The Cloud-Based Geospatial Benchmark: Challenges and LLM Evaluation

Jeffrey A. Cardille[*1,2] **Renee Johnston**[1] **Subhashini Venugopalan**[1] **Simon Ilyushchenko**[1] **Zahra Shamsi**[1] **Johan Kartiwa**[1] **Matthew Abraham**[1] **Khashayar Azad**[4] **Nuala Caughie**[2] **Emma Bergeron Quick**[2] **Karen Dyson**[5] **Andrea Puzzi Nicolau**[5] **Fernanda Lopez Ornelas**[6] **David Saah**[6] **Michael Brenner**[1,3] **Sameera Ponda**[1]

[1] *Google Research* [2] *McGill University* [3] *Harvard University* [4] *Concordia University*
[5] *Spatial Informatics Group* [6] *University of San Francisco*

Abstract

With the increasing skill and adoption of Large Language Models (LLMs) in the sciences, evaluating their capability in a wide variety of application domains is crucial. This work focuses on evaluating LLM-based agents on Earth Observation tasks, particularly those involving the analysis of satellite imagery and geospatial data. We introduce the Cloud-Based Geospatial Benchmark (CBGB), a set of challenges designed to measure how well LLMs can generate code to provide short numerical answers to 45 practical scenarios in geography and environmental science. While the benchmark questions are framed to assess broadly applicable geospatial data analysis skills, their implementation is most readily achieved using the extensive data catalogs and powerful APIs of platforms like Earth Engine. The questions and reference solutions in CBGB were curated from experts with both domain familiarity in Earth Observation and programming expertise. We also estimate and include the difficulty of each problem. We evaluate the performance of frontier LLMs on these tasks with and without access to an execution environment for error-correction based feedback. Using the benchmark we assess how LLMs operate on practical Earth Observation questions across a range of difficulty levels. We find that models with the error-correction feedback, which mirrors the iterative development process common in geospatial analyses, tend to perform consistently better with the highest performance at 71%; the reasoning variants of models outperformed the non-thinking versions. We also share detailed guidelines on curating such practical scenarios and assessing their ability to evaluate agents in the geospatial domain. The benchmark and evaluation code are available on Github[1].

Keywords: Geospatial, earth engine, LLM, evaluation, benchmark

1. Introduction

The rapid proliferation and increasing sophistication of Large Language Models (LLMs) have catalyzed transformative shifts across numerous scientific disciplines Zhang et al. (2024b). Initially demonstrating remarkable proficiency in natural language understanding and generation, the capabilities of these models are now being extended and evaluated for more specialized, complex tasks central to scientific inquiry. Among these are code gen-

* jeffrey.cardille@mcgill.ca

1. https://github.com/google/earthengine-community/tree/master/experimental/cbgb_benchmark

eration, where LLMs show promise in translating natural language instructions into executable programs Li et al. (2022), and scientific problem-solving Cui et al. (2025), encompassing domains like mathematics, physics, and chemistry Arora et al. (2023). This expansion necessitates rigorous evaluation methodologies and benchmarks to accurately gauge LLM capabilities, identify limitations, and guide future development, particularly for applications in scientific domains where reliability and precision are paramount Zhang et al. (2024b). The evaluation landscape itself is evolving. Early benchmarks often focused on static, single-turn interactions, assessing performance on discrete tasks. However, recognizing that real-world problem-solving frequently involves iterative refinement, interaction with external tools, and multi-step reasoning, newer evaluation paradigms emphasize LLM-based agents operating within interactive environments Carta et al. (2023). These agents leverage LLMs for planning and reasoning, interacting with tools like code interpreters or APIs, and adapting based on feedback or environmental state changes Yehudai et al. (2025).

Within this context, Geospatial Artificial Intelligence (GeoAI) Janowicz et al. (2020) represents a burgeoning interdisciplinary field that integrates AI and Machine Learning (ML) techniques, including LLMs, with geospatial data and methods to address complex environmental and societal questions. The application of AI, and specifically LLMs, to geospatial problems presents unique challenges stemming from the nature of the data – often multimodal (e.g., satellite imagery, vector data, sensor readings, textual descriptions) Lacoste et al. (2023), inherently spatial, requiring specialized libraries and analytical techniques, and frequently massive in scale Wu et al. (2024). While LLMs and related Vision-Language Models (VLMs) are being explored for various GeoAI tasks Chen et al. (2024b), a significant gap exists in evaluating their proficiency in generating correct and efficient geospatial code specifically for satellite-driven analytical tasks.

The Cloud-Based Geospatial Benchmark (CBGB) is introduced to address this gap, and is intended to provide a set of novel challenges that represent practical scenarios for doing cloud-based geospatial analyses (Fig. 1). The curated problems are designed to measure LLM and agent performance in generating geospatial code for satellite data analysis with particular focus on Google Earth Engine. Each problem focuses on achieving specific numerical outputs and the dataset incorporates expert knowledge, difficulty estimation, and interactive refinement capabilities. We evaluate the performance of several frontier LLMs including their reasoning variants with and without access to execution feedback with an error-correcting loop. The best performing agents achieve an accuracy of 71%. Further, we find that agents with access to the execution environment consistently outperform the base LLM solution indicating that even the best reasoning agents have room for improvement and underscores the challenging nature of the benchmark.

2. Related Works

Benchmarking and Improving Geospatial Code Generation. Generating code for geospatial tasks poses specific challenges for LLMs. These models often lack the deep, specialized knowledge required to effectively use common geospatial libraries (e.g., GeoPandas, platform-specific APIs like Google Earth Engine or ArcGIS). The use of fundamental geospatial concepts like map projections, coordinate reference systems, and spatial operations defining adjacency, containment, and proximity Ahearn et al. (2013) appear to be rare in model training data. This has the potential to lead to substantial "code hallucination", where models generate syntactically

Intermediate : Agricultural Land Use Analysis in California's Central Valley

Objective : Use Landsat and USDA datasets to calculate the total area of almond-growing fields in California's Central Valley that exhibit healthy vegetation.

- You are given the following datasets:
  - Landsat 8 Surface Reflectance: "LANDSAT/LC08/C02/T1_L2" Precise Date Range: Analyze imagery from all of 2021, removing images of more than 5% cloud cover.
  - USDA National Agricultural Statistics Service (NASS) Cropland Data Layer (CDL): "USDA/NASS/CDL", also for 2021.
- Geographic Scope: Focus on California's Central Valley. Define the region using the following bounding box coordinates: ([-121.7, 37.1], [-120.0, 37.1], [-120.0, 36.0], [-121.7, 36.0]) Analysis Task:
- Filter the Landsat 8 ImageCollection for the specified date range and geographic area.
- Calculate the median Normalized Difference Vegetation Index (NDVI) among those images.
- Identify almond-growing fields (NASS CDL code 75) within the specified Central Valley region using the 2021 CDL data.
- Calculate the area (in hectares) of almond-growing fields (pixels identified as Almonds) where the NDVI within almond-growing fields is greater than 0.25. Round your answer to the nearest 1000 hectares.

Notes: Ensure band values (e.g., radiance, temperature) are scaled to their proper units prior to use. Reflectance values should be scaled to between 0 and 1 prior to use.

Figure 1: An example challenge of intermediate difficulty from the CBGB benchmark.

plausible but functionally incorrect or non-executable code.

To explore the capacity of models to demonstrate spatial awareness and capability, a number of benchmark approaches have emerged. GeoCode-Eval / GeoCode-Bench Hou et al. (2025) introduces a comprehensive framework and a large-scale dataset featuring simple question formats (multiple-choice, fill-in-the-blank, etc.) generated via self-instruct and expert review, assessing LLMs across cognition, application, and innovation dimensions. GeoLLM-Engine Singh et al. (2024) proposes several small tasks routinely executed for geospatial analysis for "tool-use" agents. These tasks focus more on the LLM being able to call geospatial APIs for the right tools including maps, web UIs, and multimodal knowledge bases to evaluate if natural language instructions are translated into correct functional invocations for task completion. GeoLLM-Squad Lee et al. (2025) builds on GeoLLM-Engine to orchestrate multiple agents to perform some geo-spatial tasks. However their focus is more on developing subagents for specific tasks and creating an orchestrator to effectively compose multiple smaller components, and not directly focused on code generation. Closer to our work is Gramacki et. al. Gramacki et al. (2024), which categorize geospatial tasks based on their complexity and required tools and cre-

ate tasks that test model capabilities in spatial reasoning, spatial data processing, and geospatial tools usage. Their dataset consists of manually created coding problems and they propose a set of test cases to enable automatic verification of code correctness. A key difference is that their tasks are more focused simpler components of a geoanalyst workflow such as calculating areas etc. as opposed to a full fledged problem like ours. Similarly, GeoAgent Chen et al. (2024b) introduces a framework for geospatial data processing that integrates a code interpreter, static analysis, and retrieval-augmented-generation (RAG) within a Monte Carlo Tree Search algorithm, as well as a benchmark dataset to evaluate LLM-based agents in this domain. The GeoAgent coding environment leverages many geospatial libraries including the Google Earth Engine API, but, in contrast to CBGB, the GeoAgent Chen et al. (2024b) framework evaluates LLM performance by comparing how closely the function signatures within the generated code match the ground truth code, effectively treating the function signatures as "labels" rather than comparing the expected code output directly. GEE-OPs Hou et al. (2024) focuses specifically on the Earth Engine JavaScript API, building a structured knowledge base using over 185,000 Earth Engine scripts and documentation. RAG systems on these scripts improve performance on geospatial code generation by 20-30%. A key difference between existing benchmarks and our work is that our benchmark evaluates the end-to-end problem-solving capability starting with a natural language description of the geospatial problem and resulting in a specific analytical outcome, rather than solely focusing on intermediate code artifacts.

Autonomous Geospatial Agents and Systems. Beyond generating static code snippets, there is research exploring the development of autonomous geospatial agents and integrated systems capable of performing complex analysis workflows based on high-level user instructions. GIS Copilot Akinboyewa et al. (2025) is designed as an autonomous agent integrated within existing GIS platforms like QGIS. It leverages LLMs to interpret natural language commands, decompose requests into analysis steps, and intelligently select appropriate geospatial tools to generate code. GeoGPT Zhang et al. (2024a) takes natural language requests, and autonomously plans and executes sequences of GIS tools to accomplish tasks such as geospatial data collection, spatial querying, and analysis, aiming to lower the barrier for users without deep GIS expertise. Geo-OLM Stamoulis and Marculescu (2025) explores using smaller, open-source language models for geospatial agent tasks by employing state-driven reasoning to decouple planning from tool execution. Complementary to agent development, frameworks have been created to extract and reuse knowledge embedded in existing geospatial workflows (e.g., Chen et al. (2024a)). These works shift focus from generating isolated code segments to creating integrated systems. This practical consideration suggests that benchmarks aiming for real-world relevance should ideally evaluate interactions with standard tools and environments, a principle reflected in our benchmark's use of a sandbox environment.

3. Creating the questions and solutions dataset

The benchmark challenge set is designed to present practical, recognizable problems encountered in remote sensing analyses in studies around the world, span a large amount of conceptual space, and build credibility for the problems and provide answers. Using a variety of combinations of human expertise and LLM suggestions, we produced and answered a set of 45 challenge cases (in the supplement zip, Appendix A). Challenges are categorized

into three difficulty levels (Easy, Intermediate, Difficult) reflecting an estimate of the complexity of geospatial/programming knowledge required, as well as a time estimate of how long it would take an inexperienced and an experienced human to solve the challenge. They span a range of scales (local to regional), sensors (optical and SAR), techniques (e.g., area summary, time-series analysis, cloud treatment, vector analysis, etc.) and topics (e.g., forest change, land-cover classification, road analysis, rainfall calculations, etc.).

3.1. Distinguishing features

Numerical Output A distinguishing feature of the CBGB benchmark is that challenge responses exclusively require a numerical nonzero output. Although each challenge is presented as a real-world problem that accesses and interprets known data to produce maps and related visual artifacts, the solutions require a single quantity to be presented as the challenge answer. This can take the form of, for example, the value of an index at a single point in space, the slope of a line in a time series, or a calculation averaged over a region.

Single Answer Through iterative testing and analysis of draft challenges and analysis of the results, we constrained the freedom within each challenge so that any two agents or people doing the challenge should necessarily arrive at the same, intercomparable answer. The focus on a single numerical answer allowed problems to be potentially quite complex, with multiple maps made, without needing to assess the quality of maps that could contain millions or billions of pixels and resampled for viewing at a particular dpi.

Language- and Platform-independent There was no code evaluation in assessing an answer, which allowed answers to be independent of the language that might be chosen– in particular, the Earth Engine API supports both JavaScript and Python approaches. This had the additional advantage of allowing evaluation to be fully independent of programming style, which can be highly subjective. To the greatest extent possible, we made the challenges independent of the Earth Engine platform. This meant that while Earth Engine might well be the most straightforward place for solving a problem, it was not strictly necessary to use. Challenges do not make explicit appeals to use particular Earth Engine calls, and users can potentially solve problems on other platforms. For several challenges, we developed answers on both Earth Engine and on ArcGIS to confirm platform independence.

Exact Match Challenges were constrained in such a way that only exact matches could count as correct. Most problems specify that three digits to the right of the decimal point should be included in the answer. For the few challenges that allowed a tolerance around a value- for example, a specification to create a buffer to within a certain percentage of the size of the original area- we rounded the expected answer to encompass the range of valid values. In that situation, models that operated correctly within the tolerance of the problem would print the same, appropriately rounded answer.

3.2. LLM assistance to create candidate benchmark challenges ("Challenge Builder")

Book-inspired cases The book *Cloud-based Remote Sensing with Google Earth Engine: Fundamentals and Applications* Cardille et al. (2023) provided a domain-relevant template for creating and sequencing challenges within this domain. We generated draft challenges by loading each of that book's "Fundamentals" chapters into NotebookLM. Then, to derive a challenge driven by a given chapter, we toggled on all chapters prior to and including a given chapter of interest, to better target the

knowledge acquired by having done a given chapter and its preceding ones.

Because the book builds knowledge throughout the chapters, introducing new topics in each, it was feasible to develop varied real-world challenges demanding increasingly sophisticated knowledge of remote sensing techniques. The earliest chapters were more suitable for Easy challenges, while more advanced chapters could accommodate Intermediate and Difficult challenges. The draft challenges proposed by NotebookLM exercised its ability to work within a given context while also creatively envisioning distinctive challenge settings. As an example, topics include "Land Cover Classification of Milan", "Burn Severity Change Detection in the Amazon Rainforest", and "NDVI Harmonic Modeling for Cropland Phenology".

Unrestricted challenges In addition to 33 draft challenges inspired by the book, we drafted 12 challenges without reference to any specific chapter. These Unrestricted challenges were devised freely from human imagination. As an example, topics include "Combining Watersheds for Area Calculation" and "Analyzing Long-Term Land Surface Temperature Trends in an Urban Area". Drafts were shaped like those proposed by LLMs, as described below.

3.3. Shaping draft challenges for inclusion

During the challenge creation process, draft cases were seldom, if ever, sufficiently concise, challenging, and clear enough to be used without substantial modification. Draft challenges were shaped through an iterative process to be suitable for use, which typically further arrange their structure, precision, and goals to imply unambiguous processing and outcomes. This was done in two phases: the "Challenge Checker" and "Uncertainty Divergence" checks, as described below.

3.3.1. Enumerated basic checks: "Challenge Checker"

We used human judgement with LLM assistance to identify and remove any elements of a draft challenge that caused it to be underspecified. Each draft case was inspected by eye with respect to 31 criteria that we devised for producing well-constrained and well-presented challenges. To supplement human judgement of these criteria, a separate LLM instance (Gemini, chatGPT, or NotebookLM) was tasked with considering the same criteria. This process was intended to identify phrasing in the draft that, unless made more clear, could cause agents or humans to complete a problem while arriving at a different answer than the expected right answer. An example is a draft challenge that might have asked to "combine images from summer 2017" for a particular purpose. The meaning of "summer", however, is imprecise: the term could be interpreted by one model as astronomical summer, and by another as a particular set of months. In that situation, we would revise the problem to ask for images between two specific dates, further specifying whether the date endpoints were inclusive or exclusive. Similarly, direction to simply "combine" images is imprecise; in that situation, we would revise the problem to request the mean, median, or some other way of creating a single image from a set.

Challenge Checker criteria occurred in four main groups; the groups and a representative criterion are given below.

(a) Dataset Existence, Specificity, and Accuracy. Check 4 "Data needed". Each data set given in the problem must contain information that is necessary to solve the problem. Data that is "of interest" but not necessary to solve the problem should not be included.

(b) Reproducibility. Check 18 "Pixel masking". If pixel masking is needed to constitute a reproducible problem, it must be stated

explicitly. If pixel-level masking is used, the threshold or method must be clearly stated in a reproducible manner.

(c) Clarity and completeness. Check 25 "No slang". There should be no English slang in the problem that could confuse.

(d) Clearly defined outputs. Check 29 "Clear answer". The problem should necessarily lead to a numerical or Boolean output. If a map is made, a single point should be specified in the problem statement to inspect its value. If a chart is implied, a single value should be requested.

3.3.2. Cross-agent comparisons: "Uncertainty Divergence"

Despite the careful criteria of the Challenge Checker, there still remained noticeable imprecision in early drafts of each case. We then subjected each challenge to an agent-agnostic test, meant to assess whether it was yet unambiguously interpretable. Because our requirement for an effective challenge was to have only a single correct answer, the agent-agnostic test ingested the agent code outputs on a given draft. An LLM instance looked for situations in which difficult-to-detect elements of uncertainty could produce more than one viable answer to a draft challenge. An example of the prompt and a report used during the Uncertainty Divergence detection process can be seen in Appendix B.

For draft challenges in which Uncertainty-derived Divergence was detected, the analysis report offered a substantive explanation of the imprecision and proposed an amended challenge formulation. We considered those amendments, adjusted wording, and re-assessed subsequent drafts of each challenge. (It is important to emphasize that this assessment was done without regard to which particular model or models were divergent.) As we shaped challenges, where analysis reports and agent behavior indicated that our human interpretation of a problem under development was flawed or needed to change, we further constrained or clarified a problem or our approach as needed. The process of isolating and excising uncertainty to remove divergent answers for a challenge continued until three conditions applied: (1) at least one model agreed with our answer; (2) there were no clusters of agent answers indicating lingering uncertainty; and (3) the reports for a draft challenge stated that there was no lingering uncertainty in the revised problem statement.

3.4. Checks for challenge existence and topic plagiarism

As models proposed cases, we explored whether the settings and problem statements already appeared in some form online. For each challenge, we looked for similar phrasing both online and within Google Scholar. In our search, no challenge appeared directly as part of any benchmark, tutorial, or written source that models could potentially have been trained on. Only a few of the proposed challenges appeared to us to overlap substantively during this verification stage; those were either modified to a new location or time frame. To address any overlap that might be perceived by others, or for authors to link a benchmark to their similar work, we created a form (bit.ly/CBGB-overlap) that will allow an interested party to suggest an information source as a possible "For Further Reading" link, to associate with a challenge that they judge to be similar or of interest.

After shaping through iteration with the Challenge Checker and the Uncertainty Divergence tests, challenges were clear enough to produce unique results. An example Intermediate Challenge is shown in Fig. 1; example Beginner and Difficult challenges can be seen in Appendix C.

4. Experiments

Experimental Setup To enable code execution, we set up a colab-based sandbox environment including access to standard Python libraries and network access to the Google Earth Engine Python API. We ran the experiment against the following 10 models: Claude 3.5 Haiku; Claude 3.5 Sonnet; Claude 3.7 Sonnet; DeepSeek-VL Chat; DeepSeek-R1; Gemini 2.5 Pro; Gemini 2.5 Flash; Gemini 2.0 Flash; OpenAI o3; and OpenAI o4-mini. (Full versioning information is in Appendix E). For each challenge, we prompted each of the models in an isolated zero-shot manner, without any reliance on a model's notion of "chat history." We evaluated models under two execution workflows:

- **Base model**: In this case we used a single prompt (Appendix D.1) to each LLM to generate Earth Engine Python code to solve the problem. We sampled a single solution from the model with temperature 0, and evaluated performance.

- **Error-correction**: In this workflow, if an error occurred while executing the LLM's initial code, a new instance of the same LLM (with no prior chat history) was prompted with the original problem, the failing Python code, the corresponding error message, and was instructed to generate a working version of the code. This error correction prompt cycle was permitted up to 3 times (Appendix D.2).

5. Results

Distinct performance tiers are observable among the models (Figure **??**), where "reasoning" models such as o3, Gemini 2.5 Pro, Claude Sonnet, and Deepseek R1 demonstrate superior baseline performance over the baseline performance of "lighter" models such as Gemini 2.5 Flash, o4-mini, and Claude Haiku.

The range of overall model success contains distinctive patterns on Easy, Medium, and Difficult problems. In considering the error-correcting runs of the highest-performing, fifth-best (near the median performance), and lowest-performing models, there are important similarities beyond the top-line difference. Easy problems require one or two spatial operations, and often involve establishing a study area over which to calculate and summarize an index. We estimate that a beginner could solve a typical Easy problem in around 15-30 minutes; an Advanced user would need about 5-10 minutes. Even Easy problems (Appendix C, Figure **??**) demand sophistication and multi-step thinking: to be successful on an Easy problem, a model must properly parse a problem, prepare to sequence several steps, consult the Earth Engine API, and access and present the answer precisely as specified. Models succeeded similarly on the 15 questions labeled "Easy", correctly answering 12, 13, and 8 correctly among the highest-, median-, and lowest-performing models. The similarity in model success suggests a very substantial baseline competence in LLMs for reliably solving this class of problem.

For Intermediate problems, the models diverged in their capacity somewhat more than on Easy problems. We estimate that this class of problem would require about 1 hour for a Beginner, and about 20-30 minutes for an Advanced user. The highest-, middle-, and lowest-performing models answered 15, 15, and 5 of the 20 questions correctly. That the middle-performing model got about half of the Intermediate questions right suggests that an LLM user looking for help on such a problem might benefit from presenting the problem in parts.

Models diverged considerably in their ability to answer a given Difficult problem. We estimate that this class of problem would require 3 hours or more for an Advanced user, likely demanding that they have command

of sophisticated techniques for which there are limited worked examples online. We estimate that Beginner users would not be able to answer a Difficult problem. The highest-, middle-, and lowest-performing models answered 5, 1, and 1 of the 10 problems correctly. In the model space, the lowest-performing models were, in general, unable to correctly answer a given Difficult problem. It is worth noting that for a given Difficult problem, it was answered by fewer than 3 of the models. For a typical Difficult challenge, one or two of the high-performing models answered correctly, plus perhaps one of the median-performing models. Given that the Difficult problems were sometimes composed of multiple smaller steps, this suggests strongly that an LLM user could benefit greatly from breaking a Difficult problem into several components, then working interactively toward the answer over an extended period. For Beginner and Intermediate users, this could help them to work through the logic and learn any techniques currently outside their skill set; for Advanced users, this could help to increase the credibility of an LLM's answer.

6. Discussion

The benchmark is not saturated. This CBGB test set is not saturated, particularly on Difficult questions. Models in this space are rapidly improving; when we began exploring geospatial code generation in early 2024, few models could succeed on all but the easiest problems. This placed them around the position of the lowest-performing models today. In the intervening months, models have improved considerably, yet are substantially challenged to reliably solve more complex cases. Difficult problems, however, are not entirely out of reach, nor purposely confusing. At least one model was able to solve each of the Difficult problems, a fact we used to identify candidate problems that were Difficult but demonstrably solvable. Taken as a whole, the benchmark state suggests that models can be extremely helpful for solving everyday problems and to aid in coding- and, that additional meaningful progress is attainable.

A benchmark of realistic, scaled challenges is constructable. Other published benchmarks to date have mostly focused on building blocks of geospatial analysis, and have limited resemblance to the challenges presented here. This benchmark set presents problems meant to sound like what users would ask for in a real-world setting- though with the important caveat of having removed as much ambiguity as possible in the problem presentation. Challenges can be reasonably partitioned into broad categories, which illustrate a near-universal basic agent competence on "Easy" problems involving a few steps and commonly seen approaches.

Error correction is effective across models. Error correction consistently improved model performance. The performance of "lightweight" models such as o4-mini and Gemini 2.5 Flash with error correction exceeds the baseline results of most reasoning models without error correction, particularly on Easy and Intermediate questions. For example, in many cases, Gemini 2.5 Flash with error correction appears to be just as effective at solving problems as Gemini 2.5 Pro without error correction. This suggests that many of the remaining unsolved problems (perhaps especially in the Easy and Intermediate ranks) could be improved even further by more permissive and targeted error correction protocols. Exploring the precise behavior, optimal strategies and upper limits of error correction was outside the scope of this work, but could be a fruitful additional study to improve overall model performance, as could the exploration of other agentic reasoning structures.

The benchmark process can inform domain-specific improvements. Through the iterative effort to remove uncertainty-derived divergence from draft challenges, we learned that errors tended to be due to a mix of incorrect decision-making, poor data awareness, and syntax errors. In our experience, incorrect decision-making often took the form of improper ordering of steps, or making choices that may be wise scientifically but that nevertheless were counter to the problem statement (for example, deciding that it would be good to remove both cloud *and* shadow from an image when only removing clouds was requested). The nature of these limitations suggests that improvement on some of the Difficult challenges can be achieved.

Uncertainty comes in different flavors. Friction points were uncovered that were both general and specific, and are tied to this geospatial domain. High-level questions of general interest (e.g., "How much reforestation was there recently around Bolivia") are reasonable for informal scoping and discussion, but are too imprecise for any two models (or humans) to be expected to arrive at the same answer without very substantial additional shaping. The simple clarity of, for example, an algebra problem is not commonly encountered; rather, users should be guided in some way (through an interface or a checklist such as that produced here) so that results are maximally reproducible and better understood. A second type of uncertainty is when necessary details of repeatability are missing but not easily recognized by all but expert users. For example, cloud masking is a persistent challenge in most of the world's ecosystems, but the details of cloud bits on each sensor platform are not widely known by Beginner and Intermediate users. To use the benchmark set as a guide for shaping the experience of a user in this domain (notably, future users of a bespoke interface to aid in geospatial queries), overcoming these two types of uncertainty should be understood closely to aid users.

Lessons on uncertainty-derived divergence should be transferable to real problem-solving settings. As we created problems, we were surprised at how precise a problem needed to be in order to be led unambiguously to a single answer among a set of independently operating agents, and at how many iterations it took to properly constrain problems for this benchmark. The 31 characteristics of the Challenge Builder grew iteratively, as we learned that not being precise on each one would produce important uncertainty for zero-shot problem-solving. More broadly, in a real-world setting where a user would like an answer to a problem in this domain, the credibility of agent responses can be enhanced by guiding them through these issues, to confirm that a user will ultimately understand and be able to reproduce the analysis.

Limitations

Despite the success of this work, there are several limitations that must be considered.

First, although there are a substantial number of cases, small distinctions among the models cannot be seen as statistically significant. Rather, results should be taken as an indication of model success and growth, and not as a leaderboard.

Second, because models can and do produce different code on different runs, there are some cases for which a model was able to produce correct code on one day, and incorrect code on the next. As a result, we believe it is not warranted to propose blanket statements such as "Model X fails on Y percent of Intermediate cases, and fails on most Difficult cases." We see the model performance summarized here as a baseline against which future models can be compared against earlier versions.

Third, it is worth noting that Google Earth Engine is only one platform for doing cloud-based geospatial computation. Our intention was to create a benchmark that would give consistent results on any platform. We endeavored to limit the dependence of cases on Earth Engine, so that they can also be considered as challenges for LLMs operating in other environments. In cases where we think platform-specific calculations might cause different results (such as summarizing over an area), we requested a rounded answer to provide cushion for implementation differences.

Fourth, despite our efforts to excise uncertainty, there may ultimately prove to be undetected uncertainty in some of the problem statements. Since we inspected the model results for clustering of answers (for example, some models getting '8' and others getting '6'), any lingering uncertainty should not have had an effect on the model runs shown here. If additional uncertainty is detected, we expect to release enhanced versions of CBGB with any needed amendments.

7. Conclusion

This paper introduces the Cloud-Based Geospatial Benchmark (CBGB), a set of 45 challenges designed to evaluate the capabilities of Large Language Models and LLM-based agents in generating executable code for Earth Observation tasks. CBGB distinguishes itself by focusing on practical, real-world problems requiring numerical answers, curated by domain experts, and categorized by difficulty, thereby providing a structured framework for assessing geospatial code generation, particularly for platforms like Google Earth Engine.

Experiments with ten diverse LLMs, evaluated both in a base configuration and with an error-correction loop, revealed distinct performance tiers among models, in favor of more sophisticated "reasoning" models relative to 'non-thinking" counterparts. Critically, the error-correction mechanism proved quite effective, improving performance across all models and often elevating non-thinking models to the level of more powerful ones operating without correction. With accuracy at 71%, the benchmark is not saturated, particularly for "Difficult" problems, indicating substantial headroom for future LLM advancements in this complex domain. The process of developing CBGB also illuminated the critical importance of precise problem specification in avoiding ambiguity and ensuring reproducible results, offering valuable lessons for both benchmark design and practical LLM application in geospatial science.

The findings underscore the growing potential of LLMs as assistants in geospatial analysis, while also highlighting current limitations in areas like complex multi-step reasoning, nuanced understanding of geospatial data intricacies, and consistent generation of error-free code for specialized APIs. CBGB serves as a valuable tool for tracking progress and identifying areas for improvement in LLMs tailored for scientific, code-intensive domains. Future work could expand CBGB with a greater diversity of tasks, including different sensor types, analytical techniques, and even other cloud-based geospatial platforms. Investigating more sophisticated agentic frameworks beyond simple error correction, such as those incorporating RAG with geospatial knowledge bases or iterative planning, could further enhance LLM performance. Additionally, the insights gained from the "Challenge Checker" and "Uncertainty Divergence" processes could inform the development of interactive tools that guide users in formulating clearer, less ambiguous geospatial queries for LLMs.

As LLMs continue their rapid evolution, rigorous and domain-specific benchmarks like CBGB will be essential in guiding their development towards becoming helpful tools for geospatial analysts and remote sensing scientists, thereby accelerating scientific discovery.

References

S. C. Ahearn, I. Icke, R. Datta, M. N. DeMers, B. Plewe, and A. Skupin. Re-engineering the GIS&T body of knowledge. *International Journal of Geographical Information Science*, 27(11):2227–2245, 2013. doi: 10.1080/13658816.2013.802324.

Temitope Akinboyewa, Zhenlong Li, Huan Ning, and M Naser Lessani. Gis copilot: towards an autonomous gis agent for spatial analysis. *International Journal of Digital Earth*, 18(1):2497489, 2025.

Daman Arora, Himanshu Gaurav Singh, et al. Have llms advanced enough? a challenging problem solving benchmark for large language models. *arXiv preprint arXiv:2305.15074*, 2023.

J.A. Cardille, M.A. Crowley, D. Saah, and N.E. Clinton, editors. *Cloud-Based Remote Sensing with Google Earth Engine: Fundamentals and Applications*. Springer Nature, 2023.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pages 3676–3713. PMLR, 2023.

Yansong Chen, Ziyuan Wang, Guishan Li, Yixuan Liu, Mengke Zhu, Rong Wang, Qianli Chen, Jing Cao, and Chen Liu. Gpt-enhanced framework for gee knowledge extraction and reuse. *International Journal of Digital Earth*, 17(1):2398063, 2024a. doi: 10.1080/17538947.2024.2398063. URL https://www.tandfonline.com/doi/full/10.1080/17538947.2024.2398063.

Yuxing Chen, Weijie Wang, Sylvain Lobry, and Camille Kurtz. An llm agent for automatic geospatial data analysis. *arXiv preprint arXiv:2410.18792*, 2024b.

Hao Cui, Zahra Shamsi, Gowoon Cheon, Xuejian Ma, Shutong Li, Maria Tikhanovskaya, Peter Norgaard, Nayantara Mudur, Martyna Plomecka, Paul Raccuglia, et al. Curie: Evaluating llms on multitask scientific long context understanding and reasoning. *ICLR*, 2025.

Piotr Gramacki, Bruno Martins, and Piotr Szymański. Evaluation of code llms on geospatial code generation. In *Proceedings of the 7th ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery*, pages 54–62, 2024.

Shuyang Hou, Jianyuan Liang, Anqi Zhao, and Huayi Wu. Gee-ops: An operator knowledge base for geospatial code generation on the google earth engine platform powered by large language models. *arXiv preprint arXiv:2412.05587*, 2024.

Shuyang Hou, Zhangxiao Shen, Anqi Zhao, Jianyuan Liang, Zhipeng Gui, Xuefeng Guan, Rui Li, and Huayi Wu. Geocode-gpt: A large language model for geospatial code generation. *International Journal of Applied Earth Observation and Geoinformation*, page 104456, 2025.

Krzysztof Janowicz, Song Gao, Grant McKenzie, Yingjie Hu, and Budhendra Bhaduri. Geoai: spatially explicit artificial intelligence techniques for geographic knowledge discovery and beyond, 2020.

Alexandre Lacoste, Nils Lehmann, Pau Rodriguez, Evan Sherwin, Hannah Kerner, Björn Lütjens, Jeremy Irvin, David Dao, Hamed Alemohammad, Alexandre Drouin, et al. Geo-bench: Toward foundation models for earth monitoring. *Advances in Neural Information Processing Systems*, 36: 51080–51093, 2023.

Chaehong Lee, Varatheepan Paramanayakam, Andreas Karatzas, Yanan Jian, Michael Fore, Heming Liao, Fuxun Yu, Ruopu Li, Iraklis Anagnostopoulos, and Dimitrios Stamoulis. Multi-agent geospatial copilots for remote sensing workflows. *arXiv preprint arXiv:2501.16254*, 2025.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Simranjit Singh, Michael Fore, and Dimitrios Stamoulis. Geollm-engine: A realistic environment for building geospatial copilots. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 585–594, 2024.

Dimitrios Stamoulis and Diana Marculescu. Geo-olm: Enabling sustainable earth observation studies with cost-efficient open language models & state-driven workflows. *arXiv preprint arXiv:2504.04319*, 2025.

Jiayang Wu, Wensheng Gan, Han-Chieh Chao, and Philip S Yu. Geospatial big data: Survey and challenges. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*, 2025.

Yifan Zhang, Cheng Wei, Zhengting He, and Wenhao Yu. Geogpt: An assistant for understanding and processing geospatial tasks. *International Journal of Applied Earth Observation and Geoinformation*, 131:103976, 2024a.

Yu Zhang, Xiusi Chen, Bowen Jin, Sheng Wang, Shuiwang Ji, Wei Wang, and Jiawei Han. A comprehensive survey of scientific large language models and their applications in scientific discovery. *arXiv preprint arXiv:2406.10833*, 2024b.

Appendix A. Appendix: CBGB Problem Statements

A link to the full text of problem statements is contained in the zip file for this submission.

Appendix B. Appendix: Divergence/Error Classification and Summary

Distinguishing Error Types: A key part of your analysis is to differentiate between divergences caused by genuine uncertainty stemming from the problem description versus mistakes made during execution.
Define these as:

- **Uncertainty-Driven Divergence**: A divergence occurring because the problem statement lacked sufficient specificity, forcing variants to make different assumptions or choices.
- **Execution Mistake**: A divergence occurring because an agent or person made an error while attempting to execute a step correctly.

Output: Summary Table: Present your findings in a table with the following columns:

Divergence Point / Error	**Leading To Result(s) / Error**	**Divergence Type**	**Reasoning**
Example Entry	Example Result	Example Type	Example Text
...	...	... | ...	

Table 1: Example structure for Divergence/Error Summary Table.

Table Content Details:

- **Divergence Type Column**: Entries must be one of: "**Uncertainty Divergence**", "**Potential Divergence**", or "**Execution Mistake**".
- **Potential Divergence**: This type applies when you identify ambiguity in the problem statement that would likely cause divergence, even if the current set of variants failed to produce answers (e.g., all encountered errors before reaching the ambiguous point). Analyze and document this potential future divergence based on the problem's characteristics. Note it in the table as "**Potential Divergence**".
- **Sorting Order**: Present the rows in the table sorted according to the "Divergence Type" column in the following specific order:
  - Items classified as "Uncertainty Divergence".
  - Items classified as "Potential Divergence".
  - Items classified as "Execution Mistake".

Appendix C. Appendix: Example Easy and Difficult Challenges

C.1. Example Challenge (Easy)

Easy: Calculating Iron Oxide Ratio (IOR) for Hydrothermal Rock Detection

Objective : You are tasked with calculating the Iron Oxide Ratio (IOR), which is the ratio of the red band reflectance to the blue band reflectance. This ratio can help detect hydrothermally altered rocks that contain oxidized iron-bearing sulfides. Complete the following steps:

- Focus on this point in Seattle, WA, USA: (-122.2040, 47.6221).
- Access the COPERNICUS/S2_HARMONIZED ImageCollection and select images that:
  - Cover the Seattle point,
  - Are from 2020-08-15 to 2020-10-01, and
  - Have less than 10% cloud coverage.
- Select the earliest image from that set.
- Identify the red band and blue band that surround the following wavelengths: Red band, 665 nm; Blue band, 490 nm.
- Compute the IOR. Extract the calculated IOR value at the given Seattle point. Print the IOR value to the console.

Notes: Ensure band values (e.g., radiance, temperature) are scaled to their proper units prior to use. Reflectance values should be scaled to between 0 and 1 prior to use. Retrieve values at the native scale of the imagery. Write the answer to 3 decimal points of precision (e.g, 12345.678)

C.2. Example Challenge (Difficult)

Difficult: Deforestation Rate Comparison in Colombian Amazon Protected Areas

```
Objective :  This problem will compare the total deforestation assessed to have
occurred within and around two protected areas in the Colombian Amazon:  La Paya and
Tinigua.


• Use the "WCMC/WDPA/current/polygons" data set to identify the boundaries of the La
  Paya and Tinigua protected areas.  Add a 1000m buffer around each protected area's
  geometry.

• Calculate the forest loss within each protected area using the lossyear band of the
  GFC dataset, where each pixel indicates the year of deforestation.  Use the Global
  Forest Change dataset (UMD/hansen/global_forest_change_2023_v1_11).  Consider areas
  with tree cover greater than 30% in the year 2000.

• Determine the absolute value of the difference in total deforestation amounts (in
  hectares) between the area within La Paya and the area within Tinigua between 2001
  and 2023.  Provide the answer in hectares.

Notes:  Unless directed otherwise, retrieve or summarize value(s) at the native
resolution of the image band(s).  If multiple bands or sensors are used with
different resolutions, retrieve or summarize values using the finest resolution
among the inputs unless directed otherwise.  Unless directed otherwise, write the
answer to 3 decimal points of precision (e.g, 12345.678).
```

Appendix D. Appendix: Prompts

D.1. Main code generation prompt

```
You are an expert Earth Engine developer.  Write Earth Engine Python code to answer
the following question.  No extra explanation is needed.  ONLY return the Earth
Engine Python code in a ‘‘‘Python section.

Call getInfo() when you need to get the value of the final computation.  The answer
must be printed to stdout.  In cases where the answer is a number, print ONLY the
number corresponding to the answer, no other explanatory text.

For example, don't do this:  ‘print('here is my answer:  ' + x)‘

Just do:  ‘print(x)‘
```

D.2. Error correction prompt

```
Here is the question:  {question}
You previously wrote this code which had an error:  {bad_code}
This was the error:  {error_msg}
Please try again and rewrite the code.
```

Appendix E. Appendix: Agent Versions and Challenge Results

Model	Overall		Difficult		Intermediate		Easy	
	base	ec	base	ec	base	ec	base	ec
o3-2025-04-16	29	32	4	5	14	15	11	12
gemini-2.5-pro-preview-05-06	25	32	1	5	12	15	12	12
claude-3-7-sonnet-20250219	23	31	1	4	9	14	13	13
claude-3-5-sonnet-20241022	26	31	2	3	13	14	11	14
deepseek-reasoner	22	29	1	1	9	15	12	13
gemini-2.5-flash-preview-04-17	19	27	0	0	8	14	11	13
o4-mini-2025-04-16	22	27	2	2	9	11	11	14
claude-3-5-haiku-20241022	15	22	0	1	5	8	10	13
deepseek-chat	20	21	1	0	8	8	11	13
gemini-2.0-flash	8	14	0	1	3	5	5	8
TOTAL QUESTIONS	**45**	**45**	**10**	**10**	**20**	**20**	**15**	**15**

Table 2: Total correct answers of base models and error-correction (ec) agents, from a single run of the evaluation pipeline.