International Journal of Geo-Information

MDPI

Article

Generative AI for Geospatial Analysis: Fine-Tuning ChatGPT to Convert Natural Language into Python-Based Geospatial Computations

Zachary Sherman [1], Sandesh Sharma Dulal [1], Jin-Hee Cho [2], Mengxi Zhang [3] and Junghwan Kim [1,*]

[1] Department of Geography, Virginia Tech, Blacksburg, VA 24060, USA; zacfreerun10@vt.edu (Z.S.); sandesh339@vt.edu (S.S.D.)
[2] Department of Computer Science, Virginia Tech Research Center, 900 N Glebe Rd, Arlington, VA 22203, USA; jicho@vt.edu
[3] Department of Health Systems and Implementation Science, Virginia Tech Carilion School of Medicine, 2 Riverside Circle, Roanoke, VA 24016, USA; mengxizhang@vt.edu
* Correspondence: junghwankim@vt.edu

Highlights

What are the main findings?

- Fine-tuning GPT-4o-mini on geospatial queries significantly improves Python code generation for spatial analysis tasks
- The fine-tuned model achieved an 89.7% accuracy rate, improving 49.2 percentage points over the baseline.

What is the implication of the main finding?

- Integrating LLMs into geospatial dashboards enables real-time, user-friendly analysis for smart city management.
- This framework offers scalable potential for domain-specific AI tools in geospatial science and smart urban analytics.

Abstract

This study investigates the potential of fine-tuned large language models (LLMs) to enhance geospatial intelligence by translating natural language queries into executable Python code. Traditional GIS workflows, while effective, often lack usability and scalability for non-technical users. LLMs offer a new approach by enabling conversational interaction with spatial data. We evaluate OpenAI's GPT-4o-mini model in two forms: an "As-Is" baseline and a fine-tuned version trained on 600+ prompt–response pairs related to geospatial Python scripting in Virginia. Using U.S. Census shapefiles and hospital data, we tested both models across six types of spatial queries. The fine-tuned model achieved 89.7%, a 49.2 percentage point improvement over the baseline's 40.5%. It also demonstrated substantial reductions in execution errors and token usage. Key innovations include the integration of spatial reasoning, modular external function calls, and fuzzy geographic input correction. These findings suggest that fine-tuned LLMs can improve the accuracy, efficiency, and usability of geospatial dashboards when they are powered by LLMs. Our results further imply a scalable and replicable approach for future domain-specific AI applications in geospatial science and smart cities studies.

Keywords: geospatial data; dashboard; fine-tuned; ChatGPT; Large Language Model

Academic Editors: Wolfgang Kainz, Levente Juhász, Hartwig H. Hochmair and Hao Li

Received: 13 April 2025
Revised: 4 August 2025
Accepted: 14 August 2025
Published: 18 August 2025

Citation: Sherman, Z.; Sharma Dulal, S.; Cho, J.-H.; Zhang, M.; Kim, J. Generative AI for Geospatial Analysis: Fine-Tuning ChatGPT to Convert Natural Language into Python-Based Geospatial Computations. *ISPRS Int. J. Geo-Inf.* **2025**, *14*, 314. https://doi.org/10.3390/ijgi14080314

1. Introduction

1.1. Background

Recent advances in large language models (LLMs), such as OpenAI's ChatGPT, hold transformative potential for expanding stakeholder participation in geospatial analysis and visualization while enhancing smart urban management processes [1–3]. For instance, LLM-powered chatbots can translate natural language queries into programming code and execute analytical tasks with different file types [4,5], including geospatial data [6–8]. By leveraging conversational interfaces, these LLM-powered chatbots enable non-technical users (e.g., policymakers and citizens) to interact with geospatial data more intuitively, facilitating rigorous problem-solving that ultimately leads to smarter management of cities [9–11]. However, most geographic information system (GIS) tools are not designed for conversational interaction, posing a barrier to usability and accessibility.

Recent advances in LLMs can potentially fill this gap by enabling natural language interaction, opening the door to broader participation in smart decision-making based on geospatial data. This paper aims to harness the potential of LLMs to support user-friendly decision-making processes involving complex spatial data and analysis. Specifically, we conducted a methodological investigation to evaluate the capabilities of OpenAI's GPT-4o-mini model in addressing geospatial queries and examine how the refined version (i.e., fine-tuned using our proposed strategies) enhances geospatial reasoning compared to the original model. This paper begins by reviewing state-of-the-art research at the intersection of LLMs and geospatial analysis, highlighting gaps in existing studies. We then outline our research methods and materials, followed by presenting results and discussion.

1.2. State-of-the-Art: The Intersection of LLMs and Geospatial Analysis

Since the launch of OpenAI's ChatGPT in 2022, a growing number of studies in the field of geography and GIScience have explored the use of LLMs for geospatial domains [12], such as location information retrieval [13], smart urban infrastructure monitoring [14,15], and geospatial analysis workflows [16–20].

For instance, Jiang and Yang [16] evaluated LLMs for generating structured spatial queries in SQL. Their results demonstrate that while LLMs could automate basic spatial tasks, LLMs struggled with more complex operations such as spatial joins and multi-table queries. Similarly, Zhang et al. [20] introduced MapGPT, a framework for generating thematic maps through natural language inputs that simplified map creation but faced challenges in handling advanced spatial logic and maintaining accuracy. Mansourian and Oucheikh [17] further advanced this area with ChatGeoAI, a platform leveraging LLMs to translate natural language queries into geospatial code using PyQGIS. Ning et al. [18] proposed LLM-Find, an autonomous GIS agent framework that allows LLMs to retrieve geospatial data by selecting sources and generating executable code. While the system showed promising results, it still struggled with lengthy handbook inputs, vague user queries, and reliable retrieval of complex spatial datasets.

Although these advancements mark significant progress in applying LLMs to geospatial data and tasks, a growing number of studies have also reported limitations in the spatial reasoning and processing capabilities of LLMs. For example, Zhang et al. [21] found that while LLMs can generate basic spatial queries, they often fail when handling complex workflows requiring multi-layered computations. Similarly, Tao and Xu [22] highlighted how LLMs frequently generate "hallucinated data", outputs that appear correct but contain fundamental spatial errors, such as misplacing counties from Missouri and Arkansas onto a population map intended for Mississippi. This occurred because the model relied on name matching (e.g., counties named "Mississippi") rather than accurate geospatial identifiers like FIPS codes, revealing a critical limitation in its ability to perform reliable spatial joins.

In real-world applications where such LLMs inform decision-making processes, these inaccuracies may lead to misinformed policy decisions and the inefficient allocation of resources. Additionally, many LLMs are trained in text-based corpora, meaning they lack domain-specific geospatial reasoning. A recent study by Renshaw et al. [19] found potential geographic biases in LLMs' spatial reasoning capabilities. For example, LLMs performed better in high-density urban areas than in low-density rural areas when asked to identify queen-type adjacent counties or K-5 nearest neighbors.

This 'uneven geography' of LLM capabilities has been noted in previous studies. For example, Kim et al. [13] examined ChatGPT's ability to provide locally specific responses related to environmental issues across more than 3000 U.S. counties. Their findings, consistent with those of Renshaw et al. [19], revealed that ChatGPT struggled to generate local-specific information for low-density rural counties compared to high-density urban ones, highlighting a key limitation of LLMs in the geospatial context. Additionally, Jang and Kim [15] investigated multimodal LLMs and their ability to detect built environment features (e.g., trees, streetlights) from street-view images. They found that an LLM performed better on images from urban areas than rural ones, further confirming the existence of an 'uneven geography' in LLM capabilities.

1.3. Research Goal and Questions

Therefore, to effectively employ LLMs in geospatial analysis and visualization processes, these limitations must be addressed. Overcoming these limitations requires integrating GIS data structures, providing specialized training in spatial concepts, and developing spatial reasoning modules to improve accuracy in geospatial queries. Given these challenges, there remains a significant knowledge gap regarding how to enhance LLMs' ability to accurately and efficiently process geospatial data queries, with only a few studies addressing this issue [23,24] to our best knowledge. Therefore, to address this critical gap, this study aims to enhance the ability of LLMs to generate executable Python code for geospatial analysis by proposing and evaluating targeted fine-tuning strategies.

To achieve this research goal, we ask the following three research questions. First, how accurately can an existing LLM (an "As-Is" LLM that is not fine-tuned) generate Python code for handling various geospatial data and queries? Second, what types of errors are frequently observed for the "As-Is" LLM? Third, how does a fine-tuned LLM compare to an "As-Is" LLM in accuracy and efficiency in handling geospatial data and queries? To answer these questions, we adopt a case study approach focused on hospital locations in Virginia, United States. Virginia was selected for its diverse geographic characteristics, including a mix of urban, suburban, and rural regions [25,26]. While the case study is conducted within Virginia, the methodology is designed to be scalable and adaptable to broader applications in enhancing LLMs' performance in processing geospatial data queries. We also conducted an additional test using New York State as a new study area to validate our approach and framework. Furthermore, we selected and compared two LLMs (OpenAI's GPT-3.5-Turbo and GPT-4o-mini), which are accessible via API.

Our work has the following key contributions. First, we propose a novel fine-tuning framework that trains LLMs to generate executable Python code for geospatial analysis using natural language input, addressing a critical usability gap in traditional GIS tools. Second, we implement innovative mechanisms within the fine-tuned model, including modular external function calls, spatial reasoning enhancements, and fuzzy geographic input correction, which have not been widely adopted in geospatial workflows. Third, we empirically demonstrate that fine-tuning dramatically improves model performance, achieving 89.7% accuracy (vs. 40.5% in the baseline) and reducing execution errors and token usage by over 70%. These improvements enhance the model's effectiveness, computa-

tional efficiency, and cost-effectiveness, making LLMs more capable in geospatial reasoning and scalable for real-time, high-volume, industry-level applications.

2. Materials and Methods

2.1. Overview of Research Design

The primary objective of this study is to propose fine-tuning strategies to enhance LLMs' ability to process geospatial data queries with greater accuracy and efficiency. To achieve this, we first evaluated the performance of an existing, non-fine-tuned LLM ("As-Is" LLM) in handling geospatial data and queries. We assessed the model's accuracy by comparing the LLM's results with those generated by traditional geospatial analysis tools. Next, we analyzed the types of errors in the As-Is LLM to gain insights into developing fine-tuning strategies. Finally, we fine-tuned the As-Is model and compared its performance to assess the effectiveness of our fine-tuning strategies in improving the LLM's capability in handling geospatial data and queries.

We adopted a case study approach [2,23], focusing on a realistic scenario where geospatial queries are centered on a public health topic, such as understanding geospatial accessibility to hospitals. We purposefully selected this domain because geospatial data have played a critical role in public health planning, resource allocation, and addressing health disparities [27]. Public health is also a domain where place-based decisions that are informed by geospatial data and analysis are crucial, making it a useful testbed for evaluating the utility of LLMs in geospatial analysis. Additionally, healthcare accessibility is a key concern across urban, suburban, and rural contexts, enabling assessment of the model's performance under diverse spatial conditions [28].

To conduct this case study, we utilized the following empirical geospatial data. The primary datasets included:

1. Virginia polygon shapefiles representing county and ZIP code boundaries, based on the U.S. Census Bureau's TIGER/Line database, which served as the spatial units for aggregation and spatial filtering.
2. A dataset of street addresses, randomly sampled across Virginia, to simulate realistic inputs for location-based accessibility analyses.
3. A point shapefile of hospital locations, obtained from the Virginia Geographic Information Network (VGIN), which provided accurate and up-to-date healthcare facility coordinates essential for proximity and network-based accessibility calculations.

By incorporating multiple geographic scales (i.e., ZIP code, county, and individual addresses) and grounding our hospital location data into an authoritative state-level source (VGIN), we ensured that the model was trained and evaluated under conditions that reflect plausible user scenarios in the real world [16]. Furthermore, the use of standardized and widely available geospatial formats enhances the reproducibility and scalability of our approach for broader adoption across other applications.

2.2. As-Is Model: OpenAI's GPT-4o-Mini

The As-Is model served as a baseline, utilizing an unmodified version of OpenAI's GPT-4o-mini model and GPT-3.5-Turbo to generate Python code in response to spatial queries (Figure 1).

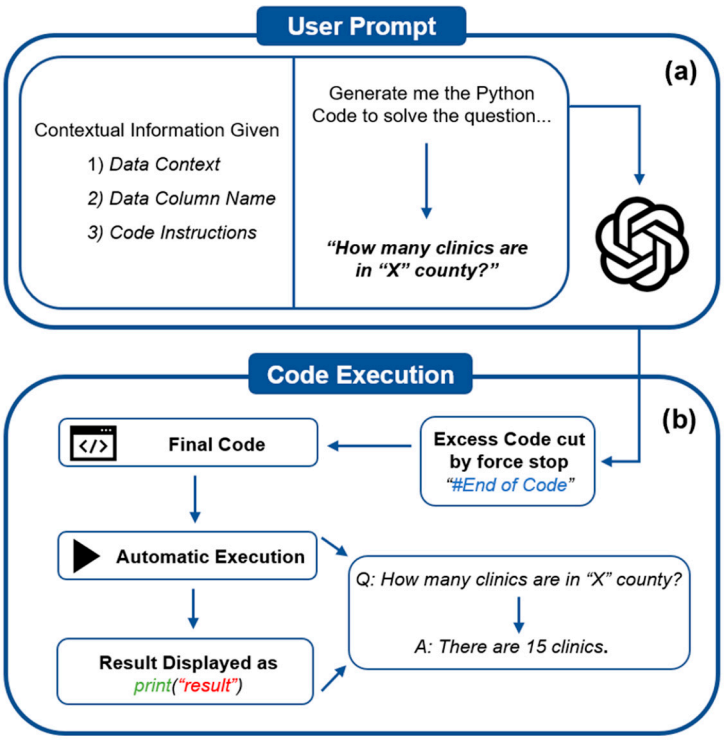

Figure 1. Workflow of the As-Is model. In step (**a**), the user provides a prompt including data context and instructions. In step (**b**), the model generates Python code, which is then automatically executed to return the final result. Excess code is removed using a forced stop tag ("#End of Code").

GPT-4o-mini was selected for its optimal balance of performance, speed, and cost, making it a suitable candidate for scalable and cost-effective applications in industry and research. According to OpenAI, GPT-4o-mini is a "fast, affordable small model for focused tasks," and is particularly ideal for fine-tuning in domain-specific applications [29,30]. Additionally, GPT-4o-mini supports distillation from larger models such as GPT-4o, achieving similar performance with lower latency and reduced computational costs, valuable for resource-constrained or high-volume deployments [29,30]. GPT-4o-mini consistently outperformed GPT-3.5-turbo across nearly every category, including accuracy, execution success, and token efficiency, making GPT-4o-mini a reliable foundation for fine-tuning in spatial code generation tasks. Although several comparable LLMs exist, GPT-4o-mini provided the most practical balance of accessibility, cost, and performance for this study's scope.

This As-Is model was not fine-tuned. However, to help the LLM's minimal understanding of our dataset, details (e.g., names of variables) were provided within each prompt. For example, each prompt specified the data context, including key shapefiles, relevant variable names, and their associated attributes. The contextual information included geographic information on counties, ZIP code areas, and clinic facility locations across Virginia. To improve query accuracy, essential columns from each dataset, such as common U.S. Census variable names (e.g., NAMESLAD for county names and ZCTA5CE20 for ZIP codes), were explicitly referenced. Providing this minimal context was necessary to fairly evaluate the As-Is model because, without context about variable names or the structure of the input data, the model would not have been able to generate meaningful or executable geospatial code [31–33]. Since language models inherently lack knowledge of dataset schemas, omitting this context would make it impossible to fairly assess the model's ability to translate spatial questions into valid Python queries.

Moreover, several formatting instructions were added to standardize the model's output to ensure the generated code could be executed directly without modification.

These instructions specified that the model should omit wrapping the code as a string, use a consistent variable structure (e.g., referencing data variables directly without reloading the shapefiles), and end each code block with "#End of Code" for easy identification [34]. For queries involving address geocoding and driving time estimations, explicit guidance was provided to use the geopy packages [35,36] for address handling and the Mapbox API [8,37] for calculating travel times. This was done to ensure the model had the necessary tools to answer location-based questions accurately. Appendix A.1 illustrates the prompts used for the As-Is model.

Despite structured prompts, formatting instructions, and domain-specific context, the As-Is model exhibited several fundamental limitations, including limited spatial awareness, frequent hallucinations, and excessive token usage, all of which compromised its ability to generate accurate geospatial queries. Details of these limitations will be illustrated in the results sections. These shortcomings emphasize the necessity of fine-tuning to enhance the model's spatial awareness, reduce hallucinations, and improve consistency in query execution.

2.3. Fine-Tuned Model Framework

The fine-tuned model addresses key limitations of the As-Is model, specifically improving geospatial reasoning, query accuracy, and computational efficiency. Figure 2 illustrates an overview of the fine-tuned model's mechanism.

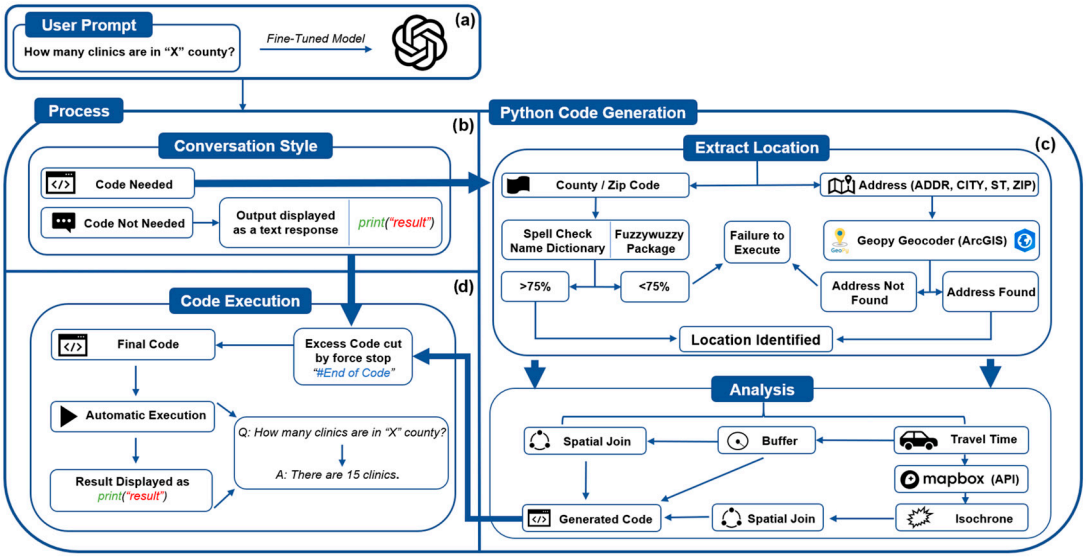

Figure 2. Workflow of the Fine-Tuned Model. (**a**) After receiving a user prompt, (**b**) the model determines whether a computational response or general information is required, formatting outputs accordingly. (**c**) Location inputs undergo correction and geocoding via fuzzy matching and ArcGIS integration. Relevant spatial analyses, including spatial joins, buffers, and travel-time calculations, are executed through pre-integrated Python libraries and APIs, reducing computational overhead. (**d**) Excess code is removed using a forced stop tag ("#End of Code"), with final results displayed automatically. The model generates Python code, which is automatically parsed and executed within a Python environment that includes pre-integrated spatial libraries and API keys. No manual copy-pasting is needed, enabling seamless interaction between the LLM and geospatial analysis tools.

2.3.1. Fine-Tuning Procedure

While the As-Is model struggled with spatial joins, buffer operations, and driving distance estimations, the fine-tuned model integrates custom training, external function calls, and error handling to improve its performance. A fine-tuned model is a customized adaptation of OpenAI's LLMs, allowing it to specialize in domain-specific tasks. In this

study, a fine-tuned version of OpenAI's GPT-4o-mini was developed to enhance geospatial analysis capabilities, addressing the As-Is model's shortcomings.

The model was trained on a dataset of 634 prompt–response pairs, each designed to refine spatial query execution. Training data consisted of customized correct answers and validated outputs from the As-Is model [16]. To improve generalization and reduce overfitting, we employed prompt augmentation, a process in which each spatial query was rewritten in multiple semantically equivalent forms to reflect the diversity of natural language input. For example, the query 'How many clinics are in X County?' was rephrased as 'What is the total number of clinics in X County?', 'Can you tell me the number of clinics operating in X County?', and other variations. This prompt augmentation approach ensured the model could interpret and process differently phrased questions while avoiding overfitting to specific patterns in the training data [38].

2.3.2. Model Usage and Operation Architecture

Fine-tuning a GPT model involves training it on pairs of prompts and completions, where the prompt is the input (e.g., a question or instruction) and the completion is the ideal response that the model should generate (e.g., correct executable Python code). This process teaches the model to produce more accurate or specialized outputs based on the specific patterns in the training data. The completion responses were designed to reflect real-world geospatial workflows by implementing appropriate spatial operations depending on the query type. These workflows are foundational geospatial methods widely used in GIS practice and public health accessibility analysis. Specifically, operations such as spatial joins, buffering for proximity analysis, nearest-neighbor queries, and isochrone-based service area analyses are extensively documented in both GIS textbooks [39,40] and software documentation [41–43] as standard practice. Similarly, studies in public health and urban planning frequently use these techniques to assess accessibility, service coverage, and spatial disparities [19,27,44]. This alignment ensures that the fine-tuning dataset mirrors the operational workflows used by GIS professionals in real-world scenarios. For example, queries asking for the number of clinics within a given distance of a ZIP code or county centroid used buffering and spatial joins, implemented using the GeoPandas library.

For queries that are required to identify the closest clinic to a specific address, the model used the ArcGIS Geocoding API to convert the textual address into geographic coordinates and then calculates the nearest point using spatial distance functions. For queries involving driving-time accessibility (e.g., "clinics within a 15-min drive"), the Mapbox Isochrone API was used to generate drivetime polygons, which were then intersected with clinic locations using GeoPandas. Across all query types, the spatial data manipulations, such as reading shapefiles, creating buffers, calculating distances, or executing spatial joins, were consistently implemented using the Python GeoPandas library to maintain reproducibility and efficiency. The core geospatial operations represented in the generated Python code are mathematically formalized in the Appendix, which outlines the spatial logic underlying the model's code generation outputs. By including these realistic geospatial operations in the completion responses, the fine-tuned model learned how to accurately convert natural language queries into executable code for location-based spatial analysis. The Appendix illustrates selective examples of prompt–response pairs that were used for fine-tuning.

Several key improvements were implemented in the fine-tuned model to overcome the As-Is model's shortcomings, including conversational adaptability, location identification, and optimized computational efficiency.

First, the fine-tuned model integrates conversational adaptability (Figure 2b), distinguishing between general inquiries and computational queries. Unlike the As-Is model, the fine-tuned model assesses whether a user query requires executable code or a con-

versational response. If a query asks for general information rather than a geospatial computation, the model formats responses as text within a Python print statement and then executes that result automatically. However, when a query demands a spatial computation, the model automatically generates and structures Python code for execution. This structured approach enables handling both general (non-spatial) queries and precise geospatial analytics within the same framework.

Second, the fine-tuned model includes a location identification mechanism that corrects misspelled or improperly formatted geographic names (Figure 2c). This mechanism is particularly important because, despite improvements in language model performance, LLMs are still prone to failure when user inputs do not precisely match the names found in the geospatial dataset [41]. For example, even a small typo in a county name (e.g., "Fairfak" instead of "Fairfax") can lead to the generation of code that references nonexistent data, resulting in execution errors or failed analyses. To address this, the model uses the Fuzzywuzzy package to compare user inputs against a predefined geographic dictionary containing valid names of counties and ZIP code areas [45]. Fuzzywuzzy applies the Levenshtein distance to compute the similarity between the input string and the valid geographic names [42]. If a confidence score of 75% is met, the model automatically corrects the input to the nearest valid location. If no match is found, the user is prompted to refine their input, preventing the generation of invalid code [43]. As shown in Figure 2c, user prompts containing location-based queries are processed through the spelling correction system before advancing to geospatial analysis, reducing the likelihood of errors due to minor spelling variations.

Third, a significant optimization in the fine-tuned model is the use of external function calls, which improves computational efficiency by reducing token usage (Figure 2c). Unlike the As-Is model, which generated full geospatial scripts within each response, the fine-tuned model calls predefined functions for key operations such as geocoding, isochrone generation, and spatial joins. For instance, when a query requires an address-to-coordinate conversion, the model invokes an ArcGIS geocoder function to retrieve latitude and longitude coordinates, ensuring that address-based queries are accurately interpreted. Another example includes cases for driving distance calculations. In this case, the model integrates the Mapbox API, generating isochrones that represent areas accessible within a specified travel time. These function calls significantly reduce token consumption, lowering computational costs and improving scalability for real-world applications [46]. It is worth mentioning that while the fine-tuned model references external functions for operations such as geocoding, spatial joins, and travel-time estimation, it does not use OpenAI's built-in function calling capabilities. Instead, the model is trained to generate Python code that invokes predefined custom functions, which are implemented externally and accessible in the execution environment.

2.4. Evaluation Setup for the Performance Comparison of As-Is and Fine-Tuned Model

To assess the effectiveness of the fine-tuned model, a comparative evaluation was conducted between the As-Is model and the fine-tuned model, which was specifically trained to process geospatial queries. Both models were tested on six predefined geospatial queries designed to convert natural language into accurate and executable Python. These questions include:

Q1. How many clinics are in "X" County?
Q2. How many clinics are in the area with the ZIP Code "X"?
Q3. What is the closest clinic to "X Address"?
Q4. How many clinics are within 10 miles of the Centroid of "X" County?

Q5. How many clinics are within 10 miles of the Centroid of the area with the ZIP Code "X"?

Q6. How many clinics are within a 15-min driving time of "X Address"?

These queries were selected to reflect typical geospatial analysis tasks relevant to practitioners, such as assessing healthcare facility locations, calculating proximity-based accessibility, and determining travel times to medical providers [27,28,44]. Specifically, by including both static location-based queries (e.g., total clinics in a county) and dynamic proximity-based queries (e.g., clinics within a 15-min drive), our evaluation approach ensures that LLMs are tested on a practical subset of core geospatial tasks.

Each query was executed 100 times using randomly selected county names, addresses, or ZIP codes from our case study region (Virginia), ensuring that the model's performance was evaluated across multiple spatial contexts. A total of 1200 evaluation queries were issued across both As-Is and fine-tuned models. The evaluation was structured to measure the models' accuracy and consistency across different spatial scales (e.g., street address vs. zip code vs. county). Accuracy was defined as the model's ability to generate correct and executable Python code, while consistency was assessed based on whether similar queries produced reliable responses across multiple iterations. Ground truth answers were established through manual geospatial analysis using Python libraries such as GeoPandas. These analyses included spatial joins, buffering, and Mapbox API-based driving-time calculations.

Any generated code that failed to execute or produced incorrect results was classified as incorrect, while code that executed successfully and returned correct outputs was considered accurate. For incorrect answers, the code was evaluated and categorized by error type. Additionally, token count was analyzed for all generated code to assess the computational efficiency of the model when handling spatial queries. In the context of LLMs, tokens are the basic units of text that the model reads and generates. A token can be a word, part of a word, or punctuation, depending on the language and tokenization rules used. Token usage is an important measure because it directly affects computational cost, API pricing, and the model's ability to process longer or more complex queries [29,30]. For example, the sentence 'How many clinics are in Fairfax County?' could be broken into tokens such as ['How', 'many', 'clinics', 'are', 'in', 'Fairfax', 'County', '?']. Token counts directly influence both computational costs and processing limits within LLM systems.

2.5. External Validation for Geographic Generalizability

To evaluate the geographic generalizability of the models, we conducted external validation using data from New York State. Two queries were selected to represent different spatial complexities: (1) "How many clinics are in X County?", administrative spatial join, and (2) "What clinics are within a 15-min driving time of X Address?", a higher-complexity proximity analysis requiring address-level geocoding and isochrone computation. This subset was intentionally chosen to reflect both low and high spatial complexity. The New York dataset was entirely separate from the Virginia data used for training, providing a test of the model's ability to generalize spatial reasoning to unseen geographies. New York was selected for external validation in part because it has been used in previous studies as a representative test case for evaluating spatial model generalizability, and also because it presents a diverse mix of dense urban, suburban, and rural geographies, making it a robust test for assessing spatial reasoning across varying spatial contexts [16].

3. Results

This section presents a comparative evaluation of the As-Is and fine-tuned LLMs across six spatial query types. We analyze model accuracy, execution success, token efficiency, and coding errors to assess the practical benefits of fine-tuning for geospatial tasks.

3.1. Performance Comparison Between As-Is and Fine-Tuned Models

Table 1 presents the total counts of correct and incorrect answers for the As-Is and fine-tuned models across all spatial queries. The fine-tuned model demonstrated substantial improvements in accuracy, reducing errors across all query types. The most notable improvement was observed for a question (Q6) that asked the LLM to calculate the number of clinics within a 15-min driving time of a given street address. The As-Is model only achieved a 12% accuracy rate for this query, whereas the fine-tuned model reached 75% accuracy, an improvement factor of 6.25 times. Similarly, for less spatially complex tasks (Q2), such as counting clinics within a specific zip code, the fine-tuned model still exhibited a 1.24 times improvement, increasing accuracy from 76% to 94%.

Table 1. Virginia Comparative Analysis of LLM Query Execution: As-Is vs. Fine-Tuned Model.

	Correct (%)	Incorrect (%)	Execution Failure Rate (%)	Non-Executable Count	Average Tokens
Q1. How many clinics are in "X" County?					
As-Is (3.5 Turbo)	22%	78%	88%	69	775
As-Is (4o-mini)	67%	33%	91%	30	746
Fine-tuned	94%	6%	100%	6	169
Improvement	+27% points improved		80% reduction in code errors		77% reduced
Q2. How many clinics are in the area with the ZIP code "X"?					
As-Is (3.5 Turbo)	23%	77%	96%	74	764
As-Is (4o-mini)	76%	24%	100%	24	747
Fine-tuned	94%	6%	100%	6	156
Improvement	+18% points improved		75% reduction in code errors		79% reduced
Q3. What is the closest clinic to "X Address"?					
As-Is (3.5 Turbo)	5%	95%	94%	89	843
As-Is (4o-mini)	20%	80%	94%	75	909
Fine-tuned	78%	22%	59%	13	297
Improvement	+58% points improved		83% reduction in code errors		67% reduced
Q4. How many clinics are within 10 miles of the centroid of the "X" County?					
As-Is (3.5 Turbo)	28%	72%	90%	65	861
As-Is (4o-mini)	39%	61%	87%	53	807
Fine-tuned	98%	2%	0%	0	160
Improvement	+59% points improved		100% reduction in code errors		80% reduced
Q5. How many clinics are within 10 miles of the centroid of the area with the ZIP code "X"?					
As-Is (3.5 Turbo)	22%	78%	88%	69	823
As-Is (4o-mini)	29%	71%	39%	28	797
Fine-tuned	99%	1%	100%	1	235
Improvement	+70% points improved		96% reduction in code errors		71% reduced
Q6. How many clinics are within a 15-min driving time of "X Address"?					
As-Is (3.5 Turbo)	6%	94%	96%	90	887
As-Is (4o-mini)	12%	88%	99%	87	924
Fine-tuned	75%	25%	84%	21	283
Improvement	+63% points improved		76% reduction in code errors		69% reduced
Summary					
As-Is (3.5 Turbo)	17.7%	82.3%	92.3%	456	826
As-Is (4o-mini)	40.5%	59.5%	83.2%	297	822
Fine-tuned	89.7%	10.3%	75.8%	47	217
Improvement	+49.2% points improved		84.2% reduction in code errors		74% reduced

Note. The improvement column represents the improvement between As-Is (4o-mini) and the Fine-Tuned model. "Non-Executable Count" represents the number of incorrect responses where the generated code failed to run. "Execution Failure Rate" is calculated as the percentage of non-executable responses relative to the total number of incorrect responses. Execution Failure Rate (%) is calculated relative to incorrect responses. High values (e.g., 100%) occur when very few errors remain, meaning all residual errors were non-executable. For each question, the variable "X" (i.e., county, ZIP code, or address) was randomly selected from the Virginia datasets. Specifically, 100 counties, 100 ZIP codes, and 100 street addresses were randomly sampled at the start of the evaluation and used consistently across both model assessments.

Additionally, Table 1 reports the execution failure rate, representing the percentage of incorrect responses that resulted in non-executable code due to syntax, logic, or missing resource errors. Generated code that executed but produced incorrect results is not included in the execution failure rate column. The As-Is model failed to execute queries in up to 100% of incorrect cases, whereas the fine-tuned model significantly reduced execution failures for Q3, Q4, and Q5. Notably, Q5, which required spatial joins, had a 90% execution failure rate in the As-Is model, while the fine-tuned model resolved this issue entirely, achieving 0% execution errors. Overall, the fine-tuned model achieved an accuracy rate of 89.7%, whereas the As-Is model only achieved 40.5% accuracy, proving the necessity of domain-specific training for geospatial applications.

The external validation conducted with New York data, shown in Table 2, confirmed the potential generalizability of the fine-tuned model beyond the Virginia training region. For the simple spatial join query ("How many clinics are in X County?"), the fine-tuned model achieved 92% accuracy, significantly outperforming GPT-4o-mini As-Is (71%) and GPT-3.5-turbo As-Is (18%). Non-executable code counts dropped from 64 (3.5-turbo) and 25 (4o-mini) to just 5 (Fine-Tuned). Token usage was reduced by 77% compared to GPT-4o-mini, demonstrating both accuracy and computational efficiency gains. For the more complex isochrone-based query ("How many clinics are within a 15-min drive of X Address?"), the fine-tuned model achieved 73% accuracy, a major improvement over 10% (4o-mini) and 9% (3.5-turbo). Non-executable code counts fell from 85 (3.5-tubro) and 80 (4o-mini) to 23 (Fine-Tuned), and token usage decreased by 65%.

Table 2. New York Comparative Analysis of LLM Query Execution: As-Is vs. Fine-Tuned Model.

	Correct (%)	Incorrect (%)	Execution Failure Rate (%)	Non-Executable Count	Average Tokens
Q1. How many clinics are in "X" County?					
As-Is (3.5 Turbo)	18%	82%	78%	64	780
As-Is (4o-mini)	71%	29%	86%	25	756
Fine-tuned	92%	8%	63%	5	171
Improvement	+21% points improved		80% reduction in code errors		77% reduced
Q6. How many clinics are within a 15-min driving time of "X Address"?					
As-Is (3.5 Turbo)	9%	91%	93%	85	892
As-Is (4o-mini)	10%	90%	89%	80	843
Fine-tuned	73%	27%	85%	23	298
Improvement	+63% points improved		71% reduction in code errors		65% reduced

Additionally, Figure 3 illustrates the relationship between token count and spatial query complexity, where the number of computational steps increases token usage. In this study, computational steps are defined as distinct spatial operations, such as geolocation, spatial joins, buffering, distance calculations, etc., required to complete a given geospatial query. The fine-tuned model required an average of 217 tokens per query, compared to 822 tokens per query for the As-Is model. This represents a considerable reduction (74%) in token usage, significantly decreasing the cost of computation in large-scale applications. Specifically, the As-Is model required longer prompts with explicit shapefile references, variable names, and geospatial processing instructions to ensure correct responses, leading to excessive token consumption. In contrast, the fine-tuned model leveraged external function calls, minimizing redundant code generation while maintaining accuracy.

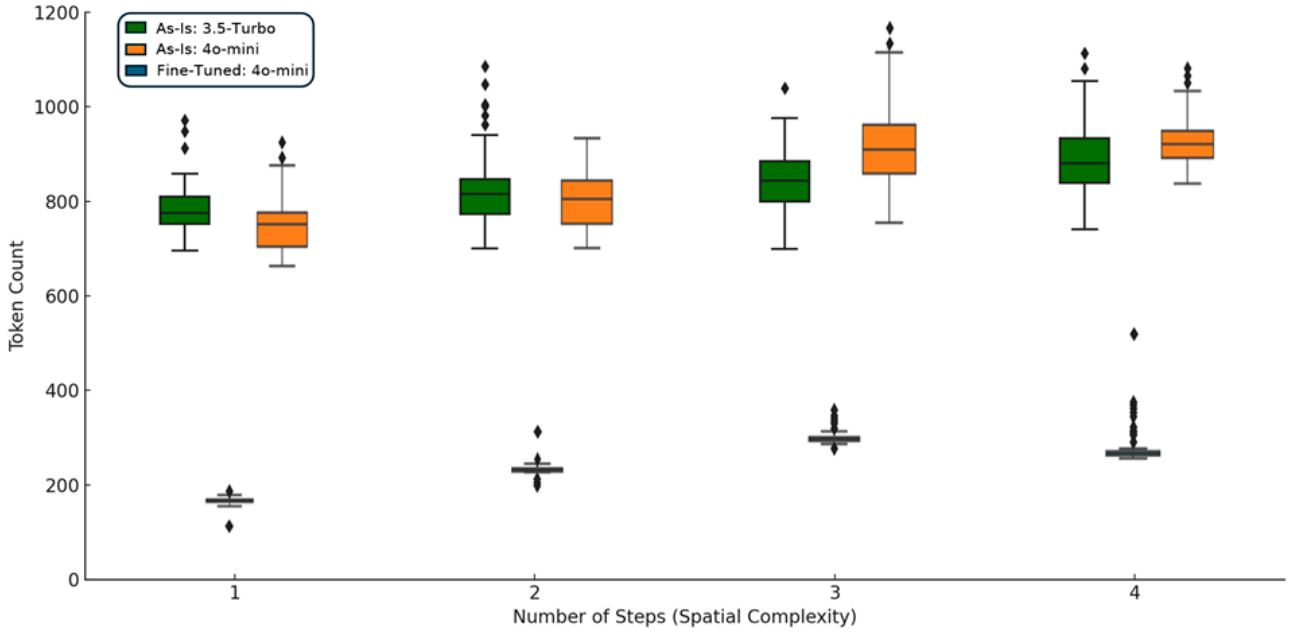

Figure 3. Comparison of token counts across increasing spatial query complexity (Steps 1–4). At each step, the Fine-Tuned model demonstrates considerably lower token usage and reduced variability, indicating greater efficiency and consistency in handling geospatial queries compared to the baseline As-Is model. The Fine-Tuned 4o-mini model's token counts are very low and consistent, so its boxplots appear compressed near the bottom of the y-axis.

Overall, these findings suggest that fine-tuning significantly enhances the model's ability to interpret and execute spatial queries. Furthermore, fine-tuning improved not only correctness but also computational efficiency, as indicated by a reduced token usage across all query types.

3.2. Analysis of Error Types of As-Is and Fine-Tuned Models

The analysis of coding errors reveals that the fine-tuned model substantially reduced errors, such as syntax errors, incorrect variable references, and coordinate reference system (CRS) issues, across all categories compared to the baseline As-Is model (Figure 4). The most frequent error in the As-Is model was syntax errors, which resulted in immediate execution failures due to missing parentheses, incorrect indentation, or improperly formatted function calls. These errors illustrate the baseline LLM's lack of structured syntax control, which frequently led to incomplete or malformed code snippets. Fine-tuning remarkably reduced syntax-related failures by refining the model's understanding of Python's syntax rules, ensuring that generated code adhered to proper structure and formatting. Errors related to file paths and missing resources also posed a frequent challenge in the baseline model, often arising when hallucinated file locations were referenced. The fine-tuned model eradicated these errors entirely.

Moreover, another prevalent issue in the As-Is model was undefined variables and function errors. These occurred when the model referenced nonexistent variables or attempted to call functions that were either incorrectly named or entirely hallucinated. This issue often led to execution failures, particularly in spatial queries requiring precise function calls. The fine-tuned model substantially mitigated this problem by learning to reference valid function names and predefined methods within the geospatial processing framework.

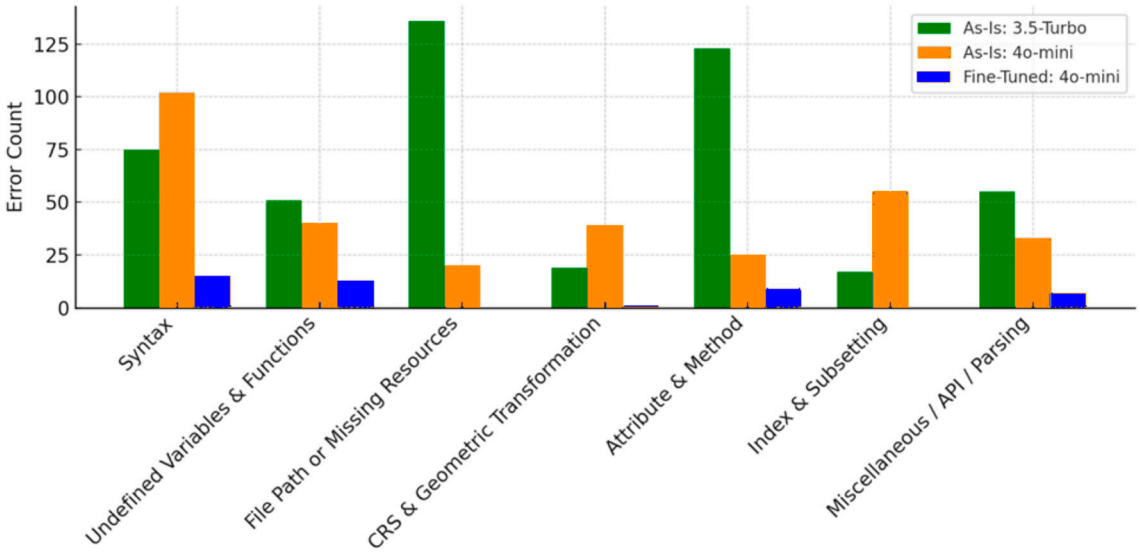

Figure 4. Distribution of coding errors across categories for As-Is and Fine-Tuned models. The fine-tuned model exhibits substantially fewer errors across all categories.

Errors involving coordinate reference systems (CRS) and geometric transformations were particularly disruptive in the As-Is model. These errors arose when the model failed to correctly assign or transform coordinate reference systems, producing failures such as "invalid projection error" and "cannot transform naive geometries". Such issues are critical in geospatial computations where accurate spatial alignment is essential. The fine-tuned model showed marked improvement in handling CRS transformations by correctly applying projection methods and ensuring appropriate geospatial references in generated queries.

It is important to note that execution accuracy, whether the generated code runs without error, is not equivalent to analytical validity. A code block may execute successfully but still produce incorrect spatial results due to semantic misunderstandings, incorrect spatial joins, or failures in geographic name resolution. While this study primarily measures execution accuracy improvements, we also report on analytical errors where outputs deviated from the correct spatial intent despite successful code execution.

To sum up, fine-tuning led to a substantial decrease in coding errors, improving the model's ability to execute geospatial queries correctly. These improvements were particularly pronounced in syntax handling, spatial data operations, and function referencing, reinforcing the effectiveness of fine-tuning for geospatial healthcare applications.

4. Discussions

Our results demonstrate substantial improvements achieved through fine-tuning a generative AI model for geospatial data queries. The fine-tuned GPT-4o-mini model exhibited higher accuracy and greater efficiency in spatial reasoning compared to the baseline As-Is model. By addressing key spatial tasks such as buffering, spatial joins, and travel time estimations, the fine-tuned model was able to produce ready-to-execute Python scripts while reducing computational costs (i.e., the number of tokens). Additionally, the fine-tuned GPT-4o-mini consistently outperformed unmodified GPT-3.5 across all tasks. The New York results further validated our approach, demonstrating similarly high accuracy and efficiency in a different geographic context. These improvements are consistent with previous research demonstrating that fine-tuning enhances LLMs' ability to handle domain-specific tasks [2,16].

A key takeaway from this study is the fine-tuned model's enhanced ability to process multi-step spatial queries, a capability that the As-Is model struggled with. For example, in response to the query "How many clinics are within a 15-min driving time of X Address?", the fine-tuned model improved accuracy by about 6.3 times, highlighting limitations in geographic and spatial reasoning capabilities of existing LLMs [19]. This aligns with prior research that has demonstrated fine-tuned LLMs significantly outperform general models in structured query generation [47–49]. In addition to increased accuracy, the fine-tuned model achieved substantial token efficiency, reducing average token usage by approximately 74% compared to the As-Is model. Token reduction was observed even for complex spatial queries, reinforcing that fine-tuning not only improves correctness but also enhances computational cost-efficiency.

The study also identified a marked reduction in error rates, particularly in syntax errors, attribute misapplications, CRS-related issues, and indexing errors. While the As-Is model frequently generated errors related to incorrect function calls, mis-referenced attributes, and missing geospatial transformations, the fine-tuned model effectively reduced these failures by refining its ability to reference geospatial attributes correctly and perform transformations accurately. These findings align with existing literature suggesting that LLM fine-tuning can mitigate hallucinations and improve structured reasoning in computational workflows [4]. However, despite these improvements, some persistent errors remained, particularly in handling advanced spatial operations such as chained geospatial computations and multi-step logic processing. This suggests that further refinement in fine-tuning strategies and the incorporation of reinforcement learning techniques could further improve the model's geospatial reasoning capabilities.

Overall, the results suggest our fine-tuning strategies were effective. The developed model demanded a domain-specific strategy grounded in spatial logic, data structure awareness, and computational efficiency. To that end, we developed a novel fine-tuning framework tailored specifically for spatial querying tasks, which we believe constitutes a foundational contribution to GIScience literature. Our approach included five key innovations: (1) a carefully curated dataset of natural language prompts paired with optimized Python code for solving real-world spatial queries; (2) purposeful variation in question phrasing to promote generalizability across query types and user styles; (3) integration of external function calls, which dramatically reduced token usage while maintaining code modularity and accuracy; (4) a fuzzy matching mechanism to auto-correct geographic names and reduce input errors; and (5) structured formatting protocols that ensured all generated code was executable, syntactically valid, and terminated consistently.

Importantly, the model is not trained to directly answer given geospatial problems; it is trained to generate executable Python code that solves them. This approach is not a limitation; it is a strength. By generating clean, domain-specific code rather than producing final outputs directly, the model improves transparency, ensures reproducibility, and allows users to interact with geospatial data through natural language. This approach dramatically increases accuracy while minimizing computational overhead, making it ideally suited for high-volume, real-time, or industry-scale applications where cost-efficiency and interpretability are paramount.

Specifically, our research moves beyond the simple general claim that "fine-tuning improves performance" by offering a replicable, purpose-built methodology for making LLMs spatially aware and providing a scalable blueprint for future applications. Thus, other studies that aim to develop LLMs that are capable of spatial querying can refer to our fine-tuning strategies that can lead to more accurate and efficient LLMs with spatial reasoning than as-is LLMs. For the application domains requiring efficient spatial analytics for decision-making, the integration of conversational AI powered by LLMs could lower

technical barriers for non-expert users while streamlining computational processes for geospatial professionals.

This study has several limitations that warrant further investigation. First, the model was fine-tuned primarily on data from Virginia, which may limit generalizability to regions with different geographic structures and naming conventions. While an external validation in New York suggests reasonable transferability, expanding the training data to include more diverse geographic areas would likely improve robustness, particularly in handling ambiguous names and boundary hierarchies.

The dataset consisted of 634 prompt–response pairs (over 70,000 tokens), covering six foundational spatial query types. Although effective, expanding both geographic diversity and query complexity, including multi-constraint queries, chained operations, and raster-based analyses, would improve generalization and semantic accuracy for broader applications. Similarly, while the current benchmark addresses operational coding tasks, it does not capture the full complexity of GIS workflows like spatiotemporal analysis, multi-layer joins, or conditional logic. Consistent with our results, Mooney et al. also found that while GPT-4 significantly outperformed GPT-3.5 in overall accuracy on GIS tasks, both models struggled with advanced GIS concepts such as spatial reasoning, mathematical computations, and complex workflows beyond basic mapping and definitions [9]. While complementary, future research should consider designing hybrid benchmarks that evaluate both conceptual GIS knowledge and operational code generation to provide a more holistic assessment of LLM spatial intelligence. Future work could build on our framework by incorporating such hybrid benchmarks to assess both conceptual reasoning and practical geospatial execution in tandem.

This study focused on OpenAI's GPT-4o-mini and compared it to GPT-3.5-Turbo. However, performance on other models and open-source models like Mistral or DeepSeek remains unknown [50,51]. Broader benchmarking across models, including alternative approaches like retrieval-augmented generation (RAG), prompt chaining, or tool-augmented systems (e.g., GeoAgent, ShapefileGPT), is a crucial next step to evaluate scalability, flexibility, and efficiency trade-offs [2,52]. While our focus was on evaluating fine-tuning in isolation, future research should extend this comparison by integrating empirical tests against these alternative architectures within a unified evaluation framework.

The current framework cannot handle conversational memory, restricting it to single-turn queries without context retention for iterative or chained spatial workflows. This limitation impacts integration into dashboards and decision-support tools. Additionally, reliance on commercial APIs like Mapbox and ArcGIS introduces operational costs; transitioning to open-source alternatives like OpenRouteService (ORS) and Nominatim would improve cost-efficiency and reproducibility [53,54].

Finally, the study did not formally assess output variability or whether semantically equivalent queries yield consistent outputs. Although prompt augmentation aimed to reduce this, variability remains an open question. Likewise, the validation framework focuses on execution correctness but does not detect semantic errors under ambiguity or flag anomalous outputs. Future research should develop trust, uncertainty quantification, and validation mechanisms to support safe deployment in real-world geospatial applications.

Despite the limitations, our study provides important implications for the potential of integrating geospatial data analysis and visualization with LLMs, especially to enhance the usability of online geospatial data interactive dashboards. Recently, there has been growing interest in the use of online dashboards as accessible tools for communicating complex spatial information, particularly in public health, urban planning, and environmental monitoring [55–57]. However, while dashboards such as those developed during the COVID-19 pandemic [58] have proven effective at visualizing data and informing policy decisions,

they remain limited in interactivity. Most dashboards require users to manually navigate filters, dropdowns, and map layers, posing challenges for non-technical users [44,59].

Our study shows that fine-tuned large language models (LLMs), when integrated with geospatial dashboards, can offer a transformative solution by enabling users to interact through natural language, asking questions like "Which counties have the fewest clinics within a 15-min drive?". LLMs can act as conversational interfaces that simplify data exploration. Unlike off-the-shelf LLMs, which frequently struggle with geospatial logic and generate incorrect or non-executable code, the fine-tuned model consistently returns valid Python code tailored to spatial analysis. Importantly, these responses can be programmed not only to return textual summaries (e.g., number of clinics, access gaps, travel times), but also to dynamically visualize the results directly on the dashboard map. This dual output, visual and textual, has the potential to enhance both usability and interpretability, helping users understand where spatial patterns occur and why they matter. By embedding our fine-tuned LLM framework into dashboards, the platform evolves from a passive visualization tool into an intelligent, user-driven decision support system, unlocking new possibilities for real-time and accessible geospatial analysis.

5. Conclusions

This study investigated how generative AI can be integrated with geospatial analysis to enhance spatial querying via a fine-tuned GPT-4o-mini model. The results highlight the potential of fine-tuning to improve accuracy, reduce computational costs, and overcome technical barriers, enabling broader access to geospatial analytical tools for various stakeholders. Key findings demonstrate the fine-tuned model's superiority over the baseline "As-Is" model, achieving higher accuracy rates across diverse spatial queries and demonstrating significant token efficiency improvements. Our study identified several effective strategies for improving LLM accuracy and efficiency in spatial tasks. These include (1) training on a curated dataset of geospatial prompts and Python completions, (2) rephrasing each question to increase generalizability and reduce overfitting, (3) enforcing structured outputs using forced stop tags to eliminate redundant generated code, (4) integrating fuzzy location matching to improve spatial input handling, and (5) using modular function calls to reduce token usage and improve code reliability. These strategies not only improved performance but also allowed non-technical users to interact with complex geospatial data in ways that were previously inaccessible, enabling natural language queries to generate accurate spatial outputs dynamically. To realize the full potential of generative AI in spatial analysis, future efforts should prioritize the development of more robust models capable of handling complex geospatial tasks and adapting to diverse geographic contexts that can contribute to a smart decision-making process.

Author Contributions: Conceptualization, Zachary Sherman, Mengxi Zhang, and Junghwan Kim; Methodology, Zachary Sherman, Mengxi Zhang, and Junghwan Kim; Formal Analysis, Zachary Sherman, Mengxi Zhang, and Junghwan Kim; Investigation, Zachary Sherman, Mengxi Zhang, and Junghwan Kim; Writing—Original Draft Preparation, Zachary Sherman, Sandesh Sharma Dulal, Jin-Hee Cho, Mengxi Zhang, and Junghwan Kim; Writing—Review & Editing, Zachary Sherman, Sandesh Sharma Dulal, Jin-Hee Cho, Mengxi Zhang, and Junghwan Kim. Zachary Sherman, Sandesh Sharma Dulal, Jin-Hee Cho, Mengxi Zhang, and Junghwan Kim; Supervision, Mengxi Zhang and Junghwan Kim; Funding Acquisition, Mengxi Zhang and Junghwan Kim. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Institute for Society, Culture and Environment (ISCE) at Virginia Tech and OpenAI's Researcher Access Program.

Data Availability Statement: The original data presented in this study are available from publicly accessible sources, including the U.S. Census Bureau and VGIN.

Acknowledgments: The authors thank the five anonymous reviewers for their thoughtful comments, which helped improve the article considerably.

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

Appendix A

Appendix A.1. Prompt Format for As-Is Model Evaluation

Below is the standardized prompt format used to evaluate the As-Is model. Each question was inserted into the {question} placeholder. Contextual information about the shapefile variables was provided to ensure the model had adequate knowledge of the geospatial data structure.

“““

Generate me the Python code to solve the question: {question}
Use the three shapefiles stored in the variables “counties”, “zip_code”, and “healthcare”.
The variable “counties”, is a polygon shapefile containing information about counties in the U.S.
Important Column names include. . .
. . .
Follow these additional instructions:
(1) Do not include ‘”python or wrap code as string. This code should be able to be directly executed.
(2) When the code is complete, please add the line “#End of Code” at the bottom of the generated code.
. . .
(6) Do not include comments in the code.
(7) When asked about an address, reference “from geopy.geocoders import ArcGIS”.
(8) Make sure the result is visible as a print statement.
(9) Do not use “op”, instead use “predicate” when generating code for spatial functions.
“““

Appendix A.2. Sample Prompt-Completion Pairs Used for Fine-Tuning

Below are selected examples of natural language prompts and corresponding Python code completions used in the fine-tuning dataset. Each completion was designed to reflect proper geospatial processing techniques using Python libraries such as GeoPandas and aligned with the real data structures provided during training.

Example 1: Driving-Time Accessibility Query Using an Address
Prompt:
How many clinics are accessible within 30 min of 1500 Maplewood Dr, Richmond, VA?
Completion:

```
address = “1500 Maplewood Dr, Richmond, VA”
coordinates = get_coordinates(address)
if coordinates:
    mapbox_token = mapbox_api
    minutes = 30
    isochrone_geojson = get_isochrone(coordinates, mapbox_token, minutes=minutes)
    if isochrone_geojson:
        clinics_within_isochrone = find_clinics_within_isochrone(isochrone_geojson, healthcare)
        if len(clinics_within_isochrone) == 0:
```

```
    print(f"Sadly, no clinics are accessible within {minutes} minutes of {address}.")
  else:
    print(f"There are {len(clinics_within_isochrone)} clinics accessible within {minutes}
minutes of {address}.")
else:
  print(f"The address '{address}' could not be found. Please try again.")
#End of Code
```

Example 2: County-Based Query with Fuzzy Name Matching
Prompt:
How many clinics are located in Fairfax County?
Completion:

```
input_name = "Fairfax County"
corrected_name = correct_name(input_name, true_county_names)
place = counties[counties["NAMELSAD"] == corrected_name]
county_boundary = place.geometry.union_all()
clinics = healthcare[healthcare.within(county_boundary)]
if len(clinics) == 0:
  print(f"Currently, there are no clinics available in {corrected_name}.")
else:
  print(f"Good news! {corrected_name} has {len(clinics)} clinics available.")
#End of Code
```

Appendix A.3. Mathematical Formalization of Geospatial Operations

This appendix presents formal mathematical descriptions of the core spatial operations that underpin the queries solved by the fine-tuned LLM in this study.

Buffer and Proximity

$$B(c,r) = \left\{ x \epsilon R^2 \middle| d(c,x) \leq r \right\}$$

$$d(c,x) = \sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2}$$

$$C = \{ p_i \epsilon P | d(c,p_i) \leq r \}$$

The buffer operation identifies all point features located within a specified straight-line distance from a reference point. In this equation, c represents the center point, such as the centroid of a county or ZIP code, with coordinates $(c_1,\ c_2)$ and r represents the buffer radius (e.g., 10 miles). The function $d(c,x)$ calculates the Euclidean distance between the center point c and any other point x in the two-dimensional space R^2. The set $B(c,r)$ defines the area that includes all points located within the distance r from point c. The set C specifically includes all points p_i from the dataset P (e.g., clinics) that satisfy the condition of being within the buffer distance r from the center point.

Driving Time Isochrones

$$I(c,t) = \left\{ x \epsilon R^2 \middle| T(c,x) \leq t \right\}$$

$$C = \{ p_i \epsilon P | p_i \epsilon I(c,t) \}$$

The driving-time isochrone operation identifies all point features that are reachable within a specified travel time along the road network from a given reference point. In this formulation, c represents the origin point, typically a user-defined address, and t represents the driving time threshold in minutes (for example, 15 min). The function $T(c,x)$ calculates

the driving time between the origin point c and any other location x in the spatial domain R^2, based on the structure of the road network. The isochrone area $I(c,t)$ represents the set of all points that can be reached from point c within the specified driving time t. The set C consists of all point features p_i in the dataset P (e.g., clinic locations) that are located within the boundaries of the isochrone $I(c,t)$, meaning those clinics are accessible within the specified travel time.

References

1. Jiang, H.; Li, M.; Witte, P.; Geertman, S.; Pan, H. Urban Chatter: Exploring the potential of ChatGPT-like and generative AI in enhancing planning support. *Cities* **2025**, *158*, 105701. [CrossRef]
2. Zhang, Y.; Wei, C.; He, Z.; Yu, W. GeoGPT: An assistant for understanding and processing geospatial tasks. *Int. J. Appl. Earth Obs. Geoinf.* **2024**, *131*, 103976. [CrossRef]
3. Ullah, A.; Qi, G.; Hussain, S.; Ullah, I.; Ali, Z. The Role of LLMs in Sustainable Smart Cities: Applications, Challenges, and Future Directions. *arXiv* **2024**, arXiv:2402.14596. [CrossRef]
4. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
5. Fang, C.; Miao, N.; Srivastav, S.; Liu, J.; Zhang, R.; Fang, R.; Tsang, R.; Nazari, N.; Wang, H.; Homayoun, H. Large Language Models for Code Analysis: Do {LLMs} Really Do Their Job? In Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24), Philadelphia, PA, USA, 14–16 August 2024; pp. 829–846.
6. Akinboyewa, T.; Li, Z.; Ning, H.; Lessani, M.N. GIS copilot: Towards an autonomous GIS agent for spatial analysis. *arXiv* **2024**, arXiv:2411.03205. [CrossRef]
7. Zhang, Y.; Wang, Z.; He, Z.; Li, J.; Mai, G.; Lin, J.; Wei, C.; Yu, W. BB-GeoGPT: A framework for learning a large language model for geographic information science. *Inf. Process. Manag.* **2024**, *61*, 103808. [CrossRef]
8. Hochmair, H.H.; Juhász, L.; Kemp, T. Correctness Comparison of ChatGPT-4, Gemini, Claude-3, and Copilot for Spatial Tasks. *Trans. GIS* **2024**, *28*, 2219–2231. [CrossRef]
9. Mooney, P.; Cui, W.; Guan, B.; Juhász, L. Towards Understanding the Geospatial Skills of ChatGPT: Taking a Geographic Information Systems (GIS) Exam. In Proceedings of the 6th ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discover, Hamburg, Germany, 13 November 2023.
10. Mahmoudi, H.; Camboim, S.; Brovelli, M.A. Development of a Voice Virtual Assistant for the Geospatial Data Visualization Application on the Web. *ISPRS Int. J. Geo-Inf.* **2023**, *12*, 441. [CrossRef]
11. Morocho, V.; Achig, R.; Bustamante, J.; Mendieta, F. Virtual Assistants to Bring Geospatial Information Closer to a Smart Citizen. In Proceedings of the 2022 IEEE Sixth Ecuador Technical Chapters Meeting (ETCM), Quito, Ecuador, 11–14 October 2022; pp. 1–6.
12. Wang, S.; Tao, H.; Huang, X.; Yun, L.; Ce, Z.; Huan, N.; Rui, Z.; Zhenlong, L.; Ye, X. GPT, Large Language Models (LLMs) and Generative Artificial Intelligence (GAI) Models in Geospatial Science: A Systematic Review. *Int. J. Digit. Earth* **2024**, *17*, 2353122. [CrossRef]
13. Kim, J.; Lee, J.; Jang, K.M.; Lourentzou, I. Exploring the limitations in how ChatGPT introduces environmental justice issues in the United States: A case study of 3108 counties. *Telemat. Inform.* **2024**, *86*, 102085. [CrossRef]
14. Akinboyewa, T.; Ning, H.; Lessani, M.N.; Li, Z. Automated floodwater depth estimation using large multimodal model for rapid flood mapping. *Comput. Urban Sci.* **2024**, *4*, 12. [CrossRef]
15. Jang, K.M.; Kim, J. Multimodal Large Language Models as Built Environment Auditing Tools. *Prof. Geogr.* **2025**, *77*, 84–90. [CrossRef]
16. 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. [CrossRef]
17. Mansourian, A.; Oucheikh, R. ChatGeoAI: Enabling Geospatial Analysis for Public through Natural Language, with Large Language Models. *ISPRS Int. J. Geo Inf.* **2024**, *13*, 348. [CrossRef]
18. Ning, H.; Zhenlong, L.; Temitope, A.; Lessani, M.N. An autonomous GIS agent framework for geospatial data retrieval. *Int. J. Digit. Earth* **2025**, *18*, 2458688. [CrossRef]
19. Renshaw, A.; Lourentzou, I.; Lee, J.; Crawford, T.; Kim, J. Comparing the Spatial Querying Capacity of Large Language Models: OpenAI's ChatGPT and Google's Gemini Pro. *Prof. Geogr.* **2025**, *77*, 186–198. [CrossRef]
20. Zhang, Y.; He, Z.; Li, J.; Lin, J.; Guan, Q.; Yu, W. MapGPT: An Autonomous Framework for Mapping by Integrating Large Language Model and Cartographic Tools. *Cartogr. Geogr. Inf. Sci.* **2024**, *51*, 717–743. [CrossRef]
21. Zhang, M.; He, J.; Lei, S.; Yue, M.; Wang, L.; Lu, C.-T. Can LLM Find the Green Circle? Investigation and Human-Guided Tool Manipulation for Compositional Generalization. In Proceedings of the ICASSP 2024—2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Seoul, Republic of Korea, 14–19 April 2024; pp. 11996–12000.

22. Tao, R.; Xu, J. Mapping with ChatGPT. *ISPRS Int. J. Geo Inf.* **2023**, *12*, 284. [CrossRef]
23. Li, Z.; Ning, H. Autonomous GIS: The next-generation AI-powered GIS. *Int. J. Digit. Earth* **2023**, *16*, 4668–4686. [CrossRef]
24. Wei, C.; Yifan, Z.; Xinru, Z.; Ziyi, Z.; Zhiyun, W.; Jianfeng, L.; Qingfeng, G.; Yu, W. GeoTool-GPT: A Trainable Method for Facilitating Large Language Models to Master GIS Tools. *Int. J. Geogr. Inf. Sci.* **2025**, *39*, 707–731. [CrossRef]
25. Rahman, F.; Oliver, R.; Buehler, R.; Lee, J.; Crawford, T.; Kim, J. Impacts of Point of Interest (POI) Data Selection on 15-Minute City (15-MC) Accessibility Scores and Inequality Assessments. *Transp. Res. Part A Policy Pract.* **2025**, *195*, 104429. [CrossRef]
26. Kim, J.; Karki, S.; Brickhouse, T.; Vujicic, M.; Nasseh, K.; Wang, C.; Zhang, M. Navigating Disparities in Dental Health-A Transit-Based Investigation of Access to Dental Care in Virginia. *Community Dent. Oral Epidemiol.* **2025**, *53*, 117–124. [CrossRef]
27. Yiannakoulias, N. Spatial intelligence and contextual relevance in AI-driven health information retrieval. *Appl. Geogr.* **2024**, *171*, 103392. [CrossRef]
28. Ong, J.C.L.; Seng, B.J.J.; Law, J.Z.F.; Low, L.L.; Kwa, A.L.H.; Giacomini, K.M.; Ting, D.S.W. Artificial intelligence, ChatGPT, and other large language models for social determinants of health: Current state and future directions. *Cell Rep. Med.* **2024**, *5*, 101356. [CrossRef]
29. OpenAI. Advancing Cost-Efficient Intelligence. Available online: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ (accessed on 2 July 2025).
30. OpenAI. Fine-Tuning Guide. Available online: https://platform.openai.com/docs/guides/fine-tuning (accessed on 2 July 2025).
31. Gramacki, P.; Martins, B.; Szymański, P. Evaluation of Code LLMs on Geospatial Code Generation. In Proceedings of the Proceedings of the 7th ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery, Atlanta, GA, USA, 29 October–1 November 2024; pp. 54–62.
32. Hou, S.; Shen, Z.; Zhao, A.; Liang, J.; Gui, Z.; Guan, X.; Li, R.; Wu, H. GeoCode-GPT: A Large Language Model for Geospatial Code Generation Tasks. *arXiv* **2024**, arXiv:2410.17031. [CrossRef]
33. Hou, S.; Jiao, H.; Shen, Z.; Liang, J.; Zhao, A.; Zhang, X.; Wang, J.; Wu, H. Chain-of-Programming (CoP): Empowering Large Language Models for Geospatial Code Generation. *arXiv* **2024**, arXiv:2411.10753. [CrossRef]
34. Ikumapayi, N. Automated Front-End Code Generation Using OpenAI: Empowering Web Development Efficiency. *SSRN Electron. J.* **2023**, 4590704. [CrossRef]
35. Gupta, D.V.; Ishaqui, A.S.A.; Kadiyala, D.K. Geode: A Zero-shot Geospatial Question-Answering Agent with Explicit Reasoning and Precise Spatio-Temporal Retrieval. *arXiv* **2024**, arXiv:2407.11014.
36. Dai, H.; Li, Y.; Liu, Z.; Zhao, L.; Wu, Z.; Song, S.; Shen, Y.; Zhu, D.; Li, X.; Li, S. Ad-Autogpt: An Autonomous Gpt for Alzheimer's Disease Infodemiology. *arXiv* **2023**, arXiv:2306.10095. [CrossRef]
37. Xu, L.; Zhao, S.; Lin, Q.; Chen, L.; Luo, Q.; Wu, S.; Ye, X.; Feng, H.; Du, Z. Evaluating Large Language Models on Spatial Tasks: A Multi-Task Benchmarking Study. *arXiv* **2024**, arXiv:2408.14438. [CrossRef]
38. Gupta, S.; Nandwani, Y.; Yehudai, A.; Mishra, M.; Pandey, G.; Raghu, D.; Joshi, S. Selective Self-Rehearsal: A Fine-Tuning Approach to Improve Generalization in Large Language Models. *arXiv* **2024**, arXiv:2409.04787.
39. Plewe, B.; Dibiase, D.; Demers, M.; Johnson, A.; Kemp, K.; Wentz, E. *Geographic Information Science & Technology Body of Knowledge*; University Consortium for Geographic Information Science: Chesapeake, VA, USA, 2006.
40. Engine, A.G. Nearest Neighbors. Available online: https://developers.arcgis.com/geoanalytics/tools/nearest-neighbors (accessed on 2 July 2025).
41. Masis, T.; O'Connor, B. Where on earth do users say they are?: Geo-entity linking for noisy multilingual user input. *arXiv* **2024**, arXiv:2404.18784. [CrossRef]
42. Bell, S.; Marlow, T.; Wombacher, K.; Hitt, A.; Parikh, N.; Zsom, A.; Frickel, S. Automated Data Extraction from Historical City Directories: The Rise and Fall of Mid-Century Gas Stations in Providence, RI. *PLoS ONE* **2020**, *15*, e0220219. [CrossRef] [PubMed]
43. Martynov, N.; Baushenko, M.; Kozlova, A.; Kolomeytseva, K.; Abramov, A.; Fenogenova, A. A methodology for generative spelling correction via natural spelling errors emulation across multiple domains and languages. *arXiv* **2023**, arXiv:2308.09435. [CrossRef]
44. Shaw, N. Geographical Information Systems and Health: Current State and Future Directions. *Healthc. Inform. Res.* **2012**, *18*, 88–96. [CrossRef]
45. Fuzzywuzzy: Fuzzy String Matching in Python. Available online: https://github.com/seatgeek/fuzzywuzzy (accessed on 2 July 2025).
46. Lee, S.; Jang, S.; Jang, S.; Lee, D.; Yu, H. Exploring Language Model's Code Generation Ability with Auxiliary Functions. *arXiv* **2024**, arXiv:2403.10575. [CrossRef]
47. Han, Y.; Liu, J.; Luo, A.; Wang, Y.; Bao, S. Fine-Tuning LLM-Assisted Chinese Disaster Geospatial Intelligence Extraction and Case Studies. *ISPRS Int. J. Geo Inf.* **2025**, *14*, 79. [CrossRef]
48. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. Bert: Pre-Training of Deep Bidirectional Transformers for Language UNDERSTANDING. In Proceedings of the 2019 conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis, MN, USA, 2–7 June 2019; Volume 1, pp. 4171–4186.

49. Manvi, R.; Khanna, S.; Mai, G.; Burke, M.; Lobell, D.; Ermon, S. Geollm: Extracting geospatial knowledge from large language models. *arXiv* **2023**, arXiv:2310.06213.
50. Frontier AI LLMs, Assistants, Agents, Services | Mistral AI. Available online: https://mistral.ai/ (accessed on 2 July 2025).
51. Deepseek. Available online: https://deep-seek.chat (accessed on 2 July 2025).
52. Lin, Q.; Hu, R.; Li, H.; Wu, S.; Li, Y.; Fang, K.; Feng, H.; Du, Z.; Xu, L. ShapefileGPT: A Multi-Agent Large Language Model Framework for Automated Shapefile Processing. *arXiv* **2024**, arXiv:2410.12376.
53. Openrouteservice. Available online: https://openrouteservice.org/ (accessed on 2 July 2025).
54. Nominatim. Available online: https://nominatim.org/ (accessed on 2 July 2025).
55. Jing, C.; Du, M.; Li, S.; Liu, S. Geospatial Dashboards for Monitoring Smart City Performance. *Sustainability* **2019**, *11*, 5648. [CrossRef]
56. Praharaj, S.; Wentz, E. Building Community Resilience Through Geospatial Information Dashboards. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2022**, *XLVIII-4/W5-2022*, 151–157. [CrossRef]
57. Lwin, K.K.; Sekimoto, Y.; Takeuchi, W.; Zettsu, K. City Geospatial Dashboard: IoT and Big Data Analytics for Geospatial Solutions Provider in Disaster Management. In Proceedings of the 2019 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM), Paris, France, 18–20 December 2019; pp. 1–4.
58. Dong, Y.; Jiang, X.; Jin, Z.; Li, G. Self-Collaboration Code Generation via ChatGPT. *ACM Trans. Softw. Eng. Methodol.* **2024**, *33*, 1–38. [CrossRef]
59. Bernhäuserová, V.; Krajňáková, L.; Hátlová, K.; Hanus, M. The Limits of GIS Implementation in Education: A Systematic Review. *ISPRS Int. J. Geo Inf.* **2022**, *11*, 592. [CrossRef]