

Article

LLM-Based Geospatial Assistant for WebGIS Public Service Applications

Gabriel Ionut Dorobantu  and Ana Cornelia Badea * 

Faculty of Geodesy, Technical University of Civil Engineering, 124 Lacul Tei Boulevard,
020396 Bucharest, Romania; gabriel-ionut.dorobantu@phd.utcb.ro

* Correspondence: ana.badea@utcb.ro

Abstract

The automation of public services represents a key area of development at the national level, with the main goal of facilitating citizens' access to comprehensive, integrated and high-quality services in the shortest possible time. National strategies emphasize the need to integrate open geospatial data and artificial intelligence into information, transparency and decision-making processes. The evolution of artificial intelligence, particularly large language models (LLMs), has led to the development of virtual assistants capable of understanding user requirements and providing answers in natural, easy-to-understand language. This paper presents directions for the development and use of large-language-model-based virtual assistants, focusing on their ability to understand and interact with the geospatial domain through an LLM API. Geospatial modeling contributes significantly to the automation of public services, but access to this technology is often limited by technical expertise or dedicated software programs. The development of AI-based virtual assistants removes these barriers, facilitating access, reducing time and ensuring transparency and accuracy of information. The proposed approach is implemented using a commercial large language model API, integrated with domain-specific geospatial functions and authoritative spatial databases. This study highlights practical examples of virtual assistants capable of understanding the geospatial field and contributing to the optimization and automation of public services in the country. In addition, the paper presents comparative analyses, challenges encountered and potential directions for future research.

Keywords: WebGIS; geospatial; artificial intelligence; LLM; GeoLLM

1. Introduction

In recent years, large language models (LLMs) have emerged as key technologies for natural language processing, enabling advances in text summarization, content generation, translation, and conversational assistance [1]. Their ability to process unstructured text at scale makes them attractive for a wide range of domains. One of the fields where their potential is increasingly recognized is geospatial science, where the integration of natural language with spatial data can facilitate more intuitive data access, analysis, and decision-making [2]. Geospatial problems are often complex and interdisciplinary, requiring the integration of heterogeneous data sources, from satellite imagery and climate records to urban infrastructure datasets. LLMs can play a key role in lowering barriers of access by enabling interaction through natural language. For example, geospatial assistants can help non-expert users query spatial databases ("Which areas in this city are at high risk of flooding given the last 10 years of precipitation data?") or assist urban planners in



Academic Editor: Juraj Gregaň

Received: 13 December 2025

Revised: 26 January 2026

Accepted: 27 January 2026

Published: 9 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\) license](https://creativecommons.org/licenses/by/4.0/).

scenario exploration by translating planning requirements into geospatial queries. Similarly, in disaster management, LLM-powered assistants can synthesize real-time reports, satellite imagery metadata, and sensor real-time data to support rapid decision-making during events such as wildfires, earthquakes, or floods. In 2013, ref. [3] pointed out that identifying and using the appropriate geospatial tools is often challenging without specific training in the field, reflecting the complexity of GIS systems at that time. The authors argued for approaches that could bring spatial thinking closer to users through simple, question-driven interactions. Although significant progress has been made since then, recent research indicates that usability challenges for non-expert users persist, particularly for complex spatial analyses and result interpretation. A systematic review by ref. [4] highlights that non-expert users still encounter substantial difficulties when performing advanced spatial tasks. In this context, large language models have emerged as a promising solution, acting as intermediaries between end users and the geospatial domain by interpreting natural language queries and translating them into meaningful spatial operations. For example, ref. [5] emphasizes that LLMs can reduce the need for specialized training while still supporting complex geospatial workflows.

Recent developments illustrate this transformation. LLMs have been explored for generating code that automates geoprocessing tasks [6], where the study reports that natural language instructions can be converted into executable GIS scripts capable of performing standard spatial analyses, although they often require expert validation. In question-driven digital cartography [7], LLMs enable the semi-automatic generation of thematic maps and visualizations from textual prompts, lowering the barrier to cartographic design. In the context of hazard and disaster management [8], LLM-based approaches have been shown to assist in synthesizing heterogeneous spatial data and scenario descriptions to support rapid situational assessment. Similarly, in urban planning applications [9], LLMs facilitate the interpretation of planning requirements and exploratory scenario analysis, improving accessibility in early planning stages while still relying on human experts for final decision-making. These applications highlight the potential of LLMs not merely as language models but as assistants that bridge human reasoning with geospatial computation.

However, a fundamental limitation of these models is that a basic LLM model does not have direct access to verified and updated geospatial databases. In the geospatial domain, where accuracy and timeliness are essential, for example, in environmental monitoring, natural hazard analysis, or resource management, this lack of connectivity significantly reduces the practical utility of LLMs when used “as is” [10]. Moreover, foundation models are typically designed for general-purpose use, aiming to cover as many domains as possible but without a specific focus on geospatial reasoning. This raises a key question for research: whether fundamental models are geospatially aware.

Several recent studies have begun to explore this issue, examining both the potential and the limitations of foundation models in handling geospatial tasks. Benchmarking efforts show that while LLMs demonstrate strong performance in geographic knowledge retrieval, GIS concepts, and code interpretation, their performance degrades for tasks requiring map generation, spatial reasoning, and complex geospatial coding [11]. Comparative evaluations across multiple models further reveal substantial variability in correctness and consistency, with even state-of-the-art models exhibiting persistent weaknesses in spatial reasoning and mapping tasks [12]. More targeted evaluations indicate that these limitations are particularly evident in geospatial code generation, where general-purpose LLMs frequently produce incomplete workflows, incorrect operators, or hallucinated functions [13]. Approaches that incorporate structured geospatial knowledge have shown measurable improvements: for example, Geo-FuB demonstrates that combining LLMs with an operator–function knowledge base and retrieval mechanisms can significantly

improve geospatial code generation accuracy and reduce hallucinations compared to baseline models [14]. Beyond generic geospatial tasks, recent domain-specific study further confirms these trends. In transportation and urban planning contexts, LLMs exhibit solid baseline geospatial and domain knowledge and can support analytical workflows and decision support, but their effectiveness remains strongly dependent on task complexity and model scale, with larger or fine-tuned models consistently outperforming smaller general-purpose models [15]. Collectively, these findings suggest that while LLMs hold substantial promise as geospatial assistants, their reliable deployment in complex spatial analysis requires systematic evaluation, domain adaptation, and tight integration with geospatial knowledge and computational tools.

Despite the rapid progress of large language models and their increasing adoption in geospatial applications, a clear research gap remains regarding their effective integration into real-world, operational geospatial systems. Existing studies predominantly focus on code generation, isolated geospatial tasks, or offline experimentation, without addressing the need for end-to-end task execution supported by authoritative, expert-maintained geospatial tools and up-to-date spatial databases. As a result, current approaches rarely demonstrate how LLMs can be reliably deployed in practical public service environments, where accuracy, data validity, and operational robustness are essential. This gap highlights the lack of integrated frameworks that combine natural language interaction, trusted geospatial data sources, and executable geospatial workflows within a unified, real-world system.

To address these gaps, this paper presents a case study that investigates the suitability of large language models (LLMs) for supporting domain-specific geospatial tasks within public institutions. The overall objective is to facilitate and optimize geospatial modeling for end users by employing LLMs as intermediaries between natural language queries and specialized geospatial tools, thereby reducing domain complexity and dependency on dedicated GIS software.

To achieve this objective, we propose an approach in which geospatial tools, developed by domain experts with controlled inputs and outputs, are connected to authoritative, well-maintained, and real-time databases. Within this framework, three LLM configurations are compared:

- Foundation model (gpt-4.1-mini-2025-04-14), used without any external augmentation.
- Same foundation model with function calling, connected to external geospatial tools.
- Fine-tuned foundation model with function calling, connected to external geospatial tools, trained on geospatial datasets for improved domain adaptation. This fine-tuning does not involve architectural modifications of the model but relies on curated instruction–response datasets designed by the authors to improve domain adaptation and function selection behavior.

This work makes three main contributions. First, it introduces a WebGIS-based architecture that enables natural language interaction with authoritative geospatial databases for public service applications, eliminating the need for dedicated GIS software and advanced technical expertise to access and execute geospatial workflows. Second, it provides a comparative analysis of three LLM configurations, base, function-calling, and fine-tuned, evaluating their suitability for real-world geospatial tasks. Third, it presents an empirical assessment of system performance, robustness, and usability using realistic public administration scenarios.

This case study lays the groundwork for the development of geospatial agents capable of performing modeling tasks across a wide range of public service applications, highlighting both current limitations and directions for future research.

The remainder of this paper is organized as follows. Section 2 presents the conceptual foundations of this study. Section 3 describes the proposed WebGIS architecture and implementation. Section 4 reports the experimental evaluation and comparative results. Section 5 discusses the findings and their implications. Section 6 addresses the main limitations and future research directions. Finally, Section 7 concludes this paper by summarizing the key contributions.

2. Conceptual Foundations

2.1. LLM Limitations

Recent progress in large language models (LLMs) such as GPT-4 [16,17] and Gemini [18], has created new opportunities for developing GIS assistants. Researchers have begun experimenting with LLMs to design autonomous GIS agents capable of retrieving and processing geospatial data [19]. Beyond data retrieval, several studies have demonstrated the integration of LLMs into specialized spatial tasks. For example, ref. [7] investigated how ChatGPT could be applied in cartography, focusing on the automation and improvement of map-making. Their experiments involved generating thematic and mental maps from natural language prompts, where ChatGPT produced Python code to drive geospatial visualizations. Other work has looked more broadly at the potential of foundation models for GeoAI. Several practical tools have also been developed. Ref. [20] introduces an LLM-based QGIS plugin, though at present its role is limited to providing step-by-step guidance rather than automating operations. In contrast, ref. [21] aims to automate geospatial workflows by generating PyQGIS scripts or building graphical models directly. While useful, its automation is constrained, as users must still manually configure parameters within the generated code. Most recently, ref. [6] presented ChatGeoAI, which fine-tunes the Llama-2 model to produce executable code for carrying out spatial analysis tasks.

While these developments represent important progress, they only partially address the broader challenge. Most of the existing approaches focus primarily on code generation, rather than delivering final geospatial results. In practice, even if the generated code were executed within dedicated GIS.

software, this would not by itself guarantee the quality, accuracy, or reliability of the outputs. The gap between code generation and validated geospatial solutions highlights the need for approaches that integrate LLMs more deeply with domain-specific tools and trusted data sources, ensuring that results are both operationally useful and scientifically robust.

2.2. Function Calling

Function calling is a capability in large language models that allows them to interact with external functions, APIs, or software tools in a structured manner [22]. Instead of providing free-form text as output, the model can produce a structured request (often in JSON format) describing the function to be executed and its arguments. The external system then processes the request and returns results, which the LLM incorporates into its final response. This mechanism transforms the LLM from a purely generative system into a hybrid reasoning agent that can both interpret natural language and execute precise computational tasks [23].

Technically, the process involves four steps:

1. Parsing the user query—the LLM interprets the natural language input and maps it to a predefined schema.

2. Generating a structured call—instead of producing free text, the model outputs a JSON object containing the function name and arguments (e.g., coordinates, layer names, buffer distance).
3. Execution by external system—the JSON is passed to an external API or software module, which executes the requested computation (e.g., a spatial join or buffer operation).
4. Integration of results—the response from the external system is returned to the LLM, which integrates the structured output into a final, human-readable answer.

Figure 1 illustrates the architecture of the function-calling workflow, explicitly mapping the four stages described in this section. The diagram shows how natural language queries are interpreted by the LLM, translated into structured function calls, executed by external geospatial services, and reintegrated into user-facing responses.

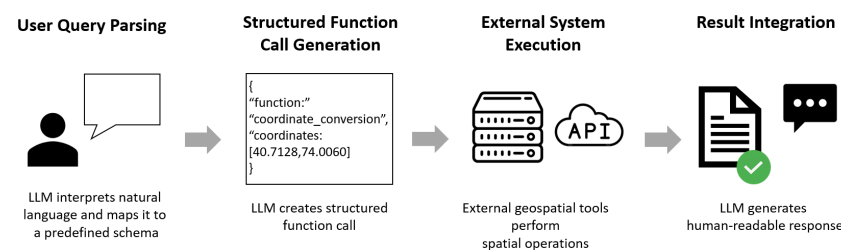


Figure 1. Architecture of the function-calling workflow illustrating the four stages of natural-language-driven geospatial task execution.

The integration of function calling provides several key advantages for extending the capabilities of LLMs. It enables direct access to real-time, verified information from external databases, ensuring that the results are not limited to static training data [24]. By connecting to domain-specific tools such as GIS platforms or spatial analysis libraries, models can perform specialized operations that exceed their intrinsic reasoning abilities. Moreover, the use of predefined input–output schemas constrains the interaction, reducing hallucinations and improving reproducibility [25]. Finally, function calling enables hybrid reasoning, where the LLM’s natural language interpretation is combined with precise external computation, leading to more accurate and contextually relevant results. The study by ref. [26] shows that this integration allows models to access up-to-date information, perform deterministic calculations, and interact with domain-specific code or services, resulting in more accurate and contextually grounded outputs.

2.3. Fine-Tuning

According to ref. [27], fine-tuning refers to the process of adapting a pre-trained foundation model to perform well on a specific task or domain by continuing its training on a smaller, specialized dataset. This is a form of transfer learning: rather than training a model from scratch (which requires massive data and computational resources), fine-tuning leverages the broad, general knowledge already learned during pre-training, focusing subsequent learning on domain-specific examples. Importantly, fine-tuning can be implemented in different ways: full fine-tuning (updating all model weights) or parameter-efficient techniques (PEFT), which freeze much of the network and only train certain parts. These PEFT methods greatly reduce the computational cost and memory requirements while helping avoid catastrophic forgetting (where the model loses general skills learned during pre-training). The general workflow of fine-tuning includes data preparation (curating, cleaning, and formatting task-specific datasets), model selection, training with hyperparameter optimization, and validation.

Supervised fine-tuning (SFT) involves training a pre-trained model on a smaller, labeled dataset designed for a downstream task. During SFT, the model's parameters are updated using supervised learning, aligning the LLM's general language representations with domain-specific objectives [28,29]. Instruction tuning is a specialized fine-tuning paradigm, a subset of SFT, in which a pre-trained LLM is adapted using large collections of instruction–response pairs, enabling the model to generalize across a wide range of tasks by learning how to follow natural language instructions. A recent survey by ref. [30] and an empirical study by ref. [28] show that this paradigm improves a model's ability to follow diverse instructions and generalize across tasks, particularly in domain-specific settings. In geospatial contexts, instruction-tuned models have been shown to effectively learn structured tool-use behaviors, enabling them to interpret complex task descriptions and generate multi-step tool-use chains, significantly outperforming both untuned and general-purpose models in task accuracy and robustness [31]. In this setup, the instruction prompt adapted for function calling encodes the user's intent (e.g., “Buffer the river layer by 500 m and return intersecting flood zones”), while the expected output represents the correct system response (e.g., a JSON call to a GIS function or a code snippet). During training, the model learns to generalize from heterogeneous instruction patterns and align them with structured outputs. From a technical perspective, instruction prompts serve as task descriptors: they transform a single dataset into a multi-task environment, where the model learns to infer not only what the task is but also how it should be solved. Figure 2 represents an example of an instruction–response pair used for fine-tuning an LLM adapted to geospatial function calling. The user query in natural language specifies a coordinate transformation from Stereo 70 to WGS84. The assistant response encodes the request as a structured function call, including the function name, arguments, and metadata describing the tool. Such examples illustrate how natural language instructions can be aligned with domain-specific geospatial functions during fine-tuning. An illustrative example of an instruction–response pair used during fine-tuning is provided in Appendix A (Figure A1).

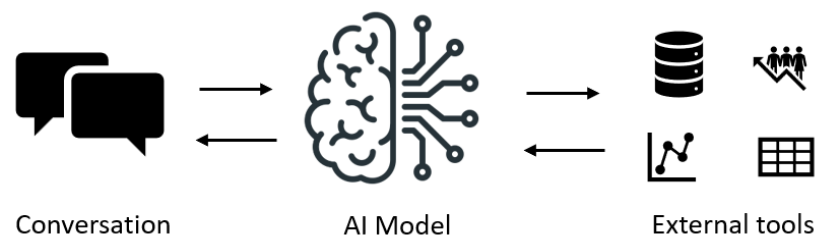


Figure 2. Function calling mechanism.

In this study, the term fine-tuning is used in its strict machine learning sense and refers to supervised fine-tuning that updates the internal parameters (weights) of the pre-trained large language model via a commercial API. Although no architectural modifications are performed, the model undergoes additional gradient-based training on domain-specific instruction–response pairs, resulting in persistent changes to its parameters. Therefore, the proposed fine-tuned model is fundamentally different from approaches based solely on prompt engineering, retrieval augmentation, or runtime tool orchestration.

The proposed approach shares certain conceptual similarities with Retrieval-Augmented Generation (RAG), as both architectures aim to enhance large language models by incorporating external, domain-specific information at inference time. In both cases, the LLM acts as a reasoning layer that integrates external knowledge to produce grounded responses.

However, there are fundamental differences between the two paradigms. In standard RAG frameworks, external information is retrieved as unstructured or semi-structured text and injected into the model's context window without modifying the model's parameters.

The LLM remains unchanged, and all domain adaptation occurs dynamically at inference time. In contrast, the proposed approach combines parameter-level adaptation through supervised fine-tuning with structured function calling over authoritative geospatial services. Fine-tuning modifies the model's internal weights to improve domain-specific behavior, such as function selection, parameter extraction, and clarification strategies. Additionally, instead of retrieving textual documents, the system invokes deterministic geospatial functions that execute validated computations on authoritative databases.

As a result, while RAG primarily augments knowledge access, the proposed framework emphasizes behavioral adaptation and executable task orchestration, enabling reliable end-to-end geospatial workflows rather than purely text-based generation.

2.4. Fine-Tuning Effects

Accordingly, the effects discussed in this section reflect the behavioral changes observed after API-level fine-tuning of a pre-trained model rather than structural changes to the model itself.

One critical effect of fine-tuning is the risk of performance degradation and the loss of generalization ability. When a model is over-specialized on a narrow set of training examples, it may adapt too strongly to those patterns and fail to handle variations or unseen tasks, a phenomenon often referred to as overfitting. Recent studies have shown that fine-tuning large language models can introduce risks related to performance degradation and loss of generalization. A comprehensive review of LLM fine-tuning highlights that adapting models to narrow or domain-specific datasets may compromise robustness and stability if broader capabilities acquired during pre-training are not carefully preserved [29]. These concerns are further supported by empirical evidence demonstrating catastrophic forgetting during continual fine-tuning, where models exhibit measurable declines in performance on previously learned or unseen tasks due to over-specialization. Together, these findings confirm that overfitting and reduced generalization are well-documented risks of fine-tuning, particularly when training data or objectives are insufficiently diverse [32]. In such cases, the fine-tuned model achieves high accuracy on the training distribution but performs poorly when confronted with new formulations of the same task. This risk is particularly significant in the geospatial domain, where users may express queries in highly diverse ways. Over-specialization can also lead to catastrophic forgetting, where the model loses some of its general capabilities acquired during pre-training, thereby reducing its versatility. The authors in article [33] used a training dataset of approximately 35,000 instructions to fine-tune the LLaMA2-7B model, along with 750 questions for evaluation, reporting promising results. However, the effectiveness of fine-tuning is not strictly determined by the number of instructions but rather by the intended objectives, the size of the model, and the way in which it was pre-trained. A small number of instructions may have little to no impact on large-scale models with billions of parameters, while an excessively large instruction set can lead to overfitting, where the model memorizes patterns instead of generalizing. Balancing dataset size with model capacity therefore remains a key challenge in designing effective fine-tuning strategies.

3. Methodology

3.1. WebGIS Application

Recent studies have proposed agent-based frameworks that integrate large language models with GIS tools to automate specific geospatial tasks. GeoGPT employs an LLM-driven agent that interprets user requests and sequentially invokes predefined GIS tools to autonomously construct geoprocessing workflows, primarily targeting professional GIS users and workflow efficiency [34]. In contrast, GeoTool-GPT focuses on instruction

tuning and domain-specific fine-tuning to enhance the intrinsic tool-use capabilities of open-source LLMs, embedding knowledge of GIS tools directly into model parameters rather than relying on runtime orchestration alone [35]. GIS Copilot advances this line of research by deeply embedding an LLM-based agent within an established GIS platform (QGIS), enabling autonomous generation and execution of spatial analysis workflows while emphasizing transparency and user interaction within desktop GIS environments [36]. Complementarily, the autonomous GIS agent framework for geospatial data retrieval concentrates on a narrower but critical task, automatic discovery and retrieval of geospatial datasets, using an LLM as a decision core to generate, debug, and execute data-fetching programs [19]. Compared to these approaches, our methodology emphasizes the integration of a virtual assistant within a WebGIS environment, focusing on end-to-end task execution over authoritative geospatial services and interactive, task-oriented workflows, thereby addressing practical deployment and usability in real-world web-based geospatial applications. The application architecture connects three main components: the user interface that contains a web map and the chat box, the LLM with function-calling capability, and a set of geospatial functions linked to authoritative databases. Users interact with the system through natural language queries, which are interpreted by the assistant and translated into function calls. These functions are executed on the server side using verified spatial datasets, and the results are returned in both textual and visual form within the WebGIS interface.

The scenario reflects the needs of public institutions, where citizens or decision-makers may not have advanced GIS expertise and software but require timely access to accurate spatial analyses. For instance, a user can issue the following request: “Show details for my land parcel with identifier X”. The assistant parses the request, generates the necessary sequence of geospatial operations, executes them through the integrated geoprocessing functions, and presents the result as an interactive map and a descriptive report.

This approach demonstrates how a WebGIS system augmented with an AI assistant can reduce complexity, improve accessibility, and support transparency in public services. By bridging natural language interaction with specialized geospatial operations, the assistant eliminates the need for dedicated software skills, thereby extending the use of geospatial modeling to a wider range of stakeholders. Figure 3 presents an example of a user query submitted through the conversational interface of the virtual assistant, which requests details about a land parcel identified by cadastral number 27521. The system returns both a textual response and a visual representation on the map. The parcel geometry and the associated textual attributes are generated through geoprocessing tools executed in the cloud on authoritative datasets, thus providing trustworthy and verifiable results, in contrast to purely generative textual outputs typically produced by LLMs.

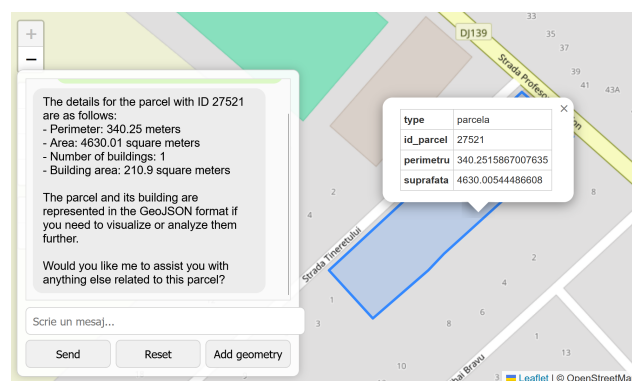


Figure 3. Example of request from WebGIS Application.

3.2. OpenAI API

The integration of the geospatial assistant into the WebGIS application was achieved through the use of the OpenAI Assistants API. This type provided by OpenAI is a powerful framework for building advanced conversational agents that go beyond simple text exchanges. Unlike traditional chat completions, the Assistants API introduces persistent structures such as threads (conversation history), messages (user and assistant inputs), and runs (execution instances of an assistant). This architecture allows developers to create assistants with memory, context management, and extensibility through external tools or custom functions (function calling). For example, an assistant can not only answer questions but also call APIs, process data, or return structured outputs such as JSON files. By combining natural language understanding with tool integration, the Assistants API enables the development of robust, interactive applications tailored to complex workflows and real-world tasks. The creation of an assistant can be carried out either programmatically, using the Python programming language together with the official OpenAI library [37] that integrates this API, or directly from the graphical interface provided by OpenAI. It is important to note that fine-tuning within commercial LLM APIs is subject to strict model availability constraints. Users cannot arbitrarily fine-tune any large language model but must select from a predefined set of models explicitly supported by the API provider. In practice, only specific model families include fine-tuning capabilities, while others are restricted to inference-only usage. The initialization process involves defining a set of essential attributes: the assistant's name, an initial prompt that guides its general behavior, the language model to be used, and, optionally, a list of functions that the assistant can call. For this scenario, gpt-4.1-mini-2025-04-14 using Python API is used as a base model. Details regarding the initialization of the geospatial assistant using the Assistants API are presented in Appendix A (Figure A2).

3.3. Function Integration

Below are detailed all the Python functions that simulate a cloud computing environment and to which the geospatial assistant has access. These functions can be further extended as needed and adapted to any geospatial scenario, ensuring flexibility and scalability in their application.

- `convert_wgs84_to_stereo70(latitude, longitude)`—converts geographic coordinates from WGS84 to the Romanian national projection (Stereo 70). The function takes latitude and longitude as inputs, applies the datum and projection transformation, and returns the projected planar coordinates x and y rounded to three decimals. The Stereo 70 system is a stereographic cartographic projection defined on the Krasovsky 1940 reference ellipsoid and the Pulkovo 1942 datum, specifically designed to minimize geometric distortions across the entire territory of Romania and adopted as the national standard reference system. It constitutes the official framework for geodetic, topographic, and cadastral activities carried out at the national level. To reduce and eliminate errors introduced during the transformation of coordinates between different reference systems, official spatial corrections are applied. These corrections are derived from gravimetric and altimetric measurements conducted across the national territory. The implemented function integrates these authorized and periodically updated corrections, thereby ensuring results characterized by a high level of accuracy and consistency, grounded in verified geodetic data and fully compliant with the current national standards. The function was developed by the authors and serves as a simulated cloud computing environment for geospatial processing.
- `convert_stereo70_to_wgs84(x, y)`—converts projected coordinates from the Stereo 70 system into the WGS84 geographic reference system. The function takes x and y

values as inputs and returns the results formatted in degrees, minutes, and seconds (DMS). The function was developed by the authors and serves as a simulated cloud computing environment for geospatial processing.

- `return_parcel(parcel_id)`—retrieves a land parcel and its associated buildings from local geospatial datasets. The function selects the parcel matching the given `parcel_id` and identifies all buildings intersecting that parcel. Both the parcel and building geometries are returned. The function was developed by the authors to simulate an API for retrieving official geospatial data.
- `land_occupancy_percentage(parcel_id)`—calculates the land occupancy percentage of a parcel, defined as the ratio between the built-up area (sum of building footprints intersecting the parcel) and the total area of the parcel. It returns the occupancy percentage as a numeric value rounded to two decimals. The function was developed by the authors to process data retrieved from a simulated official API.
- `details_parcel(parcel_id)`—extracts detailed information about a land parcel and its intersecting buildings. The function selects the parcel by ID and computes its perimeter and surface area. It also identifies all buildings located within the parcel, calculates their individual surface areas, and counts the total number of buildings. The output is a structured dictionary containing parcel ID, perimeter, area, number of buildings, building surface areas, and an associated GeoJSON. The function was developed by the authors to process data retrieved from a simulated official API.
- `search_sentinel(start_date, end_date, bbox_coords)`—searches for Sentinel-2 L2A satellite images within a specified temporal interval and geographic extent. The function connects to the Sentinel Hub Catalog API, defines a bounding box, and queries the image collection. It returns a list of dictionaries containing acquisition date and time, cloud cover percentage, and links to the image resources. The function was developed by the authors, while the input data are obtained from the official Sentinel API.
- `get_sentinel_ndvi(bbox, acc_date)`—retrieves Normalized Difference Vegetation Index (NDVI) data from Sentinel-2 imagery for a specified bounding box and acquisition date. The function was developed by the authors, while the input data are obtained from the official Sentinel API.
- `download_osm_to_geojson(query)`—sends an Overpass QL query to the OpenStreetMap Overpass API and returns the results in GeoJSON format. The function was developed by the authors to integrate the official OpenStreetMap Overpass API.

To further clarify the structure, relationships, and responsibilities of the geospatial functions, Figure 4 presents a UML component diagram of the proposed integration.

In order to execute external geoprocessing tools, the assistant must be aware of their existence, including their names and descriptions, which are used to guide the assistant in selecting the appropriate function based on the user's input. Additionally, the assistant needs to know how these functions are called, what input parameters are required, as well as the type and utility of each function. To enable a structured and standardized way of providing this information, OpenAI offers a generic JSON schema for defining functions, which ensures consistency and facilitates accurate function calling within the Assistants API. For example, Figure 5 presents the case of a geospatial function that converts WGS84 coordinates into Stereo 70 projection, while the JSON schema defines the function's name, explains its role, and formally describes the latitude and longitude inputs. Such a design ensures consistency, interoperability, and precise execution of external tools within conversational workflows.

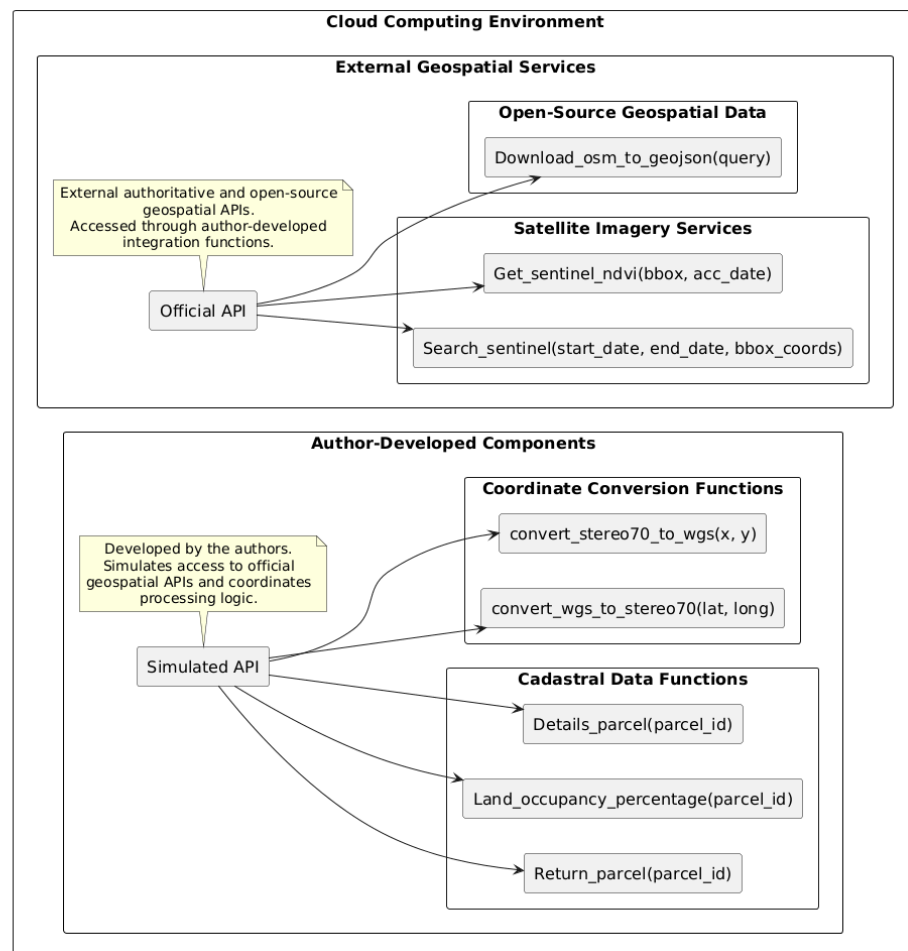


Figure 4. UML component diagram illustrating the structure, relationships, and responsibilities of the geospatial functions and their integration with geospatial APIs.

```

{
  "name": "convert_wgs84_to_stereo70",
  "description": "Converts WGS84 geographic coordinates (latitude and longitude) to Stereo 70 coordinates (x and y).",
  "strict": true,
  "parameters": {
    "type": "object",
    "required": [
      "latitude",
      "longitude"
    ],
    "properties": {
      "latitude": {
        "type": "number",
        "description": "Latitude in WGS84 format to be converted."
      },
      "longitude": {
        "type": "number",
        "description": "Longitude in WGS84 format to be converted."
      }
    }
  },
  "additionalProperties": false
}

```

Figure 5. JSON schema definition for external function.

The assistant is now initialized and equipped with a unique identifier, and the JSON schemas of the external functions have been created and transmitted to it, while the cloud computing environment is simulated through Python functions. What is further required is a core component that connects the WebGIS application, where user queries are received, to the geospatial assistant and the cloud computing functions. In this approach, the Assistant's API is integrated with a backend handler that executes geospatial tools whenever the assistant requests them. When a user submits a message, it is stored in a thread, and a

run is initiated for the assistant. If the assistant determines that a function call is necessary, the run transitions into the “requires_action” state, at which point the system inspects the requested tool and its parameters based on the JSON schema definition. The backend subsequently routes the request to the corresponding Python function (simulated cloud computing environment), for instance, calling `convert_wgs84_to_stereo70` when the user provides latitude and longitude coordinates. The function processes the input and returns the result (such as projected coordinates or a GeoJSON structure), which the assistant then incorporates into its natural language response.

A typical user interaction might look like this: “Convert latitude 45.76 and longitude 21.23 into Stereo 70 coordinates”. When a user submits this query, the assistant first interprets whether the request requires an external function. If so, the API responds not with a textual message but with a structured tool call that specifies the function name and its input parameters extracted by the model from the query in JSON format, and the run status will change to “requires_actions”.

The backend core subsequently extracts the function call and executes it within the cloud computing environment. The output function is then transmitted to the assistant, which awaits the response through the `submit_tool_outputs` method. Once the result is received, the assistant integrates it into the thread and generates the next reply. Figure 6 illustrates the complete operational workflow of the system, starting from the user’s input, followed by the semantic interpretation of the message by the LLM, the decision to invoke an external function together with the corresponding parameters, the execution of this function within the cloud computing environment, and finally the delivery of the complete response back to the user.

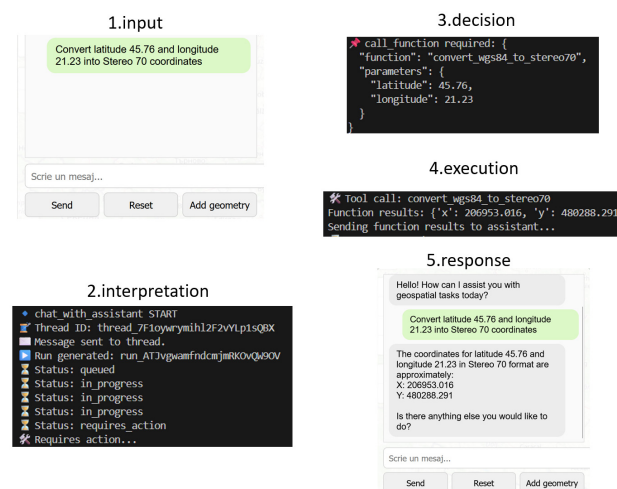


Figure 6. Assistant API response adapted with function result.

In the final implementation, the WebGIS application will provide only the natural conversation between the user and the assistant. The underlying architecture remains hidden from the user, yet it offers access to powerful tools capable of solving geospatial problems without the need for specialized software or prior technical expertise.

Figure 7 below illustrates an example from the application interface in which a user submits a natural language message, highlighted in light green, and the assistant responds within the same interface with a natural language message, displayed in gray.

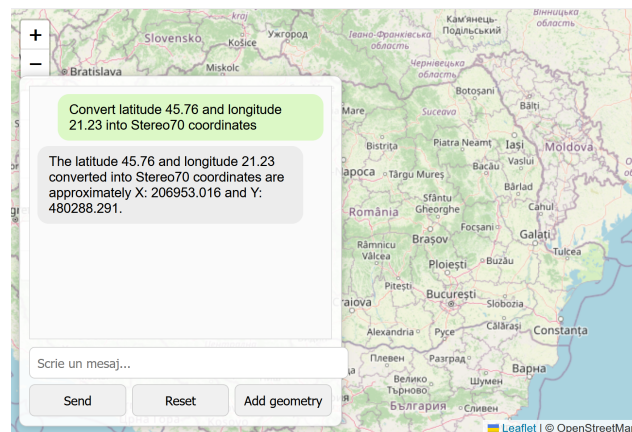


Figure 7. WebGIS application illustrating the example through natural conversation.

Figure 8 illustrates the workflow of the complete application designed to address this scenario. The user submits a request through the WebGIS application, and the message is received by the LLM model, which determines whether a function call is required. If a function is identified, the model forwards to the backend core the function name along with the parameters extracted from the conversation. The backend core then performs the actual call to the external function (e.g., Sentinel API, OSM Overpass API, or geospatial tools connected to official databases). These external functions are executed in the cloud (maintained by third-party providers), and their responses are returned to the backend, which redirects them to the LLM awaiting the function's results. Once the results are received, the LLM integrates them into natural language and transmits the response to the user's application. Through this workflow, the user interacts solely with the WebGIS application, without needing to be aware of the underlying operations. Consequently, the user is not constrained by prior knowledge or additional software requirements, and they simultaneously benefit from an assistant capable of delivering reliable results in natural language.

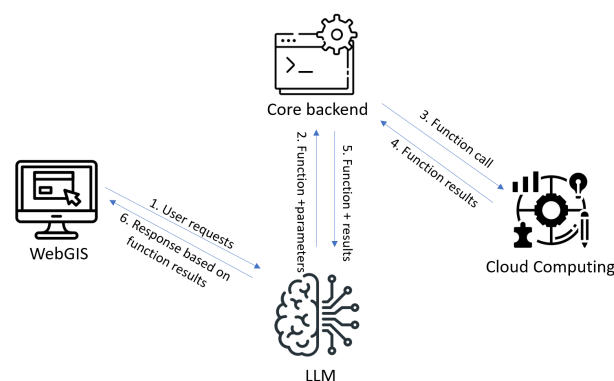


Figure 8. Application workflow.

3.4. Model Training

The next stage of this case study consisted of fine-tuning the base model gpt-4.1-mini-2025-04-14 using a dataset of approximately 100 question–answer pairs for each scenario, resulting in a total of 800 training pairs. The primary objective of fine-tuning was to improve the model's ability to correctly identify the geospatial function to be invoked, accurately extract the necessary parameters from the user's message, and adapt its responses to an appropriate tone while handling diverse user interaction scenarios. The fine-tuning dataset was constructed to cover a diverse range of geospatial interaction scenarios relevant to public service applications. The training data reflect variations in

query formulation, parameter completeness, and task complexity. To reduce bias toward specific prompt formulations, multiple linguistic variants were generated for each task type, ensuring diversity in phrasing, structure, and parameter expression. In addition, the dataset includes intentionally unclear or ambiguous prompts, for which the assistant was trained to request precise clarifications before function extraction. This strategy aims to reduce the generation of partially complete responses that could lead to ambiguity or incorrect geospatial function execution.

A separate validation set of 100 instruction–response pairs, disjoint from the training data, was used to monitor model performance during fine-tuning. Validation examples were selected to include both typical and challenging input patterns, such as incomplete parameters or alternative formatting, in order to assess the model’s ability to generalize beyond the training distribution. Model performance was evaluated by monitoring training loss and validation loss throughout the fine-tuning process.

All training and validation pairs were generated with the support of a larger LLM (GPT-5), ensuring linguistic diversity and consistency in the dataset. The fine-tuning was performed through the OpenAI web interface with the following hyperparameters: epochs = 1 (the model was exposed to the full dataset only once to minimize memorization and reduce the risk of overfitting), batch size = 4, learning rate multiplier = 1. Fine-tuning was performed for a single epoch. This stopping criterion was chosen deliberately to limit over-specialization and reduce the risk of overfitting given the relatively small size of the domain-specific dataset compared to the scale of the underlying foundation model. Training was terminated after one full pass over the dataset, as further epochs did not provide additional performance gains while increasing the risk of memorization.

The metrics obtained during fine-tuning indicate a stable and efficient learning curve. At the beginning of training, the training loss was relatively high (2.7–3.0), suggesting that the model was making many errors in reproducing the data. After the first 10–20 steps, the value began to decrease rapidly, dropping below the threshold of 1.0 and gradually reaching extremely low levels (<0.01) in the final stages. In parallel, both the validation loss and the full validation loss decreased consistently: for example, at step 10, the full validation loss was 1.874, whereas by the end it had dropped to 0.015. This convergence between training and validation indicates that the model not only memorized the training data but also succeeded in generalizing well to the validation set.

An important aspect is , although the training loss nearly reached zero (indicating complete memorization of the training data), the validation loss remained very low and stable, which demonstrates the absence of strong overfitting. The differences between training and validation loss did not increase significantly; in fact, at certain stages, the validation loss was lower than the training loss, suggesting that some validation subsets were easier for the model.

Overall, these metrics suggest that the fine-tuning process was effective: the model rapidly learned the underlying structure and patterns of the data, achieved near-perfect performance on the training set, and maintained the same level of accuracy on the entire validation set. However, further testing on unseen data is recommended, especially since the training dataset is considerably smaller relative to the model’s size.

Figure 9 shows the evolution of training and validation loss throughout the fine-tuning process. Training loss decreases rapidly during the initial steps and stabilizes near zero, while validation loss follows a similar trend with limited fluctuations. This behavior indicates effective convergence and suggests that the model generalizes well to unseen examples without exhibiting signs of overfitting. The observed loss patterns support the decision to limit fine-tuning to a single epoch, balancing domain adaptation and generalization.

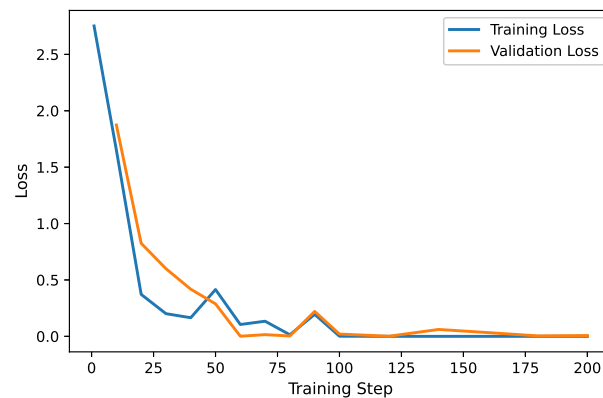


Figure 9. Training and validation loss during the fine-tuning process.

3.5. System Performance Evaluation

The performance of the proposed WebGIS-LLM architecture was evaluated using quantitative, time-based metrics designed to capture end-to-end responsiveness and to distinguish between conversational orchestration overhead and geospatial processing costs. The evaluation focuses on response times relevant to operational decision-support scenarios. Performance measurements were conducted using a set of ten representative geospatial queries, including coordinate conversion, parcel information retrieval, satellite imagery search, and parcel-based satellite imagery search requiring two-step function invocation. All queries were executed as independent requests, ensuring that no conversational context was preserved between successive runs. For each query, the following metrics were recorded:

- LLM orchestration time (*llm_time*), representing the duration required for natural language interpretation, decision-making, tool invocation planning, and response generation. This metric includes asynchronous run execution and periodic polling required by the LLM API.
- Tool execution time (*cloud_time*), corresponding to the cumulative execution time of invoked backend functions. Depending on the task, these functions may be executed locally (to simulate authoritative services) or via external production APIs. To ensure reproducibility, all queries were executed in isolated conversation threads, preventing any influence of prior conversational context on system behavior or performance.

The evaluated tool functions fall into two distinct categories. Coordinate transformation and parcel detail retrieval are executed locally and serve to simulate interactions with authoritative spatial data services. In contrast, Sentinel imagery search relies on a production external API and reflects real-world cloud service latency. This distinction is important for correctly interpreting the measured tool execution times.

Table 1 summarizes the average LLM orchestration time and tool execution time for the evaluated query categories.

Table 1. Average LLM orchestration time and tool execution time for different query types.

Query Type	LLM Orchestration Time (s)	Tool Execution Time (s)
Coordinate transformation	8.34	0.0021
Parcel details	7.62	0.0822
Sentinel imagery search	11.62	1.9853
Parcel-based Sentinel search	18.61	0.6586

The results indicate that LLM orchestration time is the dominant component of overall latency across all query types. For lightweight tasks such as coordinate transformation,

tool execution time is negligible due to local execution, and total response time is primarily influenced by asynchronous LLM run management. In particular, the relatively high LLM orchestration time reflects the execution model of the LLM API, which relies on asynchronous runs and periodic polling to monitor execution status. This orchestration process includes provider-side scheduling, run initialization, intermediate decision-making, and response finalization rather than pure inference latency. Parcel detail queries introduce a small increase in tool execution time, reflecting local data access, while overall latency remains dominated by LLM orchestration overhead. For Sentinel imagery search, tool execution time increases noticeably due to interaction with a production external API. Nevertheless, LLM orchestration remains a significant contributor to total latency. The parcel-based Sentinel search scenario exhibits the highest LLM orchestration time, reflecting increased conversational complexity and multi-step function chaining required to derive spatial extents before querying external services.

In the context of this study, real-time performance is defined as the ability to deliver validated geospatial results within a time frame suitable for operational decision-making. Given the distributed nature of cloud-based services and the asynchronous execution model of LLM APIs, the proposed system operates in a near-real-time mode rather than under strict, hard real-time constraints. Observed response times, ranging from several seconds for simple queries to under twenty seconds for more complex multi-step workflows, are compatible with decision-support use cases in public administration, where accuracy, transparency, and integration of authoritative data sources are prioritized.

It is important to clarify that the proposed system does not provide hard real-time guarantees in the strict systems-theoretic sense. Since inference and orchestration are performed on remote servers via API calls, overall response time is inherently influenced by network latency, request queuing, shared compute scheduling, and provider-side execution policies. These factors introduce variability that cannot be deterministically bounded by the application. Consequently, the term “real-time” in this study refers to soft real-time (near-real-time) performance, where responses are delivered within statistically predictable time ranges that are suitable for operational decision support rather than within strict worst-case latency bounds. The reported response times therefore reflect empirical latency distributions observed under realistic workloads rather than guaranteed upper limits.

These findings confirm that the proposed architecture is well suited for near-real-time geospatial decision support while remaining scalable as additional cloud-based services and more complex workflows are integrated.

4. Comparative Results

4.1. Need for Upgrade

At this stage, the WebGIS application is capable of responding to user queries through the virtual assistant, which relies on three approaches: the base LLM model gpt-4.1-mini-2025-04-14, the same base LLM model with integrated access to external functions (function calling), or the fine-tuned LLM model gpt-4.1-mini-2025-04-14 with access to external functions. To highlight the limitations identified in using the base LLM model, the following section presents examples of responses received within the WebGIS application.

As illustrated in Figure 10, the base LLM model is unable to provide concrete answers for coordinate transformations. Instead, it only suggests a code snippet that could be executed for coordinate conversion. Similarly, in cases where parcel data or satellite imagery are requested, the model responds by stating that it does not have access to such databases and merely recommends the steps to follow in order to obtain this information. Ultimately, the model’s output remains purely textual and does not contain reliable information that addresses the user’s actual requirement. Based on this observation, the need emerged for

a geospatial assistant capable of delivering informed responses, with access to external databases, and of solving more complex tasks.

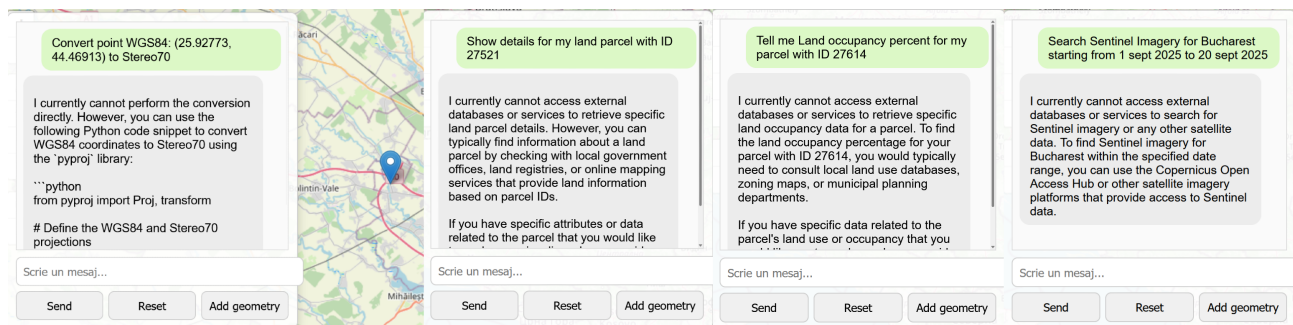


Figure 10. User query and assistants' response for gpt-4-1-mini-2025-04-14.

Figure 11 presents the same prompt examples submitted through the WebGIS application, but in this case the LLM model was replaced with the base model integrated with function calling. It can be observed that for the coordinate conversion of a point extracted from the map, real values are returned, calculated through an external third-party function maintained by specialists, which incorporates the necessary corrections for such transformations. Requests related to parcel identification now correctly recognize the spatial geometry of the parcel and display it on the map, along with accurate details (area, perimeter) computed using geospatial tools based on official cadastral data. Similarly, queries for satellite imagery now return the availability of satellite images within the requested time interval and bounding box, including cloud coverage, acquisition time, and download links for the retrieved images. In this case, the official Sentinel catalog is consulted via API. Consequently, the improvements are clearly noticeable, as the new assistant is capable of delivering accurate responses grounded in updated and reliable information.

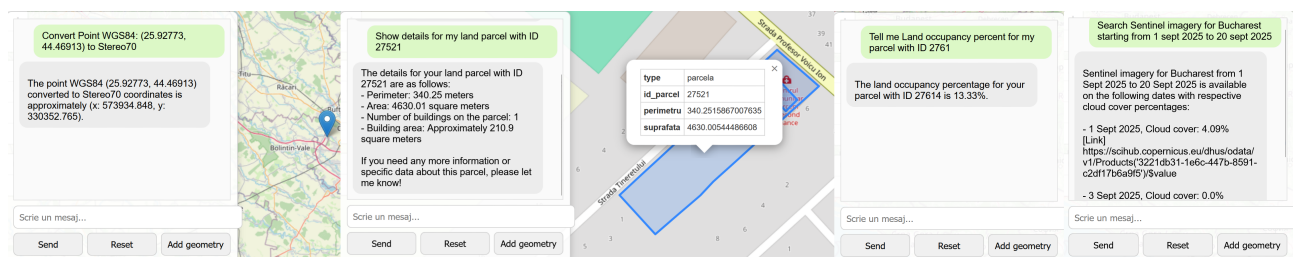


Figure 11. User query and response for gpt-4.1-mini-2025-04-14 with function calling.

4.2. Prompts Comparison

The improvements between the base model and the base model with function calling are easily noticeable when testing with simple prompts. The next step introduces a comparison between the two function-calling approaches: the gpt-4.1-mini-2025-04-14 base model and the same model fine-tuned. To objectively evaluate the two models, a set of 38 incomplete and challenging prompts, unseen during training, was prepared. A Python script was developed to sequentially submit these prompts to both models, initiating a new thread for each prompt to reset the conversation. In both cases, the prompt, the assistant's final response, and any recognized external function calls (including the extracted parameters from the conversation) were recorded.

Following data collection, a human evaluator assessed whether the assistant selected the appropriate functions and parameters or provided correct answers when function calls were unwarranted. The base model achieved a response accuracy of 89.5%, significantly outperforming the fine-tuned model, which scored 76.5%. These results indicate that

the fine-tuned model may have overfitted to the training patterns, resulting in a loss of generalization compared to the base model. As a result, it failed to recognize certain combinations of data formats. For example, when presented with the prompt “Please convert GPS coords 44.4268, 26 1025 from WGS84 to Stereo70.”, the base model successfully identified the coordinates even though one is separated by a decimal point and the other by a space; in contrast, the fine-tuned model failed to recognize coordinates separated by a space. A similar issue occurred with other separator combinations, such as in “WGS84 -> Stereo70: 47.16, 27,60”, where the fine-tuned model only recognized the decimal point as a separator. This indicates that the training data exclusively used the decimal point as a separator, which limits the model’s ability to generalize beyond the format encountered during training.

Due to the lack of generalization, the fine-tuned model fails to extract the bounding box parameter required to call the function for queries such as “Please get Sentinel NDVI for Ploiești city for 15 September 2025.” The base model leverages its general pre-trained knowledge to generate an approximate bounding box whenever regional coverage is requested. This allows it to successfully trigger functions even when parameters are not explicitly provided. In contrast, the fine-tuning dataset contained only examples where bounding box parameters were explicitly specified in coordinates. Consequently, the fine-tuned model relies exclusively on explicit inputs and fails to generate satisfactory responses without them. Furthermore, the base model demonstrates robustness by recognizing and correcting coordinate sequences that deviate from the required format [minx, miny, maxx, maxy], whereas the fine-tuned model rigidly interprets the order exactly as presented.

Nevertheless, the fine-tuned model demonstrates improved behavior for prompts such as “Search Sentinel imagery for Braşov 2025-01. ” While the base model incorrectly calls the function for downloading data from OSM, the fine-tuned model refrains from making a call and instead requests additional information to properly extract the parameters. Furthermore, the fine-tuned model shows better performance in selecting the correct function for shorter prompts. For instance, given “Parcel geometry 27690.”, the fine-tuned model correctly calls the function that displays the actual geometry of the parcel on the map, whereas the base model fails to make this fine distinction and instead calls the function returning parcel details without displaying its geometry.

The lack of generalization for the trained model was observed predominantly in the case of the `download_osm_to_geojson` function. This function is more complex than the others because it requires, as an input parameter, a specifically formatted query for the OSM Overpass API. Naturally, the user does not directly provide such a structured request but rather expresses it in natural language, for example: “I want all the pharmacies in area X.” In this situation, the assistant must formulate, based on its knowledge, a query that best satisfies the user’s requirement. Following the fine-tuning process, the dataset employed did not cover a sufficiently broad range of such queries, which consequently made the new model considerably more robust. The base model relies on queries using `area/searchArea`, which enables the selection of data by city name and allows for the inclusion of nodes, ways, and relations (e.g., streets, rivers, buildings, or hospitals), thereby returning complete geometries. In contrast, the trained model is based on queries using `around` in combination with GPS coordinates, extracts only node objects, and produces a simplified output. This approach makes it faster and more suitable for point-based extractions, but it also results in significantly reduced coverage and complexity. In the left side of Figure 12, it can be observed that the base model sequentially returns both the shops and the streets within the selected area. In contrast, the trained model does not return the streets, as in the query it generates, streets are treated as equivalent to highways.

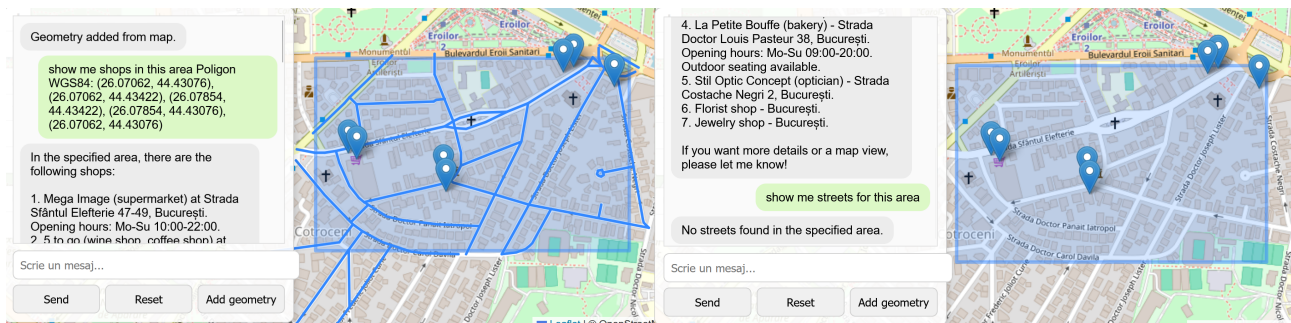


Figure 12. Conversation for `download_osm_to_geojson` function.

The comparative results reveal a clear trade-off between aggressive function selection and decision robustness. This behavior is quantitatively assessed using two decision-level metrics: function selection accuracy (FSA), which evaluates the correctness of the selected geospatial function for fully specified prompts, and clarification request accuracy (CRA), which measures the model's ability to correctly identify missing information and request appropriate clarifications for incomplete prompts. The comparison focuses exclusively on the two model configurations supporting function calling, as evaluating the base model without function calling is not meaningful in this context. The base model achieves a higher function selection accuracy (81.82%), reflecting its tendency to select a function even under incomplete input conditions, but exhibits a lower clarification request accuracy (69.70%). In contrast, the fine-tuned model shows a substantially higher clarification request accuracy (84.62%), indicating an improved ability to recognize missing information and request appropriate user input before execution, while its function selection accuracy decreases to 30.77%. This reduction in function selection accuracy is therefore an expected outcome of the model's conservative decision strategy and should not be interpreted as degraded performance. Table 2 summarizes the comparative results using decision-level metrics reflecting function selection behavior and clarification robustness.

Table 2. Comparative evaluation of decision-level performance for the two function-calling model configurations.

Model Configuration	FSA (%)	CRA (%)
Base model	81.82	69.70
Fine-tuned model	30.77	84.62

4.3. Sequential Function Call

Satisfactory results were obtained for both models in cases where the requirements involved the sequential invocation of multiple functions. The models are able to call the first function and then use its output as the input for the subsequent one. An example of this process is illustrated in the WebGIS application (see Figure 13). The user's query combines two requirements: confirmation of satellite imagery availability covering the specified parcel area during September 2025, and the immediate execution of a Normalized Difference Vegetation Index (NDVI) analysis on the most recently acquired image from that set. To manage this request, the assistant utilizes a function-chaining approach. It initially calls the function responsible for identifying the parcel and then converts the resulting geographic boundaries into the necessary bounding box parameters. These parameters are used to query for the available imagery. Essential, the system subsequently processes the returned satellite image data to select the image with the most recent acquisition date, which is then used as the specific input for the final NDVI analysis function, ensuring

complete spatial coverage of the plot. This demonstrates the system's capability to execute a multi-step, data-dependent workflow for complex geospatial tasks.

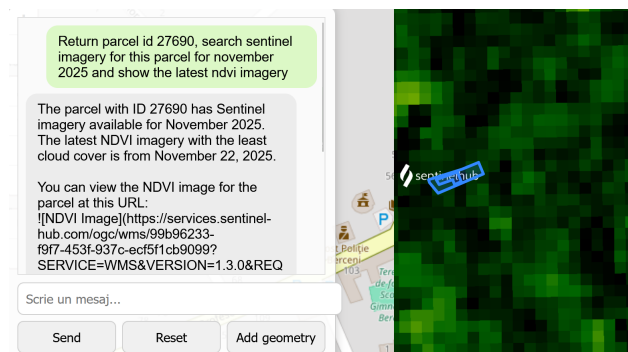


Figure 13. Sequential function invocation initiated by a simple query.

4.4. Conversation with Chat History

This comparison evaluates the models' ability to retain contextual information from the ongoing conversation and to extract parameters for function calling based on the entire context. For this purpose, the list of prompts from Figure 14 was used for both models. In this comparison, the trained model demonstrates superior results, as it adheres to the approach in which it was designed, namely, requesting additional details when essential data are missing and parameters cannot be reliably extracted. This behavior stems from its specific training to provide trustworthy responses. In contrast, the base model assigns null values to missing parameters, for example, when longitude is absent in coordinate conversion. In this case, the generalization ability of the base model proved to be less advantageous than in the previous scenarios.

```
prompts_conv = [
    "convert lat=45.803398 to Stereo70",
    "Ok, 26.1025, make the complet conversion?",
    "show parcel with cadastral identifier 27521",
    "show details for this parcel",
    "Search sentinel imagery for this parcel",
    "Search sentinel imagery for this parcel starting for 1 sept to 20 sept 2025",
    "show ndvi imagery for the latest available image",
    "Search satellite imagery for Bucharest for 1 aug 2025 to 15 aug 2025",
    "Search satellite ndvi imagery for Bucharest for 15 december 2025",
    "show parcel with 27970, calculate land occupancy percent and return geometry ",
    "Search sentinel data for this parcel for 1 sept 2025 to 20 sept 2025",
    "convert the bbox coordinates for this parcel"
]
```

Figure 14. Prompt list for test conversation with chat history.

5. Discussion

Recent agent-based geospatial systems demonstrate varying degrees of autonomy and effectiveness depending on task scope and system design. GeoGPT reports successful autonomous generation of multi-step geospatial workflows, particularly for standard GIS operations such as data acquisition from OpenStreetMap and basic spatial analyses; however, its evaluation focuses primarily on workflow correctness rather than execution within operational environments, and reported results highlight sensitivity to prompt formulation and tool availability [34]. GeoTool-GPT shows that instruction tuning and domain-specific training can significantly improve tool-selection accuracy and reduce invalid GIS function calls compared to base models, but its gains are largely confined to predefined toolsets and offline evaluation scenarios, with limited discussion of real-time system integration [35]. GIS Copilot advances practical applicability by embedding an LLM-based agent directly into QGIS, where experimental results demonstrate the successful autonomous construction and execution of spatial analysis workflows within a desktop GIS environment. Nonetheless, reported evaluations emphasize analyst productivity and workflow generation accuracy rather than deployment in web-based or service-oriented

contexts, and the system remains dependent on local GIS installations and expert-oriented interfaces [36]. In parallel, the autonomous GIS agent framework for geospatial data retrieval achieves high success rates (approximately 80–90%) in retrieving heterogeneous datasets from multiple authoritative sources by generating and debugging retrieval code, but its scope is deliberately limited to data acquisition and does not extend to end-to-end geospatial task execution or user-facing interaction [19].

In contrast, the proposed WebGIS-based virtual assistant prioritizes end-to-end task execution within an interactive, web-based geospatial environment. Rather than optimizing isolated capabilities such as workflow generation or data retrieval alone, our system integrates function calling with authoritative geospatial services to ensure deterministic execution and reproducible results. Experimental outcomes in our study demonstrate reliable execution of user-defined geospatial tasks in real-world public service scenarios, highlighting a shift from prototype-level automation toward deployment-oriented geospatial assistants. These results suggest that while existing agent-based systems validate the feasibility of LLM-driven geospatial reasoning, our approach complements and extends prior work by emphasizing operational integration, usability, and practical impact.

6. Limitations and Future Work

Despite the promising results obtained through the integration of large language models (LLMs) with geospatial tools for the automation and optimization of public services, several limitations of the present study must be acknowledged. These limitations also outline clear directions for future research and system development.

A primary limitation concerns the strong dependence on external geospatial data sources and tools. While the use of function calling significantly improves the reliability of responses by grounding them in authoritative and up-to-date datasets, the overall accuracy of the system remains contingent on the quality, completeness, and maintenance of these external resources. Any inconsistencies, delays in data updates, or inaccuracies in official databases are directly reflected in the assistant's outputs, which limits full control over end-to-end data quality.

Another important limitation relates to the generalization capability of fine-tuned models. Although fine-tuning improved the model's behavior in well-defined scenarios, particularly by encouraging cautious responses and requests for additional information, it also reduced flexibility in handling unseen or irregular input formats. Comparative evaluations showed that the fine-tuned model struggled with variations in coordinate formatting, implicit spatial references, and incomplete parameter specifications, indicating a degree of over-specialization induced by the training data.

Closely related to this issue is the relatively small size and limited diversity of the fine-tuning dataset. The training process relied on approximately 800 instruction–response pairs, which, while sufficient to demonstrate feasibility, is modest relative to the scale and complexity of the underlying language model. Although validation metrics indicated stable convergence, further evaluation on broader and more heterogeneous datasets is required to assess robustness under real-world usage conditions, where user queries may vary significantly in structure, language, and level of detail.

This study also highlights limitations in natural language geocoding and spatial disambiguation. When users refer to locations using place names, streets, or administrative areas, the models often produce approximate spatial interpretations. The absence of a dedicated geocoding service restricts spatial precision and may affect the reliability of downstream geospatial analyses. Integrating specialized geocoding APIs represents a necessary step to enhance accuracy and operational readiness.

In addition, the system exhibits a lack of native awareness of real-time temporal context. The language models operate with a static notion of time limited to their training data, which constrains their ability to handle time-sensitive queries accurately. This limitation is particularly relevant for applications involving satellite imagery acquisition dates, environmental monitoring, or dynamic public services. Future implementations should incorporate external time-aware functions to ensure temporal consistency.

From an architectural perspective, the current implementation relies on a commercial, API-based language model, which introduces constraints related to operational costs, vendor dependency, and limited transparency regarding internal model behavior. While open-source LLMs represent a potential alternative, their adoption poses additional challenges, including higher infrastructure requirements, deployment complexity, and maintenance overhead. A systematic comparison between commercial and open-source solutions remains an important avenue for future work.

Future research should therefore focus on expanding and diversifying fine-tuning datasets, integrating robust geocoding and time-awareness services, and exploring hybrid architectures that balance generalization and specialization. Additionally, evaluating the system in real institutional settings with end users from public administration would provide valuable insights into usability, trust, and long-term sustainability. Such developments are essential for transitioning from experimental prototypes to scalable, production-ready geospatial assistants capable of supporting complex public service workflows.

7. Conclusions

By introducing the capability to invoke external functions based on geospatial tools maintained by experts and connected to reliable and up-to-date databases, LLM models can successfully contribute to the development of geospatial assistants aimed at optimizing and facilitating access to public services. The integration of these functions within the above-described scenario, together with the promising results obtained, demonstrates that such geospatial assistants can be connected to any cloud computing environment to address a wide range of problems, including more complex ones, even though the case study employed a smaller model, namely, gpt-4.1-mini-2025-04-14.

An important outcome of the comparative analysis is that the effectiveness of the proposed system is primarily determined by the architectural integration between the LLM and external, authoritative geospatial functions rather than by the specific language model employed. The results demonstrate that enabling structured tool invocation (function calling) consistently provides substantial benefits, whereas fine-tuning introduces a trade-off between specialization and generalization that depends on both the underlying foundation model and the diversity of the training data. From this perspective, the proposed framework is model-agnostic. Newer versions of ChatGPT or alternative large language models, such as Gemini, Claude, or LLaMA, could be integrated without fundamental changes to the system architecture provided that they support structured tool invocation or equivalent mechanisms. In such deployments, the role of the LLM is primarily that of an orchestration and reasoning layer, while the correctness, accuracy, and timeliness of geospatial results are ensured by external geospatial services connected to authoritative data sources. This design increases the generality and long-term sustainability of the framework, allowing it to evolve alongside advances in large language models without requiring major redesign of the geospatial components.

The fine-tuning process proved highly complex and showed a critical dependence on the quality and broadness of the training data. The rigorous generation of the question-answer pairs presented to the model is therefore essential. This dataset must encompass a comprehensive range of possible scenarios, including both straightforward and challenging

examples. Furthermore, it is equally important that this training set effectively encode the desired assistant response tone and the strategies for handling incoherent or incomplete user requests. The case study revealed that the training data used for gpt-4.1-mini-2025-04-14 reduced the model's generalization capacity, thereby limiting its ability to generate responses to more difficult queries. Nevertheless, this trade-off made the newly fine-tuned model more robust and ultimately more valuable in practice. After training, the model aligns effectively with the question–answer patterns on which it was trained, makes fine distinctions in function selection, and requests additional details when it cannot generate reliable answers, thus avoiding spurious calls and overly aggressive generalization.

Although these assistants successfully fulfilled the role of a virtual geospatial assistant by acting as an intermediary between geospatial modeling and end users, this study's findings indicate specific areas for improvement. An analysis of the assistants' responses revealed deficiencies in location geocoding, as queries regarding cities, streets, or other geographical features yielded only approximate results. Integrating an external geocoding service is therefore necessary to significantly enhance performance and accuracy. Similarly, the models lack synchronization with real time, perceiving the “current time” only up to the limit of their training data. This limitation should be addressed by incorporating an external function capable of returning the present time.

A next step in extending this study could be the adoption of open-source LLMs, and from the perspective of accessibility, future efforts may focus on enabling users to interact with the virtual assistant more easily, for instance, through messaging services available on mobile devices.

Author Contributions: Conceptualization, G.I.D. and A.C.B.; methodology, G.I.D.; software, G.I.D.; validation, G.I.D. and A.C.B.; formal analysis, G.I.D. and A.C.B.; investigation, G.I.D.; resources, G.I.D. and A.C.B.; writing—original draft preparation, G.I.D.; writing—review and editing, G.I.D. and A.C.B.; visualization, G.I.D.; supervision, A.C.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data and source code supporting the findings of this study are publicly available at <https://github.com/Dorro221/geoassistant> (accessed on 2 January 2026).

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

Appendix A. Supplementary Implementation Details

```
{
  "messages": [
    {
      "role": "user",
      "content": "Convert coordinates x=500000, y=450000 from Stereo70 to WGS84."
    },
    {
      "role": "assistant",
      "tool_calls": [
        {
          "id": "call_id",
          "type": "function",
          "function": {
            "name": "convert_stereo70_to_wgs84",
            "arguments": "{\"x\":500000,\"y\":450000}"
          }
        }
      ],
      "parallel_tool_calls": false,
      "tools": [
        {
          "type": "function",
          "function": {
            "name": "convert_stereo70_to_wgs84",
            "description": "Converts coordinates from Stereo 70 to WGS84 format",
            "parameters": {
              "type": "object",
              "required": ["x", "y"],
              "properties": {
                "x": {
                  "type": "number"
                },
                "y": {
                  "type": "number"
                }
              }
            },
            "additionalProperties": false
          }
        }
      ]
    }
  ]
}
```

Figure A1. Example of an instruction–response pair used for fine-tuning the LLM.

```

from openai import OpenAI

client = OpenAI()

assistant = client.beta.assistants.create(
    name="Geospatial Assistant",
    instructions="You are an assistant who solve geospatial task.",
    model="gpt-4.1-mini-2025-04-14",
    tools=my_tools
)

```

Figure A2. Example of initializing a custom Geospatial Assistant in Python using the OpenAI Assistants API, with a specified name, role instructions, model, and toolset.

References

1. Brown, T.B.; 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. *arXiv* **2020**, arXiv:2005.14165.
2. Manvi, R.; Khanna, S.; Mai, G.; Burke, M.; Lobell, D.; Ermon, S. GeoLLM: Extracting Geospatial Knowledge from Large Language Models. *arXiv* **2024**, arXiv:2310.06213.
3. Gao, S.; Goodchild, M.F. Asking Spatial Questions to Identify GIS Functionality. In Proceedings of the 2013 Fourth International Conference on Computing for Geospatial Research and Application, San Jose, CA, USA, 22–24 July 2013; pp. 106–110. <https://doi.org/10.1109/COMGEO.2013.18>.
4. Kurniawan, D.; Rosa Indah, D.; Sari, P.; Alif, R. Understanding the Landscape of Usability Evaluation in Geographic Information Systems: A Systematic Literature Review. *J. Appl. Sci. Eng. Technol. Educ.* **2023**, *5*, 35–45. <https://doi.org/10.35877/454RI.asci1815>.
5. Pierdicca, R.; Muralikrishna, N.; Tonetto, F.; Ghianda, A. On the Use of LLMs for GIS-Based Spatial Analysis. *ISPRS Int. J. Geo. Inf.* **2025**, *14*, 401. <https://doi.org/10.3390/ijgi14100401>.
6. 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. <https://doi.org/10.3390/ijgi13100348>.
7. Tao, R.; Xu, J. Mapping with ChatGPT. *ISPRS Int. J. Geo-Inf.* **2023**, *12*, 284. <https://doi.org/10.3390/ijgi12070284>.
8. Ahmed, I.; Das (Pan), N.; Debnath, J.; Bhowmik, M.; Bhattacharjee, S. Flood hazard zonation using GIS-based multi-parametric Analytical Hierarchy Process. *Geosyst. Geoenviron.* **2024**, *3*, 100250. <https://doi.org/10.1016/j.geogeo.2023.100250>.
9. Costa, D.G.; Silva, I.; Medeiros, M.; Bittencourt, J.C.N.; Andrade, M. A method to promote safe cycling powered by large language models and AI agents. *MethodsX* **2024**, *13*, 102880. <https://doi.org/10.1016/j.mex.2024.102880>.
10. Bommasani, R.; Hudson, D.A.; Adeli, E.; Altman, R.; Arora, S.; von Arx, S.; Bernstein, M.S.; Bohg, J.; Bosselut, A.; Brunskill, E.; et al. On the Opportunities and Risks of Foundation Models. *arXiv* **2022**, arXiv:2108.07258.
11. Xu, L.; Zhao, S.; Lin, Q.; Chen, L.; Luo, Q.; Wu, S.; Ye, X.; Feng, H.; Du, Z. Evaluating large language models on geospatial tasks: A multiple geospatial task benchmarking study. *Int. J. Digit. Earth* **2025**, *18*, 2480268. <https://doi.org/10.1080/17538947.2025.2480268>.
12. Hochmair, H.; Juhász, L.; Kemp, T. Correctness Comparison of ChatGPT-4, Gemini, Claude-3, and Copilot for Spatial Tasks. *Trans. GIS* **2024**, *28*, 2219–2231. <https://doi.org/10.1111/tgis.13233>.
13. Gramacki, P.; Martins, B.; Szymański, P. Evaluation of Code LLMs on Geospatial Code Generation. In Proceedings of the 7th ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery, GeoAI '24, New York, NY, USA, October 29–November 1 2024; pp. 54–62. <https://doi.org/10.1145/3687123.3698286>.
14. Hou, S.; Zhao, A.; Liang, J.; Shen, Z.; Wu, H. Geo-FuB: A method for constructing an Operator-Function knowledge base for geospatial code generation with large language models. *Knowl. Based Syst.* **2025**, *319*, 113624. <https://doi.org/10.1016/j.knosys.2025.113624>.
15. Ying, S.; Li, Z.; Yu, M. Beyond words: Evaluating large language models in transportation planning. *Geo. Spat. Inf. Sci.* **2025**, *2025*, 1–23. <https://doi.org/10.1080/10095020.2025.2493073>.
16. OpenAI. GPT-4 Research. 2025. Available online: <https://openai.com/index/gpt-4-research/> (accessed on 15 November 2025).
17. Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv* **2023**, arXiv:2302.13971.
18. Gemma Team. Gemma: Open Models Based on Gemini Research and Technology. *arXiv* **2024**, arXiv:2403.08295.
19. Ning, H.; Li, Z.; Akinboyewa, T.; Lessani, M.N. An autonomous GIS agent framework for geospatial data retrieval. *Int. J. Digit. Earth* **2025**, *18*, 2458688. <https://doi.org/10.1080/17538947.2025.2458688>.
20. KIOS-Research. QChatGPT. GitHub Repository. 2025. Available online: <https://github.com/KIOS-Research/QChatGPT> (accessed on 12 November 2025).

21. Farnaghi, M. Intelli_Geo. GitHub Repository. 2025. Available online: https://github.com/MahdiFarnaghi/Intelli_Geo (accessed on 10 November 2025).
22. OpenAI. Function Calling and Other API Updates. 2025. Available online: <https://openai.com/index/function-calling-and-other-api-updates/> (accessed on 12 November 2025).
23. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv* **2023**, arXiv:2201.11903.
24. Cai, T.; Wang, X.; Ma, T.; Chen, X.; Zhou, D. Large Language Models as Tool Makers. *arXiv* **2024**, arXiv:2305.17126.
25. Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Zettlemoyer, L.; Cancedda, N.; Scialom, T. Toolformer: Language Models Can Teach Themselves to Use Tools. *arXiv* **2023**, arXiv:2302.04761.
26. Chen, J.; Wu, H.; Pang, J.; Wang, Y.; Zhang, D.; Sun, C. Tool learning with language models: A comprehensive survey of methods, pipelines, and benchmarks. *ViciniEarth* **2025**, *2*, 16. <https://doi.org/10.1007/s44336-025-00024-x>.
27. Bergmann, D. What is Fine-Tuning? 2025. Available online: <https://www.ibm.com/think/topics/fine-tuning> (accessed on 13 November on 2025).
28. Anisuzzaman, D.; Malins, J.G.; Friedman, P.A.; Attia, Z.I. Fine-tuning large language models for specialized use cases. *Mayo Clin. Proc. Digit. Health* **2025**, *3*, 100184. <https://doi.org/10.1016/j.mcpdig.2024.11.005>.
29. Wu, X.K.; Chen, M.; Li, W.; Wang, R.; Lu, L.; Liu, J.; Hwang, K.; Hao, Y.; Pan, Y.; Meng, Q.; et al. LLM fine-tuning: Concepts, opportunities, and challenges. *Big Data Cogn. Comput.* **2025**, *9*, 87. <https://doi.org/10.3390/bdcc9040087>.
30. Zhang, S.; Dong, L.; Li, X.; Zhang, S.; Sun, X.; Wang, S.; Li, J.; Hu, R.; Zhang, T.; Wang, G.; et al. Instruction Tuning for Large Language Models: A Survey. *ACM Comput. Surv.* **2025**, *58*, 1–36. <https://doi.org/10.1145/3777411>.
31. Zhang, Y.; Li, J.; Wang, Z.; He, Z.; Guan, Q.; Lin, J.; Yu, W. Geospatial large language model trained with a simulated environment for generating tool-use chains autonomously. *Int. J. Appl. Earth Obs. Geoinf.* **2025**, *136*, 104312. <https://doi.org/10.1016/j.jag.2024.104312>.
32. Luo, Y.; Yang, Z.; Meng, F.; Li, Y.; Zhou, J.; Zhang, Y. An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-Tuning. *IEEE Trans. Audio Speech Lang. Process.* **2025**, *33*, 3776–3786. <https://doi.org/10.1109/TASLPRO.2025.3606231>.
33. 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. <https://doi.org/10.1016/j.ipm.2024.103808>.
34. 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. <https://doi.org/10.1016/j.jag.2024.103976>.
35. Wei, C.; Zhang, Y.; Zhao, X.; Zeng, Z.; Wang, Z.; Lin, J.; Guan, Q.; 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. <https://doi.org/10.1080/13658816.2024.2438937>.
36. Akinboyewa, T.; Li, Z.; Ning, H.; Lessani, M.N. GIS Copilot: Towards an autonomous GIS agent for spatial analysis. *Int. J. Digit. Earth* **2025**, *18*, 2497489. <https://doi.org/10.1080/17538947.2025.2497489>.
37. OpenAI. OpenAI Python Library. 2025. Available online: <https://github.com/openai/openai-python> (accessed on 15 November 2025).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.