Automating Geospatial Vision Tasks with a Large Language Model Agent

Yuxing Chen[1], Weijie Wang[2], Camille Kurtz[1], and Sylvain Lobry[1] (✉)

[1] Université Paris Cité, Paris 75006, France.
{yuxing.chen,camille.kurtz, sylvain.lobry}@u-paris.fr
[2] Università degli Studi di Trento, Trento 38122, Italy. weijie.wang@unitn.it

Abstract. Large Language Models (LLMs) have shown promise in automating code generation for data science tasks, yet they struggle with complex task sequences, especially in geospatial vision tasks. These difficulties stem from challenges in managing stepwise dependencies, aligning diverse data sources with spatial constraints, and accurately applying various geospatial libraries—often resulting in logical errors or hallucinations. To address these limitations, we introduce GeoAgent, an interactive framework designed to enable LLMs to automate geospatial vision tasks effectively. GeoAgent integrates a code interpreter, static analysis, and retrieval generation within a Monte Carlo Tree Search framework, creating a robust solution tailored to the geospatial data processing workflow. We introduce a new benchmark to evaluate GeoAgent's performances on single- and multi-turn tasks, including geospatial data acquisition, analysis, and visualization across multiple Python libraries. Our experiments reveal that GeoAgent significantly outperforms baseline LLMs in function call accuracy, task pass rate and task completion, marking a substantial advancement in automating geospatial vision tasks and setting a new standard for LLM-driven geospatial data analysis.

Keywords: Large Language Model · Agent · Geospatial · Data Analysis.

1 Introduction

Large language models (LLMs) have demonstrated their potential to solve complex tasks in the geospatial domain [13, 3, 28]. Current approaches mainly rely on pre-defined, template-based prompts and third-party application programming interfaces (APIs), enabling LLMs to utilize external tools as foundational components for task completion. For instance, Remote Sensing ChatGPT [7] and ChangeAgent [17] leverage independent remote sensing (RS) vision model APIs while GEOGPT [31] uses geographic information system (GIS) API calls. These APIs provide single-line calls for a specific task without an understanding of the dependencies of their functionalities. Although GeoLLM-Engine [22] advances by integrating multiple APIs into a sequential task within a real user interface, it remains constrained by fixed task-level APIs. Recent efforts in NLP, such as QwenAgent [2] and BigCodeBench [32], demonstrate LLMs' potential to handle

open-domain data analysis utilizing any available Python libraries in code generation. These advancements suggest the potential for using LLM-based coding to address open-domain geospatial vision tasks in code execution environments.

Geospatial vision tasks pose significant challenges for LLMs, requiring them to understand complex instructions, manage interdependent inputs and outputs, and apply specialized libraries and models accurately [25]. These tasks often demand expert intervention for decomposing tasks and selecting appropriate libraries, as LLMs typically struggle with multi-library function calls, especially when under-trained with these libraries. Expert intervention is occasionally necessary for task decomposition and dynamic adjustments, which may involve fixing unsuccessful steps. General data analysis benchmarks [32, 30] rely heavily on popular Python libraries, but these are often limited to single-turn tasks and saturated by the recently released LLMs. These studies have a limited scope in geospatial vision tasks, which often require using less common geospatial Python libraries in sequential tasks. For less common geospatial Python libraries, LLMs frequently exhibit "API hallucinations" [11]. Retrieval-augmented generation (RAG) [8] has emerged as a method to supplement LLM's coding capability by incorporating domain-specific knowledge. However, effectively tackling specialized geospatial vision tasks still requires sequential reasoning and iterative refinement guided by execution feedback. Advanced techniques like Monte Carlo Tree Search (MCTS) further enhance LLMs' sequential reasoning, enabling feedback-driven, multi-step processing—a crucial aspect for handling complex geospatial tasks [10].

In addition to standalone API calls, most existing works on geospatial vision tasks leverage Vision-Language Models (VLMs) to process input data and task descriptions in a single step [28, 15, 14]. While this approach simplifies execution, it restricts detailed analysis of individual outputs and hinders integration of results across models and data sources. Key challenges arise with VLMs in geospatial applications: First, geospatial data's diverse modalities require extensive model parameters for effective initialization. Second, interdisciplinary topics often depend on domain knowledge—particularly physical models—that current VLMs cannot access. Third, geospatial tasks often require compositional reasoning, involving multiple steps like image preprocessing and analysis that are challenging to handle in a single prompt. Finally, many remote sensing data are cloud-based and accessible only via APIs, limiting VLMs' ability to query data dynamically. Unlike VLMs, LLM-based code generation offers greater flexibility, supporting multi-model integration and stepwise processing for more comprehensive geospatial data analysis.

To address these challenges, we introduce GeoAgent, an LLM-based agent that combines a code interpreter, static analysis, and RAG within an MCTS framework. We also propose a benchmark for evaluating diverse geospatial vision and vision-support tasks. GeoAgent fulfills human requirements by translating given tasks into executable Python code, utilizing dynamic task adjustment and refinement through the MCTS. This iterative refinement enables GeoAgent to manage dependencies among subtasks and dynamically refine them using execu-

tion feedback within an MCTS framework, ensuring that each code segment is logically consistent and well-developed with prior steps. The following outlines our key contributions:

- We introduce a novel LLM agent that integrates an external knowledge retriever, a code interpreter, and static analysis within an MCTS framework tailored for geospatial vision tasks. This integration enhances problem-solving, and logical capabilities in sequential task programming. Operating within Jupyter Notebook, GeoAgent enables iterative user interaction, optimizing code execution while ensuring compliance with Python libraries and best practices;
- We present GeoCode, an execution-based benchmark comprising over 18,000 single-turn and 1,356 multi-turn geospatial data analysis tasks, involving 2,313 function calls from 28 widely-used libraries across 8 task categories. GeoCode provides two evaluation models: single-turn task evaluation, which measures function call accuracy and task pass rate, and multi-turn task evaluation, which assesses task completion rates through either automatic iterative refinement or human intervention;
- We evaluate multiple LLMs on the GeoCode benchmark, showing that general-purpose LLM coders often produce incomplete workflows, while GeoAgent achieves higher task pass and completion rates. This success is attributed to its effective handling of specialized Python libraries within an MCTS framework, guided by execution feedback. The study provides a more accurate assessment of LLM coders in geospatial data processing and points to a promising future for automating geospatial vision tasks.

2 Related Work

2.1 LLM-based Code Generation

LLMs exhibit strong abilities to generate standalone function codes while struggling with multi-step and interrelated task programming. Recent works [27, 24, 30] have emphasized integrating tools like code interpreters and static analysis to enhance code-based reasoning capabilities. For example, RepairAgent [4] uses static analysis for code repair, and $STALL^+$ [20] enhances generated code fixes. Execution feedback approaches like CodeAct [24] iteratively improve code through interpreter feedback, supporting symbolic computations and logical consistency. RAG has also advanced code generation by connecting LLMs to external databases, improving precision with evolving libraries [11]. However, there is no geospatial RAG database for geospatial task programming. Tools like ToolFormer [21] and CodeAgent [29] have enabled the invocation of standalone APIs within code generation, though challenges remain for open-domain tasks. Additionally, API hallucinations are prevalent, especially with less common APIs, with rates reaching up to 15% in recent studies [18].

2.2 Geospatial Vision Tasks with LLMs

Researchers have explored integrating LLMs into geospatial vision tasks [31, 15, 3]. The preliminary attempt at bridging the gap between visual features and the semantic reasoning capabilities of LLMs is large VLMs [15, 3], which incorporate LLMs into RS image captioning, Visual Question Answering (VQA), and visual grounding tasks. Pioneering work RS-CLIP [15] created a human-annotated RS image captioning dataset, advancing VLMs in the RS domain. To leverage LLM capabilities, RS-LLaVA [3] developed the RS-instructions dataset, a benchmark integrating captioning and VQA tasks in a LLaVA [19] framework. However, the textual outputs of LLMs often fall short of meeting users' expectations. Recent advancements in GeoChat [13] have expanded VLM tasks to referring expression, region captioning, image description, and VQA. While significant progress has been made in geospatial vision tasks, challenges persist in applying these models to open-domain scenarios, especially in using multiple professional tools, procedures, and multimodal data. Recent works [16, 31] shift LLMs from operational roles to decision-makers, enabling them to select appropriate tools. For example, Change-Agent [17] directs LLMs to use segmentation and captioning tools, while GeoLLM-Engine [22] calls geospatial APIs and external knowledge bases to handle sequential tasks. However, these tools require pre-determined task-level APIs, limiting their use in open-domain tasks, especially in sequential tasks where tool calling integration becomes challenging.

3 Methodology

3.1 Framework

GeoAgent is an LLM agent designed to process geospatial vision tasks. Its architecture, shown in Figure 1, consists of two main components: 1) Task programming (Figure 1 a): GeoAgent starts by leveraging the parameterized knowledge of LLMs to generate code based on task instructions. External knowledge, such as specific Python libraries, can be integrated with retrieved items through RAG and execution feedback to enhance code generation. 2) Task refinement with MCTS (Figure 1 b, c, d): GeoAgent executes code and collects feedback within the MCTS framework. MCTS explores and evaluates multiple code candidates, while LLMs serve as the reasoner, diagnosing errors, refining prompts, and correcting failed code. This integration allows dynamic adjustments during task programming.

3.2 Dynamic Refinement on MCTS

The GeoAgent employs an MCTS framework [12] to iteratively refine subtasks within the task sequence. MCTS utilizes a tree structure where nodes represent states s and edges denote actions a. The algorithm explores the state space starting from the root s^0 to identify the terminal state s^n with the highest reward $r(s^n)$. Each node contains: the visited times count v, probability p from LLMs,

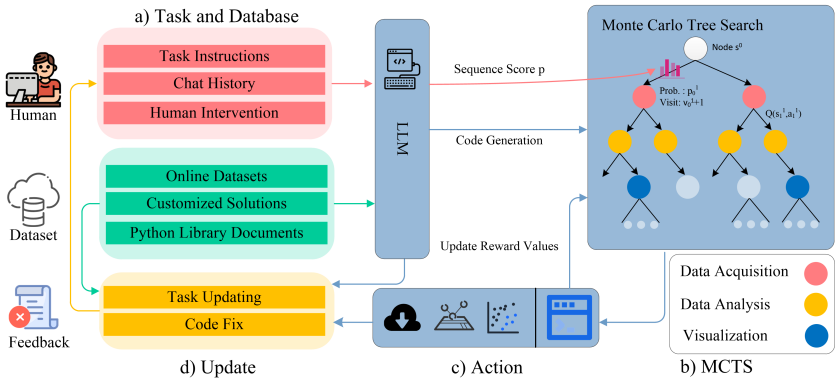

Fig. 1. GeoAgent: A geospatial vision task programming agent. This agent comprises four integral components: **a) Task and Database**, which provide LLM the task instruction and task-relevant items including Python library documents, online datasets, and solutions; **b) MCTS**, explores and evaluates multiple possible code candidates to optimize the selection of the most promising solution at each step through iterative adjustment and refinement; **c) Action**, which is an execution environment integrated with a code interpreter and static analysis and provides feedback on generated code; and **d) Update**, which suggests potential error fixes on generated code and adjustments on given tasks. In addition, the **LLM** functions as the code generator and the intelligent reasoner to propose code solutions and iteratively diagnose detected errors.

and state-action value $Q(s, a)$ (i.e., the maximum reward obtained by taking action a from state s). The reward r of state s is updated by the backpropagation of its children node's reward. Nodes with higher rewards are prioritized indicating high-quality generation. GeoAgent first transforms natural language instructions into executable codes and then dynamically refines each subtask during the MCTS selection, expansion, evaluation, and backpropagation. The full MCTS process is illustrated in Figure 2.

First, in the selection phase, GeoAgent uses the probabilistic Upper Confidence Bound (P) algorithm to select branches starting from root node s^0, which is defined as:

$$P = \arg\max_{i \in I} \left[Q\left(s^0, a_i^0\right) + \beta\left(s^0\right) \cdot \frac{p_i^0 \sqrt{\log\left(v^0\right)}}{1 + v_i^0} \right], \tag{1}$$

$Q\left(s^0, a_i^0\right)$ denotes the maximum reward obtained at action a_i^0 and is defined as:

$$Q\left(s^0, a_i^0\right) = \frac{S_{pass} - S_{node}}{S_{all} - S_{node}}, \tag{2}$$

where S_{all} represents the total number of code steps generated during the look-ahead phase, S_{pass} is the number of code steps that passed the evaluation test, and S_{node} is the number of code steps in node s^0. In addition, $\beta(s^0)$ is the weight

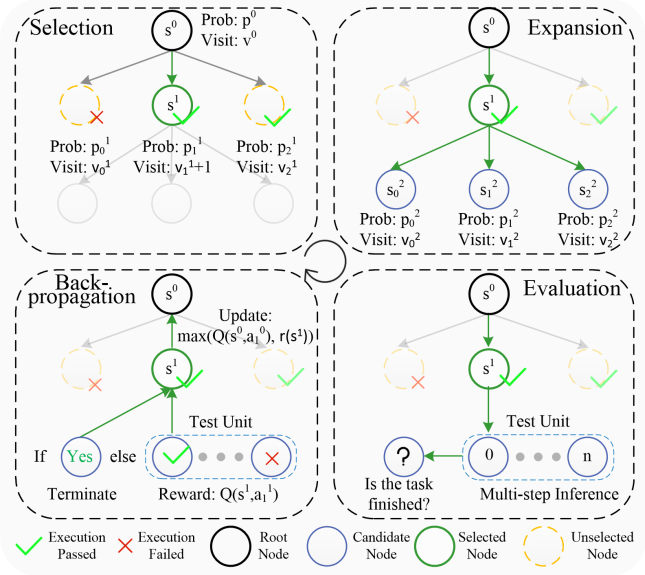

Fig. 2. Illustration of using the Monte Carlo Tree Search algorithm in geospatial vision task programming.

for exploration:

$$\beta(s^0) = \log\left(\frac{v^0 + c_{base} + 1}{c_{base}}\right) + c. \tag{3}$$

It depends on the visited number (v^0) and constants c_{base} and c, where a higher c encourages more exploration. In Equation (1), $\beta(s^0)$ is weighted by the probability p that is the LLM-determined sequence score. In the selection phase, GeoAgent begins at the root node s^0, recursively selecting subtrees until it reaches an unexplored node. This process uses P to balance exploration between known and less-visited states. Then, in the expansion phase, after selecting the initial node, potential codes for subsequent steps are generated and added to the child node list until the next subtask is reached. We sample n code snippets generated from the prompt and return the top $k = 3$ of them. These three code snippets (s_0^2, s_1^2, s_2^2) are added to the current node's (s^1) children list. For each child node, the reward Q is set to 0 for executable ones and -1 for non-executable ones. And then, in the evaluation phase, the selected node s^1 must be assessed, despite the node potentially representing only a partial program. Since the quality of partial programs is uncertain, the LLM performs a look-ahead search, generating a longer code snippet ("Test Unit"). This "Test Unit" is concatenated with the code of the current node for execution test. Finally, the reward of the current node is calculated based on the "Test Unit" and measured using Equation (2). The node terminates and receives a reward of 1 if the test code unit fully overlaps with its code. At last, in the backpropagation phase, this reward $r(s^1)$ is

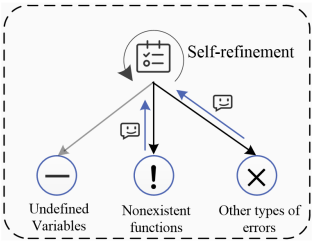

Fig. 3. The self-refinement algorithm in MCTS.

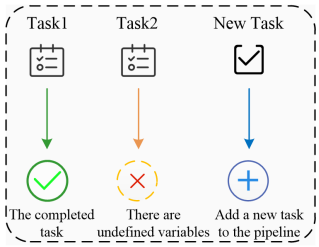

Fig. 4. Addition of new tasks to the task pool in MCTS.

then backpropagated through the tree, updating the values of its ancestor nodes accordingly.

The proposed MCTS framework incorporates an error traceback and analysis mechanism to refine subtasks. As shown in Figure 3 and Figure 4, it assesses the initial code using a code interpreter and static analysis tools. When undefined variables arise, the framework removes the affected subtree, adds a new subtask to define the variable at the top of task stacks (see Figure 4), and retries. If non-existent functions are called, tools such as Python Jedi [9] retrieve alternative functions, which are added to the prompt to guide the LLM in correcting the function call. For other errors, the interpreter provides traceback details, allowing the LLM to diagnose and suggest fixes, such as syntactic errors, logical errors, or issues with Python libraries. The failed code snippet, with suggested corrections and original instructions, is reintroduced into the task stack. This process continues until the generated code is successfully executed or the maximum number of attempts is reached.

However, even with the integration of static analysis and execution feedback, ensuring the success of task programming remains challenging due to the non-deterministic nature of LLM inference. GeoAgent addresses this by setting a maximum attempt limit. Meanwhile, manual editing is also introduced to enhance task completeness. If the problem remains unsolved, GeoAgent generates a report detailing the task, code, and encountered errors for human intervention; these modifications are then integrated as contextual inputs in subsequent runs.

4 Benchmark Setup

4.1 Benchmark Construction[3]

We outline the process of constructing the benchmark, which involves three primary steps: code collection, code refactoring, and instruction generation. Creating a high-quality, execution-based benchmark for geospatial vision tasks is challenging, as finding naturally self-contained geospatial tasks with detailed instructions is rare. GeoAgent gathers code snippets that utilize geospatial Python

[3] See Supplement 8 for additional details and 9.1 for used prompts.

Table 1. Illustration of Python libraries in GeoCode. This table categorizes Python libraries used in GeoCode according to their respective domains. Each domain is associated with specific tasks.

Domain	Library
Data Acquisition, and Data Preparation	earthengine-api, cubo, pystac, GOES-2-Go, meteostat, pystac_client, pytesmo, planetary_computer
Raster Processing	earthengine-api, eemont, geetools, GeoUtils, wxee, xarray-spatial
Vector Processing	GemGIS, GeoPandas, GeoUtils
3D analysis	Gempy
Machine Learning	scikit-eo, Verde
Deep Learning	segment-geospatial, srai
Specific Alogorithm	geeet, gstools, sen2nbar, pylandtemp, eradiate, spectramap
Visualization	geemap, leafmap, Lonboard

libraries and creates corresponding task descriptions for each code segment. Given the limited resources for geospatial vision tasks using Python libraries, we employ tutorials from various Python libraries and leverage LLMs to construct code-instruction pairs. In addition, we also constructed the RAG library to assess the function call performance under RAG assistance. Notably, we split GeoCode into two subsets: Google Earth Engine library-based tasks (GeoCode-GEE) and the other library-based tasks (GeoCode-Other).

4.2 Benchmark Statistics[4]

The proposed GeoCode benchmark, summarized in Table 1, encompasses 28 widely-used Python libraries across 8 key domains. These libraries extensively facilitate or support geospatial vision tasks, with code snippets often combining multiple library functions, thereby requiring considerable compositional reasoning ability. Table 2 compares GeoCode with existing executable Python programming benchmarks, highlighting the libraries and function calls referenced in these benchmarks. Notably, GeoCode includes 18,148 single-turn tasks and 1,356 multi-turn tasks as well as 2,313 function calls from 28 external libraries, reflecting a broader diversity compared to other benchmarks. This indicates that GeoCode offers diverse task prompts, which involve complex instructions and demand intricate implementation logic.

We provide a simple taxonomy of GeoCode-GEE single-turn tasks without context: Data Acquisition (4.4%), Data Analysis (73.6%), and Visualization (22%). Unlike common data science programming tasks where users give clean and simple data (e.g., tables or images), GeoCode works with thousands of online geospatial datasets in many different formats. These tasks often require

[4] See Supplement 8.1 for the library version and 9.2 for task examples.

Table 2. Python programming benchmark statistics: analysis by external library usage, function call frequency, task count, and task type.

Benchmark	External Library	Function Call	Single-turn Tasks	Multi-turn Tasks
DS-1000	14	540	1000	0
ODEX	13	190	439	0
BigCodeBench	62	877	1140	0
CIBench	11	171	469	73
GeoCode	28	2313	18148	1356

combining multiple datasets, and the model—not the user—must decide which data to use. This means the model must not only understand the task well, but also know what data is available and how to use it. This makes GeoCode more complex than standard data science benchmarks.

4.3 Experimental Settings

In this work, we select Llama 3.1 (8B) [5] as the LLM to support geospatial vision task programming. Additionally, we consider CodeGemma 2 (7B) [23] and Phi3.5-mini (3.8B) [1], Qwen2 (7B) [26] in our analysis of the impact of different model sizes and specialized LLMs. Inference is performed using a single NVIDIA GeForce RTX 4090 GPU, encompassing both the initial stage of instruction generation and the subsequent stage of code generation. The parameters configuration includes setting the top-p value to 0.9, the temperature parameter to 0.6, and a maximum token limit of 2048. The window size is set to at least 32k tokens, thereby supporting the retrieval of extensive contexts. For the Retriever modules, we employ the embedding model (BBAI-embedding-001) to generate high-quality embeddings for both texts and codes, enabling efficient similarity computations and retrieval processes. The constants c_{base} and c in MCTS are set as 10 and 4, the maximum attempt of each subtask is set as 3.

4.4 Metrics

This section presents the evaluation metrics for assessing function calls and task completion. To evaluate function calls in generated code, we apply multilabel classification metrics: Precision, Accuracy, Recall, F1 score, and Hamming distance[5]. Functions referenced in the generated code are extracted as predicted labels, aligning the evaluation with example-based multilabel classification [6], where partial correctness is taken into consideration. The label differences are averaged across all tasks in the test set. For a function call dataset T with n tasks (X_i, Y_i) and k classes, let h denote LLM, and $Z_i = h(X_i) = \{0,1\}^k$ represent the set of label memberships predicted by the LLM for the task X_i. The

[5] See Supplement 8.7 for other metric definitions.

Table 3. Comparison of function call performance of Llama3.1 with zero (@0) and three (@3) retrieval items across all benchmarks.

	Accuracy		F1		Hamming Loss	
	@0	@3	@0	@3	@0	@3
DS-1000	0.38	0.40	0.45	0.47	0.15	0.15
ODEX	0.49	0.53	0.50	0.55	0.28	0.26
BigCodeBench	0.37	0.61	0.45	0.72	0.11	0.10
CIBench	0.54	0.57	0.62	0.66	0.14	0.14
GeoCode-GEE	0.60	0.54	0.65	0.57	0.20	0.22
GeoCode-Others	0.61	0.67	0.66	0.72	0.19	0.17

F1 score is the harmonic mean of precision and recall, is defined as:

$$F_1 = \frac{1}{n}\sum_{i=1}^{n}\frac{2\,|Y_i \cap Z_i|}{|Y_i| + |Z_i|}. \tag{4}$$

For code completion, we consider task pass rate (pass@1) and task completion rate, where the task pass rate is mainly for single-turn task evaluation and the task completion rate is the primary measure of multi-turn tasks. For instance, in a 10-step task, the model is prompted step-by-step to generate 10 code blocks sequentially. The completion rate is calculated based on consecutive successful executions; if the first five steps succeed, the rate is 50%. Furthermore, the pass@1 metric, a standard for evaluating single-turn task execution correctness, where sequential tasks were converted into single-turn tasks to facilitate this evaluation.

5 Evaluation

This section presents the evaluation results for all benchmarks along with the proposed GeoCode-GEE and GeoCode-Others. GeoCode-GEE focuses exclusively on tasks within the GEE environment, while GeoCode-Others involves multiple Python libraries. Additionally, we include benchmarks relevant to general data science tasks, such as DS-1000, ODEX, BigCodeBench, and CIBench. We first assess the function call performance with RAG using Accuracy, F1 score, and Hamming loss metrics. To assess GeoAgent's performance on task-level tasks, we evaluate the pass rate (pass@1) on single-turn tasks and the qualitative function call (F1 score) while assessing the task completion rate on multi-turn tasks. Notably, RAG is not included in task-level evaluation avoiding introducing additional uncertainty.

5.1 Function Call Performance

We first evaluate the function call performances using Llama3.1 and RAG-powered Llama3.1, with metrics reported under settings involving zero (@0)

and three (@3) retrieval items. As shown in Table 3, most benchmarks benefit from including retrieved function documentation, leading to improved performance across all metrics. The most significant improvement is observed on BigCodeBench, where the @3 setting achieves a 0.275 increase in F1 score and a 0.007 reduction in Hamming loss. Conversely, the improvement in F1 score for other benchmarks is minimal: GeoCode-Others shows a gain of about 0.06, CIBench and ODEX about 0.04, while DS1000 exhibits almost no increase. Notably, performance on GeoCode-GEE declines with RAG, likely because the GEE Python library is heavily represented in Llama3.1's training data. Adding RAG in this context introduces noise, which detracts from generation quality. This suggests that LLMs may not yet be robust in handling extensive context, which can lead to undesired outcomes when processing varied inputs. Most geospatial Python libraries are unlikely to reach saturation in future LLM releases due to limited online documentation, making it crucial to include library documentation in the LLM's context for accurate function calls.

5.2 Single-turn Task Evaluation

To assess GeoAgent's performance in single-turn tasks, we conducted experiments on 100 randomly chosen single-turn tasks of each benchmark, given our limited computational resources. We first evaluated the pass rate (pass@1) of single-turn tasks using four different LLMs (Llama3.1, CodeGemma, Phi3.5-mini, and Qwen2) both with and without GeoAgent. As shown in Table 4, most benchmarks demonstrate a higher pass rate when using GeoAgent across all four LLMs. This improvement is because LLMs sometimes fail by calling incorrect or non-existent functions[6]. GeoAgent addresses this issue by allowing multiple attempts when the initial attempt fails. The most significant improvements are observed on the GeoCode and ODEX benchmarks. For GeoCode-GEE, GeoAgent achieves a 13% improvement with Llama3.1, a 6% improvement with CodeGemma, a 16% improvement with Phi3.5-mini, and a 20% improvement with Qwen2. For GeoCode-Others, GeoAgent shows an 18% improvement with CodeGemma, a 5% improvement with Phi3.5-mini, and a 21% improvement with Qwen2. Similarly, on ODEX, GeoAgent achieves a 17% improvement with Llama3.1, a 9% improvement with CodeGemma, a 7% improvement with Phi3.5-mini, and a 10% improvement with Qwen2.

Among the different LLMs, Phi3.5-mini achieves the best performance on general data science benchmarks while CodeGemma demonstrates the highest performance on the GeoCode benchmark, achieving a pass@1 rate of 86% on GeoCode-GEE and 59% on GeoCode-Others. This suggests that code instruction-tuned LLMs may perform better on geospatial task code generation. Across all benchmarks, DS-1000 exhibits a notably lower pass rate with both standalone LLMs and GeoAgent, highlighting the challenge of generating executable solutions for this benchmark. Nonetheless, GeoAgent still manages to improve performance on DS-1000.

[6] See Supplement 9.3 for error examples.

Table 4. Code generation pass rate (Pass@1) of the Llama3.1, CodeGemma, Phi3.5 mini, and Qwen 2 on all benchmarks.

	Llama3.1		CodeGemma		Phi3.5 mini		Qwen 2	
	LLM	Agent	LLM	Agent	LLM	Agent	LLM	Agent
DS-1000	0.05	0.34	0.10	0.19	0.03	0.06	0.15	0.18
ODEX	0.74	0.91	0.78	0.87	0.84	0.91	0.84	0.94
BigCodeBench	0.67	0.61	0.82	0.82	0.94	0.91	0.87	0.84
CIBench	0.93	0.92	0.93	0.96	0.94	0.99	0.96	0.93
GeoCode-GEE	0.76	0.89	0.86	0.92	0.66	0.82	0.61	0.81
GeoCode-Others	0.45	0.40	0.58	0.76	0.50	0.55	0.39	0.61

Table 5. Function call performance (F1 score) of the Llama3.1, CodeGemma, Phi3.5 mini, and Qwen 2 on all benchmarks.

	LLama3.1		CodeGemma		Phi3.5 mini		Qwen 2	
	LLM	Agent	LLM	Agent	LLM	Agent	LLM	Agent
DS-1000	0.83	0.86	0.75	0.75	0.80	0.80	0.79	0.79
ODEX	0.53	0.66	0.54	0.56	0.55	0.58	0.55	0.56
BigCodeBench	0.77	0.80	0.72	0.74	0.79	0.80	0.69	0.68
CIBench	0.80	0.82	0.76	0.76	0.83	0.83	0.78	0.79
GeoCode-GEE	0.78	0.86	0.75	0.71	0.70	0.77	0.76	0.74
GeoCode-Others	0.66	0.69	0.69	0.70	0.66	0.72	0.63	0.64

While the pass rate alone does not fully capture task-level performance, we also evaluate function call performance using the F1 metric. As shown in Table 5, most benchmarks benefit from GeoAgent, leading to improved function call performance. The most significant improvements are observed on GeoCode-GEE and ODEX with Llama3.1, where GeoAgent achieves a 0.08 increase on GeoCode-GEE and a 0.13 increase on ODEX. Among the different LLMs, Llama3.1 and CodeGemma demonstrated the highest function call performance on the GeoCode benchmark. Overall, considering both pass rate and function call performance, GeoAgent with Llama3.1 performs best on single-turn tasks in GeoCode-GEE, while GeoAgent with CodeGemma is the top performer across the entire GeoCode benchmark for single-turn tasks. Additionally, GeoCode-GEE consistently outperforms GeoCode-Others across all standalone LLMs and GeoAgent configurations, suggesting that tasks involving multiple libraries are more challenging than single-library tasks.

5.3 Multi-turn Task Evaluation

To assess GeoAgent's performance on multi-turn tasks[7], we use 30 randomly chosen sequential tasks from three multi-turn benchmarks. Specifically, we evaluate the completion rates of these tasks under both self-debugging and human

[7] See Supplement 9.4 for a multi-turn task case.

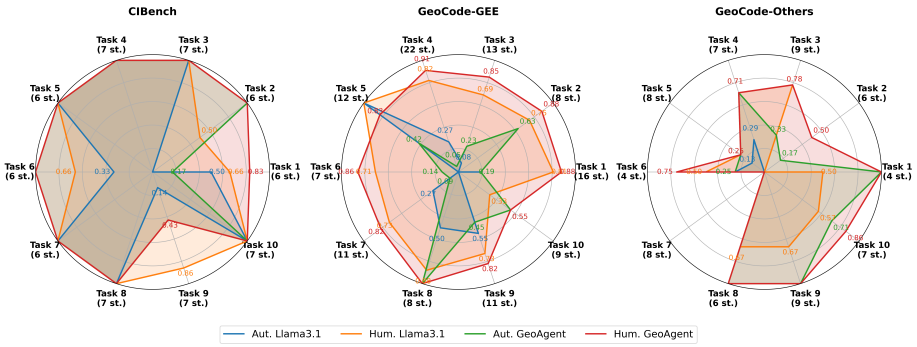

Fig. 5. Code generation complete rate (Complete@1) of the Llama3.1, GeoAgent under modes of self-debugging (Aut.) and Human Intervention (Hum.) across all benchmarks from Task 1 to Task 10 where *st.* denotes task steps.

intervention modes (see Figure 5). When a problem remains unresolved, human intervention is introduced to move the task to the next step. Consistent with the single-turn task evaluations, GeoAgent achieves a higher completion rate on all benchmarks under both the automatic and human intervention modes.

GeoAgent with Llama3.1 improves the completion rate by 20% and 6% on CIBench, 9% and 10% on GeoCode-GEE, and 48% and 22% on GeoCode-Others, under self-debugging and human intervention modes, respectively. The improvement arises because code generated solely by Llama3.1 frequently includes undefined variables, causing execution failures. In such cases, GeoAgent dynamically adjusts by refactoring the undefined variables into new subtasks and updating failed attempts within the current task loop. Compared to the automatic mode, human intervention improves the completion rate by 20% and 7% on CIBench, 46% and 47% on GeoCode-GEE, and 40% and 14% on GeoCode-Others, under LLMs alone and GeoAgent, respectively. Nearly all cases perform better with human assistance across both vanilla LLMs and GeoAgents. These observations suggest that LLMs perform better with human interaction, highlighting a promising direction for integrating LLMs to assist humans in geospatial data analysis tasks. Although GeoCode-Others is the most challenging one among the three benchmarks, with a zero completion rate for 7 out of 10 tasks using Llama3.1 alone in automatic mode, GeoAgent significantly improves the pass rate for these tasks. In contrast, CIBench is the easiest benchmark, achieving a 100% completion rate for 6 out of 10 tasks.

We compare the computational performance of Llama3.1 and GeoAgent on the GeoCode-GEE benchmark, focusing on runtime and running steps per task (Table 6). On average, Llama3.1 completes each task in approximately 6 minutes, whereas GeoAgent requires about 14 minutes. GeoAgent also tends to require more steps: across the 10 evaluated tasks, it uses an average of 14.3 steps, compared to 11.7 steps for Llama3.1—an increase of 2.6 steps per task. This additional computational burden limited our evaluation to 10 tasks per dataset.

Table 6. Runtime and step counts for multi-turn tasks using Llama3.1 and GeoAgent with self-debugging on the GeoCode-GEE benchmark.

Task No.	1	2	3	4	5	6	7	8	9	10
Aut. Llama3.1 (Steps)	16	8	13	22	12	7	11	8	11	9
Aut. GeoAgent (Steps)	16	10	15	27	16	7	15	9	11	17
Aut. Llama3.1 (Minutes)	9.0	5.5	5.2	15.0	4.4	2.6	6.5	2.9	5.8	2.7
Aut. GeoAgent (Minutes)	18.6	9.6	16.7	35.6	12.9	4.7	15.1	4.8	8.1	13.7

Furthermore, GeoAgent's performance is affected by latency from remote data access on the Google Earth Engine (GEE) server.

5.4 Discusssion[8]

Generating code with LLMs alone is challenging, as they lag behind evolving Python libraries and datasets. Our RAG implementation, simply using vector matching with metadata filters and a pre-trained embedder, limits the retrieval performance. As shown in Table 3, function call performance shows minimal improvement in DS1000 and declines in GeoCode-GEE, likely due to the saturation of relevant libraries in recent LLMs and the noise introduced by RAG. This suggests a need for selective RAG application only when LLMs cannot find suitable functions. The proposed GeoAgent integrates RAG into a structured decision-making framework that combines a code interpreter, static analysis, and MCTS. As shown in Section 5.1, RAG improves function selection, while Section 5.2 demonstrates that the code interpreter, static analysis, and MCTS still contribute significantly to task-level performance even in the absence of RAG. However, the independent effectiveness of MCTS remains unverified, as its reward computation relies on feedback from the code interpreter.

For task-level evaluation, this study focuses on process-oriented assessment. Although the generated code is executable, evaluating its performance based on output is challenging due to the variability and complexity of results. Both the quality of task instructions and the complexity of tasks can influence their results. In single-turn task evaluations, benchmarks like ODEX and GeoCode-Others exhibit the lowest function call performance, but ODEX achieves the second-highest pass rate and GeoCode-Others has the lowest pass rate, apart from DS1000 (Table 4). This suggests that ODEX contains more open-ended problems that can be solved with certain Python libraries, whereas LLMs may lack sufficient knowledge of the libraries used in GeoCode-Others. For DS1000, despite having the lowest pass rate, it achieves the second-highest function call performance (Table 5). This indicates that DS1000's low pass rate stems primarily from task instructions that are not well-suited to generating executable solutions. The reasoning capabilities of LLMs further affect outcomes, as noise may be introduced during multi-step inference. As shown in Table 4, GeoAgent

[8] See Supplement 8.2 for further discussions.

with Llama3.1 appears to struggle with inference noise on ODEX and GeoCode-Others. Similar issues are observed with GeoAgent using Phi3.5-mini on BigCodeBench and with Qwen2 on CIBench. Multi-turn task evaluations, such as Task 1 in CIBench and Task 4 in GeoCode-GEE, show similar patterns. Additionally, even human intervention does not guarantee task success if tasks are too complex, as seen in Task 7 of GeoCode-Others.

To better understand failure modes, we categorize common errors into four types: (1) Instruction-following errors, where the LLM misinterprets or oversimplifies the prompt due to insufficient domain knowledge; (2) Hallucination errors, involving invalid function calls or undefined variables—often stemming from reliance on less common geospatial libraries; (3) Lack of information errors, where the prompt lacks critical input details or context; and (4) General code errors, which reflect broader limitations in LLM reasoning or syntax handling within the geospatial programming domain.

6 Conclusion

We introduce GeoAgent, an innovative approach designed to enhance access to extensive geospatial datasets and facilitate automated geospatial vision task workflows. By leveraging the capabilities of LLMs and a diverse set of evolving Python libraries, GeoAgent transforms tasks into executable units and refines the corresponding library usage through the MCTS framework. To assess the efficacy of GeoAgent, we developed a benchmark, GeoCode, focused on geospatial vision tasks using popular geospatial Python libraries. Our experimental results on GeoCode, along with existing benchmarks, demonstrate that GeoAgent outperforms LLM baselines in both data science and geospatial vision tasks. The findings highlight that GeoAgent significantly improves the pass rate and task completion for geospatial tasks. Future work will proceed along two main directions. First, we will scale GeoAgent by fine-tuning LLMs using reinforcement learning with task completion rewards, aiming to enhance planning and long-horizon decision-making. Second, we will extend the integration of geospatial libraries and improve the LLM's ability to dynamically generate new tools based on available Python packages, thereby increasing the system's adaptability to more diverse and complex geospatial workflows. With GeoAgent, we envision a future for advanced assistance tools that can seamlessly access relevant Python libraries and extensive online data for various geospatial tasks, thereby generating tailored code for researchers. We hope this work contributes to advancing the use of geospatial data in research aimed at societal benefits and environmental conservation.

Acknowledgments. This work is supported by *Agence Nationale de la Recherche* (ANR) under the ANR-21-CE23-0011 project. The GitHub repository for this work will be made available at: https://github.com/Yusin2Chen/GeoAgent.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Abdin, M., Jacobs, S.A., Awan, A.A., Aneja, J., Awadallah, A., Awadalla, H., Bach, N., Bahree, A., Bakhtiari, A., Behl, H., et al.: Phi-3 technical report: A highly capable language model locally on your phone. CoRR **abs/2404.14219** (2024). https://doi.org/10.48550/ARXIV.2404.14219
2. Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al.: Qwen2.5 technical report. CoRR **abs/2412.15115** (2024). https://doi.org/10.48550/ARXIV.2412.15115
3. Bazi, Y., Bashmal, L., Al Rahhal, M.M., Ricci, R., Melgani, F.: RS-LLaVA: A large vision-language model for joint captioning and question answering in remote sensing imagery. Remote Sensing **16**(9), 1477 (2024). https://doi.org/10.3390/rs16091477
4. Bouzenia, I., Devanbu, P., Pradel, M.: Repairagent: An autonomous, LLM-based agent for program repair. In: 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), pp. 694–694. IEEE Computer Society, Los Alamitos, CA, USA (2025). https://doi.org/10.1109/ICSE55347.2025.00157
5. Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al.: The Llama 3 herd of models. CoRR **abs/2407.21783** (2024). https://doi.org/10.48550/ARXIV.2407.21783
6. Giraldo-Forero, A.F., Jaramillo-Garzón, J.A., Castellanos-Domínguez, C.G.: Evaluation of example-based measures for multi-label classification performance. In: Bioinformatics and Biomedical Engineering: Third International Conference, IWBBIO 2015, Granada, Spain, April 15–17, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9043, pp. 557–564. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-16483-0_54
7. Guo, H., Su, X., Wu, C., Du, B., Zhang, L., Li, D.: Remote sensing ChatGPT: Solving remote sensing tasks with ChatGPT and visual models. In: IGARSS 2024 - 2024 IEEE International Geoscience and Remote Sensing Symposium, pp. 11474–11478. IEEE (2024). https://doi.org/10.1109/IGARSS53475.2024.10640736
8. Guu, K., Lee, K., Tung, Z., Pasupat, P., Chang, M.: Retrieval augmented language model pre-training. In: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13–18 July 2020, Virtual Event. Proceedings of Machine Learning Research, vol. 119, pp. 3929–3938. PMLR (2020).
9. Halter, D.: Jedi: an awesome autocompletion tool for Python (2024), accessed: 2024-10-18. https://github.com/davidhalter/jedi
10. He, G., Singh, Z., Yoneki, E.: MCTS-GEB: Monte Carlo Tree Search is a Good E-graph Builder. In: Proceedings of the 3rd Workshop on Machine Learning and Systems, EuroMLSys 2023, Rome, Italy, 8 May 2023, pp. 26–33. ACM (2023). https://doi.org/10.1145/3578356.3592577
11. Jain, N., Kwiatkowski, R., Ray, B., Ramanathan, M.K., Kumar, V.: On mitigating code LLM hallucinations with API documentation. In: Proceedings of the 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). To appear. IEEE/ACM (2025).
12. Jiang, X., Dong, Y., Wang, L., Fang, Z., Shang, Q., Li, G., Jin, Z., Jiao, W.: Self-planning code generation with large language models. ACM Trans. Softw. Eng. Methodol. **33**(7) (2024). https://doi.org/10.1145/3672456
13. Kuckreja, K., Danish, M.S., Naseer, M., Das, A., Khan, S., Khan, F.S.: GeoChat: Grounded large vision-language model for remote sensing. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recogni-

tion, pp. 27831–27840. IEEE Computer Society, Los Alamitos, CA, USA (2024). https://doi.org/10.1109/CVPR52733.2024.02629
14. Li, X., Wen, C., Hu, Y., Yuan, Z., Zhu, X.X.: Vision-language models in remote sensing: Current progress and future trends. IEEE Geoscience and Remote Sensing Magazine **12**(2), 32–66 (2024). https://doi.org/10.1109/MGRS.2024.3383473
15. Li, X., Wen, C., Hu, Y., Zhou, N.: RS-CLIP: Zero shot remote sensing scene classification via contrastive vision-language supervision. International Journal of Applied Earth Observation and Geoinformation **124**, 103497 (2023). https://doi.org/10.1016/j.jag.2023.103497
16. Li, Z., Ning, H.: Autonomous GIS: The next-generation AI-powered GIS. International Journal of Digital Earth **16**(2), 4668–4686 (2023). https://doi.org/10.1080/17538947.2023.2278895
17. Liu, C., Chen, K., Zhang, H., Qi, Z., Zou, Z., Shi, Z.: Change-Agent: Toward interactive comprehensive remote sensing change interpretation and analysis. IEEE Transactions on Geoscience and Remote Sensing **62**, 1–16 (2024). https://doi.org/10.1109/TGRS.2024.3425815
18. Liu, F., Liu, Y., Shi, L., Huang, H., Wang, R., Yang, Z., Zhang, L.: Exploring and evaluating hallucinations in LLM-powered code generation. CoRR **abs/2404.00971** (2024). https://doi.org/10.48550/ARXIV.2404.00971
19. Liu, H., Li, C., Li, Y., Lee, Y.J.: Improved baselines with visual instruction tuning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 26286–26296 (2024). https://doi.org/10.1109/CVPR52733.2024.02484
20. Liu, J., Chen, Y., Liu, M., Peng, X., Lou, Y.: STALL+: Boosting LLM-based repository-level code completion with static analysis. CoRR **abs/2406.10018** (2024). https://doi.org/10.48550/ARXIV.2406.10018
21. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. In: Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS). Curran Associates Inc., Red Hook, NY, USA (2023). https://doi.org/10.5555/3666122.3669119
22. Singh, S., Fore, M., Stamoulis, D.: GeoLLM-Engine: A realistic environment for building geospatial copilots. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, pp. 585–594 (2024). https://doi.org/10.1109/CVPRW63382.2024.00063
23. Team, C.: CodeGemma: Open code models based on Gemma. CoRR **abs/2406.11409** (2024). https://doi.org/10.48550/ARXIV.2406.11409
24. Wang, X., Chen, Y., Yuan, L., Zhang, Y., Li, Y., Peng, H., Ji, H.: Executable code actions elicit better LLM agents. In: Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., Berkenkamp, F. (eds.) Proceedings of the 41st International Conference on Machine Learning (ICML 2024), Proceedings of Machine Learning Research, vol. 235, pp. 50208–50232. PMLR (2024). https://doi.org/10.5555/3692070.3694124
25. Wu, J., Gan, W., Chao, H.C., Philip, S.Y.: Geospatial big data: Survey and challenges. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **17**, 17007–17020 (2024). https://doi.org/10.1109/JSTARS.2024.3438376
26. Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., et al.: Qwen2 technical report. CoRR **abs/2407.10671** (2024). https://doi.org/10.48550/ARXIV.2407.10671

27. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: ReAct: Synergizing reasoning and acting in language models. In: International Conference on Learning Representations (ICLR 2023). OpenReview.net, Kigali, Rwanda (2023)
28. Zhan, Y., Xiong, Z., Yuan, Y.: SkyEyeGPT: Unifying remote sensing vision-language tasks via instruction tuning with large language model. CoRR **abs/2401.09712** (2024). https://doi.org/10.48550/ARXIV.2401.09712
29. Zhang, K., Li, J., Li, G., Shi, X., Jin, Z.: CodeAgent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In: Ku, L.W., Martins, A., Srikumar, V. (eds.) Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 13643–13658. Association for Computational Linguistics, Bangkok (2024). https://doi.org/10.18653/v1/2024.acl-long.737
30. Zhang, S., Zhang, C., Hu, Y., Shen, H., Liu, K., Ma, Z., Zhou, F., Zhang, W., He, X., Lin, D., et al.: CIBench: Evaluating your LLMs with a code interpreter plugin. CoRR **abs/2407.10499** (2024). https://doi.org/10.48550/ARXIV.2407.10499
31. Zhang, Y., Wei, C., He, Z., Yu, W.: GeoGPT: An assistant for understanding and processing geospatial tasks. International Journal of Applied Earth Observation and Geoinformation **131**, 103976 (2024). https://doi.org/10.1016/j.jag.2024.103976
32. Zhuo, T.Y., Vu, M.C., Chim, J., Hu, H., Yu, W., Widyasari, R., Yusuf, I.N.B., Zhan, H., He, J., Paul, I., et al.: BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. (2025).