International Journal of Geo-Information

MDPI

Brief Report

Is ChatGPT a Good Geospatial Data Analyst? Exploring the Integration of Natural Language into Structured Query Language within a Spatial Database

Yongyao Jiang and Chaowei Yang *

NSF Spatiotemporal Innovation Center, Department of Geography and GeoInformation Science, George Mason University, Fairfax, VA 22030, USA; yjiang8@gmu.edu
* Correspondence: cyang3@gmu.edu; Tel.: +1-703-993-4742

Abstract: With recent advancements, large language models (LLMs) such as ChatGPT and Bard have shown the potential to disrupt many industries, from customer service to healthcare. Traditionally, humans interact with geospatial data through software (e.g., ArcGIS 10.3) and programming languages (e.g., Python). As a pioneer study, we explore the possibility of using an LLM as an interface to interact with geospatial datasets through natural language. To achieve this, we also propose a framework to (1) train an LLM to understand the datasets, (2) generate geospatial SQL queries based on a natural language question, (3) send the SQL query to the backend database, (4) parse the database response back to human language. As a proof of concept, a case study was conducted on real-world data to evaluate its performance on various queries. The results show that LLMs can be accurate in generating SQL code for most cases, including spatial joins, although there is still room for improvement. As all geospatial data can be stored in a spatial database, we hope that this framework can serve as a proxy to improve the efficiency of spatial data analyses and unlock the possibility of automated geospatial analytics.

Keywords: large language models; ChatGPT; code generation; geospatial database; data analytics

Citation: Jiang, Y.; Yang, C. 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.* **2024**, *13*, 26. https://doi.org/10.3390/ijgi13010026

Academic Editors: Wolfgang Kainz, Christos Chalkias, Marinos Kavouras, Margarita Kokla and Mara Nikolaidou

Received: 12 September 2023
Revised: 12 December 2023
Accepted: 28 December 2023
Published: 10 January 2024

1. Introduction

Large language models (LLMs) are a type of artificial intelligence (AI) that have been trained on massive datasets of text and code. This allows them to generate text, translate languages, write different kinds of creative content, and answer questions in an informative way [1].

LLM research is rapidly growing and attracting significant interest [2]. In the past year, there have been more than 1000 papers published on LLMs, and the number is growing rapidly [3,4]. This growth is being driven by the increasing potential of LLMs for a wide range of applications, such as customer service, education, and healthcare [5,6]. LLMs are becoming more powerful and capable. The potential applications of LLMs are vast. They can be used for a variety of tasks, including natural language processing (NLP), text generation, and even code generation [7]. Recent advances in LLM research have led to models that are significantly larger and more complex than previous generations. For example, OpenAI's ChatGPT-4 has 175 billion parameters, making it one of the largest and most powerful language models ever created [8].

The application of LLMs to geospatial science involves leveraging these powerful language models to analyze, understand, and generate information related to geographic and spatial data [9]. This fusion of natural language processing and geospatial analysis has opened new avenues for understanding and communicating complex geographic information. For example, LLMs can assist in interpreting various forms of geospatial data, such as satellite imagery, aerial photographs, and geographic maps [10–12]. They can generate textual descriptions, classify objects, and identify features within these datasets.

Mapping based on ChatGPT that converts natural language prompts into geographic maps has also been explored [13].

Automated code generation has a long history of research. There are two major trends: rule-based program synthesis and machine learning. The main problem with the rule-based approach is that these techniques are limited to pre-defined domain-specific languages, making them less scalable to various tasks [14]. The other trend is using machine learning, especially deep learning [15,16]. These methods are more flexible, but they still suffer from a low accuracy. Recent advancements in LLMs have led to a breakthrough in automatic code generation. Recent accuracy and useability assessments of LLM-based code generation tools have shown significant improvement over prior state-of-the-art approaches [17–19]. Researchers have also explored applications like automatic code documentation generation and code refactoring [20,21]. "Text-to-SQL" [22] is a sub-domain of automated code generation in natural language processing. Numerous studies have been conducted to evaluate the accuracy of LLMs and prior machine learning approaches [2,23–25]. However, little is known about the performance of LLMs when generating code for geospatial SQL queries.

As a pioneer study, we explore the possibility of using an LLM as an interface to interact with geospatial datasets through natural language in this paper. To achieve this, we also propose a framework to (1) train an LLM to understand the data, (2) generate geospatial SQL queries based on a natural language question, (3) send the SQL query to the backend database, and (4) parse the database response back to human language. As all geospatial data can be stored in a spatial database, we hope that the framework can serve as a proxy to improve the efficiency of geospatial database analyses and reduce the barriers of geo-analytics.

2. System Framework

The major issue for generating SQL using LLMs and code generation in general is hallucination. LLMs can write SQL, but they are often prone to making up tables and fields and generally writing invalid SQL that cannot be executed against a database. The high-level idea of our framework used to overcome this problem is to provide LLMs with domain knowledge about what exists in the database so that it can write a consistent SQL query. Specifically, our framework (Figure 1) consists of three components:

- A training component that provides the LLM with the database schema and sample data through a prompt. The schema can usually be obtained through certain database commands. Please see Figure 2 for an example.
- A prompt template that combines the training data with a data analysis question in natural language. Prompt templates are pre-defined recipes for generating prompts for language models. A template may include instructions, few-shot examples, and specific context and questions appropriate for a given task. In our setup, the template is "given the following tables in PostGIS, [database schema], please write a SQL query to answer [question]". "Database schema" is a text description of the database schema and sample data, as described in Figure 2. "Question" is the data analytics question we would like to ask.
- A parser then parses the response from the database into human language following the format of the input question. This is also implemented using the LLM. For example, when the question is "what is the total population of New York City?", the framework would return "the total population of New York City is around 8 million", instead of returning a result table with the total column as the single column (Figure 3).

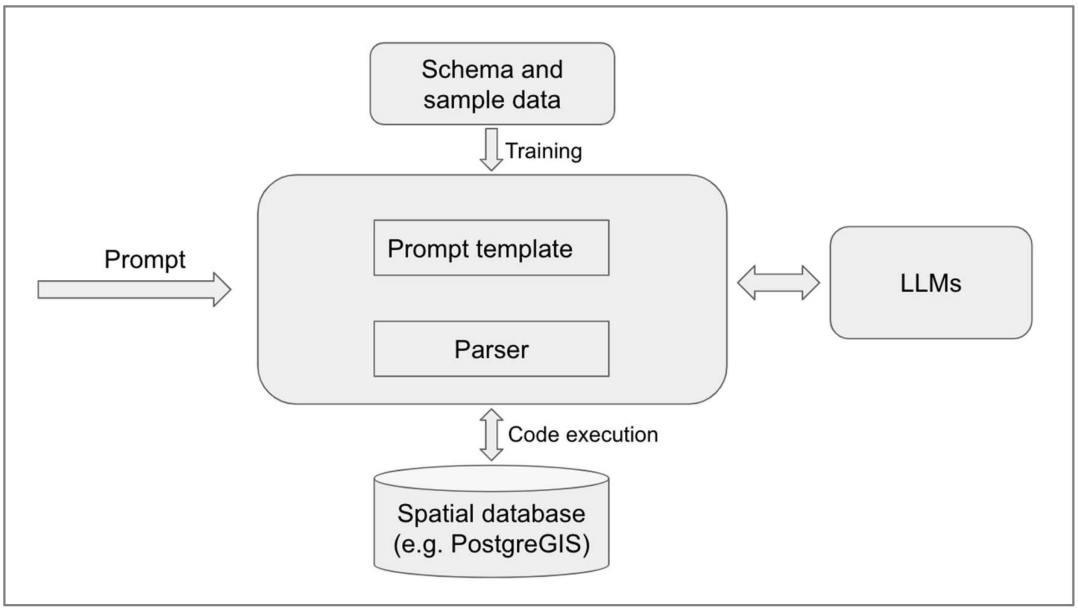

Figure 1. Framework architecture.

```
CREATE TABLE nyc_census_blocks (
        gid SERIAL NOT NULL,
        blkid VARCHAR(15),
        popn_total DOUBLE PRECISION,
        popn_white DOUBLE PRECISION,
        popn_black DOUBLE PRECISION,
        popn_nativ DOUBLE PRECISION,
        popn_asian DOUBLE PRECISION,
        popn_other DOUBLE PRECISION,
        boroname VARCHAR(32),
        CONSTRAINT nyc_census_blocks_pkey PRIMARY KEY (gid)
)

CREATE TABLE nyc_neighborhoods (
        gid SERIAL NOT NULL,
        boroname VARCHAR(43),
        name VARCHAR(64),
        CONSTRAINT nyc_neighborhoods_pkey PRIMARY KEY (gid)
)

CREATE TABLE nyc_streets (
        gid SERIAL NOT NULL,
        id DOUBLE PRECISION,
        name VARCHAR(200),
        oneway VARCHAR(10),
        type VARCHAR(50),
        CONSTRAINT nyc_streets_pkey PRIMARY KEY (gid)
)
…
```

Figure 2. Example training data.

```
SQL Query:SELECT SUM(popn_total) FROM nyc_census_blocks;

Query Result: [(8175032.0,)]

Answer:The population of the City of New York is 8175032.
```

Figure 3. Answer to "what is the population of New York City?".

3. Case Study

3.1. Data

The case study data include four shapefiles for New York City and one attribute table of sociodemographic variables. These data were imported into PostGIS, one of the most popular spatial databases.

- Census blocks [26]: A census block is the smallest geography for which census data are reported. All higher-level census geographies (metro areas, counties, etc.) can be built from unions of census blocks. Example attributes include the total number of people, number of people self-identifying as "Asian", etc.
- Neighborhoods: Neighborhoods are social constructs that do not follow the lines laid down by the government. For example, the Brooklyn neighborhoods of Carroll Gardens, Red Hook, and Cobble Hill were once collectively known as "South Brooklyn".
- Streets: The street centerlines form the transportation network of the city. These streets have been flagged as different types in order to distinguish between thoroughfares such as back alleys, arterial streets, freeways, and smaller streets.
- Subway stations: Subway stations link the upper world where people live to the invisible network of subways beneath.
- Social economic data at the census tract level: Example attributes of this variable include the number of families and median family income.

Please note that once the shapefiles are loaded into PostGIS, they all have a geometry column named "geom". The data are available through a public repository (see below).

3.2. Training Data

ChatGPT-4, one of the mainstream LLMs, is selected as the LLM for this case study. We will compare it with other LLMs in the evaluation section. For ChatGPT to understand our data, we explicitly incorporate the database schema and sample data into prompts. Instead of simply providing table names, columns, and their types, we prompt the LLM with "CREATE TABLE" commands, including column names, types, references, and keys. Researchers have conducted a performance evaluation and found that the best performances were achieved through "CREATE TABLE" commands [2]. Figure 2 shows the example schema description for the "census blocks", "neighborhoods", and "streets". To further improve ChatGPT's ability to create SQL queries, we also provide it with a few sample rows as examples of what the data look like.

3.3. Non-Spatial Query

All the following questions were randomly sampled from our experiment question collection. More details about this question collection can be found in Section 4. The framework was implemented based on Langchain. The first question is a non-spatial query: "what is the population of the City of New York?" The question was provided to the prompt template, and Figure 3 shows the answer from ChatGPT. This is a valid answer that shows that ChatGPT was able to identify the correct column (i.e., popn_total) and table (i.e., nyc_census_block). Our framework also executes and parses the query result into an answer in human language.

In case this is a coincidence, another question was examined: "How many 'neighborhoods' are in each borough?" This time (Figure 4), ChatGPT was not only able to identify the right column (i.e., boroname) and table (nyc_neighborhoods), but also select the correct operations (i.e., COUNT and GROUP BY).

SQL Query: SELECT boroname, COUNT(*) AS num_neighborhoods FROM nyc_neighborhoods GROUP BY boroname LIMIT 5;

Query Result: [('Queens', 30), ('Brooklyn', 23), ('Staten Island', 24), ('The Bronx', 24), ('Manhattan', 28)]

Answer: Queens has 30 neighborhoods, Brooklyn has 23 neighborhoods, Staten Island has 24 neighborhoods, The Bronx has 24 neighborhoods, and Manhattan has 28 neighborhoods.

Figure 4. Answer to "how many 'neighborhoods' are in each borough?".

Figure 5 shows the SQL query generated by ChatGPT for a more complicated question: "For each borough, what percentage of the population is white?" The query is very close

but has a minor syntax issue. The first argument that "ROUND" takes must be a numeric type; therefore, we need to cast the data type from float to numeric explicitly. For a human expert, significant investigation is also needed to diagnose the issue; therefore, it could be argued that this answer is still acceptable.

```
SELECT boroname, ROUND(SUM(popn_white) / SUM(popn_total) * 100, 2) AS percentage_white
FROM nyc_census_blocks
GROUP BY boroname
HAVING popn_total NOT IN (NULL)
ORDER BY percentage_white DESC
LIMIT 5;
```

Figure 5. SQL query for "for each borough, what percentage of the population is white?".

3.4. Spatial Query Based on a Single Table

PostGIS, as an extension of PostgreSQL, provides a variety of spatial functions. For example, "ST_Area" returns the area of the surface if it is a polygon or multi-polygon. "ST_X" returns the X coordinate of the point. Given ChatGPT's good performance at generating SQL for non-spatial queries, we would also like to evaluate it against spatial queries.

The first question is "what is the area of the 'West Village' neighborhood?" It can be seen from Figure 6 that ChatGPT is also capable of basic spatial analysis, and it successfully selected the right spatial operation (ST_Area) to calculate the area.

```
SQL Query: SELECT ST_Area(geom) AS area_of_west_village FROM nyc_neighborhoods
WHERE name = 'West Village';
Query Result: [(1044614.0,)]
Answer:The area of 'West Village' neighborhood is 1044614.
```

Figure 6. Answer to "what is the area of the 'West Village' neighborhood?".

The second question "what is the most westerly subway station?" is more complex, but the answer is still correct (Figure 7), as the LLM was able to interpret how to represent "most westerly" in a spatial SQL operation—the subway station that has the smallest longitude, which is the X coordinate of a point geometry type.

```
SQL Query: SELECT name, ST_X(geom) AS longitude FROM nyc_subway_stations
ORDER BY longitude ASC LIMIT 1;
Query Result: [('Tottenville', 563292.0,)]
Answer:The most westernly subway station is "'Tottenville'".
```

Figure 7. Answer to "what is the most westerly subway station?".

The third spatial question is "what is the area of Manhattan in acres?" Although the answer is correct and ChatGPT was even able to find the conversion rate between meters and acres (Figure 8), the caveat is that ChatGPT assumed that the spatial unit to be meters, which is not mentioned anywhere in the database schema. This might bring up a potential issue of LLMs: bias from online training data.

```
SQL Query: SELECT ST_Area(geom)/4046.86 AS area_in_acres
FROM nyc_neighborhoods
WHERE boroname = 'Manhattan';
```

Figure 8. Answer to "what is the area of Manhattan in acres?".

3.5. Spatial Query with Spatial Joins

Spatial databases are powerful because they not only store geometry, but they also can compare relationships between geometries. For instance, "ST_Contains (geometry A, geometry B)" returns true if and only if no points of B lie in the exterior of A and at least one point of the interior of B lies in the interior of A. "ST_Intersects (geometry A, geometry B)" returns TRUE if the geometries/geography "spatially intersect" and FALSE if they do not [27]. ChatGPT has demonstrated a powerful capability in answering geospatial queries measuring geometries based on a single table. But what about questions about spatial relationships involving multiple spatial objects?

The first question in this category is "The 'Battery Park' neighborhood was off limits for several days after 9/11. How many people had to be evacuated?" The answer is correct, and there a few things worth highlighting (Figure 9):

- ChatGPT knows how to translate the question into a query with the spatial join: "ST_Intersects ()"
- The right aggregation operation "SUM ()" was selected.
- The right tables and geometry types were identified, which are "nyc_census_blocks" and "nyc_neighborhoods".

```
SQL Query: SELECT SUM(popn_total)
FROM nyc_census_blocks
WHERE ST_Intersects(geom,
        (SELECT geom FROM nyc_neighborhoods WHERE name = 'Battery Park'))
Query Result: [17153]
Answer: 17153 people had to be evacuated.
```

Figure 9. Answer to "The 'Battery Park' neighborhood was off limits for several days after 9/11. How many people had to be evacuated?"

Another question is "What subway station is in 'Little Italy'? What subway route is it on?" Figure 10 shows that the answer ChatGPT gave is very close, and the correct spatial join operation were selected: "ST_Contains". However, the order of the two arguments inside "ST_Contains" were wrong. Conceptually, a neighborhood should contain a subway station, not the other way around.

```
SQL Query: SELECT name, routes FROM nyc_subway_stations
WHERE ST_Contains(geom, (SELECT geom FROM nyc_neighborhoods
        WHERE name = 'Little Italy'))
```

Figure 10. Answer to "What subway station is in 'Little Italy'? What subway route is it on?".

The third question is "What is the population and racial make-up of the neighborhoods of Manhattan?" The right tables and spatial join operation (i.e., ST_Intersects) were identified, but there are a few things that went wrong (Figure 11). One is that there is no

such column called "nghbhd", which was a made up by the LLM completely. The other issue is that the LLM should have aggregated "popn_total" based on different races rather than making up columns such as "popn_white".

```
SQL Query: SELECT nghbhd, popn_total AS total_population,
           SUM(popn_white) AS total_white,
           SUM(popn_black) AS total_black,
           SUM(popn_nativ) AS total_nativ,
           SUM(popn_asian) AS total_asian,
           SUM(popn_other) AS total_other
           FROM nyc_census_blocks AS cb
           INNER JOIN nyc_neighborhoods AS nb ON ST_Intersects(cb.geom, nb.geom)
           WHERE nb.boroname = 'Manhattan'
           GROUP BY nghbhd;
```

Figure 11. Answer to "What is the population and racial make-up of the neighborhoods of Manhattan?".

To verify how ChatGPT performs across multiple datasets, a complex question that requires an understanding of four tables was asked: "What is the closest street to the subway stations in the neighborhood that has the highest population density?" Figure 12 shows that the LLM handled this complex case reasonably well. It successfully identified the target tables and the corresponding spatial join operations. There are only two minor issues. The first issue is that it tries to reproject both the subway stations and streets tables to reference "4326". This is unnecessary, because both datasets share the same reference with the neighborhood and census blocks datasets. The other problem "0.01" was picked arbitrarily as the buffer size.

```
SQL Query:
WITH densest_neighborhood AS (
  SELECT n.name AS name, n.geom AS geom
  FROM nyc_census_blocks b
  INNER JOIN nyc_neighborhoods n
  ON ST_Intersects(b.geom, n.geom)
  GROUP BY n.name
  ORDER BY SUM(b.popn_total) / ST_Area(n.geom) DESC
  LIMIT 1
),
station_locations AS (
  SELECT id, name AS station_name, ST_Transform(geom, 4326) AS geom
  FROM nyc_subway_stations
),
street_lines AS (
  SELECT id, name AS street_name, ST_Transform(geom, 4326) AS geom
  FROM nyc_streets
)
SELECT
  sl.station_name,
  s.street_name AS closest_street,
  ST_Distance(sl.geom, s.geom) AS distance
FROM densest_neighborhood dn
INNER JOIN station_locations sl
ON ST_Intersects(dn.geom, sl.geom)
INNER JOIN street_lines s
ON ST_DWithin(sl.geom, s.geom, 0.01)
ORDER BY sl.station_name, distance ASC;
```

Figure 12. Answer to "What is the closest street to the subway stations in the neighborhood that has the highest population density?".

4. Quantitative Evaluation

Based on the case study, the overall performance of ChatGPT to generate SQL queries from natural language questions appears to be very promising. As the questions become more complex, it tends to make mistakes more easily. A quantitative evaluation was

conducted to compare the accuracy of different LLMs and hyperparameter settings. Since our work is one of the first attempts to evaluate an LLM against spatial SQL queries, there is a lack of a benchmarking dataset. As a starting point, we randomly selected 45 questions with 15 for each of the three categories above: non-spatial, single table-based spatial queries, and spatial join queries.

Temperature is a hyperparameter of LLMs that regulates the randomness and creativity of the output of an LLM. The higher the value, the more flexible and creative the model will be. Increasing the temperature value typically makes the output more diverse but might also increase its likelihood of straying from the context. For example, a temperature of 0 is deterministic, meaning that the highest probability response is always selected. A few temperature values were tested using ChatGPT. Table 1 shows that a temperature of zero tends to result in the best performance across all three question categories. An interesting finding is that although a temperature of 0.5 could be used to solve some difficult questions, a temperature of 0 could not, and this performance improvement was not consistent. This is why the overall performance of a temperature of 0.5 is lower than a temperature of 0.

Table 1. Evaluation results of the different temperatures used for ChatGPT.

	Non-Spatial	Single Table Spatial Query	Spatial Join Query
Temperature = 0	0.93	1	0.73
Temperature = 0.5	0.93	0.93	0.6
Temperature = 0.8	0.67	0.6	0.27

Since there are some other notable LLMs other than ChatGPT, we also compared three well-recognized models: GPT-4, PALM 2, and LLAMA 2. All the temperature settings were set to zero in this experiment. As can be seen from Table 2, GPT-4 had the best performance overall. PALM 2 seemed to be great at non-spatial queries, but still needs to be improved for queries involving spatial relationships.

Table 2. Evaluation results for different LLMs.

	Non-Spatial	Single Table Spatial Query	Spatial Join Query
GPT-4	0.93	1	0.73
PALM 2	1	0.93	0.46
LLAMA 2	0.8	0.93	0.67

5. Conclusions and Discussion

In this pilot study, we explored the possibility of using natural language to interact with geospatial datasets with the help of LLMs. The results show that LLMs can be accurate in generating SQL code for most cases, including spatial joins, although there is still room for improvement. Even when it did not generate the correct solution, most of the answers were still on the right track. This method might not completely replace human geospatial data analysts at this point, but it could serve as an assistant or drafting partner to provide a reasonable starting point. We hope that the framework can serve as a proxy to improve the efficiency of geo-analytics and reduce barriers in geospatial analysis.

As an important future study, benchmarking against a large collection of sample questions can provide more convincing results. A reasonable dataset should have at least multiple thousands of entries to try to represent all types of geospatial tasks and NLP challenges (e.g., fuzziness). This can be challenging because there is a lack of text-to-SQL datasets in the geospatial community. However, determining how to compose a comprehensive benchmarking dataset presents a significant research opportunity.

Improving LLMs for SQL generation is a challenging but valuable task, as it can automate and simplify database interactions for a wide range of geo-analytical applica-

tions. There are several ways to further improve the SQL generation performance. The first approach is through better prompt engineering. Rather than zero-shot learning, we could try few-shot learning, which means providing a few examples when asking a question [28,29]. However, based on recent research, improvements with this approach tend not to be consistent. Small changes such as wording or order changes can impact the performance. The other promising approach is to fine-tune the model using domain-specific knowledge [30–32]. For example, we could try incorporating geospatial domain-specific knowledge or ontologies into the fine-tuning process.

Author Contributions: Conceptualization, Yongyao Jiang and Chaowei Yang; methodology, Yongyao Jiang; Validation, Chaowei Yang and Yongyao Jiang; writing—original draft preparation, Yongyao Jiang; writing—review and editing, Chaowei Yang. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by NSF (1841520) and NASA AIST.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: https://github.com/Yongyao/postgis-data-for-llm, accessed on 1 September 2023.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wei, J.; Tay, Y.; Bommasani, R.; Raffel, C.; Zoph, B.; Borgeaud, S.; Yogatama, D.; Bosma, M.; Zhou, D.; Metzler, D. Emergent abilities of large language models. *arXiv* **2022**, arXiv:2206.07682.
2. Rajkumar, N.; Li, R.; Bahdanau, D. Evaluating the text-to-sql capabilities of large language models. *arXiv* **2022**, arXiv:2204.00498.
3. Bahrini, A.; Khamoshifar, M.; Abbasimehr, H.; Riggs, R.J.; Esmaeili, M.; Majdabadkohne, R.M.; Pasehvar, M. ChatGPT: Applications, opportunities, and threats. In *Proceedings of the 2023 Systems and Information Engineering Design Symposium (SIEDS), Charlottesville, VA, USA, 27–28 April 2023*; IEEE: Piscataway, NJ, USA, 2023; pp. 274–279.
4. Lund, B.D.; Wang, T. Chatting about ChatGPT: How may AI and GPT impact academia and libraries? *Libr. Hi Tech News* **2023**, *40*, 26–29. [CrossRef]
5. Biswas, S.S. Role of chat gpt in public health. *Ann. Biomed. Eng.* **2023**, *51*, 868–869. [CrossRef] [PubMed]
6. Fraiwan, M.; Khasawneh, N. A Review of ChatGPT Applications in Education, Marketing, Software Engineering, and Healthcare: Benefits, Drawbacks, and Research Directions. *arXiv* **2023**, arXiv:2305.00237.
7. Dong, Y.; Jiang, X.; Jin, Z.; Li, G. Self-collaboration Code Generation via ChatGPT. *arXiv* **2023**, arXiv:2304.07590.
8. Zhang, C.; Zhang, C.; Zheng, S.; Qiao, Y.; Li, C.; Zhang, M.; Dam, S.K.; Thwal, C.M.; Tun, Y.L.; Huy, L.L. A complete survey on generative ai (aigc): Is chatgpt from gpt-4 to gpt-5 all you need? *arXiv* **2023**, arXiv:2303.11717.
9. Zhang, Y.; Wei, C.; Wu, S.; He, Z.; Yu, W. GeoGPT: Understanding and Processing Geospatial Tasks through An Autonomous GPT. *arXiv* **2023**, arXiv:2307.07930.
10. Mai, G.; Huang, W.; Sun, J.; Song, S.; Mishra, D.; Liu, N.; Gao, S.; Liu, T.; Cong, G.; Hu, Y. On the opportunities and challenges of foundation models for geospatial artificial intelligence. *arXiv* **2023**, arXiv:2304.06798.
11. Osco, L.P.; Lemos, E.L.D.; Gonçalves, W.N.; Ramos, A.P.M.; Marcato Junior, J. The Potential of Visual ChatGPT For Remote Sensing. *Remote Sens.* **2023**, *15*, 3232. [CrossRef]
12. Wang, D.; Lu, C.-T.; Fu, Y. Towards automated urban planning: When generative and chatgpt-like ai meets urban planning. *arXiv* **2023**, arXiv:2304.03892.
13. Tao, R.; Xu, J. Mapping with ChatGPT. *ISPRS Int. J. Geo-Inf.* **2023**, *12*, 284. [CrossRef]
14. Gulwani, S. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Not.* **2011**, *46*, 317–330. [CrossRef]
15. Balog, M.; Gaunt, A.L.; Brockschmidt, M.; Nowozin, S.; Tarlow, D. Deepcoder: Learning to write programs. *arXiv* **2016**, arXiv:1611.01989.
16. Le, T.H.; Chen, H.; Babar, M.A. Deep learning for source code modeling and generation: Models, applications, and challenges. *ACM Comput. Surv. CSUR* **2020**, *53*, 1–38. [CrossRef]
17. Liu, A.; Hu, X.; Wen, L.; Yu, P.S. A comprehensive evaluation of ChatGPT's zero-shot Text-to-SQL capability. *arXiv* **2023**, arXiv:2303.13547.
18. Liu, J.; Xia, C.S.; Wang, Y.; Zhang, L. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv* **2023**, arXiv:2305.01210.
19. Vaithilingam, P.; Zhang, T.; Glassman, E.L. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In Proceedings of the Chi Conference on Human Factors in Computing Systems Extended Abstracts, New Orleans, LA, USA, 29 April–5 May 2022; pp. 1–7.
20. Khan, J.Y.; Uddin, G. Automatic code documentation generation using gpt-3. In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, Rochester, MI, USA, 10–14 October 2022; pp. 1–6.

21. Poldrack, R.A.; LU, T.; Beguš, G. AI-assisted coding: Experiments with GPT-4. *arXiv* **2023**, arXiv:2304.13187.
22. Qin, B.; Hui, B.; Wang, L.; Yang, M.; Li, J.; Li, B.; Geng, R.; Cao, R.; Sun, J.; Si, L. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv* **2022**, arXiv:2208.13629.
23. Finegan-Dollak, C.; Kummerfeld, J.K.; Zhang, L.; Ramanathan, K.; Sadasivam, S.; Zhang, R.; Radev, D. Improving text-to-sql evaluation methodology. *arXiv* **2018**, arXiv:1806.09029.
24. Li, J.; Hui, B.; Qu, G.; Li, B.; Yang, J.; Li, B.; Wang, B.; Qin, B.; Cao, R.; Geng, R. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv* **2023**, arXiv:2305.03111.
25. Trummer, I. CodexDB: Generating Code for Processing SQL Queries using GPT-3 Codex. *arXiv* **2022**, arXiv:2204.08941.
26. Leslie, M.; Ramsey, P. Introduction to PostGIS. 2023. Available online: https://postgis.net/workshops/postgis-intro/ (accessed on 1 September 2023).
27. Obe, R.; Hsu, L.S. *PostGIS in Action*; Simon and Schuster: New York, NY, USA, 2021.
28. Loukas, L.; Stogiannidis, I.; Malakasiotis, P.; Vassos, S. Breaking the Bank with ChatGPT: Few-Shot Text Classification for Finance. *arXiv* **2023**, arXiv:2308.14634.
29. Scaringi, G.; Loche, M. An Interview with ChatGPT: Discussing Artificial Intelligence in Teaching, Research, and Practice. Available online: https://eartharxiv.org/repository/view/5041/ (accessed on 1 September 2023).
30. Kasneci, E.; Sessle, K.; Küchemann, S.; Bannert, M.; Dementieva, D.; Fischer, F.; Gasser, U.; Groh, G.; Günnemann, S.; Hüllermeier, E. ChatGPT for good? On opportunities and challenges of large language models for education. *Learn. Individ. Differ.* **2023**, *103*, 102274. [CrossRef]
31. Sallam, M. ChatGPT utility in healthcare education, research, and practice: Systematic review on the promising perspectives and valid concerns. *Healthcare* **2023**, *11*, 887. [CrossRef]
32. Zhong, Q.; Ding, L.; Liu, J.; Du, B.; Tao, D. Can chatgpt understand too? a comparative study on chatgpt and fine-tuned bert. *arXiv* **2023**, arXiv:2302.10198.