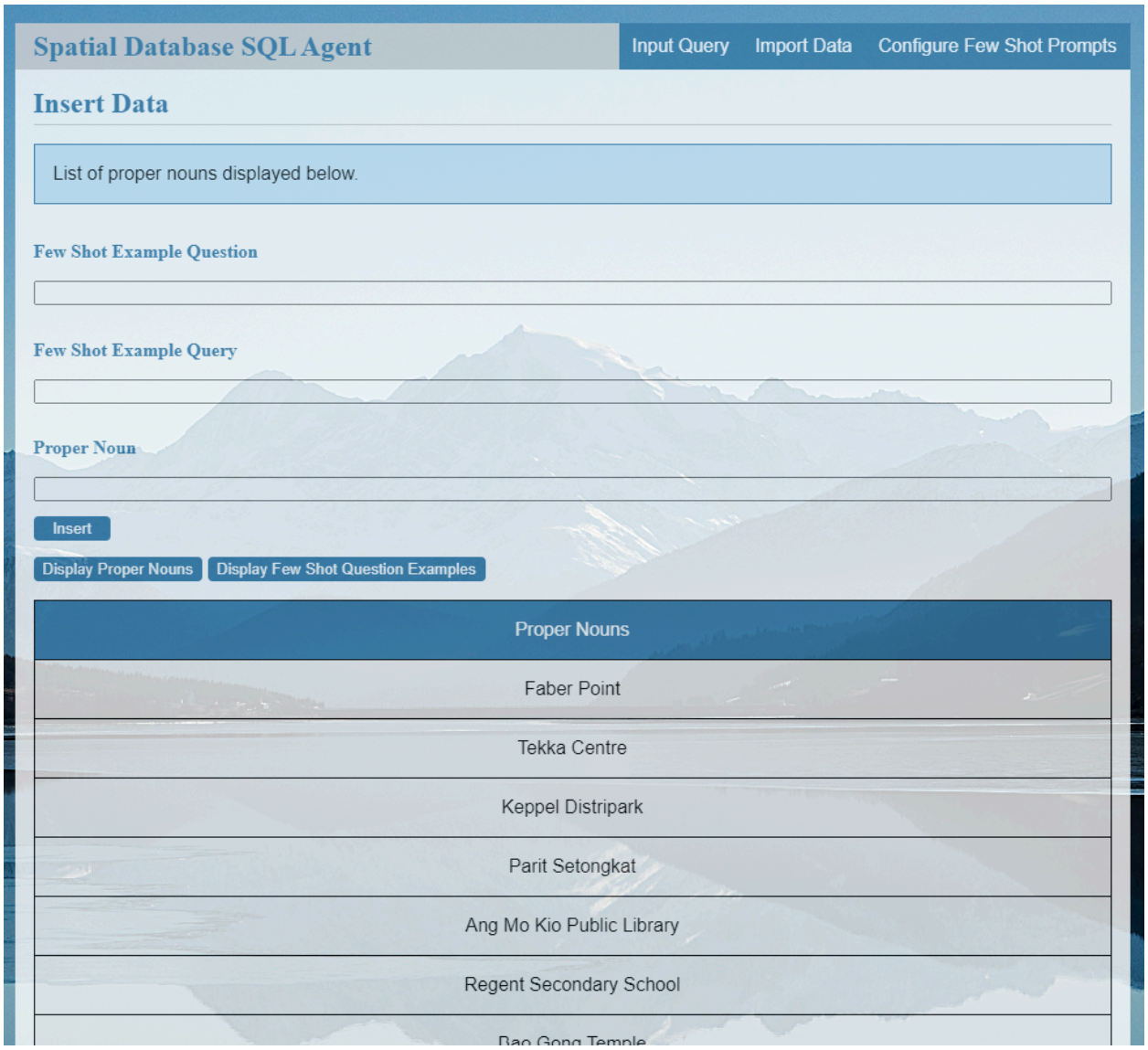

Figure 25: Displaying all stored proper nouns

To perform data insertion and retrieval between the database and the webpage, a separate module containing Python functions built for this purpose was written. For instance, the function in the figure 26, select_query_list, would accept a SQL query and database connection as an argument, and return the results returned from the query as a list:

```
def select_query_list(sql, connection):
    cursor = connection.cursor()
    cursor.execute(sql)
    results = cursor.fetchall()
    resultlist = [result[0] for result in results]
    return resultlist
```

Figure 26: select_query_list function

Finally, the initial setup of the webpage skeleton and styling was achieved by referencing various online guides on CSS [52] and Flask [51]. The background image was sourced from an image sharing site [53].

4. Application Testing

Testing of the application took place after implementation of the spatial data pipeline, SQL agent, and application UI. The test case IDs and the test string inputs are shown in table 7. For each test case, the test input was entered into the “Query Input” field at langchainmodel/input and queried against the agent. The data stored in the “fewshotexamples” and “propernouns” database tables remain unchanged from the Detailed Implementation section. The results captured on the langchainmodel/input webpage, as well as the reasoning output by the agent in the command terminal, are displayed as well.

Test Case ID	Test Input
1	Where is Tekka Centre located?
2	How long is Parit Setongkat?
3	Is Keppel Distripark on the east of Faber Point?
4	Are there any banks near offices?
5	Which is the smallest army airfield by area?
6	Which buildings are beside Katng Par?
7	Which clinics are at most 9km from Rgnt Secodary Scool?

Table 7: Test case IDs and respective test inputs

The test cases will be explained in the sections below. The correctness of the test case results were verified manually by querying the PostgreSQL database.

4.1. Test Case 1

After invoking the search_proper_nouns to verify the correct spelling of “Tekka Centre” as shown in figure 27, the sql_get_similar_examples tool was invoked to retrieve the most relevant few-shot example.

```
Invoking: `search_proper_nouns` with `{'query': 'Tekka Centre'}`


[Document(page_content='Tekka Centre'), Document(page_content='Katong Park'), Document(page_content='Keppel
Distripark'), Document(page_content='Bao Gong Temple'), Document(page_content='Ang Mo Kio Public Library')]
Invoking: `sql_get_similar_examples` with `{'query': 'Where is Tekka Centre located?'}`
```

Figure 27: Invocation of search_proper_nouns in test case 1

Figure 28 shows that the SQL agent has chosen few-shot example with Question ID 1 constructed from tables 4 and 5 as the most relevant few-shot example. For the sake of brevity, the explanations for the remaining test cases will denote a specific few shot example constructed from tables 4 and 5 solely by its Question ID (QID). For instance, a mention of "QID4" would refer to the few-shot example with Question ID 4 constructed from tables 4 and 5.

The reasoning process of the SQL agent is shown as well in figure 28. In the original SQL query, the original WHERE clause was "WHERE name = 'National Cancer Centre Singapore'". 'National Cancer Centre Singapore' was adapted to fit the user's question and changed to 'Tekka Centre' in the SQL query generated by the agent in figure 28, before the query was passed to the database.

```
responded: Based on the similar examples, I can adapt the SQL query from the example "Where is the National
Cancer Centre Singapore located?" to fit the user question "Where is Tekka Centre located?". The adapted SQL
 query would be:

```sql
SELECT name, ST_AsText(way) as coordinates
FROM planet_osm_polygon AS outer_poly
WHERE ST_Contains(outer_poly.way, (
    SELECT way
    FROM planet_osm_polygon
    WHERE name = 'Tekka Centre'
    ORDER BY way_area DESC
    LIMIT 1
));
```
```

Figure 28: SQL agent reasoning process in test case 1

The final reply from the agent is shown in figure 29. The reply shows that the agent has successfully identified the geographical area in which 'Tekka Centre' is located, which in this case is 'Little India'.

Figure 29: Final result in test case 1

4.2. Test Case 2

The search_proper_nouns tool was invoked again to verify the spelling of ‘Parit Setongkat’ in a similar fashion to test case 1, as shown in figure 30.

```
Invoking: `search_proper_nouns` with `{'query': 'Parit Setongkat'}`


[Document(page_content='Parit Setongkat'), Document(page_content='Katong Park'), Document(page_content='Kepp
el Distripark'), Document(page_content='Tekka Centre'), Document(page_content='Bao Gong Temple')]
Invoking: `sql_get_similar_examples` with `{'query': 'How long is Parit Setongkat?'}`
```

Figure 30: Invocation of search_proper_nouns in test case 2

Figure 31 shows the invocation of the sql_get_similar_examples tool by the agent to retrieve and adapt the most relevant few-shot example, QID2. Besides recognizing that the ST_Length() function should be used on the ‘way’ column to calculate length in the agent-generated SQL query, the agent has also adapted the WHERE clause, “WHERE name = ‘Bishan Street 13’”, to reflect the name of the road being queried, ‘Parit Setongkat’. The new WHERE clause is “WHERE name = ‘Parit Setongkat’”.

```
responded: Based on the similar examples, I can adapt the SQL query to fit the user's question. The adapted
SQL query would be:

```sql
SELECT SUM(ST_Length(way)) AS total_distance
FROM (
    select way from planet_osm_line where name = 'Parit Setongkat'
) AS linestrings;
```
```

Figure 31: SQL agent reasoning process in test case 2

Figure 32 shows the final reply from the agent, which has successfully calculated the length of ‘Parit Setongkat’ to be 1902.74 units.

Figure 32: Final result in test case 2

As can be seen from the above test cases, the agent selects the SQL query in the few-shot examples annotated by the natural language question most semantically similar to the user question for adaptation. To further test the versatility of the agent, the test cases below introduce slight changes in semantics in user input, from their closest few-shot example natural language question.

4.3. Test Case 3

In this test case, the technical difference introduced was the use of “east” in the test input as opposed to “west” in the natural language question in QID3. Like the previous two test cases, the test input location names were also changed from “National Cancer Centre Singapore” and “Tekka Centre” to “Keppel Distripark” and “Faber Point”. The agent invoked the sql_get_similar_examples tool as shown in figure 33 and recognized the most relevant few-shot example to reference, QID3, after verifying the spelling of the proper nouns.

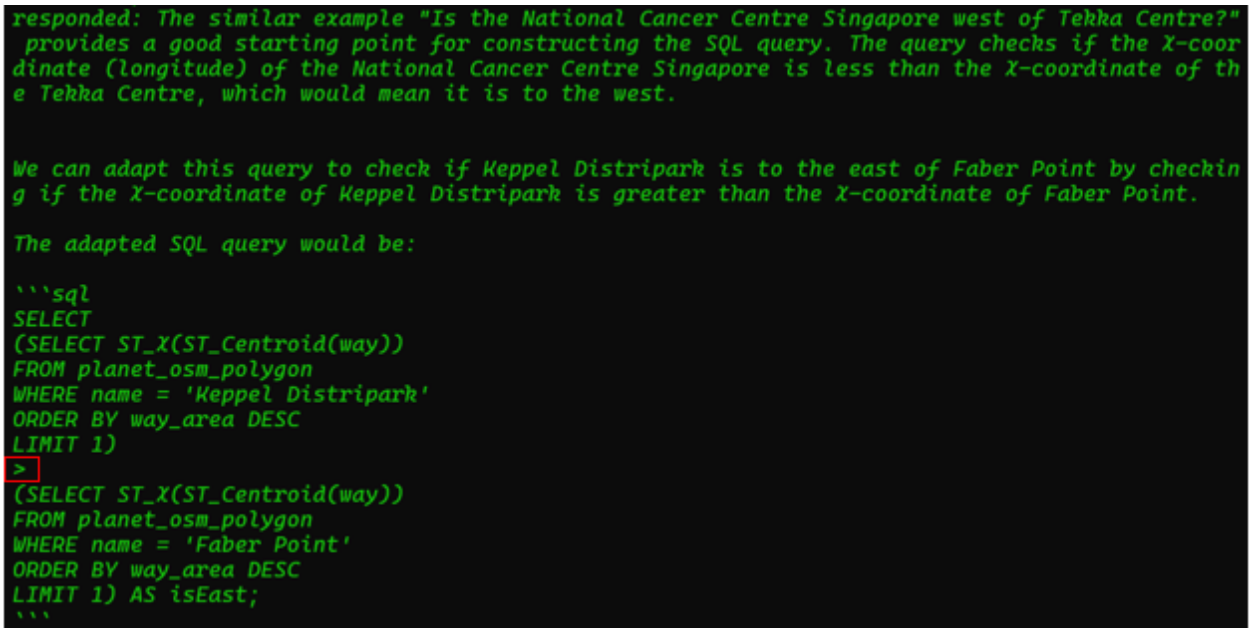

Figure 33: SQL agent reasoning process in test case 3

Originally, in QID3, the SQL query was written to check if the x-coordinate of the National Cancer Centre Singapore was less than the x-coordinate of Tekka Centre. As the test input was checking if Keppel Distripark was to the east, rather than the west, of Faber Point, the agent replaced the “<” operator in the original SQL query with “>” in its generated SQL query, to check if the x-coordinate of Keppel Distripark was greater than Faber Point instead. This is shown by the red box in figure 33. As a result, the agent was able to give the correct reply, as shown in figure 34 below.

Figure 34: Final result in test case 3

4.4. Test Case 4

This test input was structured to have a slightly different meaning from the natural language question in QID6, “Which universities are near kindergartens?”. Instead of asking for names of universities near kindergartens, this test input inquires about the existence of banks near offices in general.

It is worth noting the lack of proper nouns in this test input, causing the agent to invoke the search_proper_nouns tool against the nouns “banks” and “offices” which were not present in the “propernouns” table as shown in figure 35. Nevertheless, the agent continued with its reasoning and invoked the sql_get_similar_examples tools against the original test input, “Are there any banks near offices?”, without changing anything.

```
Invoking: `search_proper_nouns` with `{'query': 'banks'}`


[Document(page_content='Katong Park'), Document(page_content='Bao Gong Temple'), Document(page
_content='Parit Setongkat'), Document(page_content='Regent Secondary School'), Document(page_c
ontent='Ang Mo Kio Public Library')]
Invoking: `search_proper_nouns` with `{'query': 'offices'}`


[Document(page_content='Faber Point'), Document(page_content='Tekka Centre'), Document(page_co
ntent='Parit Setongkat'), Document(page_content='Keppel Distripark'), Document(page_content='A
ng Mo Kio Public Library')]
Invoking: `sql_get_similar_examples` with `{'query': 'Are there any banks near offices?'}`
```

Figure 35: Invocation of search_proper_nouns in test case 4

From the red box in figure 36, it can be seen that the agent was still able to adapt the SQL query in QID6 correctly, changing the conditions to check on for poly1.amenity and poly2.amenity from “university” and “kindergarten” to “bank” and “office” respectively.

```
responded: Based on the similar examples, I can adapt the SQL query to fit the user question.
The adapted SQL query would be:

    SELECT poly1.name
    FROM planet_osm_polygon AS poly1
    INNER JOIN planet_osm_polygon AS poly2
    ON ST_DWithin(poly1.way, poly2.way, 1000)
    WHERE poly1.name != 'null'
    AND poly1.amenity = 'bank'
    AND poly2.amenity = 'office';
```

Figure 36: SQL agent reasoning process in test case 4

As a result, the agent answered the test input not only in terms of the existence of banks near offices, but also by giving the name of the bank, as shown in figure 37.

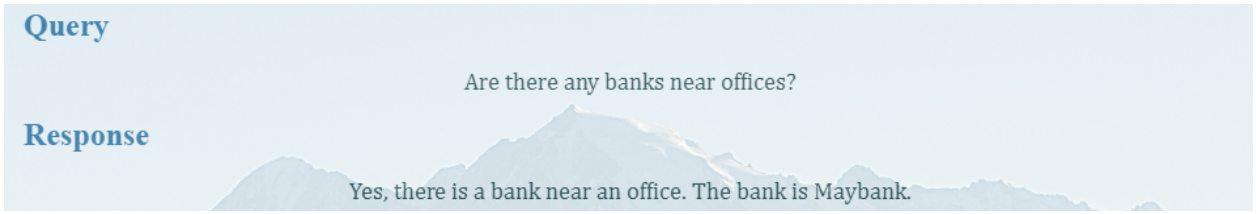

Figure 37: Final result in test case 4

4.5. Test Case 5

The test input in test case 5 asks for the SQL agent to identify the smallest army airfield by area, as opposed to the largest park by area in the natural language question in QID7. Similarly to test case 4, the agent checks the general noun “army airfield” against the search_proper_nouns tool but was able to continue with its reasoning despite not finding “army airfield” in the “propernouns” table. This is shown in figure 38.

```
Invoking: `search_proper_nouns` with `{'query': 'army airfield'}`


[Document(page_content='Katong Park'), Document(page_content='Keppel Distripark'), Document(pa
ge_content='Faber Point'), Document(page_content='Bao Gong Temple'), Document(page_content='Te
kka Centre')]
Invoking: `sql_get_similar_examples` with `{'query': 'smallest army airfield by area'}`
```

Figure 38: Invocation of search_proper_nouns in test case 5

Figure 39 depicts the adaptations made by the agent to the SQL query in QID7 in generating its own input-specific SQL query. In the first red box, “AND leisure = ‘park’” is changed to “AND military = ‘airfield’” to reflect the type of land use the test input is asking for. In the second red box, the agent has changed “DESC LIMIT 1” to “ASC LIMIT 1”, in order to sort and retrieve the airfield with the smallest geographical area, rather than the park with the largest geographical area.

```
responded: Based on the similar examples, I can adapt the SQL query from the first example to
fit the user's question. The adapted query would be:

```sql
SELECT name
FROM planet_osm_polygon
WHERE name != 'null'
AND military = 'airfield'
ORDER BY ST_Area(way)
ASC LIMIT 1;
```
```

Figure 39: SQL agent reasoning process in test case 5

The agent is hence able to provide the name of the smallest army airfield stored in the database, Changi Air Base (West), as shown in figure 40.

Figure 40: Final result in test case 5

4.6. Test Case 6

This test case introduces a test input containing misspellings in the proper noun, Katong Park. Other than that, it is semantically similar to the natural language question in QID5. As can be seen from figure 41, the agent invoked the search_proper_nouns tool against the misspelled proper noun “Katng Par” and corrected it to the proper spelling, “Katong Park”, retrieved from the “propernouns” table.

```
Invoking: `search_proper_nouns` with `{'query': 'Katng Par'}`


[Document(page_content='Katong Park'), Document(page_content='Parit Setongkat'), Document(page
_content='Keppel Distripark'), Document(page_content='Ang Mo Kio Public Library'), Document(pa
ge_content='Tekka Centre')]
Invoking: `sql_get_similar_examples` with `{'query': 'Which buildings are beside Katong Park?'
```

Figure 41: Invocation of search_proper_nouns in test case 6

The agent was hence able to invoke the sql_get_similar_examples tool to generate an SQL query filtering on the correctly spelled “Katong Park” proper noun as shown in figure 42. This enabled the agent’s SQL query to return the correct result as shown in figure 43.

```
responded: Based on the similar examples, I can adapt the SQL query to fit the user question.
The query should select the names of the buildings from the planet_osm_polygon table where the
 buildings touch the way of 'Katong Park'. Let's construct and check this query.

SELECT poly1.name FROM planet_osm_polygon AS poly1 INNER JOIN planet_osm_polygon AS poly2 ON S
T_Touches(poly1.way, poly2.way) WHERE poly1.name IS NOT NULL AND poly2.name = 'Katong Park';
Invoking: `sql_db_query` with `SELECT poly1.name FROM planet_osm_polygon AS poly1 INNER JOIN p
lanet_osm_polygon AS poly2 ON ST_Touches(poly1.way, poly2.way) WHERE poly1.name IS NOT NULL AN
D poly2.name = 'Katong Park';`
```

Figure 42: SQL agent reasoning process in test case 6

Figure 43: Final result in test case 6

4.7. Test Case 7

Test case 7 was designed to test both the search_proper_nouns and sql_get_similar_examples tools, by introducing a misspelling in a proper noun and a slight technical difference from the natural language question in QID4. Figure 44 shows the successful invocation of the search_proper_nouns tool correcting the misspelled “Rgnt Secodary Scool” to “Regent Secondary School”.

```
Invoking: `search_proper_nouns` with `{'query': 'Rgnt Secodary Scool'}`


[Document(page_content='Regent Secondary School'), Document(page_content='Parit Setongkat'), D
ocument(page_content='Ang Mo Kio Public Library'), Document(page_content='Keppel Distripark'),
 Document(page_content='Katong Park')]
```

Figure 44: Invocation of search_proper_nouns in test case 7

In figure 45, it can be seen that the agent was subsequently still able to reason and generate a correct SQL query, by changing the value “2000” found in the original SQL query in QID4, to “9000” as shown in the red box. This reflected the distance of 9km asked by the test input, as opposed to the original distance of 2km. The correct names of the clinics are hence returned by the agent in figure 46.

```
responded: The similar examples provided a SQL query that can be adapted to answer the user's
question. The query finds amenities that are within a certain distance from another amenity. I
n this case, we need to find clinics that are within 9km from Regent Secondary School.

The adapted SQL query would be:

```sql
SELECT poly1.name
FROM planet_osm_polygon AS poly1
INNER JOIN planet_osm_polygon AS poly2
ON ST_DWithin(poly1.way, poly2.way, 9000)
WHERE poly1.name != 'null'
AND poly1.amenity = 'clinic'
AND poly2.name = 'Regent Secondary School';
```
```

Figure 45: SQL agent reasoning process in test case 7

Figure 46: Final result in test case 7

The successful test results in all the above test cases demonstrate the versatility of the interaction workflow built upon the SQL agent in handling misspelled proper nouns, as well as semantically and technically diverse input from a layman user.

5. Conclusion and Future Improvements

5.1. Challenges and Limitations

5.1.1. Project Aim Rework

The original aim of this project was to develop a spatial search plugin for ChatGPT that would help it better understand spatial information and incorporate it accurately into its responses, in line with the original project title, “ChatGPT Plugin Development - Spatial Search Plugin”. To achieve this, it would have been necessary to gain ChatGPT plugin developer access. Although the author of this report requested for developer access from OpenAI and joined the request waitlist [54], access still had not been granted in 2024, leading to a rework of the project aim. This report’s author notes that the report title was changed accordingly to “Helping Language Models Process Spatial Data Using LangChain”, to reflect the current objective.

5.1.2. LangChain and OpenAI

LangChain is a fast-evolving framework, and piecing together the correct versions of the different dependencies and libraries to work with was not trivial. This was further compounded by the fact that some of the LangChain libraries were still experimental and unstable. The documentation for LangChain also updates regularly together with the framework, and LangChain documentation sources provided in this report may end up pointing to different content in the future. Additionally, as the project also relies on external OpenAI API calls, there exists a possibility of the project code breaking in the event of deprecations to the OpenAI GPT models, such as a deprecation that took place recently on 1 January 2024 [55]. Finally, as this project’s OpenAI API calls were funded by the author of this report, scaling this project for use

on a larger scale would require more funding owing to the greater volume of API calls made by multiple users.

5.2. Future Improvements

5.2.1. Other Geospatial Datasets

The OSM dataset used for this project consisted of geospatial data from Singapore, Malaysia, and Brunei. In line with this project's goal of transferability, different agents may be customized for use with different geospatial data by downloading data from other geographical regions via Geofabrik [31], and importing the data using the webpage at langchainmodel/dataimportview. New few-shot examples and proper nouns may be added using the webpage at langchainmodel/fewshotsview, for use with the different geospatial data.

5.2.2. More Few-shot Examples

In the original study featuring template questions on geospatial data [41], the size of the entire benchmark question dataset used was 201. Increasing the number of few-shot examples to a comparable number would thus further improve the robustness of the SQL Agent and the interaction workflow built upon it. This would also necessitate using a larger number of test cases than the 7 detailed in this report, in order to have a more complete assurance of the SQL agent's ability to utilize a larger volume of few-shot examples.

5.3. Conclusion

This project explored the creation of an interaction workflow for an LLM to better interpret and process spatial data. The creation of an application UI for hiding the technical details of this workflow helped to achieve some form of accessibility to layman users, while the workflow's

flexibility with other geographical datasets as explained above help to achieve transferability. The development of the workflow and application UI required an understanding of technical concepts like few-shot learning and geospatial data representation, as well as software development skills like implementing code and reading documentation. Testing of the SQL agent's versatility in applying few-shot learning to technically and semantically different test inputs was also conducted. Ultimately, the interaction workflow built in this project demonstrates the workability of integrating cutting-edge technologies like LangChain [11] with more mature technologies like PostGIS [16], which can serve as a baseline for more customized application development in the future.

Appendix

This project has been published to a public GitHub repository at https://github.com/therealaxax/fyp. The setup instructions are contained in the readme file in the repository itself, and have been replicated below as well.

Database Setup

1. Setup PostgreSQL databases and install the PostGIS extension. Remember to enable the PostGIS extension as well if necessary. Set the user to "postgres", the host to "localhost" and the port number to 5432. Refer to https://postgis.net/documentation/getting_started/#installing-postgis [56].

2. Create 2 databases named "fewshots" and "osm2pgsqldb". There is no need to create any tables in "osm2pgsqldb".

3. In the "fewshots" database, create a table named "fewshotexamples". "fewshotexamples" should have 2 columns named "question" and "query" for containing text data.

4. In the "fewshots" database, create a table named "propernouns". "propernouns" should have 1 column named "propernoun" for containing text data.

Setting Up Dependencies

1. It is recommended to use conda for managing dependencies. Install conda following the instructions at https://docs.conda.io/projects/conda/en/stable/user-guide/install/index.html [57]. This Anaconda Distribution installer was used for this project.

2. Create a new conda environment. 'geoenv' in the command below denotes the name of the environment.

   conda create -n geoenv

3. Activate the conda environment.

   conda activate geoenv

4. Install geopandas. Refer to the instructions at https://geopandas.org/en/stable/getting_started/install.html [58]. The command below installs geopandas via the conda-forge channel.

   conda install --channel conda-forge geopandas

5. Install the rest of the dependencies into the same conda environment. This project was built using the versions of langchain, langchain_community, openai, and langchain_openai listed in the commands below.

   pip install psycopg2

   pip install Flask

   pip install langchain==0.0.353

   pip install langchain_community==0.0.10

pip install openai==1.7.0

pip install langchain_openai==0.0.2

6. This project uses the OpenAI "gpt-4" model and requires an OpenAI API key. Details of OpenAI models can be found at https://platform.openai.com/docs/models/overview [59]. Set up an OpenAI API key following the instructions at https://platform.openai.com/docs/quickstart?context=python [60] and note down the key. Note that each API call will be charged by OpenAI.

Running The Project

The data used in this project is OpenStreetMap data for Malaysia, Singapore, and Brunei. However, it is possible to use geospatial data from other regions, by downloading different data files from the Geofabrik server in step 4.

1. git clone this project

2. In the command line, change the working directory to osm2pgsql. Then inside the osm2pgsql folder, change the working directory to osm2pgsql-bin.

3. Create a new directory folder named osmdata_geofabrik inside the osm2pgsql/osm2pgsql-bin directory folder.

4. Download the data file named "malaysia-singapore-brunei-latest.osm.pbf" from https://download.geofabrik.de/asia/malaysia-singapore-brunei.html [61] into the osmdata_geofabrik directory.

5. In the importdata.py file, change the string of the directory variable to the path of the osm2pgsql-bin folder on the local machine.

6. In dbsearch_module.py, in the create_connection(database) function, change the string of the password variable to the one used for setting up the local PostgreSQL databases.

7. In the sqlagentwithtools_module.py file, change the string of the openai_api_key variable to the OpenAI API key. Do this inside the 3 functions, fewshots(), propernounsearchtool(), and createsqlagentwithtools(custom_tool_list).

8. Change the working directory to the flaskr folder.

9. Run the project with the following command:
   flask --app flaskapp.py run --debug

10. Go to http://127.0.0.1:5000/langchainmodel/input to view the landing site of the project.

Configuring Few-Shot Examples and Proper Nouns

1. Go to http://127.0.0.1:5000/langchainmodel/dataimportview and click on "Import Data" to import the downloaded OpenStreetMap data for Malaysia, Singapore, and Brunei into

the "osm2pgsqldb" database. Note that the database password will have to be entered into the terminal as part of the import process when prompted.

2. Go to http://127.0.0.1:5000/langchainmodel/fewshotsview. Pairs of few-shot example questions and SQL queries can be entered row-by-row on this page. The original pairs of few-shot example questions and SQL queries used can be found from Tables 4 and 5 from this project's report, which is also contained in this git repository. Click "Insert" to perform the insertion.

3. Proper nouns can also be entered one-by-one in http://127.0.0.1:5000/langchainmodel/fewshotsview. The original list of proper nouns used can also be found from Table 6 of this project's report. Click "Insert" to perform the insertion.

4. Natural language questions about the geospatial data can be entered in http://127.0.0.1:5000/langchainmodel/input. The original test cases used are documented in the "Application Testing" section of this project's report.

Additional Notes

The sql files contained in this repository are not essential in the running of the Flask application. They contain examples of spatial data-specific SQL queries which can be used to verify the data in the database. Similarly, the qgz file in this repository is not essential, but can be used with QGIS for visualizing the spatial data.

Bibliography

[1] S. Lock, “What is AI chatbot phenomenon ChatGPT and could it replace humans?,” *The Guardian*, Dec. 05, 2022. Accessed: Mar. 24, 2024. [Online]. Available: https://www.theguardian.com/technology/2022/dec/05/what-is-ai-chatbot-phenomenon-chatgpt-and-could-it-replace-humans

[2] K. Roose, “How ChatGPT Kicked Off an A.I. Arms Race,” *The New York Times*, Feb. 03, 2023. Accessed: Mar. 24, 2024. [Online]. Available: https://www.nytimes.com/2023/02/03/technology/chatgpt-openai-artificial-intelligence.html

[3] “What Is ChatGPT? Everything You Need to Know,” WhatIs. Accessed: Mar. 24, 2024. [Online]. Available: https://www.techtarget.com/whatis/definition/ChatGPT

[4] “GPT-4 and GPT-4 Turbo.” Accessed: Mar. 24, 2024. [Online]. Available: https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo

[5] K. Pal, “What Are the Best ChatGPT Alternatives? 12 ChatGPT Rivals,” Techopedia. Accessed: Mar. 24, 2024. [Online]. Available: https://www.techopedia.com/who-are-the-competitors-of-chatgpt

[6] S. Bordow, “Do the math: ChatGPT sometimes can’t, expert says.” Accessed: Mar. 24, 2024. [Online]. Available: https://www.newswise.com/articles/do-the-math-chatgpt-sometimes-can-t-expert-says

[7] BoredGeekSociety, “A First Comprehensive Analysis of ChatGPT’s limitations,” Medium. Accessed: Mar. 24, 2024. [Online]. Available: https://medium.com/@boredgeeksociety/a-first-comprehensive-analysis-of-chatgpts-limitations-4ead13a62bf9

[8] Y. Yamada, Y. Bao, A. K. Lampinen, J. Kasai, and I. Yildirim, “Evaluating Spatial Understanding of Large Language Models.” arXiv, Mar. 05, 2024. Accessed: Mar. 24, 2024. [Online]. Available: http://arxiv.org/abs/2310.14540

[9] M. Ouasti, “The Future of GIS with ChatGPT; will we require humans to write Python Scripts?,” Medium. Accessed: Mar. 24, 2024. [Online]. Available: https://medium.com/@moradouasti/the-future-of-gis-with-chatgpt-a9d2588e841a

[10] Geoapify, “The Importance of GIS: 5 Key Benefits,” Geoapify. Accessed: Mar. 24, 2024. [Online]. Available: https://www.geoapify.com/gis-importance/

[11] “Introduction | 🦜🔗 Langchain.” Accessed: Mar. 24, 2024. [Online]. Available: https://python.langchain.com/docs/get_started/introduction

[12] “langchain_experimental.sql.base.SQLDatabaseChain — 🦜🔗 LangChain 0.1.13.” Accessed: Mar. 24, 2024. [Online]. Available: https://api.python.langchain.com/en/latest/sql/langchain_experimental.sql.base.SQLDatabaseChain.html

[13] ZKS, “Answer to ‘What is the difference between “SQLDatabaseChain” and “create_sql_agent” in langchain?,’” Stack Overflow. Accessed: Mar. 24, 2024. [Online]. Available: https://stackoverflow.com/a/76936960

[14] A. Battula, “SQLDatabaseChain: Answering Questions with SQL Databases,” Medium. Accessed: Mar. 24, 2024. [Online]. Available: https://medium.com/@anushabattula/sqldatabasechain-answering-questions-with-sql-databases-2fb88a458e29

[15] “6. About our data — Introduction to PostGIS.” Accessed: Jan. 27, 2024. [Online]. Available: https://postgis.net/workshops/postgis-intro/about_data.html

[16] “PostGIS,” PostGIS. Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/

[17] “ST_Contains.” Accessed: Jan. 27, 2024. [Online]. Available: https://postgis.net/docs/ST_Contains.html

[18] “Standards,” Open Geospatial Consortium. Accessed: Mar. 24, 2024. [Online]. Available: https://www.ogc.org/standards/

[19] “Chapter 4. Data Management.” Accessed: Mar. 24, 2024. [Online]. Available:

https://postgis.net/docs/using_postgis_dbmanagement.html#OGC_Geometry

[20] “Chapter 4. Data Management.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/using_postgis_dbmanagement.html#Polygon

[21] “Chapter 4. Data Management.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/using_postgis_dbmanagement.html#Point

[22] “Chapter 4. Data Management.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/using_postgis_dbmanagement.html#LineString

[23] “Chapter 7. PostGIS Reference.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/reference.html

[24] “ST_X.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/ST_X.html

[25] “ST_Y.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/ST_Y.html

[26] “ST_Area.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/ST_Area.html

[27] “ST_Perimeter.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/ST_Perimeter.html

[28] H. Su *et al.*, “Selective Annotation Makes Language Models Better Few-Shot Learners.” arXiv, Sep. 05, 2022. doi: 10.48550/arXiv.2209.01975.

[29] L. Knight, “Build your First SQL Database Agent with LangChain,” Medium. Accessed: Jan. 27, 2024. [Online]. Available: https://medium.com/@LawrencewleKnight/build-your-first-sql-database-agent-with-langchain-19af8064ae18

[30] “OpenStreetMap Wiki.” Accessed: Mar. 24, 2024. [Online]. Available: https://wiki.openstreetmap.org/wiki/Main_Page

[31] “Geofabrik Download Server.” Accessed: Jan. 27, 2024. [Online]. Available: https://download.geofabrik.de/

[32] “Home - osm2pgsql.” Accessed: Jan. 27, 2024. [Online]. Available: https://osm2pgsql.org/

[33] “Elements - OpenStreetMap Wiki.” Accessed: Mar. 24, 2024. [Online]. Available: https://wiki.openstreetmap.org/wiki/Elements

[34] “Osm2pgsql Manual - osm2pgsql.” Accessed: Mar. 24, 2024. [Online]. Available: https://osm2pgsql.org/doc/manual.html#the-pgsql-output

[35] “Map features - OpenStreetMap Wiki.” Accessed: Mar. 24, 2024. [Online]. Available: https://wiki.openstreetmap.org/wiki/Map_features#Tourism

[36] “ST_AsText.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/ST_AsText.html

[37] “Discover QGIS.” Accessed: Mar. 24, 2024. [Online]. Available: https://www.qgis.org/en/site/about/index.html

[38] “Agents | 🦜🔗 Langchain.” Accessed: Mar. 24, 2024. [Online]. Available: https://python.langchain.com/docs/modules/agents/

[39] “Prompting strategies | 🦜🔗 Langchain.” Accessed: Jan. 27, 2024. [Online]. Available: https://python.langchain.com/docs/use_cases/sql/prompting

[40] Y. Jiang and C. Yang, “Is ChatGPT a Good Geospatial Data Analyst? Exploring the Integration of Natural Language into Structured Query Language within a Spatial Database,” *ISPRS Int. J. Geo-Inf.*, vol. 13, no. 1, Art. no. 1, Jan. 2024, doi: 10.3390/ijgi13010026.

[41] D. Punjani *et al.*, “Template-Based Question Answering over Linked Geospatial Data.” arXiv, Apr. 29, 2021. Accessed: Jan. 27, 2024. [Online]. Available: http://arxiv.org/abs/2007.07060

[42] “ST_Length.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/ST_Length.html

[43] “ST_Centroid.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/ST_Centroid.html

[44] “ST_DWithin.” Accessed: Mar. 24, 2024. [Online]. Available: https://postgis.net/docs/ST_DWithin.html

[45] “ST_Touches.” Accessed: Mar. 24, 2024. [Online]. Available:

https://postgis.net/docs/ST_Touches.html

[46] “Faiss | 🦜🔗 Langchain.” Accessed: Jan. 27, 2024. [Online]. Available: https://python.langchain.com/docs/integrations/vectorstores/faiss

[47] “Faiss: A library for efficient similarity search,” Engineering at Meta. Accessed: Jan. 27, 2024. [Online]. Available: https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

[48] “Retrievers | 🦜🔗 Langchain.” Accessed: Mar. 24, 2024. [Online]. Available: https://python.langchain.com/docs/modules/data_connection/retrievers

[49] “langchain.tools.retriever.create_retriever_tool — 🦜🔗 LangChain 0.1.13.” Accessed: Mar. 24, 2024. [Online]. Available: https://api.python.langchain.com/en/latest/tools/langchain.tools.retriever.create_retriever_tool.html

[50] “langchain_community.agent_toolkits.sql.base.create_sql_agent — 🦜🔗 LangChain 0.1.13.” Accessed: Mar. 24, 2024. [Online]. Available: https://api.python.langchain.com/en/latest/agent_toolkits/langchain_community.agent_toolkits.sql.base.create_sql_agent.html

[51] “Welcome to Flask — Flask Documentation (3.0.x).” Accessed: Mar. 24, 2024. [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/

[52] “CSS Buttons.” Accessed: Mar. 24, 2024. [Online]. Available: https://www.w3schools.com/css/css3_buttons.asp

[53] “Photo by Bri Schneiter on Pexels,” Pexels. Accessed: Mar. 24, 2024. [Online]. Available: https://www.pexels.com/photo/calm-body-of-lake-between-mountains-346529/

[54] “ChatGPT Plugins waitlist.” Accessed: Mar. 24, 2024. [Online]. Available: https://openai.com/waitlist/plugins

[55] “Deprecations.” Accessed: Mar. 24, 2024. [Online]. Available:

https://platform.openai.com/docs/deprecations

[56]“Getting Started,” PostGIS. Accessed: Apr. 15, 2024. [Online]. Available: https://postgis.net/documentation/getting_started/#installing-postgis

[57]“Installing conda — conda 24.3.0 documentation.” Accessed: Apr. 15, 2024. [Online]. Available: https://docs.conda.io/projects/conda/en/stable/user-guide/install/index.html

[58]“Installation — GeoPandas” Accessed: Apr. 15, 2024. [Online]. Available: https://geopandas.org/en/stable/getting_started/install.html

[59]“Models - Overview.” Accessed: Apr. 15, 2024. [Online]. Available: https://platform.openai.com/docs/models/overview

[60]“Developer quickstart.” Accessed: Apr. 15, 2024. [Online]. Available: https://platform.openai.com/docs/quickstart?context=python

[61]“Geofabrik Download Server.” Accessed: Apr. 15, 2024. [Online]. Available: https://download.geofabrik.de/asia/malaysia-singapore-brunei.html