An LLM Agent for Automatic Geospatial Data Analysis

YUXING CHEN, WEIJIE WANG, SYLVAIN LOBRY, and CAMILLE KURTZ

Large language models (LLMs) are being used in data science code generation tasks, but they often struggle with complex sequential tasks, leading to logical errors. Their application to geospatial data processing is particularly challenging due to difficulties in incorporating complex data structures and spatial constraints, effectively utilizing diverse function calls, and the tendency to hallucinate less-used geospatial libraries. To tackle these problems, we introduce GeoAgent, a new interactive framework designed to help LLMs handle geospatial data processing more effectively. GeoAgent pioneers the integration of a code interpreter, static analysis, and Retrieval-Augmented Generation (RAG) techniques within a Monte Carlo Tree Search (MCTS) algorithm, offering a novel approach to geospatial data processing. In addition, we contribute a new benchmark specifically designed to evaluate the LLM-based approach in geospatial tasks. This benchmark leverages a variety of Python libraries and includes both single-turn and multi-turn tasks such as data acquisition, data analysis, and visualization. By offering a comprehensive evaluation among diverse geospatial contexts, this benchmark sets a new standard for developing LLM-based approaches in geospatial data analysis tasks. Our findings suggest that relying solely on knowledge of LLM is insufficient for accurate geospatial task programming, which requires coherent multi-step processes and multiple function calls. Compared to the baseline LLMs, the proposed GeoAgent has demonstrated superior performance, yielding notable improvements in function calls and task completion. In addition, these results offer valuable insights for the future development of LLM agents in automatic geospatial data analysis task programming.

Additional Key Words and Phrases: Code Generation, Agent, LLM, Benchmark, Geospatial Data Analysis

1 Introduction

Large language models have demonstrated their potential to solve complex tasks in the geospatial domain [1–9]. These initial efforts focus on enabling LLM to utilize external tools, plugins, etc, for addressing geospatial data processing tasks. However, existing approaches are based on pre-defined low-level template-based prompts which often involve standalone third-party application programming interfaces (APIs) as foundational components for task completion. Among these works, Remote Sensing ChatGPT [9] and ChangeAgent [2] focus on utilizing independent remote sensing (RS) vision model APIs while GEOGPT [4] uses third-party API calls provided by the GIS system. These APIs, particularly based on given systems, provide single-line API calls for a specific task without a deep understanding of the dependencies of different functionalities. A more recent effort, GeoLLM-Engine [1], integrates several APIs into a sequential task on a real user interface platform. This is closer to real-world geospatial data processing tasks including the multi-steps of online data acquisition, data analysis, and visualization in one system. However, this work is still limited by the task-level APIs and lacks flexibility in complex and open-domain geospatial data analysis

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.
Authors' address: Yuxing Chen, chenyuxing16@mails.ucas.ac.cn; Weijie Wang; Sylvain Lobry; Camille Kurtz.

An LLM Agent for Automatic Geospatial Data Analysis

YUXING CHEN, WEIJIE WANG, SYLVAIN LOBRY, and CAMILLE KURTZ

大型语言模型（LLMs）正在被用于数据科学代码生成任务，但它们在处理复杂的顺序任务时经常遇到困难，导致逻辑错误。它们在地理空间数据处理中的应用尤其具有挑战性，因为难以结合复杂的数据结构和空间约束，有效利用多样的函数调用，并且倾向于幻想较少使用的地理空间库。为了解决这些问题，我们引入了GeoAgent，这是一个新的交互式框架，旨在帮助LLMs更有效地处理地理空间数据。 GeoAgent开创性地将代码解释器、静态分析和检索增强生成（RAG）技术集成到蒙特卡洛树搜索（MCTS）算法中，为地理空间数据处理提供了新的方法。此外，我们贡献了一个新的基准测试，专门设计用于评估基于LLM的方法在地理空间任务中的表现。该基准测试利用了多种Python库，包括单轮和多轮任务，如数据获取、数据分析和可视化。通过在不同的地理空间背景下提供全面的评估，该基准测试为基于LLM的方法在地理空间数据分析任务中的开发设定了新的标准。我们的发现表明，仅依赖LLM的知识不足以实现准确的地理空间任务编程，这需要连贯的多步骤过程和多个函数调用。与基线LLMs相比，所提出的GeoAgent表现出更优的性能，在函数调用和任务完成方面取得了显著的改进。此外，这些结果为未来基于LLM的代理在自动地理空间数据分析任务编程中的发展提供了宝贵的见解。

Additional Key Words and Phrases: Code Generation, Agent, LLM, Benchmark, Geospatial Data Analysis

1 Introduction

大型语言模型已经展示了其在地理空间领域解决复杂任务的潜力 [1–9]。这些初步努力集中在使 LLM 能够利用外部工具、插件等，以解决地理空间数据处理任务。然而，现有的方法基于预定义的低级模板提示，通常涉及独立的第三方应用程序编程接口（API）作为任务完成的基础组件。在这些工作中，Remote Sensing ChatGPT [9] 和 ChangeAgent [2] 专注于利用独立的遥感（RS）视觉模型 API，而 GEOGPT [4] 使用 GIS 系统提供的第三方 API 调用。这些 API，特别是基于给定系统的 API，为特定任务提供单行 API 调用，而无需深入了解不同功能之间的依赖关系。最近的一项工作，GeoLLM-Engine [1]，将多个 API 集成到一个真实用户界面平台上的顺序任务中。这更接近于现实世界中的地理空间数据处理任务，包括在线数据获取、数据分析和可视化的多步骤在一个系统中完成。然而，这项工作仍然受到任务级 API 的限制，并且在复杂和开放领域的地理空间数据分析场景中缺乏灵活性。为了解决这些挑战，自然语言处理（NLP）领域的最新努力，如 QwenAgent [10]、DS1000 [11]、ODEX [12]、BigCodeBench [13] 和 CIBench [14]，旨在利用 LLM 的代码生成能力来执行广泛的开放领域数据分析任务，利用任何可用的 Python 库。这些研究展示了 LLM 在代码执行环

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.
Authors' address: Yuxing Chen, chenyuxing16@mails.ucas.ac.cn; Weijie Wang; Sylvain Lobry; Camille Kurtz.

scenarios. To tackle these challenges, more recent efforts in natural language processing (NLP), such as QwenAgent [10], DS1000 [11], ODEX [12], BigCodeBench [13] and CIBench [14], aim to leverage the code generation capabilities of LLM for executing a wide range of open domain data analysis tasks, utilizing any available Python libraries. These studies illustrate the potential of LLMs in solving open-domain data analysis tasks within code execution environments and offer promising directions for applying similar approaches to geospatial data analysis in open-domain settings.

Geospatial data analysis tasks [15], consisting of data collection, data storage, data retrieval, data analysis, prediction, and visualization, present huge challenges for LLMs to follow complex instructions and accurately utilize specialized libraries and models. The tasks require a comprehensive understanding of instructions, including the inter-relationships among various inputs and outputs, systematic task decomposition, and the application of specialized domain expertise. Expert intervention is occasionally necessary for task decomposition and dynamic adjustments, which may involve assigning specific libraries. Developing executable solutions requires invoking multiple function calls from various libraries, which is often challenging for LLMs, particularly when they are under-trained with these libraries. General data analysis work typically relies on existing benchmarks [11–14], which primarily involve frequently used APIs from popular Python libraries. The efficacy of API usage in these studies has been saturated by the recently released models accompanied by extensive online documentation and limited to single-turn question assessments. These studies have a limited scope in geospatial data analysis, which often requires using several geospatial Python libraries in a given task with consecutive steps. For less frequent APIs from geospatial Python libraries, LLMs tend to be troubled by the API hallucinations [16], especially when under-trained on domain-specific knowledge. This difficulty has been highlighted in recent studies [13]. In NLP, retrieval-augmented generation (RAG) [17] has been proposed to incorporate domain-specific knowledge into task programming. Retrieval-augmented approaches [18] can enable models to base their predictions on external knowledge using various search techniques. Simultaneously, advancements in new architectures and techniques have empowered models to process extensive sequences of tokens [19]. However, effectively tackling specialized tasks, particularly in the geospatial domain, requires precise domain expertise, sequential multi-step reasoning, and iterative refinement guided by execution feedback. This feedback is particularly crucial. For example, the availability of online data may not be evident before task execution, highlighting the limitations of currently available general-purpose LLM coders. In addition, incorporating sequential reasoning capabilities often relies on Monte Carlo Tree Search (MCTS), which has demonstrated robust reasoning capabilities in many tasks where LLMs tackle complex mathematical problems [20, 21].

境中解决开放领域数据分析任务的潜力，并为在开放领域设置中应用类似方法进行地理空间数据分析提供了有希望的方向。

地理空间数据分析任务 [15]，包括数据收集、数据存储、数据检索、数据分析、预测和可视化，对 LLM 遵循复杂指令并准确利用专业库和模型提出了巨大挑战。这些任务需要全面理解指令，包括各种输入和输出之间的相互关系、系统任务分解以及应用专业领域知识。专家干预有时是必要的，用于任务分解和动态调整，这可能涉及分配特定库。开发可执行解决方案需要从各种库中调用多个函数，这通常对 LLM 来说具有挑战性，尤其是在这些库的训练不足时。一般的数据分析工作通常依赖于现有的基准 [11–14]，这些基准主要涉及常用 Python 库的 API。这些研究中的 API 使用效果已被最近发布的模型所饱和，这些模型附带了广泛的在线文档，并且仅限于单轮问题评估。这些研究在地理空间数据分析中的范围有限，通常需要在给定任务中使用多个地理空间 Python 库并进行连续步骤。对于地理空间 Python 库中较少使用的 API，LLM 往往会因 API 幻觉 [16] 而困扰，特别是在特定领域知识训练不足时。这一困难在最近的研究中得到了强调 [13]。在 NLP 中，检索增强生成（RAG）[17] 已被提出，以将特定领域的知识纳入任务编程。检索增强方法 [18] 可以使模型基于外部知识使用各种搜索技术进行预测。同时，新架构和技术的进步使模型能够处理大量标记序列 [19]。然而，有效解决特定任务，特别是在地理空间领域，需要精确的领域专业知识、顺序多步推理以及由执行反馈指导的迭代改进。这种反馈尤为重要。例如，在任务执行之前，可能无法明显看出在线数据的可用性，这突显了目前可用的通用 LLM 编码器的局限性。此外，结合顺序推理能力通常依赖于蒙特卡洛树搜索（MCTS），这在许多任务中已经展示了强大的推理能力，特别是在 LLM 解决复杂数学问题时 [20, 21]。除了独立的API调用之外，大多数现有的工作[3, 5–8]通过将输入数据与任务描述整合在一个操作中，利用大型视觉-语言模型（VLMs）[22]来解决地理空间数据分析任务。尽管这两种方法更为直接，但它们缺乏对单个结果的全面分析，也不利于将各种模型的输出和数据资源整合起来进行深入分析。在实际的地理空间任务中使用大型模型时，出现了几个关键挑战。首先，理解地理空间数据是复杂的，因为其各种模态携带的信息比RGB图像更丰富。VLMs为了有效初始化这些知识，需要大量的模型参数，这通常是不切实际的。其次，调用领域知识对于解决跨学科问题至关重要。例如，特定的RS数据的物理模型嵌入在线教程和地理空间库中，这些通常是当前VLMs无法访问的。从合成孔径雷达（SAR）数据中估计水分等地理空间任务，涉及多个物理模型函数，这不能通过VLMs中的简单文本提示来实现。第三，组合推理提出了另一个挑战，因为大多数地理空间任务涉及多个组件，如图像预处理和空间关系分析。与提示VLMs不同，LLM编码器通过整合不同的模型和促进全面分析，提供了更灵活的解决方案。最后，在线数据访问是一个重大限制；在遥感中，大量数据集存储在云中并通过API访问，但VLMs目前缺乏执行在线数据查询的能力。

为了解决上述挑战，我们介绍了一种基于LLM的代理，称为GeoAgent，它在一个MCTS算法中集成了代码解释器、静态分析和RAG。此外，我们提出了一种评估基准，涵盖了多样化的地理空

In addition to the standalone API call, most existing works [3, 5–8] addressing geospatial data analysis tasks through large Vision-Language Models (VLMs) [22] by integrating input data with task descriptions in a single operation. Although both methods are more direct, they lack comprehensive analysis of individual results and do not facilitate the integration of various models' outputs and data resources for in-depth analysis. Several key challenges arise when employing large models in practical geospatial tasks. First, understanding geospatial data is complex due to its various modalities, which carry richer information than RGB images. The requirement of substantial model parameters in VLMs to effectively initialize this knowledge is often impractical. Second, invoking domain knowledge is essential for addressing interdisciplinary topics. For example, the physics models of specific RS data are embedded within online tutorials and geospatial libraries, which are typically inaccessible to current VLMs. Geospatial tasks such as moisture estimation from Synthetic Aperture Radar (SAR) data, involve several physics model functions, which cannot be achieved through a simple text prompt in VLMs. Third, compositional reasoning presents another challenge, as most geospatial tasks involve multiple components, such as image preprocessing and spatial relation analysis. Unlike prompting VLMs, LLM coder offers more adaptable solutions by integrating different models and facilitating comprehensive analyses. Lastly, online data access is a significant limitation; in remote sensing, vast datasets are stored in the cloud and accessed through APIs, but VLMs currently lack the ability to perform online data querying.

To address the aforementioned challenges, we introduce an LLM-based agent, dubbed GeoAgent, which integrates a code interpreter, static analysis, and RAG within an MCTS algorithm. Additionally, we propose an evaluation benchmark encompassing diverse geospatial data analysis tasks, supplemented by an RAG library comprising geospatial library documents and solution examples. This agent fulfills human requirements by translating a given task into executable Python code, utilizing dynamic task adjustment and refinement through the MCTS. During the code generation process, RAG invokes external knowledge into the LLM, particularly when employing less common geospatial libraries. While RAG enhances task-specific coding proficiency by integrating corresponding library documents, it can negatively impact the performance of popular Python modules due to occasional irrelevant augmentations from suboptimal retrievers. The process of dynamic refinement necessitates LLM iteratively improving code generation based on the feedback, especially when tasks involve multiple consecutive steps that build upon each other. This method identifies dependencies among subtasks and dynamically refines them using execution feedback within an MCTS framework, ensuring that each code segment is logically consistent and well-developed in relation to 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 MCTS, tailored for geospatial data analysis. This integration enhances

An LLM Agent for Automatic Geospatial Data Analysis • 3

间数据分析任务，并补充了一个包含地理空间库文档和解决方案示例的RAG库。该代理通过将给定的任务翻译成可执行的Python代码来满足人类需求，利用MCTS进行动态任务调整和优化。在代码生成过程中，RAG将外部知识引入LLM，特别是在使用不太常见的地理空间库时。虽然RAG通过整合相应的库文档增强了特定任务的编码能力，但由于次优检索器的偶尔无关增强，它可能会对流行Python模块的性能产生负面影响。动态优化的过程需要LLM根据反馈迭代改进代码生成，特别是在任务涉及多个连续步骤时。这种方法通过在MCTS框架内使用执行反馈来识别子任务之间的依赖关系并动态优化它们，确保每个代码段在逻辑上一致且与先前步骤紧密相关。以下概述了我们的主要贡献：

- 我们介绍了一种新颖的LLM代理，该代理在MCTS中集成了外部知识检索器、代码解释器和静态分析，专为地理空间数据分析而设计。这种集成增强了顺序任务编程中的问题解决和逻辑能力。该代理在Jupyter Notebook环境中运行，允许用户与模型进行迭代交互，优化任务执行。这确保了与Python库和最佳实践的合规性，同时各种库的API令牌在远程服务器上安全管理，减轻了终端用户获取多个令牌的负担。
- 我们介绍了GeoCode，这是一个基于执行的基准测试，包含超过18,000个单轮和1,356个多轮地理空间数据分析任务，涉及来自28个广泛使用的库的2,313个函数调用，涵盖8个任务类别。GeoCode 特有两个不同的评估模型：单轮任务评估，评估函数调用的准确性及单个任务的成功率；多轮任务评估，关注任务完成率。后者以两种模式运行：自动迭代改进顺序和对失败子任务的人工干预。
- 我们在GeoCode基准上评估了各种大型语言模型（LLM）。尽管通用的LLM编码器可能会生成不完整或不兼容的工作流程，GeoAgent实现了更高的通过率和任务完成率，展示了卓越的性能。这一成功归因于其在MCTS框架内有效处理专业Python库的能力，并由执行反馈指导。本研究为LLM编码器在地理空间数据处理中的表现提供了更准确的评估，并指出了自动地理空间数据分析的光明未来。

本文其余部分结构如下。

2 2

介绍了基于LLM的代码生成和LLM增强的地理空间数据分析的相关工作。

3 3

通过描述GeoAgent的框架、API检索和MCTS上的动态细化来介绍所提出的方法。

4 4

描述了实验设置、基准统计、RAG库、LLM和基线设置以及评估指标。

problem-solving, and logical capabilities in sequential task programming. This agent operates within a Jupyter Notebook environment, allowing users to interact with the model iteratively, refining and optimizing task execution. This ensures compliance with Python libraries and best practices, while API tokens for various libraries are securely managed on a remote server, reducing the burden on end-users to acquire multiple tokens.

- 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 features two distinct evaluation models: single-turn task evaluation, which assesses function call accuracy and individual task success, and multi-turn task evaluation, which focuses on task completion rates. The latter operates in two modes: automatic iterative refinement sequentially and human intervention for failed subtasks.
- We evaluate various LLMs on the GeoCode benchmark. While general-purpose LLM coders may produce incomplete or incompatible workflows, GeoAgent achieves higher pass and task completion rates, demonstrating superior performance. 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 automatic geospatial data analysis.

The rest of this paper is organized as follows. Section 2 presents the related works on LLM-based code generation and LLM-empowered geospatial data analysis. Section 3 introduces the proposed method by describing the framework of GeoAgent, API retrieval, and dynamic refinement on MCTS. The descriptions of the experimental setup, benchmark statistics, RAG library, LLM, and baseline settings as well as the evaluation metrics in Section 4. Evaluation results obtained on function call performance, task-level performance, as well as discussion, are illustrated in Section 5. Finally, Section 6 concludes the paper.

2 Related Works

2.1 LLM-based Code Generation

LLMs exhibit a strong ability to generate code based on the pre-training of large-volume code sets. Although these LLM coders have proven highly effective in generating standalone functions, they face challenges dealing with interrelated tasks. To address these challenges, researchers have proposed several strategies to enhance the programming capabilities of LLMs, including instruction tuning [23], self-debugging [24, 25], and in-context learning [26, 27]. More recent pre-trained LLM coders have leveraged instruction tuning further for repository-level coding task [28, 29]. Instruction-tuned LLMs generally excel at producing code snippets that are well-aligned with specific natural language (NL) task instructions. Despite these

5 5

展示了在函数调用性能、任务级性能方面的评估结果以及讨论。最后，

6 6

对全文进行了总结。

7 Related Works

7.1 LLM-based Code Generation

LLMs 展现出基于大量代码集预训练生成代码的强大能力。尽管这些 LLM 编码器在生成独立函数方面表现出色，但在处理相互关联的任务时面临挑战。为了解决这些挑战，研究人员提出了几种策略来增强 LLM 的编程能力，包括指令调优 [23]、自我调试 [24, 25] 和上下文学习 [26, 27]。最近的预训练 LLM 编码器进一步利用了指令调优，以应对仓库级别的编码任务 [28, 29]。经过指令调优的 LLM 通常在生成与特定自然语言（NL）任务指令高度一致的代码片段方面表现出色。尽管有这些改进，LLM 仍然在需要多步骤逻辑的编程任务中遇到困难，特别是在与数据科学处理管道的集成方面。当前的努力 [14, 30–32] 专注于提高 LLM 在数据科学应用中的性能，并通过调用错误回溯和分析来实现基于代码的推理能力。例如，将代码解释器 [14] 和静态分析 [32] 纳入工作流程中是值得注意的。例如，RepairAgent [33] 利用静态分析提供基于 LLM 的修复见解，而 CoderGen [32] 则利用静态分析验证变量和语句。此外，*STALL*$^+$ [34] 使用静态分析生成上下文信息，以进行基于 LLM 的代码迭代修复。这些研究强调了将静态分析与 LLM 结合用于编程任务的好处。在执行反馈的背景下，CodeAct [31] 通过与 Python 解释器的多轮交互和来自静态分析的反馈，执行并动态改进代码生成。该迭代框架中的解释器能够管理符号计算、逻辑推理和精确的数值操作，其反馈有助于提高循环一致性。在检索增强的 LLM 编码器 [16, 18, 26, 27] 领域，将 LLM 与最新的外部数据库链接可以提高代码生成的精度，特别是在利用不太流行且不断发展的库来解决特定挑战时。然而，我们缺乏一个地理空间 RAG 数据库来提示地理空间任务的编程。另一个重要的方向是将外部工具的独立 API 集成到处理循环中。例如，Schick 等人提出了 ToolFormer [35]，它使用额外的信息来指导 LLM 如何调用现有 API。随后，Zhang 等人开发了 ToolCoder [36]，以在代码生成过程中对 LLM 进行 API 利用的微调。尽管 ToolCoder 在代码生成中有效地处理了独立 API 的调用，但未能充分解决涉及开放库的开放域任务。此外，Liu 等人 [37] 发现，在最先进的 LLM 编码器中，API 幻觉占所有幻觉的 15%，尤其是在不太常见的 API 中。

improvements, LLMs still face difficulties in programming tasks requiring multi-step logic, particularly in integrating with data science processing pipelines. Current efforts [14, 30–32] focus on boosting the LLMs' performance in data science applications and enabling code-based reasoning capabilities by invoking error traceback and analysis. For instance, the incorporation of code interpreters [14] and static analysis [32] within the workflow is noteworthy. For example, RepairAgent [33] utilizes static analysis to provide LLM-based repair insights, while CoderGen [32] employs static analysis to verify variables and statements. Additionally, *STALL*$^+$ [34] uses static analysis to generate contextual information for iterative LLM-based code repairs. These studies underscore the benefits of combining static analysis with LLMs for coding tasks. In the context of execution feedback, CodeAct [31] executes and dynamically improves code generation through multi-turn interactions with a Python interpreter and feedback from static analysis. The interpreter within this iterative framework is capable of managing symbolic computations, logical reasoning, and precise numerical operations, with its feedback contributing to the enhancement of loop consistency. In the realm of retrieval-augmented LLM coders [16, 18, 26, 27], linking LLMs to up-to-date external databases improves the precision of code generation, particularly when leveraging less popular and continuously evolving libraries to address specific challenges. However, we lack a geospatial RAG database to prompt programming for geospatial tasks. Another important direction is to integrate standalone APIs from external tools into the processing loop. For instance, Schick et al. propose ToolFormer [35], which uses extra information to instruct LLMs on how to invoke existing APIs. Zhang et al. subsequently developed ToolCoder [36] to finetune LLMs on API utilization during code generation. Although ToolCoder effectively handles the invocation of standalone APIs within code generation, it does not adequately address open-domain tasks involving open libraries. Additionally, Liu et al. [37] have identified that API hallucinations account for up to 15% of all hallucinations in state-of-the-art LLM coders, particularly with less common ones.

2.2 Geospatial Analysis with Large Language Models

Researchers [38–45] have explored the incorporation of LLMs into the processing and analysis of geospatial data. The initial efforts have focused on employing LLMs with text-based inputs such as Geographic Information System (GIS) tasks [40–42]. For instance, Li et al, [38] have tested the spatial semantic reasoning ability of ChatGPT [46] on geospatial tasks including toponym recognition, location description, and time series forecasting. This study indicates that LLMs are capable of comprehending and parsing geospatial data processing and analytical tasks based solely on their intrinsic knowledge. While this approach leverages the spatial analytical potential of LLMs, it is constrained by the lack of direct access to visual information, which is often crucial in geospatial analysis. The preliminary attempt at bridging the gap between visual features and the semantic reasoning capabilities of LLMs is large VLMs [43–45, 47], which incorporate LLMs into RS

An LLM Agent for Automatic Geospatial Data Analysis • 5

7.2 Geospatial Analysis with Large Language Models

研究人员 [38–45] 探索了将大语言模型（LLMs）纳入地理空间数据的处理和分析中。最初的尝试集中在使用基于文本的输入，如地理信息系统（GIS）任务 [40–42]，来应用 LLMs。例如，Li 等人 [38] 测试了 ChatGPT [46] 在地理空间任务中的空间语义推理能力，包括地名识别、位置描述和时间序列预测。这项研究表明，LLMs 仅凭其内在知识就能理解和解析地理空间数据处理和分析任务。虽然这种方法利用了 LLMs 的空间分析潜力，但受限于无法直接访问视觉信息，而视觉信息在地理空间分析中通常至关重要。初步尝试将视觉特征与 LLMs 的语义推理能力结合起来的是大型视觉-语言模型（VLMs）[43–45, 47]，这些模型将 LLMs 纳入遥感（RS）任务，如 RS 图像描述、视觉问答（VQA）和视觉定位。开创性工作 RSGPT [43] 构建了一个高质量的人工标注 RS 图像描述数据集，推动了 VLMs 在 RS 领域的发展。此外，RSGPT 引入了一个专门为 RS 图像描述和 VQA 设计的基于 GPT 的模型。为了更好地利用 LLMs 的能力，RS-Llava [45] 创建了 RS-instructions 数据集，这是一个全面的基准数据集，整合了四个与描述和 VQA 相关的单一任务数据集，纳入了 Llava [48] 框架。然而，LLMs 的文本输出往往未能完全满足用户的精确期望。除了生成文本输出外，GeoChat [47] 的最新进展还扩展了 VLMs 中的任务范围，包括指代表达、区域描述、图像描述和 VQA。尽管这些发展取得了良好的成果，但它们仍然局限于 RGB 图像，并且仅专注于计算机视觉任务。

尽管这些模型在各种地理空间数据分析任务中取得了显著成就，但利用这些模型解决跨开放领域任务中的复杂挑战仍然是一项重大难题。空间分析和多模态数据分析的挑战在 LLMs 和 VLMs 范式中提出了重大障碍。具体而言，地理空间数据处理领域的任务本质上依赖于多种专业工具、程序和多模态数据的使用。最近的尝试 [40, 49] 探索了将 LLMs 从纯粹的操作角色转变为决策者的潜力。这些方法不是直接解决任务，而是利用 LLMs 在语言理解和推理方面的卓越能力来评估需求并选择适当的专业工具。随后，使用这些选定的工具来执行任务。在这种背景下，一个早期的项目 GEOGPT [40] 提供了一个解决 GIS 任务的可行框架。 GeoGPT 将 LLMs 与 GIS 领域内已建立的工具集成，以应对各种地理空间挑战。 Change-Agent [2] 使用 LLM 根据用户指令调用分割和描述工具，以完成 RS 变化检测任务。此外，GeoLLM-Engine [1] 调用更多的地理空间 API 工具和外部知识库，使代理能够处理复杂的地理空间任务。这项工作明确了这些工具的具体要求，这些要求必须预先确定并封装到任务级 API 中。这限制了其在开放领域任务中的应用。此外，工具调用在顺序任务中整合任务级 API 方面也存在不足。

8 Methodology

8.1 Framework

在本节中，我们介绍了GeoAgent，这是一种专为处理和分析地理空间数据而设计的LLM代理，旨在满足研究人员的需求。GeoAgent的整体架构如图1所示，突出了两个组件：1) 任务编程：GeoAgent首

tasks, such as RS image captioning, Visual Question Answering (VQA), and visual grounding. Pioneering work RSGPT [43] built a high-quality human-annotated RS image captioning dataset that advances the development of VLMs in the RS domain. Additionally, RSGPT introduced a GPT-based model specifically designed for RS image captioning and VQA. To better leverage the ability of LLMs, RS-Llava [45] create the RS-instructions dataset, a comprehensive benchmark dataset that integrates four diverse single-task datasets related to captioning and VQA in a Llava [48] framework. However, the textual outputs of LLMs often fall short of meeting users' precise expectations. In addition to generating textual output, recent advancements in GeoChat [47] have incorporated a broader range of tasks within VLMs, encompassing referring expression, region captioning, image description, and VQA. Although these developments have yielded good results, they remain constrained to RGB images and are exclusively focused on computer vision tasks.

Despite the remarkable achievements of these models in various geospatial data analysis tasks, leveraging these models to address complex challenges across open-domain tasks remains a significant difficulty. The challenges associated with spatial analysis and multimodal data analysis present significant hurdles within the LLMs and VLMs paradigm. Specifically, tasks within the geospatial data processing field inherently rely on the utilization of multiple professional tools, procedures, and multimodal data. Recent attempts [40, 49] explore the potential of converting LLMs from a purely operational role to a decision-maker. Rather than directly addressing tasks, these approaches capitalize on the exceptional capabilities of LLMs in language comprehension and reasoning to assess requirements and select appropriate professional tools. Subsequently, the tasks are carried out using these selected tools. In this context, a prior initiative known as GEOGPT [40] offers a viable framework for addressing GIS tasks. GeoGPT integrates LLMs with established tools within the GIS field to tackle a range of geospatial challenges. Change-Agent [2] uses an LLM to follow user instructions to call segmentation and captioning tools for the RS change detection task. In addition, GeoLLM-Engine [1] calls more geospatial API tools and external knowledge bases, enabling agents to handle complex geospatial tasks. This work delineates the precise requirements for these tools, which must be pre-determined and wrapped into a task-level API. This limits the application on open-domain tasks. Furthermore, tool calling defeats on the integration of task-level APIs in a sequential task.

3 Methodology

3.1 Framework

In this section, we present GeoAgent, an LLM agent designed for the processing and analysis of geospatial data, tailored to meet the needs of researchers. The overall architecture of GeoAgent is illustrated in Fig. 1, which highlights two components: 1) Task programming: GeoAgent starts by leveraging parameterized

6 • Chen and Wang, et al.

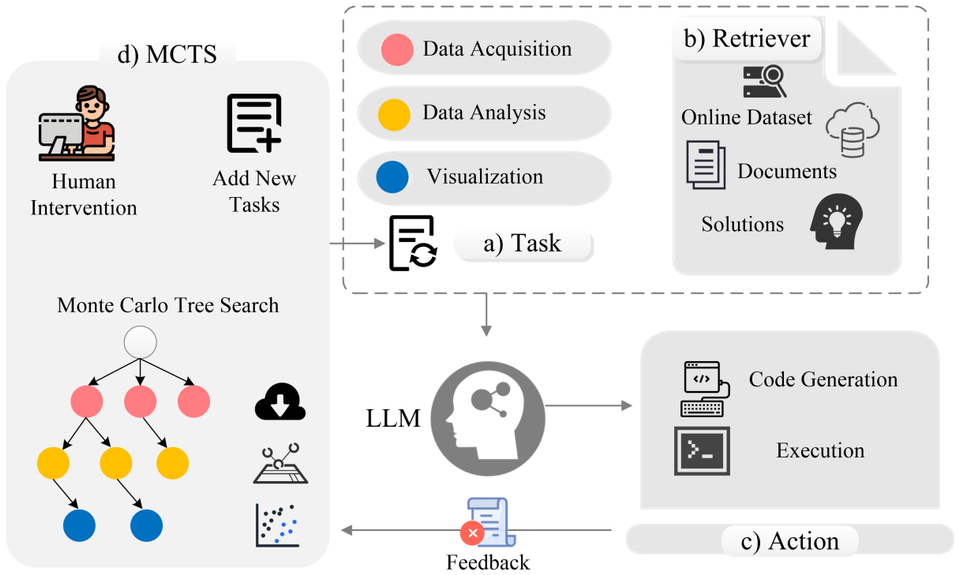

Fig. 1. GeoAgent: 一个地理空间数据分析任务编程代理。该代理由四个核心组件组成：**b) 检索器**，从提供的文档数据中检索与任务相关的内容，包括Python库文档、在线教程和函数级别的解决方案；**c) 行动**，这是一个集成了代码解释器和静态分析的执行环境，可以对生成的代码提供反馈并建议潜在的修复方案；**d) MCTS**，通过迭代调整和优化，探索和评估多个可能的代码候选方案，以在每一步选择最有希望的解决方案；**LLMs**，既作为代码生成器，又作为智能推理器，提出代码解决方案并迭代地诊断和修复错误。

先利用LLM的参数化知识根据任务指令生成代码。当提到特定的Python库时，GeoAgent从RAG数据库（见第3.2节）中检索相关API，并根据检索到的项目生成代码。 2) 基于MCTS的任务细化：GeoAgent在MCTS框架内执行并收集执行反馈，MCTS迭代地探索和评估多个代码候选方案，以优化选择最有希望的解决方案，而LLM则作为智能推理器，诊断和修复错误。值得注意的是，RAG使GeoAgent能够与不同地球科学领域的各种Python库接口。这一功能使GeoAgent能够将外部知识纳入特定领域的任务中。虽然RAG为特定任务提供了必要的信息，GeoAgent仍然需要动态调整，例如改进初始提示和纠正任何生成失败的代码。在本研究中，我们将LLM与MCTS集成，以促进任务编程过程中的动态调整（见第3.3节）。我们的自完善算法在搜索树框架内采用迭代细化过程，实现逐步改进。

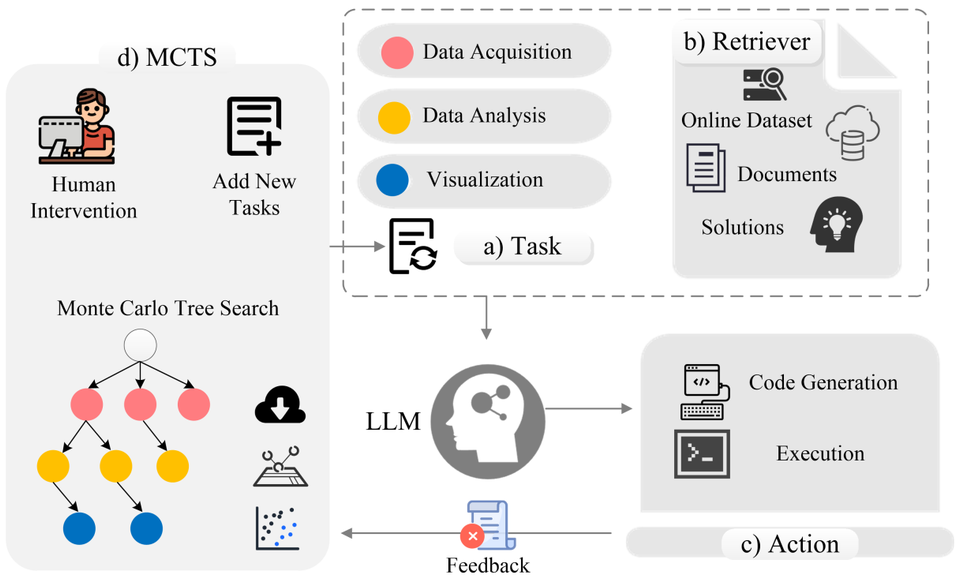

Fig. 1. GeoAgent: A geospatial data analysis task programming agent. This agent comprises four integral components: **b) Retriever**, which retrieves task-relevant items from provided document data, including Python library documents, online tutorials, and function-level solutions; **c) Action**, which is an execution environment integrated with a code interpreter and static analysis, provides feedback on generated code and suggests potential fixes; **d) 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; and **LLMs**, function as both the code generator and the intelligent reasoner to propose code solutions and iteratively diagnose and fix errors.

knowledge of LLMs to generate code based on task instructions. When a specific Python library is mentioned, GeoAgent retrieves relevant APIs from the RAG database (see Section 3.2) and generates codes conditioned on the retrieved items. 2) Task refinement on MCTS: GeoAgent executes and collects execution feedback within the MCTS framework, where MCTS iteratively explores and evaluates multiple code candidates to optimize the selection of the most promising solution, while LLMs serve as the intelligent reasoner, diagnosing and fixing errors. Notably, RAG enables GeoAgent to interface with various Python libraries of different geoscience domains. This capability allows GeoAgent to incorporate external knowledge for specialized domain tasks. While RAG provides essential information for specific tasks, GeoAgent still requires dynamic adjustments, such as improving initial prompts and correcting any failed generated code.

An LLM Agent for Automatic Geospatial Data Analysis • 7

8.2 API Retrieval

在本工作中，RAG 数据库整合了多样的 Python 库文档、在线教程和可供 LLMs 使用的功能级解决方案。 GeoAgent 利用外部 RAG 数据库根据提供的任务指令和先前的代码识别相关功能。一旦检索到合适的解决方案和最佳匹配的功能，GeoAgent 就利用 LLMs 的强大功能将给定的指令有效转换为可执行代码。此外，当生成的代码由于 API 幻觉而无法调用特定功能时，GeoAgent 会从指定的 Python 库中检索其使用详情。对于给定的任务 t_i，GeoAgent 使用检索模块从 RAG 数据库中提取相关功能信息和使用示例。 GeoAgent 使用现成的嵌入模型作为选择器。检索过程涉及三个步骤：首先，计算给定任务描述 t_i 和检索到的功能文档的嵌入特征。其次，计算这些嵌入之间的余弦相似度。最后，返回最相关的 k 个功能，表示为 $A_i = \langle Func., Usage \rangle$，为给定任务 t_i 向 LLMs 提供上下文信息。

8.3 Dynamic Refinement on MCTS

GeoAgent 采用 MCTS 框架 [50] 以迭代方式优化顺序任务中的每个子任务。 MCTS 使用树结构，其中节点表示状态 s，边表示动作 a。算法从根节点 s^0 开始，探索状态空间以识别具有最高奖励 $r(s^n)$ 的终止状态 s^n。每个节点包含以下组件：访问次数 v：每个节点的访问次数；概率 p：从 LLMs 导出；状态-动作值 $Q(s,a)$：从状态 s 采取动作 a 获得的最大奖励。状态 s 的奖励 r 定义为成功通过的步骤比例。该算法优先访问具有较高奖励值的节点，因为这些节点表示高质量的生成。此外，它还探索访问次数较少的节点，以确保对状态空间中未充分探索的区域进行充分调查。

在这个框架中，GeoAgent 首先将自然语言指令转换为可执行代码，然后执行这些代码。然而，地理空间任务涉及广泛的操作和复杂的流程，使得一次性生成连贯的多步骤代码变得困难。这种复杂性需要对上下文有全面的理解，以及对每一步的持续反馈和调整。为了解决这一挑战，GeoAgent 在 MCTS 扩展过程中采用束搜索结合执行过滤进行子节点采样和提示更新。 MCTS 的整个过程如图 2 所示。

该过程从根节点开始，根节点从第一个子任务开始并依次通过前一个子任务。在选择阶段，利用概率上界 ($p-UCB$) 算法从根节点 s^0 选择合适的分支：

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

其中 I 是子节点的集合；$\beta(s^0)$ 是探索的权重，它取决于访问次数（v^0）和常数 c_{base} 和 c。

$$\beta(s^0) = \log\left(\frac{v^0 + c_{base} + 1}{c_{base}}\right) + c. \tag{2}$$

$p-UCB$ 算法包含一个探索项 $\beta(s^0)$，该探索项由 LLM 确定的序列得分的概率 p 加权，其中 c 值的增加会促进更多的探索。例如，GeoAgent 从根节点 s^0 开始，递归地选择子树，直到到达一个未被

In this study, we integrate LLMs with MCTS to facilitate dynamic adjustments (see Section 3.3) during task programming. Our self-refinement algorithm employs an iterative refinement process within a search tree framework, enabling a step-by-step improvement.

3.2 API Retrieval

In this work, the RAG database incorporates diverse Python library documents, online tutorials, and function-level solutions accessible to LLMs. GeoAgent utilizes the external RAG database to identify relevant functions based on the provided task instructions and previous codes. Once suitable solutions and best-fit functions are retrieved, GeoAgent leverages the power of LLMs to effectively transform given instructions into executable codes. In addition, when the generated code fails to call specific functions due to API hallucinations, GeoAgent retrieves their usage details from the specified Python library. For a given task t_i, GeoAgent employs the Retriever module to extract relevant function information and usage examples from the RAG database. GeoAgent utilizes the off-the-shelf embedding model as the selector. The retrieval process involves three steps: First, compute the embedding features of the given task description t_i and the retrieved function documents. Second, calculate the cosine similarity between these embeddings. Finally, the k most relevant functions are returned, denoted as $A_i = \langle Func., Usage \rangle$, providing contextual information of the given task t_i to LLMs.

3.3 Dynamic Refinement on MCTS

The GeoAgent employs an MCTS framework [50] to iteratively refine each subtask within a sequential task. MCTS utilizes a tree structure where nodes represent states s and edges denote actions a. The algorithm begins at the root node s^0 and explores the state space to identify the terminal state s^n with the highest reward $r(s^n)$. Each node comprises the following components: visited count v: the visited times of each node; probability p: derived from LLMs; state-action value $Q(s, a)$: the maximum reward obtained by taking action a from state s. The reward r of state s is defined as the proportion of successfully passed steps. This algorithm prioritizes visiting nodes with higher reward values, as these indicate high-quality generation. Additionally, it explores nodes with fewer visits to ensure under-explored areas of the state space are adequately investigated.

In this framework, GeoAgent first transforms NL instructions into executable codes and then executes them. Nevertheless, geospatial tasks involve extensive operations and intricate workflows, making it challenging to generate coherent multi-step code in one attempt. This complexity necessitates a comprehensive understanding of the context, as well as continuous feedback and adjustments of each step. To tackle this challenge, GeoAgent employs beam search combined with execution filtering in child node sampling and

8 • Chen and Wang, et al.

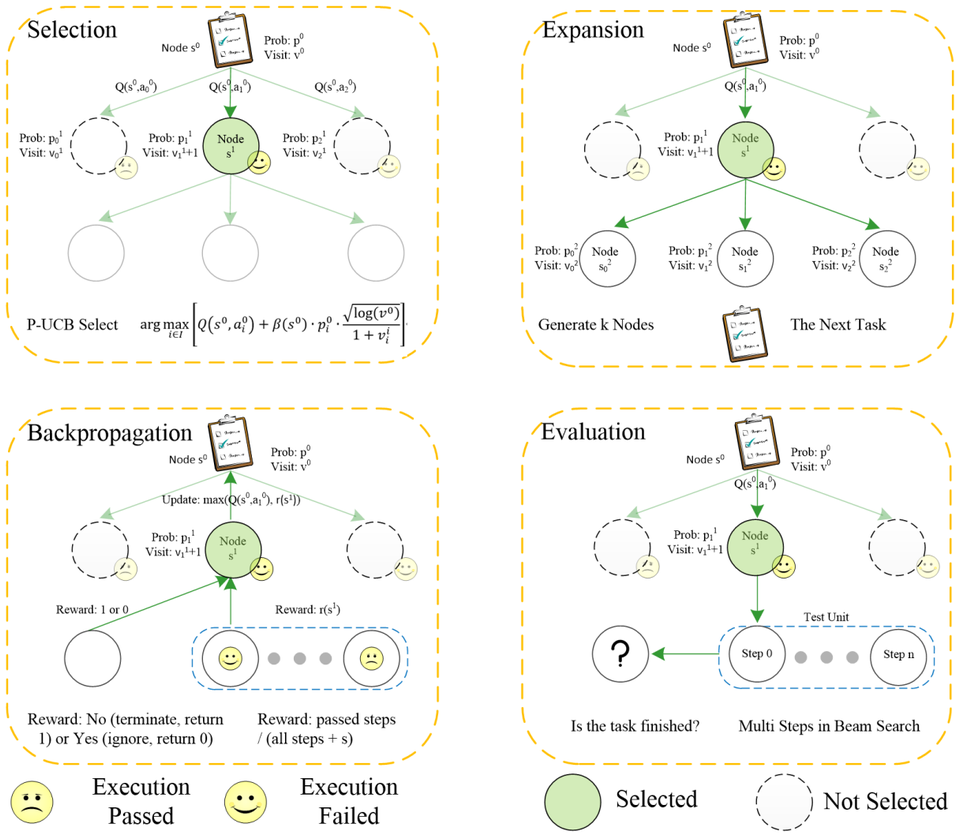

Fig. 2. 在地理空间数据分析任务编程中使用蒙特卡洛树搜索算法的示例。

扩展的节点。在这个过程中，$p-UCB$ 在已知良好的状态和较少访问的状态之间平衡探索。在扩展阶段，选择初始节点后，为后续步骤生成潜在代码，并将其作为新节点添加到子节点列表中，直到到达下一个子任务。在这个过程中，我们从提示生成的输出序列中采样 n 个输出序列。从这些输出序列中，只有排名前 $k = 3$ 的序列被返回。这些 k 个输出序列 (s_0^2, s_1^2, s_2^2) 随后被添加到当前代码 s^1 的子节点列表中。对于每个子节点，如果它们是非终止推理步骤，我们将其奖励值 Q 定义为 0；如果它们未能通过评估测试，则将其 Q 值定义为 -1。

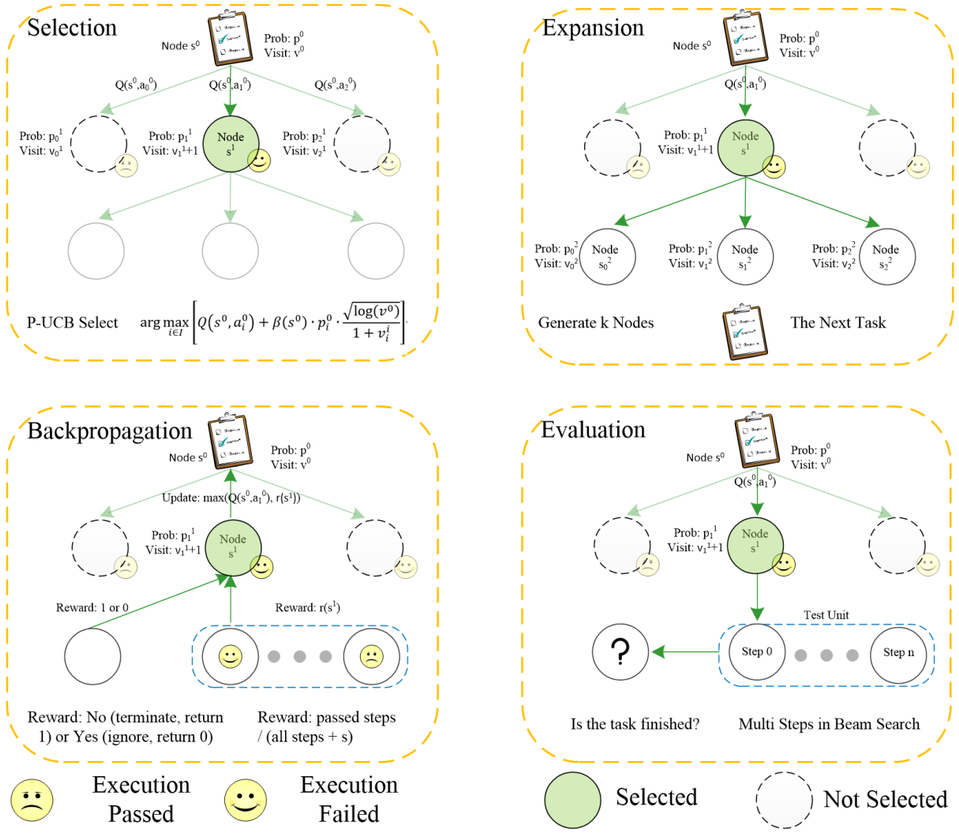

Fig. 2. Illustration of using the Monte Carlo tree search algorithm in geospatial data analysis task programming.

prompt updating throughout the MCTS expansion process. The entire process of MCTS is illustrated in Fig. 2.

The process begins at the root node, which initiates from the first subtask and progresses sequentially through previous ones. During the selection phase, the appropriate branch starting from root node s^0 is chosen utilizing the probabilistic Upper Confidence Bound ($p-UCB$) algorithm:

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

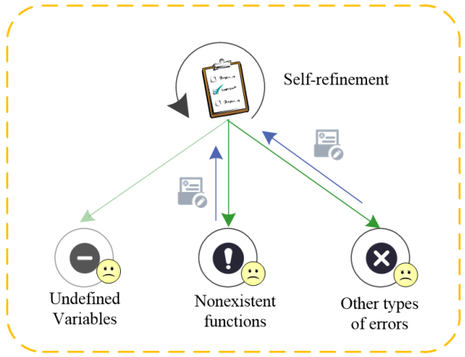

Fig. 3. The self-refinement algorithm in MCTS.

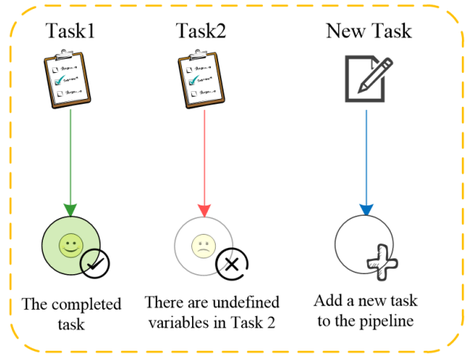

Fig. 4. 在MCTS中向任务池添加新任务。

在评估阶段，必须评估选定的节点 s^1，尽管该节点可能仅代表一个部分程序。部分程序的质量无法直接评估，因为其解决任务和遵循指令的能力仍然不确定。为了解决这个问题，LLM 从当前节点进行前瞻搜索，一次性生成多步代码（测试单元）。具体来说，给定当前节点的代码，我们首先收集由 LLM 生成的代码候选。然后，将生成的代码候选与先前节点的代码连接起来，以验证组合代码是否能通过执行测试，确保静态分析器和代码解释器未检测到任何解析或完成错误。最后，根据多步代码计算当前节点的奖励，该奖励以可执行步骤与所有步骤的比例来衡量。此外，当多步代码与当前节点完全重叠时，当前节点将获得 1 的奖励。此奖励 $r(s^1)$ 然后通过树进行反向传播，相应地更新其祖先节点的值。

所提出的MCTS框架结合了一个全面的错误回溯和分析机制，以动态地优化每个子任务。该机制如图3和图4所示，通过将最初生成的代码传递给代码解释器和静态分析工具来评估其功能。LLM常见的问题是调用未定义的变量。我们的方法通过移除使用未定义变量的子树，并在后续任务中使用该变量之前插入一个新任务来定义变量，从而解决此问题，如图4所示。如果当前节点中存在不正确的函数调用，我们使用静态分析工具（如Python Jedi [51]）来检索相关对象的所有可访问API（例如，方法名、变量名和参数名）。这些信息随后被纳入提示中，指导LLM选择合适的选项并相应地重新生成当前节点。对于其他类型的错误，解释器捕获错误回溯，提供导致失败的执行路径的详细记录。 LLM随后分析这个回溯，确定失败点并建议可能的修复方案，无论是语法错误、逻辑错误还是与Python库相关的问题。失败的代码片段、建议的修正和原始指令被重新引入到LLM中，随后生成修订版本。此过程将持续进行，直到代码成功执行或达到最大尝试次数。

然而，即使将静态分析和执行反馈整合到提示过程中，由于LLM推理的固有非确定性和有限的可解释性，确保任务完成的成功仍然具有挑战性。为了减少无限循环的风险，为每个任务设定了最大尝试次数。框架从错误回溯和实时反馈中学习的能力是GeoAgent与传统代码生成过程区别的

where I is the collection of children nodes; $\beta(s^0)$ is the weight for exploration, which depends on the number of visits (v^0) and constants c_{base} and c.

$$\beta(s^0) = \log\left(\frac{v^0 + c_{base} + 1}{c_{base}}\right) + c. \tag{2}$$

The $p - UCB$ algorithm incorporates an exploration term $\beta(s^0)$, weighted by the probability p of the LLM-determined sequence score, where an increased value of c promotes greater exploration. For instance, GeoAgent starts from the root node s^0 and selects subtrees recursively until it reaches a node that has not been expanded. In this process, $p - UCB$ balances the exploration between known-to-be-good states and less visited states. In the expansion phase, after selecting the initial node, potential codes for subsequent steps are generated and added as new nodes to the child node list until the next subtask is reached. In this process, we sample n output sequences generated from the prompt. From these output sequences, only top $k = 3$ sequences are returned. These k output sequences (s_0^2, s_1^2, s_2^2) are subsequently added to the children list of the current code s^1. For each child node, we define their reward value Q as 0 if they are non-terminal reasoning steps and Q value as -1 if they failed the evaluation test.

During the evaluation phase, the selected node s^1 must be assessed, despite the node potentially representing only a partial program. The quality of a partial program cannot be directly evaluated, as its ability to solve the task and adhere to instructions remains uncertain. To address this, the LLM does a look-ahead search from the current node by generating a multi-step code (test unit) in one attempt. Specifically, given the code of the current node, we first collect code candidates generated by LLMs. The generated code candidate is then concatenated with the code from previous nodes to verify whether the combined code can pass the execution test, ensuring that no parsing or completion errors are detected by the static analyzer and code interpreter. Finally, the reward of the current node is calculated based on the multi-step code, measured as the ratio of executable steps to all steps. Additionally, a reward of 1 is assigned to the current node when the multi-step code completely overlaps with the current node. This reward $r(s^1)$ is then backpropagated through the tree, updating the values of its ancestor nodes accordingly.

The proposed MCTS framework incorporates a comprehensive error traceback and analysis mechanism to dynamically refine each subtask. This mechanism, depicted in Fig. 3 and Fig. 4, operates by passing the initially generated code within a code interpreter and static analysis tools to assess its functionality. It is common for LLMs to invoke undefined variables. Our methodology addresses this issue by removing the subtree that uses undefined variables and inserting a new task into the workflow to define the variable before its use in subsequent tasks, as shown in Fig. 4. If incorrect function calls are present within the current nodes, we utilize static analysis tools, such as Python Jedi [51], to retrieve all accessible APIs (e.g., method names, variable names, and parameter names) of the relevant object. This information is then

显著特征。 MCTS中的自我调试可以显著减少科学家进行地理空间任务编程所需的时间和精力。同时，引入了手动编辑以增强顺序任务的完整性和准确性。如果问题未解决，GeoAgent将生成一份详细的任务、代码和遇到的错误报告。在此阶段，引入人工干预，允许根据打印的错误信息对失败的子任务进行手动编辑。在后续尝试中，这些人工修改将作为上下文输入被纳入。此外，GeoAgent提供了利用聊天历史来改进不成功的代码生成并将其纳入新轮次的功能。通过在MCTS框架中结合自我调试和手动编辑策略，GeoAgent旨在生成不仅正确而且高效和可维护的代码，反映与任务相关的最佳实践和外部知识。

9 Experimental Setup

9.1 Benchmark Construction

在本节中，我们概述了构建基准的过程，该过程涉及三个主要步骤：代码收集、代码重构和指令生成。与简短的代码练习不同，构建一个高质量的、基于执行的基准，专注于地理空间编程任务，并非易事。主要挑战在于难以自然地获取带有详细说明的自包含地理空间编程任务。虽然许多 GitHub 仓库包含现实的源代码，但这些仓库中的脚本通常需要跨文件的函数。此外，地理空间任务编程场景的多样性进一步复杂化了这一努力。同时，为了评估 GeoAgent 在地理空间任务上的性能，评估基准必须能够回答几个关键问题：1）LLM 是否可以遵循复杂的地理空间任务指令？2）基于 LLM 的地理空间任务编程是否可以从外部知识中受益？3）LLM 是否可以通过集成 MCTS 来提高任务完成度？

本工作特别关注 Python 代码，因为 Python 在地理空间数据分析中广泛使用。虽然本任务中采用的方法是可适应的，并且可以在未来的基准中扩展到更多的编程语言。 GeoAgent 收集使用地理空间 Python 库的代码片段，并为每个代码段创建相应的任务描述。通常，仓库级别的代码用于生成函数级别的编程任务，但这种方法通常需要跨文件的信息，这可能难以隔离成独立的脚本。鉴于 Python 脚本中用于地理空间数据处理和分析的资源有限，GeoAgent 利用各种 Python 库的教程并借助 LLM 构建代码-指令对。值得注意的是，我们将 GeoCode 分为两个子集：基于 Google Earth Engine 库的任务（GeoCode-GEE）和其他库的任务（GeoCode-Other）。

9.2 Benchmark Statistics

所提出的GeoCode基准测试，总结在表 1 中，涵盖了8个关键领域的28个广泛使用的Python库。这些库在地理空间数据分析任务中被广泛使用，GeoCode中的代码片段通常涉及来自多个库的组合函数调用，因此需要相当的组合推理能力。此外，表 2 将GeoCode与可执行的Python编程基准测试进行了比较，突出了这些基准测试中引用的库和函数调用。值得注意的是，GeoCode包括18148个单轮任务和1356个多轮任务，以及来自28个外部库的2313个函数调用，反映出与其他基准测试相比更广泛

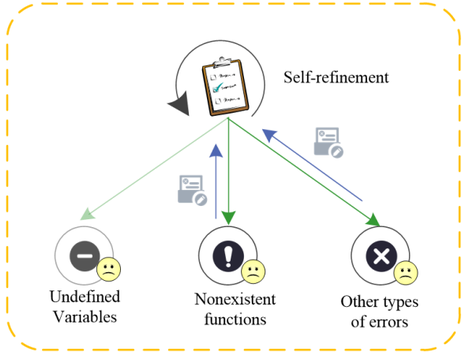

Fig. 3. The self-refinement algorithm in MCTS.

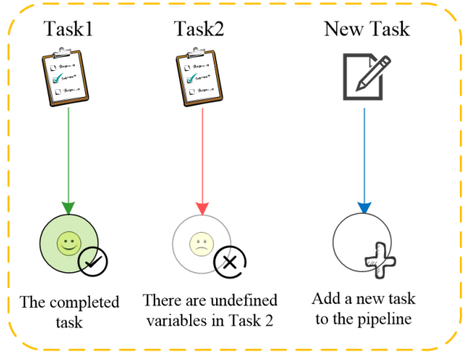

Fig. 4. Addition of new tasks to the task pool in MCTS.

incorporated into the prompt, guiding the LLM to select the appropriate option and regenerate the current node accordingly. For other error types, the interpreter captures an error traceback, providing a detailed record of the execution path that led to the failure. This traceback is then analyzed by the LLM, which identifies the point of failure and suggests potential fixes, whether they are syntactic errors, logical errors, or issues with Python libraries. The failed code snippet, alongside the suggested corrections and the original instructions, is reintroduced into the LLM, which subsequently generates a revised version. This process continues until the code successfully executes or the maximum number of attempts is reached.

However, even with the integration of static analysis and execution feedback into the prompting process, ensuring the success of task completion remains challenging due to the inherent non-determinism and limited interpretability of LLM inference. To mitigate the risk of an endless loop, a maximum number of attempts for each task is established. The capacity of the framework to learn from error tracebacks and real-time feedback is a distinctive feature that differentiates GeoAgent from traditional code generation processes. The self-debugging in MCTS can significantly reduce the time and effort required for scientists to conduct geospatial task programming. Meanwhile, manual editing is also introduced to enhance the completeness and accuracy of sequential tasks. If the problem goes unsolved, GeoAgent generates a report detailing the task, code, and encountered errors. At this stage, human intervention is introduced, allowing for manual edits to failed subtasks based on the printed error information. In subsequent attempts, these human modifications are incorporated as contextual inputs. Additionally, GeoAgent offers functionalities to utilize chat histories for improving unsuccessful code generations and include them in the new round. By incorporating both self-debugging and manual editing strategies within the MCTS framework, GeoAgent

An LLM Agent for Automatic Geospatial Data Analysis • 11

表 1. GeoCode 中 Python 库的说明。此表根据各自的领域对 GeoCode 中使用的 Python 库进行分类。每个领域都与特定的任务相关联。

Domain	Library
Data Acquisition & 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

表 2. Python 编程基准统计：通过外部库使用、函数调用频率、任务数量和任务类型进行分析。

Benchmark	Ext. 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

的多样性。这表明GeoCode提供了多样化的任务提示，涉及复杂的指令，需要开发具有复杂实现逻辑的解决方案代码。

9.3 RAG Library

仅基于参数化知识使用LLM生成代码是具有挑战性的，因为它无法跟上不断演变的公共库和数据集的步伐。在本研究中，我们整合了三种关键资源，以使模型能够检索相关上下文：特定解决方案、在线教程和库文档。 1) 特定解决方案：我们编制了一份包含特定功能的基本解决方案的文档，其中包括自然语言描述和代码解决方案。 2) 在线教程：我们从各个网站汇总了教程，每一页都包含代码片段和文本解释，主题从变化检测到计算机视觉不等。 3) 库文档：我们收集了表 1 中列出的所有Python库的官方文档，这对于在线教程稀缺的库尤其有价值。然而，当前的检索系统在来源有

aims to produce code that is not only correct but also efficient and maintainable, reflecting best practices and external knowledge relevant to the task.

4 Experimental Setup

4.1 Benchmark Construction

In this section, we outline the process of constructing the benchmark, which involves three primary steps: code collection, code refactoring, and instruction generation. Unlike the short code exercise, constructing a high-quality, execution-based benchmark that focuses on geospatial programming tasks is non-trivial. The primary challenge lies in the difficulty of naturally sourcing self-contained geospatial programming tasks with detailed instructions. While many GitHub repositories contain realistic source code, the scripts inside these repositories often require cross-file functions. Additionally, the diversity of geospatial task programming scenarios further complicates this effort. Meanwhile, to assess the performance of GeoAgent on geospatial tasks, the evaluation benchmark must be able to reply to several key questions: 1) Can LLMs follow complex geospatial task instructions? 2) Can LLMs-based geospatial task programming benefit from external knowledge? 3) Can LLMs enhance task completion by integrating MCTS?

This work specifically focuses on Python code due to its widespread use in geospatial data analysis. Whereas the methodology employed in this task is adaptable and can be expanded with more programming languages in future benchmarks. GeoAgent gathers code snippets that utilize geospatial Python libraries and creates corresponding task descriptions for each code segment. Typically, repository-level code is used to generate function-level programming tasks, but this approach often necessitates cross-file information, which can be challenging to isolate into standalone scripts. Given the limited resources for geospatial data processing and analysis within Python scripts, GeoAgent employs tutorials from various Python libraries and leverages LLMs to construct code-instruction pairs. 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

The proposed GeoCode benchmark, summarized in Table 1, encompasses 28 widely-used Python libraries across 8 key domains. These libraries are extensively utilized in geospatial data analysis tasks, and the code snippets in GeoCode often involve combined function calls from multiple libraries, thereby requiring considerable compositional reasoning ability. Additionally, Table 2 compares GeoCode with executable Python programming benchmarks, highlighting the libraries and function calls referenced in these benchmarks. Notably, GeoCode includes 18148 single-turn tasks and 1356 multi-turn tasks as well as 2313 function calls from 28 external libraries, reflecting a broader diversity compared to other benchmarks. This indicates that

限的共享词汇时难以获取有用的上下文。并且，代码生成器在整合不相关上下文时往往无法提高性能。为了解决这些挑战，我们实施了增强元数据过滤器的RAG。

9.4 Experimental Settings

在本研究中，我们选择了Llama 3.1 (8B) [52] 作为支持地理空间数据分析编程的LLM。此外，我们在分析不同模型大小和专门化LLM的影响时，还考虑了CodeGemma 2 (7B) [53] 和 Phi3.5 mini (3.8B) [54]，以及 Qwen2 (7B) [55]。推理使用单个NVIDIA GeForce RTX 4090 GPU进行，包括指令生成的初始阶段和随后的代码生成阶段。参数配置包括将top-p值设置为0.9，温度参数设置为0.6，最大令牌限制为2048。窗口大小设置为至少32k令牌，从而支持广泛上下文的检索。对于检索模块，我们使用BBAI-embedding-001模型生成高质量的文本和代码嵌入，以实现高效的相似性计算和检索过程。

9.5 Metrics

在本节中，我们介绍用于评估函数调用和任务完成的评估指标。为了评估生成代码中函数调用的性能，我们采用通常用于多标签分类任务的评估指标：精确度（Precision）、准确率（Accuracy）、召回率（Recall）、F1分数和汉明距离（Hamming distance）。我们从生成的代码中提取引用的函数作为预测标签。由于函数列表作为标签集，预测可能是完全正确、部分正确或完全错误。这与基于示例的多标签分类 [56] 评估一致，其中部分正确性被考虑在内。我们采用基于示例的策略，该策略涉及计算每个任务的预测标签和实际标签之间的平均差异，然后计算测试集中所有任务的这些差异的平均值。设 T 为一个包含 n 个任务（X_i, Y_i）的函数调用评估数据集，$1 \leq I \leq n$ 且 $x_i \in X, y_i \in Y = \{0,k\}^k$，其中包含 k 个类别。设 h 表示大语言模型（LLM），$Z_i = h(x_i) = \{0,1\}^k$ 表示 LLM 为任务 x_i 预测的标签集。每个实例的准确率定义为正确预测的标签数与总标签数（包括预测和实际标签）的比例。总体准确率是所有实例的这些比例的平均值：

$$A = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \tag{3}$$

召回率（Recall）定义为正确预测的标签数占实际标签总数的比例，跨所有实例的平均值：

$$R = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|Z_i|} \tag{4}$$

精确度定义为正确预测的标签数占总预测标签数的比例，跨所有实例的平均值：

$$P = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|Y_i|} \tag{5}$$

F1分数，即精确率和召回率的调和平均值，计算如下：

$$F_1 = \frac{1}{n}\sum_{i=1}^{n}\frac{2\,|Y_i \cap Z_i|}{|Y_i| + |Z_i|} \tag{6}$$

表 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 & 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

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

Benchmark	Ext. 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

GeoCode offers diverse task prompts, which involve complex instructions and require the development of solution code with intricate implementation logic.

4.3 RAG Library

Generating code solely with LLMs based on parametric knowledge is challenging, as it cannot keep pace with evolving public libraries and datasets. In this work, we integrate three key resources to enable models to retrieve relevant contexts: specific solutions, online tutorials, and library documents. 1) Specific solutions: We compiled a document containing basic solutions to specific functionalities consisting of natural language descriptions and code solutions. 2) Online tutorials: We aggregated tutorials from various websites, each page comprising code snippets and textual explanations on topics ranging from change detection to computer vision. 3) Library documents: We collected official documentation for all Python libraries shown in Table 1,

表 3. 在所有基准测试中，函数调用性能与零（@0）和三个（@3）检索项的比较。

	Accuracy		Recall		Precision		F1		Hamming Loss	
	@0	@3	@0	@3	@0	@3	@0	@3	@0	@3
DS-1000	0.379	0.395	0.423	0.443	0.495	0.523	0.448	0.472	0.153	0.153
ODEX	0.489	0.528	0.508	0.549	0.512	0.555	0.509	0.551	0.277	0.264
BigCodeBench	0.369	0.611	0.411	0.675	0.524	0.805	0.449	0.724	0.108	0.101
CIBench	0.543	0.571	0.604	0.634	0.658	0.713	0.621	0.662	0.139	0.138
GeoCode-GEE	0.604	0.535	0.640	0.562	0.656	0.581	0.645	0.568	0.199	0.216
GeoCode-Others	0.610	0.670	0.651	0.708	0.673	0.729	0.659	0.716	0.190	0.170

Hamming loss 衡量标签被错误预测或相关标签被遗漏的频率。它通过总类别数和样本数进行归一化。

$$\text{Hamming loss} = \frac{1}{kn}\sum_{i=1}^{n}\sum_{l=1}^{k}\left[I\left(l \in Z_i \wedge l \notin Y_i\right) + I\left(l \notin Z_i \wedge l \in Y_i\right)\right] \tag{7}$$

其中 I 是指示函数，较小的汉明损失表示更好的性能。为了分析 RAG 对函数调用的影响，我们分别比较了使用零个和三个检索项时的评估指标。

在评估代码补全性能时，考虑了通过率和任务完成率（成功执行的步骤数除以总步骤数）等指标，其中通过率主要用于单轮任务评估，而任务完成率是多轮任务的主要衡量标准。例如，如果一个任务包含 10 个步骤，我们向模型提供 10 个提示，使其依次生成 10 个代码块。完成率是根据连续成功的执行情况计算的，如果前五个步骤通过，则完成率为 50%。此外，pass@k 指标（在这种情况下 k=1）是一种广泛认可的评估措施，用于评估单轮任务的执行正确性。对于多轮任务，顺序任务被转换为单轮任务以方便评估。

10 Evaluation

在本节中，我们展示了所有基准测试的结果，包括所提出的GeoCode-GEE和GeoCode-Others。GeoCode-GEE专门针对GEE环境内的任务，而GeoCode-Others需要使用多个Python库。此外，我们还包括与一般数据科学任务相关的基准测试，如DS-1000、ODEX、BigCodeBench和CIBench。为了解决基准构建中提出的问题，我们首先使用精确度、召回率、F1分数和汉明损失指标分析定性函数调用。随后，我们使用通过率和任务完成率评估GeoAgent在任务完成中的表现。

10.1 Function Call Performance

我们首先评估了使用Llama3.1和RAG增强的Llama3.1获得的结果来考虑函数调用性能。所有指标都在涉及零（@0）和三个（@3）最相关检索项的设置下报告。如表3所示，大多数基准测试从检索

which is particularly valuable for libraries with scarce online tutorials. However, current retrieval systems struggle to source useful contexts, especially when shared vocabulary is limited. And, code generators often fail to improve performance when integrating irrelevant contexts. To address these challenges, we implement RAG enhanced with metadata filters.

4.4 Experimental Settings

In this work, we select Llama 3.1 (8B) [52] as the LLM to support geospatial data analysis programming. Additionally, we consider CodeGemma 2 (7B) [53] and Phi3.5 mini (3.8B) [54], Qwen2 (7B) [55] 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 parameter configurations include 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 BBAI-embedding-001 model to generate high-quality embeddings for both text and code, enabling efficient similarity computations and retrieval processes.

4.5 Metrics

In this section, we introduce the evaluation metrics utilized for assessing the function call and task completion. To evaluate the performance of function calls within generated code, we employ evaluation metrics typically used in multilabel classification tasks: Precision, Accuracy, Recall, F1 score, and Hamming distance. We extract the functions referenced in the generated code as predicted labels. Since the function list serves as a set of labels, predictions may be entirely correct, partially correct, or entirely incorrect. This aligns the evaluation with the exampled-based multilabel classification [56], where partial correctness is taken into consideration. We utilize an example-based strategy, which involves calculating the average difference between predicted and actual labels for each task and then averaging these differences across all tasks in the test set. Let T be a function call evaluation dataset consisting n tasks (X_i, Y_i), $1 \leq I \leq n$ and $x_i \in X, y_i \in Y = \{0, k\}^k$, with k classes. Let h denote the 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. Accuracy for each instance is defined as the proportion of correctly predicted labels to the total labels (both predicted and actual ones) for one instance. The overall accuracy is the average of these proportions across all instances:

$$A = \frac{1}{n} \sum_{i=1}^{n} \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \tag{3}$$

到的函数文档的包含中受益，从而在所有指标上提高了性能。最显著的改进是在BigCodeBench上，@3的结果在F1分数上提高了0.275，在汉明损失上减少了0.007。这可以归因于Llama3.1中对不太流行的Python库的训练不足，其中从库文档中获取的函数定义和使用示例可以显著提高代码生成。相反，其他基准测试的F1分数改进最小，GeoCode-Others显示增加了约0.06，CIbench和ODEX增加了约0.04，而DS1000几乎没有增益。这种有限的改进可能是因为DS1000主要涉及流行的Python库，这些库在最近发布的Llama3.1模型中已经饱和。值得注意的是，GeoCode-GEE的性能在包含RAG后出现了下降。这种下降可能是由于GEE Python库在Llama3.1训练数据中过度表示，导致饱和，而RAG在上下文中引入了额外的噪声，损害了生成。这表明，当处理过多的上下文时，大语言模型可能不够稳健，导致在处理不同上下文输入时出现不希望的行为。重要的是要注意，大多数地理空间Python库在未来的大语言模型版本中不太可能饱和，因为可用于进一步训练的在线文档有限。这突显了在大语言模型的上下文中纳入库文档的重要性，以确保最佳函数的准确调用。此外，检索准确性在函数调用性能中起着关键作用，需要进一步探索改进的RAG算法。在这项工作中，我们选择仅在提到库时实现RAG，旨在减轻无关检索项的影响。

10.2 Task level performance

为了评估 GeoAgent 在任务导向代码生成中的表现，我们进行了实验，涉及每个基准的 100 个单轮任务和三个多轮任务基准的 10 个顺序任务，鉴于我们有限的计算资源。我们首先分别使用四种不同的大语言模型（即 Llama3.1、CodeGemma、Phi3.5-mini、Qwen2）单独和结合 GeoAgent 评估了单轮任务的通过率（pass@1）。对于单轮任务评估（如表 4 所示），Llama3.1 赋能的 GeoAgent 似乎因多步推理中引入的噪声而受到影响。除了 DS1000 之外的所有基准中，它仅将 GeoCode-GEE 的通过率提高了 13%，将 ODEX 的通过率提高了 17%，与单独使用基线 Llama3.1 相比。值得注意的是，通过在循环中添加额外的步骤可以避免这种噪声。然而，如果我们考虑所有四个不同的大语言模型在所有基准上的表现，单独使用大语言模型的通过率较低，而 GeoAgent 在大多数基准上提高了通过率。这是因为大语言模型有时会因调用错误或不存在的函数而失败。 GeoAgent 通过在初始尝试失败时允许多次尝试来解决这个问题。特别是对于 GeoCode-GEE，GeoAgent 为 Llama3.1 提高了 13% 的通过率，为 CodeGemma 提高了 6% 的通过率，为 Phi3.5-mini 提高了 16% 的通过率，为 Qwen2 提高了 20% 的通过率。在不同的大语言模型中，CodeGemma 在 GeoCode 基准上表现出最高的性能，其中在 GeoCode-GEE 上达到了 86% 的 pass@1，在 GeoCode-Others 上达到了 59%。这表明代码指令调优与大语言模型的性能之间存在明显的相关性，代码后训练通常能取得更好的结果。在所有基准中，DS-1000 在所有大语言模型和 GeoAgent 上的通过率都明显较低。这个问题是由于提示中缺少必要的代码，导致执行失败。然而，GeoAgent 仍然提高了其性能。至于 ODEX、BigCodeBench 和 CiBench，它们的通过率改进不如提出的 GeoCode 显著。此外，所有大语言模型和 GeoAgent 在 GeoCode-GEE 上的表现都优于 GeoCode-Others，这表明基于多库的任务编程比基于单库的任务更

Recall is defined as the proportion of correctly predicted labels to the total number of actual labels, averaged across all instances:

$$R = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|Z_i|} \tag{4}$$

Precision is defined as the proportion of correctly predicted labels to the total number of predicted labels, averaged across all instances:

$$P = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|Y_i|} \tag{5}$$

The F1 score, which is the harmonic mean of precision and recall, is calculated as follows:

$$F_1 = \frac{1}{n}\sum_{i=1}^{n}\frac{2\,|Y_i \cap Z_i|}{|Y_i| + |Z_i|} \tag{6}$$

Hamming loss measures the frequency with which a label is incorrectly predicted or a relevant label is missed. It is normalized by the total number of classes and examples.

$$\text{Hamming loss} = \frac{1}{kn}\sum_{i=1}^{n}\sum_{l=1}^{k}\left[I\left(l \in Z_i \wedge l \notin Y_i\right) + I\left(l \notin Z_i \wedge l \in Y_i\right)\right] \tag{7}$$

where I is the indicator function and a smaller Hamming loss indicates better performance. To analyze the impact of RAG on function calls, we compare the evaluation metrics under conditions where zero and three retrieved items are used, respectively.

For assessing performance in code completion, metrics such as pass rate and task completion rate (the number of successfully executed steps achieved divided by the total number of steps) are considered, where pass rate is mainly for single-turn task evaluation and task completion rate being the primary measure of multi-turn tasks. For example, if a task consists of 10 steps, we provide the model with 10 prompts, allowing it to generate 10 code blocks sequentially. The completion rate is then calculated based on the consecutive successful executions, with a 50% completion rate if the first five steps are passed. Furthermore, the pass@k metric (with k=1 in this case) is a widely recognized evaluation measure that assesses the execution correctness of single-turn tasks. For multi-turn tasks, sequential tasks were converted into single-turn tasks to facilitate this evaluation.

5 Evaluation

In this section, we present the evaluation results for all benchmarks along with the proposed GeoCode-GEE and GeoCode-Others. GeoCode-GEE is focused exclusively on tasks within the GEE environment, while GeoCode-Others requires the use of multiple Python libraries. Additionally, we include benchmarks relevant to general data science tasks, such as DS-1000, ODEX, BigCodeBench, and CIBench. To address the questions

表 4. Llama3.1 (8B)、CodeGemma (7B)、Phi3.5 mini (3.8B) 和 Qwen 2 (7B) 在所有基准测试中的代码生成通过率 (Pass@1)。

	Llama3.1 (8B)		CodeGemma (7B)		Phi3.5 mini (3.8B)		Qwen 2 (7B)	
	LLM	GeoAgent	LLM	GeoAgent	LLM	GeoAgent	LLM	GeoAgent
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

表 5. Llama3.1 (8B)、CodeGemma (7B)、Phi3.5 mini (3.8B) 和 Qwen 2 (7B) 在所有基准测试中的函数调用性能（F1 分数）。

	LLama3.1 (8B)		CodeGemma (7B)		Phi3.5 mini (3.8B)		Qwen 2 (7B)	
	LLM	GeoAgent	LLM	GeoAgent	LLM	GeoAgent	LLM	GeoAgent
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

具挑战性。然而，通过率本身并不能完全反映任务级别的性能。因此，我们展示了四种不同大语言模型和 GeoAgent 在单轮任务中函数调用的 F1 分数（表 5）。可以看出，与原始大语言模型相比，GeoAgent 提高了函数调用性能。然而，我们在某些情况下观察到一些下降，特别是在原始大语言模型已经表现出高性能的情况下。

对于顺序任务评估（表 6），我们分别评估了在自调试和人工干预模式下所有基准的完成率。当问题未解决时，引入人工干预以推进任务到下一步。与单轮任务不同，顺序任务在逐步生成代码时对大语言模型的推理能力提出了挑战。与单轮任务评估一致，GeoAgent 在自动和人工干预模式下所有基准的完成率都更高。与原始的 Llama3.1 相比，GeoAgent 在 CIBench 上的完成率分别提高了 20% 和 6%，在 GeoCode-GEE 上提高了 9% 和 10%，在 GeoCode-Others 上提高了 48% 和 22%。可能的原因是单独使用 Llama3.1 生成的代码经常出现未定义变量的使用，导致执行失败。在这种情况下，GeoAgent 的动态调整将未定义变量重构为新的子任务，并在当前任务循环中更新失败的尝试。

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

	Accuracy		Recall		Precision		F1		Hamming Loss	
	@0	@3	@0	@3	@0	@3	@0	@3	@0	@3
DS-1000	0.379	0.395	0.423	0.443	0.495	0.523	0.448	0.472	0.153	0.153
ODEX	0.489	0.528	0.508	0.549	0.512	0.555	0.509	0.551	0.277	0.264
BigCodeBench	0.369	0.611	0.411	0.675	0.524	0.805	0.449	0.724	0.108	0.101
CIBench	0.543	0.571	0.604	0.634	0.658	0.713	0.621	0.662	0.139	0.138
GeoCode-GEE	0.604	0.535	0.640	0.562	0.656	0.581	0.645	0.568	0.199	0.216
GeoCode-Others	0.610	0.670	0.651	0.708	0.673	0.729	0.659	0.716	0.190	0.170

posed in the benchmark construction, we first analyze qualitative function calls using Precision, Recall, F1 score, and Hamming loss metrics. Subsequently, we assess GeoAgent's performance in task completion using the pass rate and task completion metrics.

5.1 Function Call Performance

We first evaluate the function call performance considering the results obtained using Llama3.1 and RAG powered-Llama3.1. All metrics are reported under settings that involve zero (@0) and three (@3) most relevant retrieval items. As one can see in Table 3, most benchmarks benefit from the inclusion of retrieved function documents, leading to enhanced performance on all metrics. The most significant improvement is observed on BigCodeBench, where the result of @3 achieves a 0.275 increase in F1 score and a 0.007 reduction in hamming loss. This can be attributed to the less popular Python libraries being under-trained in Llama3.1, where the function definitions and usage examples from library documentation can substantially improve code generation. Conversely, the improvement in F1 score for other benchmarks is minimal, with GeoCode-Others showing an increase of around 0.06, CIbench and ODEX seeing a rise of around 0.04, and DS1000 exhibiting almost no gain. This limited improvement is likely due to DS1000 predominantly involving popular Python libraries, which are already saturated in the recently released Llama3.1 model. Notably, the performance on GeoCode-GEE shows a decline of including RAG. This decline is likely because the GEE Python library is overly represented in the Llama3.1 training data, leading to saturation while the RAG includes additional noise in context and harms the generation. This suggests that LLMs may not be sufficiently robust when dealing with excessive context, resulting in undesired behavior when processing varied context inputs. It is important to note that most geospatial Python libraries are unlikely to become saturated in future LLM releases, as there are limited online documents available for further training. This highlights the importance of incorporating library documents in the context of LLMs to ensure the accurate

16 • Chen and Wang, et al.

表 6. Llama3.1 和 GeoAgent 在自我调试（Aut.）和人工干预（Hum.）模式下在所有基准测试中的代码生成完成率（Complete@1）。*idx* 指的是任务索引，*steps* 表示任务的总步骤数。

complete@1	idx	steps	Aut. Llama3.1	Hum. Llama3.1	Aut. GeoAgent	Hum. GeoAgent
CIBench	0	6	0.50	0.66	0.17	0.83
	1	6	0.00	0.50	1.00	-
	2	7	1.00	-	1.00	-
	3	7	1.00	-	1.00	-
	4	6	1.00	-	1.00	-
	5	6	0.33	0.66	1.00	-
	6	6	1.00	-	1.00	-
	7	7	1.00	-	1.00	-
	8	7	0.14	0.86	0.43	0.43
	9	7	1.00	-	1.00	-
GeoCode-GEE	0	16	0.19	0.81	0.19	0.88
	1	8	0.00	0.75	0.63	0.88
	2	13	0.08	0.69	0.23	0.85
	3	22	0.27	0.82	0.05	0.91
	4	12	1.00	-	0.42	0.83
	5	7	0.00	0.71	0.14	0.86
	6	11	0.27	0.73	0.09	0.82
	7	8	0.50	0.88	1.00	-
	8	11	0.55	0.73	0.45	0.82
	9	9	0.00	0.33	0.55	0.55
GeoCode-Others	0	4	0.00	0.50	1.00	-
	1	6	0.00	0.00	0.17	0.50
	2	9	0.00	0.78	0.33	0.78
	3	7	0.29	0.71	0.71	0.71
	4	8	0.13	0.25	0.25	0.25
	5	4	0.25	0.50	0.25	0.75
	6	8	0.00	0.00	0.00	0.00
	7	6	0.00	0.67	1.00	-
	8	9	0.00	0.67	1.00	-
	9	7	0.00	0.57	0.71	0.86

invocation of best-fit functions. Additionally, retrieval accuracy plays a crucial role in the performance of function calls, necessitating further exploration of improved RAG algorithms. In this work, we opt to implement RAG only when the library is mentioned, aiming to mitigate the impact of irrelevant retrieved items.

5.2 Task level performance

To assess GeoAgent's performance in task-oriented code generation, we conducted experiments involving 100 single-turn tasks of each benchmark and 10 sequential tasks of three multi-turn task benchmarks, given our limited computation resources. We first evaluated the pass rate (pass@1) of single-turn tasks under four different LLMs (i.e. Llama3.1, CodeGemma, Phi3.5-mini, Qwen2) using LLM alone and GeoAgent, respectively. For single-turn task evaluation (as shown in Table 4), the code generated by Llama3.1 empowered GeoAgent seems troubled by the noise introduced in the multiple-step inference. Across all benchmarks except DS1000, it only increases the pass rate of GeoCode-GEE by 13% and of ODEX by 17% compared with the use of baseline Llama3.1 alone. Notably, this noise can be avoided by adding an additional step in the loop. However, if we consider all four different LLMs over all benchmarks, the LLMs alone have a lower pass rate, while GeoAgent increases the pass rate on most benchmarks. This is because LLMs sometimes fail by calling incorrect or non-existent functions. GeoAgent addresses this issue by allowing multiple attempts to solve a task when the initial attempt fails. Specially for GeoCode-GEE, GeoAgent achieves a 13% improvement in pass rate for Llama3.1, a 6% improvement for CodeGemma, and a 16% improvement for Phi3.5-mini as well as an improvement of 20% for Qwen2. Among the different LLMs, CodeGemma demonstrates the highest performance on the GeoCode benchmark, where achieved the pass@1 by 86% on GeoCode-GEE and 59% on GeoCode-Others. This indicates a clear correlation between code instruction tuning and the performance of LLMs, with code post-training generally achieving better results. Among all benchmarks, DS-1000 shows a much lower pass rate overall LLMs and GeoAgents. This issue is due to the absence of necessary code in the prompt, which leads to failed execution. However, GeoAgent still improved its performance. As for ODEX, BigCodeBench, and CiBench, they show less pass rate improvement than the proposed GeoCode. Furthermore, the performance on GeoCode-GEE outperforms GeoCode-Others across all LLMs and GeoAgents, indicating that multi-library-based task programming is more challenging than single-library-based tasks. Whereas the pass rate alone does not fully capture task-level performance. Hence, we present the F1 score of function calling of single-turn tasks under four different LLMs and GeoAgents (Table 5). As one can see, GeoAgent improves function call performance compared to the vanilla LLM. However, we observed some declines in a few cases, particularly where the vanilla LLM had already achieved high performance.

An LLM Agent for Automatic Geospatial Data Analysis • 17

与自动模式相比，人工干预在 LLMs 单独使用和 GeoAgent 下分别在 CIBench 上提高了 20% 和 7%，在 GeoCode-GEE 上提高了 46% 和 47%，在 GeoCode-Others 上提高了 40% 和 14%。在自调试和人工干预模式下，几乎所有情况在人工专业知识的辅助下表现更好。尽管少数困难案例即使在人工干预下完成率也保持不变，例如 GeoCode-Others 中的第六个任务。这些观察结果表明，通过与人类互动，大语言模型可以取得更好的结果，这为将大语言模型整合到地理空间数据分析任务中以辅助人类提供了有希望的途径。在个别案例中，我们观察到实施 GeoAgent 后性能有所下降。这种下降归因于 GeoAgent 推理阶段引入的额外噪声，可以通过在过程中增加一个额外步骤来缓解。在三个基准中，GeoCode-Others 相比其他两个基准更具挑战性，因为其在自动模式下有 7 个任务的完成率为零。尽管如此，GeoAgent 显著提高了这些任务的通过率。相比之下，CIBench 是最简单的基准，10 个任务中有 6 个任务的完成率为 100%。

10.3 Discusssion

在构建我们的数据集时，我们考虑了各种用于开放领域地理空间数据分析任务的库。选择合适的库具有挑战性，需要广泛的专业知识。我们的研究表明，GEE库是唯一一个在开源社区中提供许多资源的库，而其他库大多仅提供使用示例。包括这些库需要大量的人力。为了确保我们实验的公平性和有效性，我们单独测试了GEE与其他库。我们从提供的代码片段生成了代码指令，尽管这些指令可能并不总是与代码完全匹配。此外，使用Llama3.1的提示生成强调了提示措辞的重要性，以生成准确、特定任务的代码。库和函数依赖关系的表示在模型的预训练阶段处理，但在提示中探索这些依赖关系是有限的。进一步研究在提示中表示这些依赖关系的多样化方法可能是未来研究的重要领域。

在GeoAgent中，仅使用Python解释器进行错误检测通常不够，甚至可能产生误导。在复杂的工作流程中，代码运行无误并不保证其正确性。我们的研究发现，不准确的执行反馈会限制动态调整的优势。目前，MCTS中的动态调整策略过于依赖执行反馈。为了解决这个问题，我们需要创建更灵活的在线评估策略。此外，过度依赖执行反馈和RAG在模型推理过程中增加了显著的成本。因此，将这些方法集成到MCTS框架时，我们必须考虑效率。一个可能的解决方案是在动态调整过程中实施选择性执行反馈，而不是在每个生成步骤中都使用它。

在生成代码的评估中，我们确定了四种常见的错误类型。首先，指令跟随错误发生在模型在特定领域知识方面训练不足时，导致难以遵循任务指令。其次，幻觉错误涉及错误的或不存在的函数调用和未定义的变量，通常是由使用较少见的地理空间Python库引起的。这些错误揭示了将LLM应用于开放领域地理空间任务的挑战。第三，信息不足错误发生在提示未提供足够或准确的信息时，LLM无法自行解决，需要人类帮助。最后，一般代码错误反映了LLM在有效生成代码方面的当前局限性。

表 4. Code generation pass rate (Pass@1) of the Llama3.1 (8B), CodeGemma (7B), Phi3.5 mini (3.8B) and Qwen 2 (7B) on all benchmarks.

	Llama3.1 (8B)		CodeGemma (7B)		Phi3.5 mini (3.8B)		Qwen 2 (7B)	
	LLM	GeoAgent	LLM	GeoAgent	LLM	GeoAgent	LLM	GeoAgent
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

表 5. Function call performance (F1 score) of the Llama3.1 (8B), CodeGemma (7B), Phi3.5 mini (3.8B) and Qwen 2 (7B) on all benchmarks.

	LLama3.1 (8B)		CodeGemma (7B)		Phi3.5 mini (3.8B)		Qwen 2 (7B)	
	LLM	GeoAgent	LLM	GeoAgent	LLM	GeoAgent	LLM	GeoAgent
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

For sequential task evaluation (Table 6), we assess the completion rates of all benchmarks under self-debugging and human intervention modes, respectively. When a problem remains unresolved, human intervention is introduced to advance the task to the next step. Unlike single-turn tasks, sequential tasks challenge the reasoning capabilities of LLMs when generating code step by step. Consistent with single-turn task evaluations, the GeoAgent achieves a higher completion rate on all benchmarks under both automatic and human intervention modes. Compared with the vanilla Llama3.1, GeoAgent brings an improvement of complete rate by 20% and 6% on CIBench, 9% and 10% on GeoCode-GEE, 48% and 22% on GeoCode-Others under self-debugging and human intervention modes, respectively. The possible reason is that the generated code using Llama3.1 alone often suffers from the use of undefined variables, leading to execution failures. In such cases, dynamic adjustment of GeoAgent refactors undefined variables into new subtasks and updates failed attempts in the current task loop. Compared with the automatic mode, human intervention brings an

18 • Chen and Wang, et al.

这项工作有两个主要局限性。首先，GeoAgent完全依赖于LLM的能力，而许多开源地理空间库可以提高其地理空间编码技能。其次，GeoCode的评估指标仅关注过程导向的评估。尽管代码可执行，但基于输出测量其性能是困难的，因为结果可能是多样和复杂的。

11 Conclusion

我们介绍了GeoAgent，这是一种创新的地理空间任务编程方法，旨在增强对广泛地理空间数据集的访问，并利用具有多样化且不断发展的Python库的大型语言模型（LLMs）实现自动化工作流程。GeoAgent利用LLMs的能力将任务转换为可执行单元，随后通过从知识数据库中检索相应的API（使用RAG）并动态地在MCTS中优化子任务。此外，我们开发了一个基准测试GeoCode，以评估所提出的框架在使用流行的地理空间Python库进行各种地理空间数据分析任务中的有效性。我们的实验结果表明，GeoAgent在GeoCode和现有基准测试中，无论是通用编程任务还是地理空间数据分析任务，都超过了LLM基线。研究结果表明，GeoAgent显著提高了地理空间任务的通过率和任务完成率。通过GeoAgent，我们展望了一个未来，高级辅助工具可以无缝访问相关的Python库和广泛的在线数据，为各种地理空间任务生成定制代码。我们希望这项工作能够促进地理空间数据在旨在实现社会利益和环境保护的研究中的应用。

Acknowledgements

我要向Laurent Wendling教授表示衷心的感谢，感谢他在本工作的初期讨论中提供的宝贵见解和指导。

References

[1] S. Singh, M. Fore, and D. Stamoulis, “Geollm-engine: A realistic environment for building geospatial copilots,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2024, pp. 585–594.

[2] C. Liu, K. Chen, H. Zhang, Z. Qi, Z. Zou, and Z. Shi, “Change-agent: Toward interactive comprehensive remote sensing change interpretation and analysis,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 62, pp. 1–16, 2024.

[3] K. Kuckreja, M. S. Danish, M. Naseer, A. Das, S. Khan, and F. S. Khan, “Geochat: Grounded large vision-language model for remote sensing,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2024, pp. 27 831–27 840.

[4] Y. Zhang, C. Wei, Z. He, and W. Yu, “Geogpt: An assistant for understanding and processing geospatial tasks,” *International Journal of Applied Earth Observation and Geoinformation*, vol. 131, p. 103976, 2024.

[5] W. Xu, Z. Yu, Y. Wang, J. Wang, and M. Peng, “Rs-agent: Automating remote sensing tasks through intelligent agents,” *arXiv preprint arXiv:2406.07089*, 2024.

[6] Y. Hu, J. Yuan, C. Wen, X. Lu, and X. Li, “Rsgpt: A remote sensing vision language model and benchmark,” *arXiv preprint arXiv:2307.15266*, 2023.

[7] Y. Bazi, L. Bashmal, M. M. Al Rahhal, R. Ricci, and F. Melgani, “Rs-llava: A large vision-language model for joint captioning and question answering in remote sensing imagery,” *Remote Sensing*, vol. 16, no. 9, p. 1477, 2024.

[8] Y. Zhan, Z. Xiong, and Y. Yuan, “Skyeyegpt: Unifying remote sensing vision-language tasks via instruction tuning with large language model,” *arXiv preprint arXiv:2401.09712*, 2024.

[9] H. Guo, X. Su, C. Wu, B. Du, L. Zhang, and D. Li, “Remote sensing chatgpt: Solving remote sensing tasks with chatgpt and visual models,” *arXiv preprint arXiv:2401.09083*, 2024.

表 6. Code generation complete rate (Complete@1) of the Llama3.1, GeoAgent under modes of self-debugging (Aut.) and Human Intervention (Hum.) across all benchmarks. *idx* refers to the task index, and *steps* denotes the total number of task steps.

complete@1	idx	steps	Aut. Llama3.1	Hum. Llama3.1	Aut. GeoAgent	Hum. GeoAgent
CIBench	0	6	0.50	0.66	0.17	0.83
	1	6	0.00	0.50	1.00	-
	2	7	1.00	-	1.00	-
	3	7	1.00	-	1.00	-
	4	6	1.00	-	1.00	-
	5	6	0.33	0.66	1.00	-
	6	6	1.00	-	1.00	-
	7	7	1.00	-	1.00	-
	8	7	0.14	0.86	0.43	0.43
	9	7	1.00	-	1.00	-
GeoCode-GEE	0	16	0.19	0.81	0.19	0.88
	1	8	0.00	0.75	0.63	0.88
	2	13	0.08	0.69	0.23	0.85
	3	22	0.27	0.82	0.05	0.91
	4	12	1.00	-	0.42	0.83
	5	7	0.00	0.71	0.14	0.86
	6	11	0.27	0.73	0.09	0.82
	7	8	0.50	0.88	1.00	-
	8	11	0.55	0.73	0.45	0.82
	9	9	0.00	0.33	0.55	0.55
GeoCode-Others	0	4	0.00	0.50	1.00	-
	1	6	0.00	0.00	0.17	0.50
	2	9	0.00	0.78	0.33	0.78
	3	7	0.29	0.71	0.71	0.71
	4	8	0.13	0.25	0.25	0.25
	5	4	0.25	0.50	0.25	0.75
	6	8	0.00	0.00	0.00	0.00
	7	6	0.00	0.67	1.00	-
	8	9	0.00	0.67	1.00	-
	9	7	0.00	0.57	0.71	0.86

An LLM Agent for Automatic Geospatial Data Analysis • 19

[10] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[11] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-t. Yih, D. Fried, S. Wang, and T. Yu, "Ds-1000: A natural and reliable benchmark for data science code generation," in *International Conference on Machine Learning*. PMLR, 2023, pp. 18 319–18 345.

[12] Z. Wang, S. Zhou, D. Fried, and G. Neubig, "Execution-based evaluation for open-domain code generation," *arXiv preprint arXiv:2212.10481*, 2022.

[13] T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul *et al.*, "Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions," *arXiv preprint arXiv:2406.15877*, 2024.

[14] S. Zhang, C. Zhang, Y. Hu, H. Shen, K. Liu, Z. Ma, F. Zhou, W. Zhang, X. He, D. Lin *et al.*, "Cibench: Evaluating your llms with a code interpreter plugin," *arXiv preprint arXiv:2407.10499*, 2024.

[15] J. Wu, W. Gan, H.-C. Chao, and S. Y. Philip, "Geospatial big data: Survey and challenges," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024.

[16] N. Jain, R. Kwiatkowski, B. Ray, M. K. Ramanathan, and V. Kumar, "On mitigating code llm hallucinations with api documentation," *arXiv preprint arXiv:2407.09726*, 2024.

[17] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, "Retrieval augmented language model pre-training," in *International conference on machine learning*. PMLR, 2020, pp. 3929–3938.

[18] Z. Z. Wang, A. Asai, X. V. Yu, F. F. Xu, Y. Xie, G. Neubig, and D. Fried, "Coderag-bench: Can retrieval augment code generation?" *arXiv preprint arXiv:2406.14497*, 2024.

[19] T. Li, G. Zhang, Q. D. Do, X. Yue, and W. Chen, "Long-context llms struggle with long in-context learning," *arXiv preprint arXiv:2404.02060*, 2024.

[20] D. Zhang, J. Li, X. Huang, D. Zhou, Y. Li, and W. Ouyang, "Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b," *arXiv preprint arXiv:2406.07394*, 2024.

[21] M. Liao, W. Luo, C. Li, J. Wu, and K. Fan, "Mario: Math reasoning with code interpreter output–a reproducible pipeline," *arXiv preprint arXiv:2401.08190*, 2024.

[22] X. Li, C. Wen, Y. Hu, Z. Yuan, and X. X. Zhu, "Vision-language models in remote sensing: Current progress and future trends," *IEEE Geoscience and Remote Sensing Magazine*, 2024.

[23] N. Muennighoff, Q. Liu, A. Zebaze, Q. Zheng, B. Hui, T. Y. Zhuo, S. Singh, X. Tang, L. Von Werra, and S. Longpre, "Octopack: Instruction tuning code large language models," *arXiv preprint arXiv:2308.07124*, 2023.

[24] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," *arXiv preprint arXiv:2304.05128*, 2023.

[25] N. Jiang, X. Li, S. Wang, Q. Zhou, S. B. Hossain, B. Ray, V. Kumar, X. Ma, and A. Deoras, "Training llms to better self-debug and explain code," *arXiv preprint arXiv:2405.18649*, 2024.

[26] J. Li, G. Li, C. Tao, H. Zhang, F. Liu, and Z. Jin, "Large language model-aware in-context learning for code generation," *arXiv preprint arXiv:2310.09748*, 2023.

[27] A. Patel, S. Reddy, D. Bahdanau, and P. Dasigi, "Evaluating in-context learning of libraries for code generation," *arXiv preprint arXiv:2311.09635*, 2023.

[28] F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J.-G. Lou, and W. Chen, "Repocoder: Repository-level code completion through iterative retrieval and generation," *arXiv preprint arXiv:2303.12570*, 2023.

[29] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "Swe-bench: Can language models resolve real-world github issues?" *arXiv preprint arXiv:2310.06770*, 2023.

[30] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[31] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji, "Executable code actions elicit better llm agents," *arXiv preprint arXiv:2402.01030*, 2024.

[32] Y. Xia, Y. Chen, T. Shi, J. Wang, and J. Yang, "Aicodereval: Improving ai domain code generation of large language models," *arXiv preprint arXiv:2406.04712*, 2024.

[33] I. Bouzenia, P. Devanbu, and M. Pradel, "Repairagent: An autonomous, llm-based agent for program repair," *arXiv preprint arXiv:2403.17134*, 2024.

[34] J. Liu, Y. Chen, M. Liu, X. Peng, and Y. Lou, "Stall+: Boosting llm-based repository-level code completion with static analysis," *arXiv preprint arXiv:2406.10018*, 2024.

improvement of complete rate by 20% and 7% on CIBench, 46% and 47% on GeoCode-GEE, 40% and 14% on GeoCode-Others under LLMs alone and GeoAgent, respectively. Across both self-debugging and human intervention modes, almost all cases perform better with the assistance of human expertise. Whereas few hard cases still keep the same completion rate even with human intervention, such as the sixth task in GeoCode-Others. These observations suggest that LLMs can achieve better results with human interaction, indicating a promising avenue for integrating LLMs to assist humans in geospatial data analysis tasks. In individual cases, we observed a decline in performance after implementing GeoAgent. This drop is attributed to the additional noise introduced during the GeoAgent inference phases, which can be mitigated by incorporating an extra step in the process. Among the three benchmarks, GeoCode-Others presents a greater challenge compared to the other two, as it has a zero completion rate for 7 out of 10 tasks in automatic mode. Despite this, GeoAgent significantly improved the pass rate for these tasks. In contrast, CIBench is the easiest benchmark, achieving a 100% completion rate for 6 out of 10 tasks.

5.3 Discusssion

In constructing our dataset, we considered various libraries for open-domain geospatial data analysis tasks. Choosing the right library is challenging and requires extensive expert knowledge. Our research shows that the GEE library is the only one that offers many resources within the open-source community, while other libraries mostly provide only usage examples. Including these libraries requires a lot of human effort. To ensure the fairness and validity of our experiments, we tested GEE separately from the other libraries. We generated code instructions from the provided code snippets for our dataset, although these instructions may not always match perfectly with codes. Additionally, prompt generation, which uses Llama3.1, highlights the importance of how prompts are worded to produce accurate, task-specific code. The representation of library and function dependencies is handled during the model's pre-training phase, but the exploration of these dependencies within prompts is limited. Further investigation into diverse ways to represent these dependencies in prompts could be an important area for future research.

In GeoAgent, using only Python interpreters for error detection is often not enough and can be misleading. In complex workflows, running code without errors does not guarantee its correctness. Our findings show that inaccurate execution feedback can limit the advantages of making dynamic adjustments. Currently, the dynamic adjustment strategies in MCTS focus too much on execution feedback. To address this problem, we need to create more flexible online evaluation strategies. Additionally, relying too heavily on execution feedback and RAG adds significant costs during model inference. Therefore, we must consider efficiency when integrating these methods into the MCTS framework. A possible solution is to implement selective execution feedback during dynamic adjustments instead of using it in every generation step.

20 • Chen and Wang, et al.

[35] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[36] K. Zhang, H. Zhang, G. Li, J. Li, Z. Li, and Z. Jin, "Toolcoder: Teach code generation models to use api search tools," *arXiv preprint arXiv:2305.04032*, 2023.

[37] F. Liu, Y. Liu, L. Shi, H. Huang, R. Wang, Z. Yang, and L. Zhang, "Exploring and evaluating hallucinations in llm-powered code generation," *arXiv preprint arXiv:2404.00971*, 2024.

[38] Z. Li and H. Ning, "Autonomous gis: the next-generation ai-powered gis," *International Journal of Digital Earth*, vol. 16, no. 2, pp. 4668–4686, 2023.

[39] R. Manvi, S. Khanna, G. Mai, M. Burke, D. Lobell, and S. Ermon, "Geollm: Extracting geospatial knowledge from large language models," *arXiv preprint arXiv:2310.06213*, 2023.

[40] Y. Zhang, C. Wei, Z. He, and W. Yu, "Geogpt: An assistant for understanding and processing geospatial tasks," *International Journal of Applied Earth Observation and Geoinformation*, vol. 131, p. 103976, 2024.

[41] Z. Lin, C. Deng, L. Zhou, T. Zhang, Y. Xu, Y. Xu, Z. He, Y. Shi, B. Dai, Y. Song *et al.*, "Geogalactica: A scientific large language model in geoscience," *arXiv preprint arXiv:2401.00434*, 2023.

[42] C. Deng, T. Zhang, Z. He, Q. Chen, Y. Shi, Y. Xu, L. Fu, W. Zhang, X. Wang, C. Zhou *et al.*, "K2: A foundation language model for geoscience knowledge understanding and utilization," in *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, 2024, pp. 161–170.

[43] Y. Hu, J. Yuan, C. Wen, X. Lu, and X. Li, "Rsgpt: A remote sensing vision language model and benchmark," *arXiv preprint arXiv:2307.15266*, 2023.

[44] W. Zhang, M. Cai, T. Zhang, Y. Zhuang, and X. Mao, "Earthgpt: A universal multi-modal large language model for multi-sensor image comprehension in remote sensing domain," *IEEE Transactions on Geoscience and Remote Sensing*, 2024.

[45] Y. Bazi, L. Bashmal, M. M. Al Rahhal, R. Ricci, and F. Melgani, "Rs-llava: A large vision-language model for joint captioning and question answering in remote sensing imagery," *Remote Sensing*, vol. 16, no. 9, p. 1477, 2024.

[46] P. P. Ray, "Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 121–154, 2023.

[47] K. Kuckreja, M. S. Danish, M. Naseer, A. Das, S. Khan, and F. S. Khan, "Geochat: Grounded large vision-language model for remote sensing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 27 831–27 840.

[48] H. Liu, C. Li, Y. Li, and Y. J. Lee, "Improved baselines with visual instruction tuning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 26 296–26 306.

[49] H. Ning, Z. Li, T. Akinboyewa, and M. N. Lessani, "An autonomous gis agent framework for geospatial data retrieval."

[50] S. Zhang, Z. Chen, Y. Shen, M. Ding, J. B. Tenenbaum, and C. Gan, "Planning with large language models for code generation," *arXiv preprint arXiv:2303.05510*, 2023.

[51] D. Halter, "jedi: an awesome autocompletion tool for python," 2024, accessed: 2024-10-18.

[52] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[53] C. Team, "Codegemma: Open code models based on gemma," *arXiv preprint arXiv:2406.11409*, 2024.

[54] M. Abdin, S. A. Jacobs, A. A. Awan, J. Aneja, A. Awadallah, H. Awadalla, N. Bach, A. Bahree, A. Bakhtiari, H. Behl *et al.*, "Phi-3 technical report: A highly capable language model locally on your phone," *arXiv preprint arXiv:2404.14219*, 2024.

[55] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.

[56] A. F. Giraldo-Forero, J. A. Jaramillo-Garzón, and C. G. Castellanos-Domínguez, "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 3*. Springer, 2015, pp. 557–564.

[57] N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore, "Google earth engine: Planetary-scale geospatial analysis for everyone," *Remote sensing of Environment*, vol. 202, pp. 18–27, 2017.

In the evaluation of generated code, we identified four common types of errors. First, instruction-following errors occur when the model lacks adequate training in specific domain knowledge, leading to difficulties in following task instructions. Second, hallucination errors involve incorrect or non-existent function calls and undefined variables, often caused by using less common geospatial Python libraries. These errors reveal the challenges of applying LLMs to open-domain geospatial tasks. Third, the lack of information errors happens when the prompt does not provide enough or accurate information, which LLMs cannot fix on their own and need human help to address. Lastly, general code errors reflect the current limitations of LLMs in generating code effectively.

This work has two main limitations. First, GeoAgent relies solely on the capabilities of LLMs, while many open-source geospatial libraries could improve their geospatial coding skills. Second, the evaluation metrics for GeoCode focus only on process-oriented assessments. Although the code is executable, measuring its performance based on output is difficult because the results can be varied and complex.

6 Conclusion

We introduce GeoAgent, an innovative approach for geospatial task programming designed to enhance access to extensive geospatial datasets and facilitate automated workflow using LLMs with diverse and ever-evolving Python libraries. GeoAgent harnesses the capabilities of LLMs to transform tasks into executable units, subsequently retrieving the corresponding APIs through RAG from a knowledge database and dynamically refining subtasks in MCTS. Additionally, we develop a benchmark, GeoCode, to assess the efficacy of the proposed framework on diverse geospatial data analysis tasks using popular geospatial Python libraries. Our experimental results on GeoCode and existing benchmarks indicate that GeoAgent surpasses LLM baselines in both general programming tasks and geospatial data analysis tasks. The findings reveal that GeoAgent significantly enhances geospatial task pass rate and task completion. 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 will contribute to improving the use of geospatial data in research aimed at societal benefits and environmental conservation.

Acknowledgements

I would like to express my sincere gratitude to Professor Laurent Wendling for his valuable insights and guidance during the initial discussions of this work.

An LLM Agent for Automatic Geospatial Data Analysis • 21

12 Appendices

12.1 Motivation

本研究旨在为评估LLM编码器在地理空间数据分析编程中的能力建立一个基准。目前的研究已经证明了自然语言处理在遥感任务中的有效性。与LVLMs不同，地理空间数据涵盖了多种模态，这些模态无法通过自然语言接口轻松访问。此外，传感器技术的持续更新以及相应的Python库对不熟悉这些工具的用户在数据处理中提出了挑战。在这种背景下，我们寻求一种更基础的方法来处理多样化的模态和地理空间数据来源。对这种数据的编程访问是一种有前景的方法，这一点在NLP和CV社区的进展中得到了证明。编程在地理空间数据分析中的优势包括在线数据的可用性和能够即时执行代码并接收反馈。此外，大多数任务可以通过简单的脚本完成，而无需跨文件编程。

尽管有这些优势，但使用LLM代理进行地理空间任务编程的综合研究仍然缺乏。在这项工作中，我们特别关注与地理空间数据分析相关的挑战，即：（1）用于后训练的计算资源相对较少，（2）函数调用的多样性，以及（3）顺序任务中的逻辑连贯性。我们还创建了一个基准，以增强我们对当前LLM技术在应对这些挑战方面的潜力的理解，并指导未来的研究方向。我们根据三个主要指南构建了这个数据集，以鼓励社区对这一基准的贡献。首先，每个基准项目都应该是可执行的，代码解释器应即时提供反馈。存储库或单个函数级别的基准不适合此目的。其次，基准应是实用的，涵盖各种地理空间数据分析场景，包括图像获取、空间分析等。第三，评估应挑战任务的完成，要求LLMs展示强大的组合推理和指令遵循能力。单独的API调用不足以满足此目的。这一初步努力旨在帮助社区更好地评估LLMs在解决地理空间任务中的编程能力，从而在一个开放的框架内促进这一研究领域的发展。

References

[1] S. Singh, M. Fore, and D. Stamoulis, "Geollm-engine: A realistic environment for building geospatial copilots," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2024, pp. 585–594.

[2] C. Liu, K. Chen, H. Zhang, Z. Qi, Z. Zou, and Z. Shi, "Change-agent: Toward interactive comprehensive remote sensing change interpretation and analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 62, pp. 1–16, 2024.

[3] K. Kuckreja, M. S. Danish, M. Naseer, A. Das, S. Khan, and F. S. Khan, "Geochat: Grounded large vision-language model for remote sensing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2024, pp. 27 831–27 840.

[4] Y. Zhang, C. Wei, Z. He, and W. Yu, "Geogpt: An assistant for understanding and processing geospatial tasks," *International Journal of Applied Earth Observation and Geoinformation*, vol. 131, p. 103976, 2024.

[5] W. Xu, Z. Yu, Y. Wang, J. Wang, and M. Peng, "Rs-agent: Automating remote sensing tasks through intelligent agents," *arXiv preprint arXiv:2406.07089*, 2024.

[6] Y. Hu, J. Yuan, C. Wen, X. Lu, and X. Li, "Rsgpt: A remote sensing vision language model and benchmark," *arXiv preprint arXiv:2307.15266*, 2023.

[7] Y. Bazi, L. Bashmal, M. M. Al Rahhal, R. Ricci, and F. Melgani, "Rs-llava: A large vision-language model for joint captioning and question answering in remote sensing imagery," *Remote Sensing*, vol. 16, no. 9, p. 1477, 2024.

[8] Y. Zhan, Z. Xiong, and Y. Yuan, "Skyeyegpt: Unifying remote sensing vision-language tasks via instruction tuning with large language model," *arXiv preprint arXiv:2401.09712*, 2024.

[9] H. Guo, X. Su, C. Wu, B. Du, L. Zhang, and D. Li, "Remote sensing chatgpt: Solving remote sensing tasks with chatgpt and visual models," *arXiv preprint arXiv:2401.09083*, 2024.

[10] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[11] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-t. Yih, D. Fried, S. Wang, and T. Yu, "Ds-1000: A natural and reliable benchmark for data science code generation," in *International Conference on Machine Learning*. PMLR, 2023, pp. 18 319–18 345.

[12] Z. Wang, S. Zhou, D. Fried, and G. Neubig, "Execution-based evaluation for open-domain code generation," *arXiv preprint arXiv:2212.10481*, 2022.

[13] T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul *et al.*, "Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions," *arXiv preprint arXiv:2406.15877*, 2024.

[14] S. Zhang, C. Zhang, Y. Hu, H. Shen, K. Liu, Z. Ma, F. Zhou, W. Zhang, X. He, D. Lin *et al.*, "Cibench: Evaluating your llms with a code interpreter plugin," *arXiv preprint arXiv:2407.10499*, 2024.

[15] J. Wu, W. Gan, H.-C. Chao, and S. Y. Philip, "Geospatial big data: Survey and challenges," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024.

[16] N. Jain, R. Kwiatkowski, B. Ray, M. K. Ramanathan, and V. Kumar, "On mitigating code llm hallucinations with api documentation," *arXiv preprint arXiv:2407.09726*, 2024.

[17] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, "Retrieval augmented language model pre-training," in *International conference on machine learning*. PMLR, 2020, pp. 3929–3938.

[18] Z. Z. Wang, A. Asai, X. V. Yu, F. F. Xu, Y. Xie, G. Neubig, and D. Fried, "Coderag-bench: Can retrieval augment code generation?" *arXiv preprint arXiv:2406.14497*, 2024.

[19] T. Li, G. Zhang, Q. D. Do, X. Yue, and W. Chen, "Long-context llms struggle with long in-context learning," *arXiv preprint arXiv:2404.02060*, 2024.

[20] D. Zhang, J. Li, X. Huang, D. Zhou, Y. Li, and W. Ouyang, "Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b," *arXiv preprint arXiv:2406.07394*, 2024.

[21] M. Liao, W. Luo, C. Li, J. Wu, and K. Fan, "Mario: Math reasoning with code interpreter output–a reproducible pipeline," *arXiv preprint arXiv:2401.08190*, 2024.

[22] X. Li, C. Wen, Y. Hu, Z. Yuan, and X. X. Zhu, "Vision-language models in remote sensing: Current progress and future trends," *IEEE Geoscience and Remote Sensing Magazine*, 2024.

[23] N. Muennighoff, Q. Liu, A. Zebaze, Q. Zheng, B. Hui, T. Y. Zhuo, S. Singh, X. Tang, L. Von Werra, and S. Longpre, "Octopack: Instruction tuning code large language models," *arXiv preprint arXiv:2308.07124*, 2023.

[24] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," *arXiv preprint arXiv:2304.05128*, 2023.

12.2 Benchmark Construction Details

12.2.1 Python Library Version Control and Domain Classification

表 7. 在 GeoCode 中使用的 Python 库版本及其所属的功能领域。

库	版本	领域
earthengine-api	0.1.412	遥感数据分析
cubo	2024.8.0	遥感数据获取
pystac	1.10.1	遥感数据获取
eemont	0.3.6	遥感数据分析
geemap	0.34.0	可视化
geetools	1.4.0	遥感数据分析
GemGIS	1.1.8	GIS处理
Gempy	2024.2.0.2	地质模型
geeet	0.3.0	蒸散
GeoPandas	1.0.1	GIS处理
GOES-2-Go	2024.7.0	遥感数据获取
GeoUtils	0.1.8	栅格和地理信息处理
gstools	1.6.0	地统计学
leafmap	0.37.1	制图和地理空间分析
Lonboard	0.9.3	大型地理空间数据可视化
meteostat	1.6.8	气象和气候数据
pystac_client	0.8.3	遥感数据获取
pytesmo	0.16.0	土壤湿度观测
scikeo	0.2.32	机器学习
segment-geospatial	0.10.7	深度学习
sen2nbar	2024.6.0	法向BRDF校正反射率
srai	0.7.6	深度学习
planetary_computer	1.0.0	遥感数据获取
verde	1.8.1	机器学习
wxee	0.4.2	遥感数据分析
xarray-spatial	0.4.0	基于栅格的空间分析
pylandtemp	0.0.1a1	全球地表温度
eradiate	0.28.0	3D辐射传输模型

[25] N. Jiang, X. Li, S. Wang, Q. Zhou, S. B. Hossain, B. Ray, V. Kumar, X. Ma, and A. Deoras, "Training llms to better self-debug and explain code," *arXiv preprint arXiv:2405.18649*, 2024.

[26] J. Li, G. Li, C. Tao, H. Zhang, F. Liu, and Z. Jin, "Large language model-aware in-context learning for code generation," *arXiv preprint arXiv:2310.09748*, 2023.

[27] A. Patel, S. Reddy, D. Bahdanau, and P. Dasigi, "Evaluating in-context learning of libraries for code generation," *arXiv preprint arXiv:2311.09635*, 2023.

[28] F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J.-G. Lou, and W. Chen, "Repocoder: Repository-level code completion through iterative retrieval and generation," *arXiv preprint arXiv:2303.12570*, 2023.

[29] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "Swe-bench: Can language models resolve real-world github issues?" *arXiv preprint arXiv:2310.06770*, 2023.

[30] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[31] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji, "Executable code actions elicit better llm agents," *arXiv preprint arXiv:2402.01030*, 2024.

[32] Y. Xia, Y. Chen, T. Shi, J. Wang, and J. Yang, "Aicodereval: Improving ai domain code generation of large language models," *arXiv preprint arXiv:2406.04712*, 2024.

[33] I. Bouzenia, P. Devanbu, and M. Pradel, "Repairagent: An autonomous, llm-based agent for program repair," *arXiv preprint arXiv:2403.17134*, 2024.

[34] J. Liu, Y. Chen, M. Liu, X. Peng, and Y. Lou, "Stall+: Boosting llm-based repository-level code completion with static analysis," *arXiv preprint arXiv:2406.10018*, 2024.

[35] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[36] K. Zhang, H. Zhang, G. Li, J. Li, Z. Li, and Z. Jin, "Toolcoder: Teach code generation models to use api search tools," *arXiv preprint arXiv:2305.04032*, 2023.

[37] F. Liu, Y. Liu, L. Shi, H. Huang, R. Wang, Z. Yang, and L. Zhang, "Exploring and evaluating hallucinations in llm-powered code generation," *arXiv preprint arXiv:2404.00971*, 2024.

[38] Z. Li and H. Ning, "Autonomous gis: the next-generation ai-powered gis," *International Journal of Digital Earth*, vol. 16, no. 2, pp. 4668–4686, 2023.

[39] R. Manvi, S. Khanna, G. Mai, M. Burke, D. Lobell, and S. Ermon, "Geollm: Extracting geospatial knowledge from large language models," *arXiv preprint arXiv:2310.06213*, 2023.

[40] Y. Zhang, C. Wei, Z. He, and W. Yu, "Geogpt: An assistant for understanding and processing geospatial tasks," *International Journal of Applied Earth Observation and Geoinformation*, vol. 131, p. 103976, 2024.

[41] Z. Lin, C. Deng, L. Zhou, T. Zhang, Y. Xu, Y. Xu, Z. He, Y. Shi, B. Dai, Y. Song *et al.*, "Geogalactica: A scientific large language model in geoscience," *arXiv preprint arXiv:2401.00434*, 2023.

[42] C. Deng, T. Zhang, Z. He, Q. Chen, Y. Shi, Y. Xu, L. Fu, W. Zhang, X. Wang, C. Zhou *et al.*, "K2: A foundation language model for geoscience knowledge understanding and utilization," in *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, 2024, pp. 161–170.

[43] Y. Hu, J. Yuan, C. Wen, X. Lu, and X. Li, "Rsgpt: A remote sensing vision language model and benchmark," *arXiv preprint arXiv:2307.15266*, 2023.

[44] W. Zhang, M. Cai, T. Zhang, Y. Zhuang, and X. Mao, "Earthgpt: A universal multi-modal large language model for multi-sensor image comprehension in remote sensing domain," *IEEE Transactions on Geoscience and Remote Sensing*, 2024.

[45] Y. Bazi, L. Bashmal, M. M. Al Rahhal, R. Ricci, and F. Melgani, "Rs-llava: A large vision-language model for joint captioning and question answering in remote sensing imagery," *Remote Sensing*, vol. 16, no. 9, p. 1477, 2024.

[46] P. P. Ray, "Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 121–154, 2023.

[47] K. Kuckreja, M. S. Danish, M. Naseer, A. Das, S. Khan, and F. S. Khan, "Geochat: Grounded large vision-language model for remote sensing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 27 831–27 840.

[48] H. Liu, C. Li, Y. Li, and Y. J. Lee, "Improved baselines with visual instruction tuning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 26 296–26 306.

库	版本	领域
spectramap	0.6.0.0	光谱学家的高光谱包

12.2.2 Code Collection: 本工作中收集的所有数据集均来自公开可用的GitHub仓库和解决方案共享网站。最初，我们收集了一个地理空间主题的仓库通用语料库，以便进行后续分析。这一过程首先通过GitHub的搜索功能获取地理空间数据处理仓库的列表，特别针对那些具有许可许可证的仓库。在初步数据收集之后，我们从Python项目中提取使用示例。这些示例由库作者提供，展示了库的功能和用例。在地理空间数据处理库中，Google Earth Engine [57] 脱颖而出，成为最重要和最广泛使用的库。然而，该库的大多数现有脚本都是用JavaScript编写的。为了利用这些丰富的代码资源，我们首先使用大型语言模型（LLMs）自动将JavaScript翻译成Python脚本，然后通过程序分析手动纠正任何运行时错误。除了GitHub仓库，我们还从网站收集数据。获取地理空间数据是有效进行地理空间数据处理和分析的基础。现代地理空间数据平台促进了多样化数据集的收集，这些数据集可以根据特定用户需求进行定制。因此，我们非常重视将在线数据获取库整合到所提出的基准中。

12.2.3 Code Refactoring: 本文基准测试中使用的大多数代码源自GEE，其中通常包括注释的代码块、冗余甚至死代码。因此，重构收集的代码过程至关重要。初步方法是利用大型语言模型（LLMs）自动重构脚本，尽管LLMs在处理长上下文时经常省略部分内容。为了解决这个问题，脚本首先根据代码注释被划分为多个步骤，然后每个步骤独立地重构为Python代码。尽管如此，脚本仍然存在几个问题，如不必要的代码块、未使用的导入库和运行时错误。为了开发高质量的基于执行的基准测试，有必要验证每个代码块的正确性，识别错误，并移除冗余的代码和库。然而，对于人类开发者来说，这个重构过程是复杂且耗时的。为了提高代码质量和减少人工工作，采用了代码解释器和抽象语法树解析器来分析每个代码块。该过程包括锚定每个代码块，收集所有相关的代码块以构建脚本，并过滤掉少于三个可执行单元的脚本。最后，代码必须通过代码解释器测试；否则，当代码失败时，LLMs将重新参与重构代码。此外，必须移除未使用的库，并在代码片段中添加任何缺失的必要库。当LLMs参与重构循环时，发现了两个问题：首先，代码片段中缺少必要的变量，这阻碍了顺利执行；其次，LLMs在解决运行时错误时陷入困境，导致迭代改进。持续的人类反馈对于提供可行的解决方案和确保过程顺利进行至关重要。相同的方法也应用于来自其他Python库的代码。

12.2.4 Code Instruction Generation: 评估LLM编码器依赖于使用指令-代码对，其中用户向LLM提出问题或提供特定任务的指令以进行任务编程。为了构建指令-代码对，我们首先通过Llama3.1指令版本处理代码片段，该版本被提示为每个代码块生成指令。这种方法生成了逐步指令，LLM可以遵循这些指令使用给定的库生成脚本。然后，我们手动修正缺乏必要信息以生成相应代码的指令。该过程从遍历脚本中的每个代码块开始。然后，使用LLM作为摘要器为每个块生成任务描述。摘要过程的

[49] H. Ning, Z. Li, T. Akinboyewa, and M. N. Lessani, "An autonomous gis agent framework for geospatial data retrieval."

[50] S. Zhang, Z. Chen, Y. Shen, M. Ding, J. B. Tenenbaum, and C. Gan, "Planning with large language models for code generation," *arXiv preprint arXiv:2303.05510*, 2023.

[51] D. Halter, "jedi: an awesome autocompletion tool for python," 2024, accessed: 2024-10-18.

[52] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[53] C. Team, "Codegemma: Open code models based on gemma," *arXiv preprint arXiv:2406.11409*, 2024.

[54] M. Abdin, S. A. Jacobs, A. A. Awan, J. Aneja, A. Awadallah, H. Awadalla, N. Bach, A. Bahree, A. Bakhtiari, H. Behl *et al.*, "Phi-3 technical report: A highly capable language model locally on your phone," *arXiv preprint arXiv:2404.14219*, 2024.

[55] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.

[56] A. F. Giraldo-Forero, J. A. Jaramillo-Garzón, and C. G. Castellanos-Domínguez, "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 3*. Springer, 2015, pp. 557–564.

[57] N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore, "Google earth engine: Planetary-scale geospatial analysis for everyone," *Remote sensing of Environment*, vol. 202, pp. 18–27, 2017.

24 • Chen and Wang, et al.

数学表示如下：

$$< s_i, S_i >= LLMs(c_i) \tag{8}$$

其中 s_i 对应于代码片段 c_i 的摘要， $S_i = \{s_1, ..., s_m\}$ 指的是先前块的描述。摘要 s_i 被视为代码片段 c_i 的子任务描述，而聚合描述 S_i 被视为整体任务描述。

7 Appendices

7.1 Motivation

This study aims to establish a benchmark for evaluating the capabilities of LLM coders in geospatial data analysis programming. Current research has demonstrated the efficacy of natural language processing in remote sensing tasks. Unlike LVLMs, geospatial data encompasses a multitude of modalities that are not readily accessible through natural language interfaces Furthermore, the continuous updates in sensor technology and the corresponding Python libraries present challenges for users unfamiliar with these tools in data processing. In this context, we seek a more foundational approach to processing diverse modalities and sources of geospatial data. Programming access to such data is a promising approach, as evidenced by advancements in NLP and CV communities. The advantages of programming for geospatial data analysis include the availability of online data and the ability to execute code instantly and receive feedback. Moreover, most tasks can be completed using simple scripts without requiring cross-file programming.

Despite these advantages, there is a lack of comprehensive study of geospatial task programming using LLM agents. In this work, we focus particularly on the challenges associated with geospatial data analysis, namely: (1) the relatively low computational resources available for post-training, (2) the diversity of function calls, and (3) logical coherence in sequential tasks. We also create a benchmark that can enhance our understanding of the potential of current LLM techniques in addressing these challenges and inform future research directions. We constructed this dataset following three primary guidelines to encourage community contributions to this benchmark. First, each benchmark item should be executable, with feedback provided by the code interpreter instantaneously. Benchmarks at the repository or single-function level are unsuitable for this purpose. Second, the benchmark should be practical, encompassing a variety of geospatial data analysis scenarios, including image acquisition, spatial analysis, and beyond. Third, the evaluation should challenge the completion of tasks, requiring LLMs to demonstrate strong compositional reasoning and instruction-following capabilities. Standalone API calls are inadequate for this purpose. This initial effort aims to assist the community in better assessing the LLMs' programming capabilities in solving geospatial tasks within an open-ended framework, thereby contributing to the advancement of this research area.

An LLM Agent for Automatic Geospatial Data Analysis • 25

12.3 Detailed Prompts

Prompt for Program Refactoring

Here is a task script written in JavaScript using GEE; please refactor it into a Python version using the Earth Engine API.

Prompt for instruction generation

Here are the used Python libraries { } and the previous steps { }. I want you to become my Expert Prompt Creator. Your goal is to help me reverse Python code into prompts. The prompt you provide should be a very detailed text description, including all necessary information, such as exact parameter values, input files, date, location, etc. Provide the prompt for this step's code { } in the format { 'prompt': detailed prompts with all parameters and values }.

Instruction-tuning Prompt

I will give you a task description that needs to use some of the given library { } to implement it. You need to provide the Python code for the given task description. Here is the previous code { }. Please provide the Python code for the current step with this description.

Inference Prompt

I want you to become my expert programmer. Your goal is to help me write Python code for the given task using the Python library { }. You need to write code according to the detailed prompt { }. Please provide the corresponding code.

Updating the Node Prompt

Your goal is to fix the error in the initial prompt. I first give you the Python code { }. Give your solution for fixing the error { } and add it to the initial prompt.

7.2 Benchmark Construction Details

7.2.1 Python Library Version Control and Domain Classification

表 7. The version of Python libraries used in GeoCode, as well as their parent functionality domain.

Library	Version	Domain
earthengine-api	0.1.412	RS data analysis
cubo	2024.8.0	RS data acqusition
pystac	1.10.1	RS data acquisition
eemont	0.3.6	RS data analysis
geemap	0.34.0	visualization
geetools	1.4.0	RS data analysis
GemGIS	1.1.8	GIS processing
Gempy	2024.2.0.2	geological models
geeet	0.3.0	evapotranspiration
GeoPandas	1.0.1	GIS processing
GOES-2-Go	2024.7.0	RS data acquisition
GeoUtils	0.1.8	raster and geographic information processing
gstools	1.6.0	geostatistical
leafmap	0.37.1	mapping and geospatial analysis
Lonboard	0.9.3	visualizing large geospatial datasets
meteostat	1.6.8	weather and climate data
pystac_client	0.8.3	RS data acquisition
pytesmo	0.16.0	Soil Moisture Observations
scikeo	0.2.32	machine learning
segment-geospatial	0.10.7	deep learning
sen2nbar	2024.6.0	Nadir BRDF Adjusted Reflectance
srai	0.7.6	deep learning
planetary_computer	1.0.0	RS data acquisition
verde	1.8.1	machine learning
wxee	0.4.2	RS data analysis
xarray-spatial	0.4.0	Raster-based Spatial Analytics
pylandtemp	0.0.1a1	global land surface temperature

26 • Chen and Wang, et al.

Prepend New Task Prompt

Defining the undefined variables for the next step task: { next step instruction }. Give your code for the undefined variables in this step:

Library	Version	Domain
eradiate	0.28.0	3D radiative transfer model
spectramap	0.6.0.0	Hyperspectral package for spectroscopists

7.2.2 Code Collection: All these datasets collected in this work are from publicly available GitHub repositories and solution-sharing websites. Initially, we collect a general corpus of geospatial-topic repositories for subsequent analysis. This process begins by obtaining a list of geospatial data processing repositories through GitHub's search function, specifically targeting those with permissive licenses. Following the preliminary data collection, we extract usage examples from the Python projects. These examples, provided by the library authors, demonstrate the capabilities and use cases. Among the geospatial data processing libraries, Google Earth Engine [57] stands out as the most significant and widely used. However, most existing scripts for this library are written in JavaScript. To harness these abundant code resources, we first employ LLMs to automatically translate JavaScript into Python scripts, followed by manual correction of any runtime errors through program analysis. In addition to GitHub repositories, we also collect data from websites. The acquisition of geospatial data is fundamental to effective geospatial data processing and analysis. Modern geospatial data platforms facilitate the collection of diverse datasets, which can be tailored to meet specific user requirements. As a result, we place a strong emphasis on integrating online data acquisition libraries into the proposed benchmark.

7.2.3 Code Refactoring: The majority of the code utilized in this benchmark originates from GEE, where it often includes commented code blocks and redundant and even dead codes. Consequently, the process of refactoring the collected code is crucial. The initial approach involves employing LLMs to automatically refactor the scripts, although LLMs frequently omit parts of the content when processing long contexts. To address this, the scripts are first divided into multiple steps based on code comments, after which each step is independently refactored into Python code. Despite these efforts, the scripts still present several issues, such as unnecessary code blocks, unused imported libraries, and runtime errors. To develop a high-quality execution-based benchmark, it is necessary to verify the correctness of each block, identify bugs, and remove redundant code and libraries. However, this refactoring process is complex and time-consuming for human developers. To enhance code quality and reduce manual effort, a code interpreter and an abstract syntax tree parser are employed to analyze each code block. The process involves anchoring each block, gathering all related code blocks to construct a script, and filtering out scripts with fewer than three executable units. At last, the code must pass the code interpreter test; otherwise, LLMs are re-engaged to refactor the code when it fails. Additionally, unused libraries must be removed, and any necessary libraries that are missing in the code snippet should be added. When LLMs are involved in the refactoring loop, two issues are

An LLM Agent for Automatic Geospatial Data Analysis • 27

12.4 Benchmark Samples

Traning examples

```
## Instruction
# Collect Sentinel-2 data for the period from June 1st, 2019 to September
    30th, 2019, and filter out images with more than 20
## Ground Truth
import ee
# Sentinel-2 data collection
Date_1 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED') \
                  .filterDate('2019-06-01', '2019-09-30') \
                  .filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE',20)) \
                  .map(maskS2clouds)
Date_2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED') \
                  .filterDate('2021-06-01', '2021-09-30') \
                  .filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE',20)) \
                  .map(maskS2clouds)
```

Evaluation Cases

```
## Instruction
#1. Calculate the median value of Date_1 for S1_img; 2. Clip S1_img using
    the ROI defined by "seoul"; 3. Calculate the median value of Date_2 for
    S2_img; 4. Clip S2_img using the ROI defined by "seoul"; 5. Combine the
    clipped S1_img and S2_img to create a composite image. Here is the
    previous code:
import ee
# Sentinel-2 data collection
Date_1 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED') \
                  .filterDate('2019-06-01', '2019-09-30') \
                  .filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE',20)) \
                  .map(maskS2clouds)

```

identified: first, the absence of essential variables in the code snippet, which prevents smooth execution, and second, the LLMs become stuck when resolving runtime bugs, leading to iterative refinement. Continuous human feedback is crucial for providing viable solutions, and ensuring the process stays on track. The same methodology is applied to code from other Python libraries.

7.2.4 Code Instruction Generation: The evaluation of LLM coders relies on using the instruct-code pairs, where users pose questions or provide task-specific instructions to LLMs for task programming. To construct instruct-code pairs, we initially process the code snippet through the Llama3.1 instruction version, which is prompted to produce an instruction for each code block. This approach results in step-by-step instructions that the LLM can then follow to generate a script using given libraries. Then, we manually fix the instruction lacking the necessary information for corresponding code generation. The process starts by iterating over each code block in the scripts. The LLMs are then employed as summarizers to generate task descriptions for each block. The summarization process is mathematically represented as follows:

$$< s_i, S_i >= LLMs(c_i) \tag{8}$$

where s_i corresponds to the summarization of code snippet c_i and $S_i = \{s_1, ..., s_m\}$ refers to the description of previous blocks. The summarization s_i is considered as the subtask description of the code snippet c_i, while the aggregate description S_i is treated as the overall task description.


```
Date_2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED') \
                  .filterDate('2021-06-01', '2021-09-30') \
                  .filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE',20)) \
                  .map(maskS2clouds)
```

12.5 Error Samples

Instruction Following Errors

```
## Instruction
# Import Necessary Feature Collections and Image Collections:
# Import the Lower Deschutes feature collection.
# Set visualization parameters for the annual NPP data: [Opacity: 1, Bands:
    ["annualNPP"], Min: 2178.6000000000004, Max: 10727.4, Gamma: 1]
# Import the Landsat 30m NPP image collection.
# Import the MODIS 250m NPP image collection.

## Generation
# Import necessary feature collections and image collections
Lower_Deschtues = ee.FeatureCollection("users/yusin/Lower_Deschtues")
# Set visualization parameters for the annual NPP data
Map.addLayer(Annual_NPP, {'opacity':1,
    'bands':["annualNPP"],'min':2178.6000000000004,'max':10727.4,
    'gamma':1}, "Annual NPP")
Landsat_30m_NPP = ee.ImageCollection("LANDSAT/LC08/C01/T1_8DAY_NPP")
MODIS_250m_NPP = ee.ImageCollection("MODIS/006/MOD17A2HGF")

## Error Feedback
NameError : name 'Annual_NPP' is not defined

## Ground Truth
# Import the Lower Deschutes feature collection
```

7.3 Detailed Prompts

Prompt for Program Refactoring

Here is a task script written in JavaScript using GEE; please refactor it into a Python version using the Earth Engine API.

Prompt for instruction generation

Here are the used Python libraries { } and the previous steps { }. I want you to become my Expert Prompt Creator. Your goal is to help me reverse Python code into prompts. The prompt you provide should be a very detailed text description, including all necessary information, such as exact parameter values, input files, date, location, etc. Provide the prompt for this step's code { } in the format { 'prompt': detailed prompts with all parameters and values }.

Instruction-tuning Prompt

I will give you a task description that needs to use some of the given library { } to implement it. You need to provide the Python code for the given task description. Here is the previous code { }. Please provide the Python code for the current step with this description.

Inference Prompt

I want you to become my expert programmer. Your goal is to help me write Python code for the given task using the Python library { }. You need to write code according to the detailed prompt { }. Please provide the corresponding code.

Updating the Node Prompt

Your goal is to fix the error in the initial prompt. I first give you the Python code { }. Give your solution for fixing the error { } and add it to the initial prompt.


```
lower_deschutes_fc =
    ee.FeatureCollection('users/your_username/Lower_Deschutes')
# Set visualization parameters for the annual NPP data
npp_vis_params = {
  'opacity': 1,
  'bands': ['annualNPP'],
 'min': 2178.6,
 'max': 10727.4,
  'gamma': 1
}
# Import the Landsat 30m NPP image collection
landsat_npp_collection = ee.ImageCollection('LANDSAT/LT05/C01/T1_TOA')
# Import the MODIS 250m NPP image collection
modis_npp_collection = ee.ImageCollection('MOD11C3')
```

Non-existent APIs

```
## Instruction
# Clip the buildings dataset to the geometry of the selected region.

## Generation
#Clipping the Buildings Dataset:
clipped_buildings = buildings.clip(geometry)
Map.addLayer(clipped_buildings, {}, 'Google Open Buildings')

## Error Feedback
AttributeError: 'FeatureCollection' object has no attribute 'clip'

## Ground Truth
# Clip the buildings dataset to the region of interest
buildings_clipped = buildings.map(lambda img: img.clip(roi))
```

Prepend New Task Prompt

Defining the undefined variables for the next step task: { next step instruction }. Give your code for the undefined variables in this step:

Hallucination of API parameters

```
## Instruction
# Utilize ee.Algorithms.Landsat.simpleComposite to create a simple composite image:- Collection: `l8raw` - Parameter: `asFloat = true`

## Generation
image = ee.Algorithms.Landsat.simpleComposite({
  'collection': l8raw,
  'asFloat': True
})
Map.addLayer(image)

## Error Feedback
EEException: Unrecognized argument type to convert to an ImageCollection: 'collection': <ee.imagecollection.ImageCollection object at 0x7f7e384acac0>, 'asFloat': True

## Ground Truth
# Utilize ee.Algorithms.Landsat.simpleComposite to create a simple composite image
composite_image = ee.Algorithms.Landsat.simpleComposite(l8raw, asFloat=True)
```

Undefined Variables

```
## Instruction
# Filter and Aggregate Global Fishing Effort by Date Range: - Start Date: `2016-01-01` - End Date: `2017-01-01` - The dataset is filtered by the date range and the global flag state ('WLD'). - The filtered collection is aggregated to a single image representing the total fishing effort.

## Generation
gfwFiltered = gfw \
 .filterDate(startDate, endDate) \
 .filter(ee.Filter.eq('flag', 'WLD'))
```

7.4 Benchmark Samples

Traning examples

```
## Instruction
# Collect Sentinel-2 data for the period from June 1st, 2019 to September
    30th, 2019, and filter out images with more than 20
## Ground Truth
import ee
# Sentinel-2 data collection
Date_1 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED') \
                  .filterDate('2019-06-01', '2019-09-30') \
                  .filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE',20)) \
                  .map(maskS2clouds)
Date_2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED') \
                  .filterDate('2021-06-01', '2021-09-30') \
                  .filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE',20)) \
                  .map(maskS2clouds)
```

Evaluation Cases

```
## Instruction
#1. Calculate the median value of Date_1 for S1_img; 2. Clip S1_img using
    the ROI defined by "seoul"; 3. Calculate the median value of Date_2 for
    S2_img; 4. Clip S2_img using the ROI defined by "seoul"; 5. Combine the
    clipped S1_img and S2_img to create a composite image. Here is the
    previous code:
import ee
# Sentinel-2 data collection
Date_1 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED') \
                  .filterDate('2019-06-01', '2019-09-30') \
                  .filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE',20)) \
                  .map(maskS2clouds)

```

An LLM Agent for Automatic Geospatial Data Analysis • 31

```
gfwFilteredTotal = gfwFiltered.sum()

## Error Feedback
NameError: name 'startDate' is not defined

## Ground Truth
start_date = '2016-01-01'
end_date = '2017-01-01'
gfwFiltered = gfw \
.filterDate(start_date, end_date)\
.filter(ee.Filter.eq('flag_state', 'WLD'))
gfwFilteredTotal = gfwFiltered.reduce(ee.Reducer.sum())
```

Syntax Errors

```
## Instruction
# A polygon defined by the coordinates [[11.3258, 47.0801], [11.6746,
    47.1773], [11.5318, 47.3730], [11.1692, 47.2687]].

## Generation
geometry = ee.Geometry.Polygon(
        [[11.3258, 47.0801], [11.6746, 47.1773],
        [11.5318, 47.3730],[11.1692, 47.2687]]])

## Error Feedback
SyntaxError: closing parenthesis ']' does not match opening parenthesis '('

## Ground Truth
geometry = ee.Geometry.Polygon(
    [[11.3258, 47.0801], [11.6746, 47.1773],
     [11.5318, 47.3730],[11.1692, 47.2687]])
```

```
Date_2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED') \
                    .filterDate('2021-06-01', '2021-09-30') \
                    .filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE',20)) \
                    .map(maskS2clouds)
```

7.5 Error Samples

Instruction Following Errors

```
## Instruction
# Import Necessary Feature Collections and Image Collections:
# Import the Lower Deschutes feature collection.
# Set visualization parameters for the annual NPP data: [Opacity: 1, Bands:
    ["annualNPP"], Min: 2178.6000000000004, Max: 10727.4, Gamma: 1]
# Import the Landsat 30m NPP image collection.
# Import the MODIS 250m NPP image collection.

## Generation
# Import necessary feature collections and image collections
Lower_Deschtues = ee.FeatureCollection("users/yusin/Lower_Deschtues")
# Set visualization parameters for the annual NPP data
Map.addLayer(Annual_NPP, {'opacity':1,
    'bands':["annualNPP"],'min':2178.6000000000004,'max':10727.4,
    'gamma':1}, "Annual NPP")
Landsat_30m_NPP = ee.ImageCollection("LANDSAT/LC08/C01/T1_8DAY_NPP")
MODIS_250m_NPP = ee.ImageCollection("MODIS/006/MOD17A2HGF")

## Error Feedback
NameError : name 'Annual_NPP' is not defined

## Ground Truth
# Import the Lower Deschutes feature collection
```


12.6 Example Studies

我们的基准引入了有助于地理空间数据分析的实例。虽然这项工作的实际应用尚未完全探索，但我们的读者可能渴望了解其对地球科学界潜在的影响。在本节中，我们重点介绍了一个利用Sentinel-1 GRD图像和对数比阈值算法的变更检测用例。所展示的示例涉及每个任务编程中的指令、生成失败、错误反馈、生成成功、自我修复生成以及人工干预（当问题未解决时）。此演示不仅展示了GeoAgent的当前能力和局限性，还为读者提供了一个评估的视角。

Change Detection

```
## Instruction 1:
# Import library GEE python and geemap, then authenticate with GEE; at last
    start a Map instance.
## Sucessful Generation:
import ee
ee.Authenticate()
import geemap
Map = geemap.Map()
## Instruction 2:
# Define Parameters:\n\nDate Range: - Start Date: '2018-01-01' - End Date:
    '2019-01-01'\nArea of Interest (AOI): Polygon with coordinates defining
    the area [[-122.0, 37.0],[-122.0, 38.0],[-121.0, 38.0],[-121.0,
    37.0],[-122.0, 37.0]].
## Sucessful Generation:
start_date = '2018-01-01'
end_date = '2019-01-01'
aoi = ee.Geometry.Polygon([[-122.0, 37.0], [-122.0, 38.0], [-121.0, 38.0],
    [-121.0, 37.0], [-122.0, 37.0]])
Map.addLayer(ee.ImageCollection(MODIS/006/MOD13Q1).filterDate(start_date,end_date)
.filterBounds(aoi_polygon).first())
## Instruction 3:
```

```
lower_deschutes_fc =
    ee.FeatureCollection('users/your_username/Lower_Deschutes')
# Set visualization parameters for the annual NPP data
npp_vis_params = {
  'opacity': 1,
  'bands': ['annualNPP'],
 'min': 2178.6,
 'max': 10727.4,
  'gamma': 1
}
# Import the Landsat 30m NPP image collection
landsat_npp_collection = ee.ImageCollection('LANDSAT/LT05/C01/T1_TOA')
# Import the MODIS 250m NPP image collection
modis_npp_collection = ee.ImageCollection('MOD11C3')
```

Non-existent APIs

```
## Instruction
# Clip the buildings dataset to the geometry of the selected region.

## Generation
#Clipping the Buildings Dataset:
clipped_buildings = buildings.clip(geometry)
Map.addLayer(clipped_buildings, {}, 'Google Open Buildings')

## Error Feedback
AttributeError: 'FeatureCollection' object has no attribute 'clip'

## Ground Truth
# Clip the buildings dataset to the region of interest
buildings_clipped = buildings.map(lambda img: img.clip(roi))
```

An LLM Agent for Automatic Geospatial Data Analysis • 33

```
#Select Sentinel-1 Image Collection:'COPERNICUS/S1_GRD'; Filter Criteria:
    Date Range: '2018-01-01' to '2019-01-01', Orbit Properties:
    'DESCENDING', Geographical Bounds: Defined by aoi; Sort sentinel-1 image
    according to system.
## Sucessful Generation:
sentinel1_data =
    ee.ImageCollection('COPERNICUS/S1_GRD').filterDate(start_date,
    end_date).filter(ee.Filter.eq('orbitProperties_pass', 'DESCENDING'))
           .filterBounds(aoi)
collection_S1data = sentinel1_data.sort('system:time_start')
## Instruction 4:
#Define box kernel Kernel Size: 33 meters Write a Python function named
    smooth that takes an input image. The function should: Revert the input
    image to its original form by dividing it by 10 and raising it to the
    power of 10. Apply a boxcar convolution filter to the reverted image to
    smooth it. Convert the filtered image back to the backscatter
    coefficient by taking the base-10 logarithm and multiplying it by 10.
    Ensure that the resulting image retains the system:time_start and
    cycleNumber properties from the input image. Return the final image as a
    float type.
## Failed Generation:
def smooth(image):
    reverted_image = ee.Image(image).divide(10).pow(10)
    box_kernel = ee.Kernel.box(33)
    smoothed_image = reverted_image.focal(box_kernel, 'gauss', 1)
    log_image = smoothed_image.log10().multiply(10)
    filtered_image = log_image.addBands(image.select([0, 1, 2,
        3]).copyProperties(image, ['system:time_start', 'cycleNumber']))
    return filtered_image
smoothed_collection = collection_S1data.map(smooth)
## Error Feedback
AttributeError: type object 'Kernel' has no attribute 'box'
## Human Intervention:
boxcar = ee.Kernel.square(33, 'meters')
def smooth(image):
```

Hallucination of API parameters

```
## Instruction
# Utilize ee.Algorithms.Landsat.simpleComposite to create a simple composite image:- Collection: `l8raw` - Parameter: `asFloat = true`

## Generation
image = ee.Algorithms.Landsat.simpleComposite({
  'collection': l8raw,
  'asFloat': True
})
Map.addLayer(image)

## Error Feedback
EEException: Unrecognized argument type to convert to an ImageCollection: 'collection': <ee.imagecollection.ImageCollection object at 0x7f7e384acac0>, 'asFloat': True

## Ground Truth
# Utilize ee.Algorithms.Landsat.simpleComposite to create a simple composite image
composite_image = ee.Algorithms.Landsat.simpleComposite(l8raw, asFloat=True)
```

Undefined Variables

```
## Instruction
# Filter and Aggregate Global Fishing Effort by Date Range: - Start Date: `2016-01-01` - End Date: `2017-01-01` - The dataset is filtered by the date range and the global flag state ('WLD'). - The filtered collection is aggregated to a single image representing the total fishing effort.

## Generation
gfwFiltered = gfw \
 .filterDate(startDate, endDate) \
 .filter(ee.Filter.eq('flag', 'WLD'))
```

34 • Chen and Wang, et al.

```
    natimg = ee.Image(10.0).pow(image.divide(10.0))
    filtered = natimg.convolve(boxcar)
    return filtered.log10().multiply(10.0).set('system:time_start', image.get('system:time_start')).set('cycleNumber', image.get('cycleNumber')).float()
collection_S1data_Filtered = collection_S1data.map(smooth)
## Instruction 5:
#Create a visualization using the Google Earth Engine (GEE) Map object to display Sentinel-1 Ground Range Detected (GRD) data before and after a filtering step. The code should add two layers to the map: The first layer should represent the unfiltered Sentinel-1 data. Use the first image from the collection_S1data ImageCollection. The visualization parameters should specify the VH band with a minimum value of -23 and a maximum value of -11. Label this layer as Sentinel-1 GRD non filtered. The second layer should represent the filtered Sentinel-1 data. Use the first image from the collection_S1data_Filtered ImageCollection. The visualization parameters should also specify the VH band with a minimum value of -23 and a maximum value of -11. Label this layer as Sentinel-1 GRD filtered.The layers should be added to the map in the order specified.
## Sucessful Generation:
Map.addLayer(collection_S1data.first().select('VH'), {'min': -23, 'max': -11}, 'Sentinel-1 GRD non filtered')
Map.addLayer(collection_S1data_Filtered.first().select('VH'), {'min': -23, 'max': -11}, 'Sentinel-1 GRD filtered')
## Instruction 6:
#Write a Python function that adds a new band to an image, representing the ratio between the 'VH' and 'VV' bands from Sentinel-1 satellite data. The function should compute the ratio as the difference between the 'VH' and 'VV' bands, convert it to a floating-point number, and then add it as a new band named 'ratio'. After defining this function, apply it to a pre-existing image collection named collection_S1data_Filtered, mapping it across the entire collection to create a new image collection named collection_S1data_Filtered_Ratio.
## Passed Generation:
```

```
gfwFilteredTotal = gfwFiltered.sum()

## Error Feedback
NameError: name 'startDate' is not defined

## Ground Truth
start_date = '2016-01-01'
end_date = '2017-01-01'
gfwFiltered = gfw \
.filterDate(start_date, end_date)\
.filter(ee.Filter.eq('flag_state', 'WLD'))
gfwFilteredTotal = gfwFiltered.reduce(ee.Reducer.sum())
```

Syntax Errors

```
## Instruction
# A polygon defined by the coordinates [[11.3258, 47.0801], [11.6746,
    47.1773], [11.5318, 47.3730], [11.1692, 47.2687]].

## Generation
geometry = ee.Geometry.Polygon(
        [[11.3258, 47.0801], [11.6746, 47.1773],
        [11.5318, 47.3730],[11.1692, 47.2687]]])

## Error Feedback
SyntaxError: closing parenthesis ']' does not match opening parenthesis '('

## Ground Truth
geometry = ee.Geometry.Polygon(
    [[11.3258, 47.0801], [11.6746, 47.1773],
     [11.5318, 47.3730],[11.1692, 47.2687]])
```

An LLM Agent for Automatic Geospatial Data Analysis • 35

```
def add_ratio_band(image):
    ratio = image.select('VH').subtract(image.select('VV'))\
    .divide(image.select('VV')).multiply(100)
    ratio_image = image.addBands(ratio.rename('ratio'))
    return ratio_image
collection_S1data_Filtered_Ratio =
    collection_S1data_Filtered.map(add_ratio_band)
## Human Intervention
def add_vhvv_ratio(image):
    vh_vv_ratio =
        image.select('VH').subtract(image.select('VV')).float().rename('ratio')
    image = image.addBands(vh_vv_ratio)
    return image
collection_S1data_Filtered_Ratio =
    collection_S1data_Filtered.map(add_vhvv_ratio)
## Instruction 7:
#Generate Python code using the Google Earth Engine (GEE) Python API to
    visualize Sentinel-1 data. Convert the first 30 images in
    collection_S1data_Filtered_Ratio to a list using the toList() method.
    Assign this list to the variable S1_listData. Visualize the first image
    in this list on a map, clipped to the Area of Interest (aoi). Display
    the image in RGB format using the VV, VH, and ratio bands. Set the
    visualization parameters with minimum values of [-18, -23, 3] and
    maximum values of [-4, -11, 15]. Add this visualization to the map with
    the label sentinel-1 GRD RGB First. Repeat the visualization for the
    last image in the list, adding it to the map with the label Sentinel-1
    GRD RGB Last.
## Failed Generation:
first_image = S1_listData.get(0)
last_image = S1_listData.get(-1)
first_image_clipped = first_image.clip(aoi)
Map.addLayer(first_image_clipped.select('VV', 'VH', 'ratio'), {'min': [-18,
    -23, 3], 'max': [-4, -11, 15]}, 'sentinel-1 GRD RGB First')
last_image_clipped = last_image.clip(aoi)
S1_listData = collection_S1data_Filtered_Ratio.limit(30).toList(30)
```

7.6 Example Studies

Our benchmark introduces instances that facilitate geospatial data analysis. While the practical applications of this work have not yet been fully explored, our audience may be eager to understand its potential impact on the geoscience community. In this section, we highlight a change detection use case utilizing Sentinel-1 GRD images in conjunction with a log-ratio thresholding algorithm. The presented example involves instructions, failed generation, error feedback, passed generation, successful generation, self-fixed generation, and human intervention (when it goes to be unsolved) in terms of each task programming. This demonstration not only showcases the current capabilities and limitations of GeoAgent but also provides an evaluative perspective for the audience.

Change Detection

```
## Instruction 1:
# Import library GEE python and geemap, then authenticate with GEE; at last start a Map instance.
## Sucessful Generation:
import ee
ee.Authenticate()
import geemap
Map = geemap.Map()
## Instruction 2:
# Define Parameters:\n\nDate Range: - Start Date: '2018-01-01' - End Date: '2019-01-01'\nArea of Interest (AOI): Polygon with coordinates defining the area [[-122.0, 37.0],[-122.0, 38.0],[-121.0, 38.0],[-121.0, 37.0],[-122.0, 37.0]].
## Sucessful Generation:
start_date = '2018-01-01'
end_date = '2019-01-01'
aoi = ee.Geometry.Polygon([[-122.0, 37.0], [-122.0, 38.0], [-121.0, 38.0], [-121.0, 37.0], [-122.0, 37.0]])
Map.addLayer(ee.ImageCollection(MODIS/006/MOD13Q1).filterDate(start_date, end_date)
.filterBounds(aoi_polygon).first())
## Instruction 3:
```

36 • Chen and Wang, et al.

```
Map.addLayer(first_image.clip(aoi), {'min': [-18, -23, 3], 'max': [-4, -11, 15]}, 'sentinel-1 GRD RGB First')
Map.addLayer(last_image.clip(aoi), {'min': [-18, -23, 3], 'max': [-4, -11, 15]}, 'Sentinel-1 GRD RGB Last')
## Error Feedback
NameError : name 'S1_listData' is not defined
## Self-fixed Generation:
S1_listData = collection_S1data_Filtered_Ratio.toList(30)
first_image = ee.Image(S1_listData.get(0))
last_image = ee.Image(S1_listData.get(-1))
first_image_clipped = first_image.clip(aoi)
Map.addLayer(first_image_clipped.select('VV', 'VH', 'ratio'), {'min': [-18, -23, 3], 'max': [-4, -11, 15]}, 'sentinel-1 GRD RGB First')
last_image_clipped = last_image.clip(aoi)
Map.addLayer(ee.Image(S1_listData.get(29)).clip(aoi),{'bands': ['VV', 'VH', 'ratio'],min: [-18, -23, 3], max: [-4, -11, 15]}, 'Sentinel-1 GRD RGB Last')
## Instruction 8:
#The first step is to select the Sentinel-1 radar images before and after the event from the provided list (S1_listData). Both images are clipped to a specified area of interest (aoi). Specifically, select the VH polarization band from the images. S1_before is the image before the event, and it's selected as the first image in the list. S1_after is the image after the event, selected as the 30th image in the list. Calculation of Log Ratio: Compute the Log Ratio by subtracting the S1_after image from the S1_before image. The result is then divided by 10 to normalize the values, making the output more scalable. This operation results in a new band named 'logratio'. Visualization: Finally, visualize the resulting Log Ratio image on the map. Set the visualization parameters with a min value of -1.5 and a max value of 1.5.
## Sucessful Generation:
S1_before = ee.Image(S1_listData.get(0)).select('VH').clip(aoi)
S1_after = ee.Image(S1_listData.get(-1)).select('VH').clip(aoi)
LogRatio = S1_before.subtract(S1_after).divide(10).rename('logratio')
print('Log Ratio Band (Change Image):', LogRatio)
```

```
#Select Sentinel-1 Image Collection:'COPERNICUS/S1_GRD'; Filter Criteria: Date Range: '2018-01-01' to '2019-01-01', Orbit Properties: 'DESCENDING', Geographical Bounds: Defined by aoi; Sort sentinel-1 image according to system.
## Sucessful Generation:
sentinel1_data = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate(start_date, end_date).filter(ee.Filter.eq('orbitProperties_pass', 'DESCENDING'))
       .filterBounds(aoi)
collection_S1data = sentinel1_data.sort('system:time_start')
## Instruction 4:
#Define box kernel Kernel Size: 33 meters Write a Python function named smooth that takes an input image. The function should: Revert the input image to its original form by dividing it by 10 and raising it to the power of 10. Apply a boxcar convolution filter to the reverted image to smooth it. Convert the filtered image back to the backscatter coefficient by taking the base-10 logarithm and multiplying it by 10. Ensure that the resulting image retains the system:time_start and cycleNumber properties from the input image. Return the final image as a float type.
## Failed Generation:
def smooth(image):
    reverted_image = ee.Image(image).divide(10).pow(10)
    box_kernel = ee.Kernel.box(33)
    smoothed_image = reverted_image.focal(box_kernel, 'gauss', 1)
    log_image = smoothed_image.log10().multiply(10)
    filtered_image = log_image.addBands(image.select([0, 1, 2, 3]).copyProperties(image, ['system:time_start', 'cycleNumber']))
    return filtered_image
smoothed_collection = collection_S1data.map(smooth)
## Error Feedback
AttributeError: type object 'Kernel' has no attribute 'box'
## Human Intervention:
boxcar = ee.Kernel.square(33, 'meters')
def smooth(image):
```



```
Map.addLayer(LogRatio, {'bands': ['logratio'], min: [-1.5], max: [1.5]}, 'Sentinel-1 Log-Ratio Image')
## Instruction 9:
#Extract the first image from the Sentinel-1 dataset (collection_S1data). Center the map on the area of interest (aoi) with a zoom level of 10. Add the original Sentinel-1 image layer to the map with a visualization range (min: -20, max: 0).
#Display Filtered Data: Filter the smoothed Sentinel-1 dataset (collection_S1data_Filtered) to retrieve images within a specified date range (start_date to end_date). Extract the first image from the filtered dataset. Add this filtered image layer to the map with the same visualization range (min: -20, max: 0).
#Display VH/VV Ratio Data (collection_S1data_Filtered_Ratio): Extract the VH/VV ratio band from the first image of the filtered smoothed dataset. Extract the VH/VV ratio band from the last image of the filtered smoothed dataset, sorted by the system:index.\nAdd these VH/VV ratio layers to the map with a visualization range (min: 0, max: 1). Label the layers as VH/VV Ratio (First Image) and VH/VV Ratio (Last Image) respectively.
## Failed Generation:
filtered_S1_layer = collection_S1data.select('VH').clip(aoi)
Map.addLayer(filtered_S1_layer, {'bands': ['VH'], 'min': [-20], 'max': [0]}, 'Filtered Sentinel-1 Image')
vh_vv_ratio_first_image = ee.Image(collection_S1data_Filtered.get(0)).select('vhvv_ratio')
vh_vv_ratio_last_image = ee.ImageCollection(collection_S1data_Filtered).filterDate(start_date, end_date).sort('system:index').last().select('vhvv_ratio')
Map.addLayer(vh_vv_ratio_first_image.clip(aoi), {'bands': ['vhvv_ratio'], 'min': [0], 'max': [1]}, 'VH/VV Ratio (First Image)')
Map.addLayer(vh_vv_ratio_last_image.clip(aoi), {'bands': ['vhvv_ratio'], 'min': [0], 'max': [1]}, 'VH/VV Ratio (Last Image)')
## Error Feedback
AttributeError : 'ImageCollection' object has no attribute 'clip'
## Human Intervention
```

```
    natimg = ee.Image(10.0).pow(image.divide(10.0))
    filtered = natimg.convolve(boxcar)
    return filtered.log10().multiply(10.0).set('system:time_start',
        image.get('system:time_start')).set('cycleNumber',
        image.get('cycleNumber')).float()
collection_S1data_Filtered = collection_S1data.map(smooth)
## Instruction 5:
#Create a visualization using the Google Earth Engine (GEE) Map object to
    display Sentinel-1 Ground Range Detected (GRD) data before and after a
    filtering step. The code should add two layers to the map: The first
    layer should represent the unfiltered Sentinel-1 data. Use the first
    image from the collection_S1data ImageCollection. The visualization
    parameters should specify the VH band with a minimum value of -23 and a
    maximum value of -11. Label this layer as Sentinel-1 GRD non filtered.
    The second layer should represent the filtered Sentinel-1 data. Use the
    first image from the collection_S1data_Filtered ImageCollection. The
    visualization parameters should also specify the VH band with a minimum
    value of -23 and a maximum value of -11. Label this layer as Sentinel-1
    GRD filtered.The layers should be added to the map in the order
    specified.
## Sucessful Generation:
Map.addLayer(collection_S1data.first().select('VH'), {'min': -23, 'max':
    -11}, 'Sentinel-1 GRD non filtered')
Map.addLayer(collection_S1data_Filtered.first().select('VH'), {'min': -23,
    'max': -11}, 'Sentinel-1 GRD filtered')
## Instruction 6:
#Write a Python function that adds a new band to an image, representing the
    ratio between the 'VH' and 'VV' bands from Sentinel-1 satellite data.
    The function should compute the ratio as the difference between the 'VH'
    and 'VV' bands, convert it to a floating-point number, and then add it
    as a new band named 'ratio'. After defining this function, apply it to a
    pre-existing image collection named collection_S1data_Filtered, mapping
    it across the entire collection to create a new image collection named
    collection_S1data_Filtered_Ratio.
## Passed Generation:
```

38 • Chen and Wang, et al.

```
original_image = collection_S1data.first()
Map.setCenter(aoi.getCenter(), 10)
Map.addLayer(original_image, {'min': -20,'max': 0}, 'Original Data')
filtered_image =collection_S1data_Filtered.filterDate(start_date,
    end_date).first()
Map.addLayer(filtered_image, {'min': -20,'max': 0}, 'Filtered Data')
first_image = collection_S1data_Filtered_Ratio.filterDate(start_date,
    end_date).first()
last_image = collection_S1data_Filtered_Ratio.filterDate(start_date,
    end_date).sort('system:index').last()
Map.addLayer(first_image.select('VH_VV_ratio'), {'min': 0,'max': 1}, 'VH/VV
    Ratio (First Image)')
Map.addLayer(last_image.select('VH_VV_ratio'), {'min': 0,'max': 1}, 'VH/VV
    Ratio (Last Image)')
## Instruction 10:
#Create a binary image by applying a threshold on LogRatio, where values
    greater than 1 are classified as change (True) and others as no change
    (False). Add the binary change detection layer to the map using the
    Map.addLayer() function. The visualization should use the 'logratio'
    band with values ranging from 0 to 1.
## Passed Generation:
vh_vv_ratio_image_logratio = LogRatio.gt(1).log().rename('logratio')
Map.addLayer(vh_vv_ratio_image_logratio, {'min': 0, 'max': 1}, 'VH/VV Ratio
    (Logratio)')
## Human Intervention
SARchange = LogRatio.gt(1)
Map.addLayer(SARchange, {'bands': ['logratio'], min: [0], max: [1]},
    'Sentinel-1 Log-Ratio Image')
```

```
def add_ratio_band(image):
    ratio = image.select('VH').subtract(image.select('VV'))\
    .divide(image.select('VV')).multiply(100)
    ratio_image = image.addBands(ratio.rename('ratio'))
    return ratio_image
collection_S1data_Filtered_Ratio =
    collection_S1data_Filtered.map(add_ratio_band)
## Human Intervention
def add_vhvv_ratio(image):
    vh_vv_ratio =
        image.select('VH').subtract(image.select('VV')).float().rename('ratio')
    image = image.addBands(vh_vv_ratio)
    return image
collection_S1data_Filtered_Ratio =
    collection_S1data_Filtered.map(add_vhvv_ratio)
## Instruction 7:
#Generate Python code using the Google Earth Engine (GEE) Python API to
    visualize Sentinel-1 data. Convert the first 30 images in
    collection_S1data_Filtered_Ratio to a list using the toList() method.
    Assign this list to the variable S1_listData. Visualize the first image
    in this list on a map, clipped to the Area of Interest (aoi). Display
    the image in RGB format using the VV, VH, and ratio bands. Set the
    visualization parameters with minimum values of [-18, -23, 3] and
    maximum values of [-4, -11, 15]. Add this visualization to the map with
    the label sentinel-1 GRD RGB First. Repeat the visualization for the
    last image in the list, adding it to the map with the label Sentinel-1
    GRD RGB Last.
## Failed Generation:
first_image = S1_listData.get(0)
last_image = S1_listData.get(-1)
first_image_clipped = first_image.clip(aoi)
Map.addLayer(first_image_clipped.select('VV', 'VH', 'ratio'), {'min': [-18,
    -23, 3], 'max': [-4, -11, 15]}, 'sentinel-1 GRD RGB First')
last_image_clipped = last_image.clip(aoi)
S1_listData = collection_S1data_Filtered_Ratio.limit(30).toList(30)
```

An LLM Agent for Automatic Geospatial Data Analysis • 39

12.7 Ethics Statement

我们的基准测试是使用Llama3.1和各种在线资源开发的。它由专家精心设计，以确保排除任何私人信息。此外，我们利用GPT-4o对基准测试和论文写作中的文本进行润色，纠正拼写和语法错误。虽然Llama3.1存在伦理风险，例如可能生成有害内容，但本基准测试中生成的所有指令均经过人工审核，代码在沙箱环境中执行，以减轻这些风险。大型语言模型（LLMs）以其高计算需求而著称，这引发了环境问题。然而，我们的主要工作不涉及训练如此大规模的模型；相反，仅涉及在基准测试上的推理，这导致计算足迹相对较低。本研究进行的所有实验均在单个NVIDIA RTX 4090机器上完成。

```
Map.addLayer(first_image.clip(aoi), {'min': [-18, -23, 3], 'max': [-4, -11, 15]}, 'sentinel-1 GRD RGB First')
Map.addLayer(last_image.clip(aoi), {'min': [-18, -23, 3], 'max': [-4, -11, 15]}, 'Sentinel-1 GRD RGB Last')
## Error Feedback
NameError : name 'S1_listData' is not defined
## Self-fixed Generation:
S1_listData = collection_S1data_Filtered_Ratio.toList(30)
first_image = ee.Image(S1_listData.get(0))
last_image = ee.Image(S1_listData.get(-1))
first_image_clipped = first_image.clip(aoi)
Map.addLayer(first_image_clipped.select('VV', 'VH', 'ratio'), {'min': [-18, -23, 3], 'max': [-4, -11, 15]}, 'sentinel-1 GRD RGB First')
last_image_clipped = last_image.clip(aoi)
Map.addLayer(ee.Image(S1_listData.get(29)).clip(aoi),{'bands': ['VV', 'VH', 'ratio'],min: [-18, -23, 3], max: [-4, -11, 15]}, 'Sentinel-1 GRD RGB Last')
## Instruction 8:
#The first step is to select the Sentinel-1 radar images before and after the event from the provided list (S1_listData). Both images are clipped to a specified area of interest (aoi). Specifically, select the VH polarization band from the images. S1_before is the image before the event, and it's selected as the first image in the list. S1_after is the image after the event, selected as the 30th image in the list. Calculation of Log Ratio: Compute the Log Ratio by subtracting the S1_after image from the S1_before image. The result is then divided by 10 to normalize the values, making the output more scalable. This operation results in a new band named 'logratio'. Visualization: Finally, visualize the resulting Log Ratio image on the map. Set the visualization parameters with a min value of -1.5 and a max value of 1.5.
## Sucessful Generation:
S1_before = ee.Image(S1_listData.get(0)).select('VH').clip(aoi)
S1_after = ee.Image(S1_listData.get(-1)).select('VH').clip(aoi)
LogRatio = S1_before.subtract(S1_after).divide(10).rename('logratio')
print('Log Ratio Band (Change Image):', LogRatio)
```

```
Map.addLayer(LogRatio, {'bands': ['logratio'], min: [-1.5], max: [1.5]},
    'Sentinel-1 Log-Ratio Image')
## Instruction 9:
#Extract the first image from the Sentinel-1 dataset (collection_S1data).
    Center the map on the area of interest (aoi) with a zoom level of 10.
    Add the original Sentinel-1 image layer to the map with a visualization
    range (min: -20, max: 0).
#Display Filtered Data: Filter the smoothed Sentinel-1 dataset
    (collection_S1data_Filtered) to retrieve images within a specified date
    range (start_date to end_date). Extract the first image from the
    filtered dataset. Add this filtered image layer to the map with the same
    visualization range (min: -20, max: 0).
#Display VH/VV Ratio Data (collection_S1data_Filtered_Ratio): Extract the
    VH/VV ratio band from the first image of the filtered smoothed dataset.
    Extract the VH/VV ratio band from the last image of the filtered
    smoothed dataset, sorted by the system:index.\nAdd these VH/VV ratio
    layers to the map with a visualization range (min: 0, max: 1). Label the
    layers as VH/VV Ratio (First Image) and VH/VV Ratio (Last Image)
    respectively.
## Failed Generation:
filtered_S1_layer = collection_S1data.select('VH').clip(aoi)
Map.addLayer(filtered_S1_layer, {'bands': ['VH'], 'min': [-20], 'max': [0]},
    'Filtered Sentinel-1 Image')
vh_vv_ratio_first_image =
    ee.Image(collection_S1data_Filtered.get(0)).select('vhvv_ratio')
vh_vv_ratio_last_image =
    ee.ImageCollection(collection_S1data_Filtered).filterDate(start_date,
    end_date).sort('system:index').last().select('vhvv_ratio')
Map.addLayer(vh_vv_ratio_first_image.clip(aoi), {'bands': ['vhvv_ratio'],
    'min': [0], 'max': [1]}, 'VH/VV Ratio (First Image)')
Map.addLayer(vh_vv_ratio_last_image.clip(aoi), {'bands': ['vhvv_ratio'],
    'min': [0], 'max': [1]}, 'VH/VV Ratio (Last Image)')
## Error Feedback
AttributeError : 'ImageCollection' object has no attribute 'clip'
## Human Intervention
```

```
original_image = collection_S1data.first()
Map.setCenter(aoi.getCenter(), 10)
Map.addLayer(original_image, {'min': -20,'max': 0}, 'Original Data')
filtered_image =collection_S1data_Filtered.filterDate(start_date,
    end_date).first()
Map.addLayer(filtered_image, {'min': -20,'max': 0}, 'Filtered Data')
first_image = collection_S1data_Filtered_Ratio.filterDate(start_date,
    end_date).first()
last_image = collection_S1data_Filtered_Ratio.filterDate(start_date,
    end_date).sort('system:index').last()
Map.addLayer(first_image.select('VH_VV_ratio'), {'min': 0,'max': 1}, 'VH/VV
    Ratio (First Image)')
Map.addLayer(last_image.select('VH_VV_ratio'), {'min': 0,'max': 1}, 'VH/VV
    Ratio (Last Image)')
## Instruction 10:
#Create a binary image by applying a threshold on LogRatio, where values
    greater than 1 are classified as change (True) and others as no change
    (False). Add the binary change detection layer to the map using the
    Map.addLayer() function. The visualization should use the 'logratio'
    band with values ranging from 0 to 1.
## Passed Generation:
vh_vv_ratio_image_logratio = LogRatio.gt(1).log().rename('logratio')
Map.addLayer(vh_vv_ratio_image_logratio, {'min': 0, 'max': 1}, 'VH/VV Ratio
    (Logratio)')
## Human Intervention
SARchange = LogRatio.gt(1)
Map.addLayer(SARchange, {'bands': ['logratio'], min: [0], max: [1]},
    'Sentinel-1 Log-Ratio Image')
```

7.7 Ethics Statement

Our benchmark is developed using Llama3.1 and a variety of online resources. It is carefully propossed by experts to ensure the exclusion of any private information. Furthermore, we utilize GPT-4o for refining text and correcting typographical errors and syntax in both the benchmark and paper writing. Although Llama3.1 is associated with ethical risks, such as the potential generation of harmful content, all generated instructions within this benchmark undergo human review, and the code is executed in sandbox environments to mitigate these risks. LLMs are known for their high computational demands, which pose environmental concerns. However, our primary work does not involve training such a large model; instead, it involves only inference on benchmarks, which results in a relatively low computational footprint. All experiments conducted for this study were performed on a single NVIDIA RTX 4090 machine.