Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

GeoMind: An Autonomous Multi-Agent Framework for Verified Geospatial Analysis

Jakaria Habib

jakariahabibjim2@gmail.com

Pabna University of Science and Technology

Monoarul Islam Srabon

Daffodil International University

Mst Lutfun Nahar

Daffodil International University

Mariam Lutfy

Pabna University of Science and Technology

Most. Amina Khatun

Rajshahi University of Engineering and Technology

Md. Basim Al Zabir Shammo

Pabna University of Science and Technology

Md. Atik Hasan

Pabna University of Science and Technology

Research Article

Keywords: Earth Observation, Large Language Models, Geospatial AI, Multi-Agent Systems, Retrieval-Augmented Generation, Autonomous GIS

Posted Date: June 24th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10034367/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License

Additional Declarations: No competing interests reported.

Abstract

The rapid growth of the Earth observation satellite constellation has started to shift the paradigm to autonomous geospatial computing. But until now, using general-purpose Large Language Models (LLMs) as independent GIS coding agents has been hampered by the “semantic gap”: syntactic hallucination, meaning that models generate code that runs successfully but yields topologically incorrect geometry, and geographic hallucination, where models output topologically correct code but provide an incorrect geometry. To overcome this, we propose a new closed-loop autonomous multiagent system for verified geospatial analysis, called GeoMind. GeoMind's core is a decentralized agentic hierarchy based on a local Qwen2.5-Coder-7B-Instruct model. A Planner Agent breaks down complex natural language queries into structured Directed Acyclic Graphs (DAGs) of atomic tasks, while a Context Injector based on ChromaDB with a Retrieval-Augmented Generation (RAG) mechanism syntactically grounds these tasks into semantic context using natural language. The Coder Agent creates Python code that is executed in an isolated sandbox previously validated by a deterministic Geospatial Validator which guarantees the topological integrity constraints as per the Open Geospatial Consortium (OGC). The autonomous self-correction loop with dynamic temperature scaling is triggered after failed validations. The framework is measured on a GeoTaskBench benchmark of 150 tasks. The empirical results show that the mean Intersection over Union (IoU) for the baseline open-loop LLM is 0.140 and the Task Success Rate (TSR) is 30.0%, while the combination of RAG brings the TSR to 73.3%. The complete closed-loop GeoMind framework achieves state-of-the-art TSR of 90.0% and moves the mean spatial IoU to 0.793 (absolute gain of + 0.653). The results of the paired t-test indicate that this improvement is highly statistically significant ($t = 22.74$, $p < 1.0 \times 10^{-10}$). The above results illustrate the clear benefit of closed-loop symbolic verification over exhaustive parametric expansion when applied to autonomous Earth science analytics.

1. Introduction

1.1. Background: The Data Deluge in Earth Observation

The exponential increase in Earth Observation (EO) satellite constellations of the past two decades has initiated a paradigm change for environmental monitoring from a traditional data-scarce science to a data-rich high-throughput discipline referred to as Big Earth Data (M. Sudmanns et al., 2020). In the past, this task was simply impossible: getting high resolution multispectral imagery from space was prohibitively expensive and highly classified; either a luxury of well-funded military applications or those national space agencies with hundreds of billions of dollars at their disposal. The relatively recent democratization of access to space and the launch of open-access satellite missions, most famously by ESA with its Copernicus Sentinel program and USGS large-scale Landsat continuity missions had removed these historical barriers to acquisition (D. Tiede et al., 2016). Current satellite constellations produce telemetry at a scale and speed that has never been seen before, pouring petabytes of freely available optical, thermal, and Synthetic Aperture Radar (SAR) imagery into public cloud repositories each year. Such a phenomenal volume of high-dimensional spatial data potentially enables mankind to observe climate change on the earth, furthermore, track the deforestation in Amazon Basin and map large-scale inundation of flooding areas comprehensively (M. Reichstein et al., 2019). Provided the Sentinel-2 constellation alone covers all of Earth every five days at 10 m spatial resolution, which probably yields thousands of multidimensional arrays within a temporal stack. As a result, the dominant scientific challenge in Earth Science has suddenly changed; after decades, obtaining geospatial data is no longer an issue, instead it is the

computational and algorithmic capacity required to process, analyze and summarize this now petabyte-scale deluge of pixels into actionable intelligence.

The massive storage and processing demand of Big Earth Data have required the geospatial industry to transition away from local, desktop Geographic Information System (GIS) architectures to planetary-scale cloud computing platforms. Although the actual remote sensing datasets are massive in volume, velocity and variety, the amounts of images a standard consumer hardware can process has been dwarfed by this tsunami of earth observation data, requiring novel distributed computational infrastructures such as Google Earth Engine, Amazon Web Services (AWS) Open Data Registry and Copernicus Data Space Ecosystem (N. Gorelick et al., 2017). However, despite these technical achievements of cloud infrastructure and serverless processing pipelines, the translation of raw pixel Arrays into scientific knowledge remains a highly manual, expert-intensive task. The real danger of running a thorough geospatial analytical workflow is that it requires highly specialized, cross-disciplinary domain expertise in the hands of a human operator (X. X. Zhu et al., 2017). They require a deep understanding of intricate physical geography and atmospheric physics, but also require exceptional advanced Python programming expertise including the use of low-level C-libraries like GEOS, PROJ, and GDAL to manipulate topological vectors and multispectral matrices (M. F. Goodchild, 2018). The human operator needs to write hundreds of lines of code in order to authenticate against Spatio-Temporal Asset Catalogs (STAC), define elaborate bounding geometries, align different Coordinate Reference Systems (CRS) and apply the cloud-masking algorithm. This results in a dramatic and unsustainable processing bottleneck spanning the industry: exabytes of high-fidelity Earth Observation data sitting in cloud repositories while we lack the human capital and software engineering needed to process this information efficiently, at planetary scales.

At the same time, Deep Learning still made computer science all the more radically different with Huge Large Language Models (LLMs) based on Transformer architecture. The models are originally built for Natural Language Processing (NLP) tasks like translation or text summarization, yet outright perform outstanding zero-shot generalized software engineering and algorithmic code generation (M. A. Wulder et al., 2008). Models like GPT-4, LLaMA-3, and Qwen2.5 have been trained on enormous multi-lingual corpora. consisting of trillions of tokens including many, many terabytes worth of programming type code scraped from public GitHub repositories. This massive pre-training allows those neural networks to generate complex, executable algorithmic syntax purely from unstructured natural language prompts (T. B. Brown et al., 2020). The incorporation of generative models into IDEs (e.g., GitHub Copilot, OpenAI's Codex) has shown demonstrable and statistically significant increases in standard software developer productivity (M. Chen et al., 2021). These LLMs can seamlessly scaffold boilerplate applications, write unit tests and even debug basic syntax errors in common programming languages by predicting the statistically most probable sequence of code tokens. Considering these developments in AI, there is significant theoretical and practical potential to be able to deploy LLMs in an autonomous fashion as coding agents to finally solve the human-capital bottleneck that has long paralyzed the geospatial data industry (A. Ziegler et al., 2022).

GeoAI, the meeting point between geographic analysis and artificial intelligence mostly consisted of using deep learning to train Convolutional Neural Networks (CNNs) for image segmentation and feature extraction. Nonetheless, using Foundation Models to independently automate writing geospatial software scripts is a very new and extremely difficult challenge (G. Mai et al., 2023). The theoretical premise is compelling: a climate scientist types 'calculate drought index over California last 10 years' and an autonomous LLM agent will write all the data-grab code/raster maths/output map/formatting etc without needing humans to even write code. However, existing systematic empirical evaluations also show that generalized LLMs with an open-loop setup

perform very poorly on specialized spatial computing tasks and frequently fail during execution (OpenAI et al., 2024). We find that when researchers attempt to use ChatGPT (or generalized coding agents) to write complex GeoPandas or Rasterio pipelines, the models make so many coding errors it becomes unacceptable and often the code cannot compile; worse still, some generated code successfully compiles, but produces mathematically invalid geographic data. This systematic failing exposes a core deficiency of current AI architectures: predicting the next word in a Python program is not computationally analogous to calculating the physical and mathematical laws that govern the surface of our planet.

This low task success rate can be largely attributed to what researchers refer to as the 'semantic gap': the very real divide between linguistic likelihood and geometric actuality. LLMs are fundamentally probabilistic token predictors, (they have native text-genre understanding of textual linguistics and standard software logic but they possess exactly none of the options conceptually spatial knowledge making Euclidean geometry, non-Euclidean map projections or physical planetary bounds) (Z. Ji et al., 2023). At zero-shot, open-looping configurations, LLMs can often experience hallucination (persistence to make up API parameters that do not exist for complex geospatial libraries)(P. Lewis et al., 2021). More insidiously, they hallucinate their topologies awayproducing self-intersecting WGS84 bounding polygons or otherwise assigning dividing (latitudinal) coordinates which live outside the physical 90-degree limit of our poles. Without the ability to run the code that they write, standard LLMs can only operate sight unseen; unaware that their generated syntax has produced a CRS mismatch or that their spatial join output an empty file. Therefore, a new architectural paradigm is needed to successfully leverage artificial intelligence for the Earth Sciences which mathematically bounds the probabilistic reasoning of an LLM with deterministic spatial guardrails, Retrieval-Augmented Generation (RAG), and closed-loop self-correction algorithms(S. Hong et al., 2024). This paper proposes precisely such a solution and introduces an entirely new framework for immediate closing of the semantic gap, GeoMind.

1.2. The Semantic Gap in Artificial Intelligence

However, while the advent of Large Language Models (LLMs) in standard software development has triggered a substantial jump in coding productivity, their application to Earth Science and geospatial analytics reveals a deep-rooted architectural challenge when it comes to bridging the so-called 'semantic gap'(W. Liu et al., 2025). To make sense of this limitation, we must decompose the basic mechanics of the modern neural network architectures (e.g. the LLaMA series or Qwen), which are mainly designed for linguistic pattern recognition rather than deterministic mathematics (H. Touvron et al., 2023). These models are, in effect, sophisticated auto-regressive token predictors with output calculated as the statistical distribution of sub-word tokens comprised of an immense unstructured training dataset (B. Rozière et al., 2024). Thus, while it is fairly trivial for an LLM to generate the boilerplate syntax of a Python function, as discussed in Section 1.1, what is really missing is that the LLM lacks the embodied cognition which grounds that syntax into real-world, physical phenomena (J. Feng et al., 2025). This idea is commonly discussed in the field of computational linguistics, suggesting that language models have an amazing understanding of how language 'looks' but little comprehension of the 'reality' or meaning the language represents (E. M. Bender and A. Koller, 2020). In other, geospatial intelligence-related words, an AI can write a script that knows how to plot a coordinate against a map but will sacrifice any contextual understanding of what the map is or how the curve of the Earth bends that map and why two polygons overlapped over one another might be topologically invalid (Y. Bisk et al., 2020). As a result, these models function in an open-loop void when presented with complex Earth Observation workflows and offer solutions based on probabilistic guessing alone rather than geometric reasoning.

Standard LLM benchmarking approaches demonstrate that this semantic gap causes severe failures in spatial reasoning and topological integrity (F. Shiri et al., 2024). Most recent empirical evaluations with established spatial reasoning benchmarks (ex: StepGame) reveal that even state-of-the-art models experience an almost complete collapse in performance when asked to compute nontrivial spatial relations between several objects (F. Li et al., 2024). A generalized AI, when instructed by an end-user with the prompt 'Give me the intersection of vegetation index between two overlapping satellite footprints over the Northern Hemisphere' has to do linguistic parsing (that is mapping user query into API calls), and generate both API syntax in addition to high-dimensional cognitive geometric mental rotation. But since the LLM is a 1D textual sequence, it cannot literally 'see' the two-dimensional intersection matrix that manifests this relation, hence yielding topological errors with increasing task complexity(A. Yang et al., 2025). So far, researchers exploring these limitations have observed that LLMs often get 'stuck in the matrix' when doing basic quadrants identification tasks and producing spatial boundaries that even break basic Euclidean geometry laws (M. Bai et al., 2025). This is not merely a syntactic error, this is the deep cognitive hallucination of language where AI generates mathematical impossibilities: latitudes that are more than 90 degrees or geometries loop back on themselves to create unclosed rings. Therefore, in mission-critical Earth Science applications such as disaster response or climate modeling, these geometric hallucinations should be completely eliminated via deterministic validation.

Using standard prompt engineering techniqueslike zero-shot Chain-of-Thought reasoningto address the semantic gap has not been successful for geospatial workflows (Y. Gao et al., 2024). Chain-of-Thought prompting works well on logical or arithmetic word problems as it encourages the model to output intermediate reasoning steps, but does not synthetically create geometric knowledge that is uncongenial with respect to the pre-training data of the model (T. Kojima et al., 2023). Moreover, a standard implementation of Retrieval-Augmented Generation (RAG) alone is not enough (Y. Gao et al., 2024). Traditional RAG architectures retrieve textual documents from vector databases optimized by high-performance similarity search algorithms and enter them to the LLM context window (J. Johnson et al., 2021). This achieves the desired effect of grounding the LLM in correct API syntax (e.g., training it on functions dGo of a geopandas function and which arguments to fill), but does not equip the model with measure-theoretic tools needed to verify if the spatial output of that function is mathematically valid (M. M. Bronstein et al., 2017). For example, an RAG enabled model might write perfect syntax to crop a raster image given by the bounding box but if that bounding box itself was hallucinated due to the semantic gap then there is no way this output will be scientifically useful. This means that a closed-loop architectural paradigm is an absolute necessity to get over the built-in spatial constraints of neural networks.

1.3. The Research Problem: Syntactic and Geographic Hallucination

In order to properly explain the problem addressed by GeoMind, we need a closer look at emerging phenomenon 'code hallucination' and what happens when high precision scientific programming breaks down. Transformer-based architectures have evolved rapidly and produced very complex models fine-tuned on large corpora of source code, changing the way in which we think about automated program synthesis (A. Vaswani et al., 2023). For instance, the pioneering systems like AlphaCode have shown the ability to solve competition programming problems by generating millions of candidate solutions and filtering them based on execution based feedback (Y. Li et al., 2022). Likewise, open-source models such as StarCoder and CodeGen have shifted the limits of multi-turn program synthesis to allow developers to scaffold applications in a number of programming languages through many iterations (E. Nijkamp et al., 2023). Yet gross-performing surveys of these Large Language Models for natural-language-to-code (NL2Code) translation expose a fundamental backdoor: while these models excel in

translating highly-specified, well-documented algorithms (ex: sorting arrays or simple web scraping), they exhibit adverse performance on niche, mathematically-rich APIs (D. Zan et al., 2023). Recent empirical investigations of code hallucination mathematically classify these failures into a number of types, and observe that LLMs often invent functions that do not exist or hallucinate class attributes outside their primary training distribution (F. Liu et al., 2026). For Earth Observation, where the ecosystem is dominated by libraries like GDAL, Rasterio and GeoPandas, these syntactic hallucinations are more than just outlier cases; they pose the main challenge to fully autonomous operation.

Earth Science APIs are especially fragile, making small syntactic errors in the compositional pipeline leads to complete downstream pipeline failures this exacerbates the hallucination problem with direct effects on Earth Science contexts. To test the functional correctness of LLM-generated code, researchers rely on standard benchmarks that evaluate individual self-contained algorithmic functions such as the Mostly Basic Python Problems (MBPP) (J. Austin et al., 2021). But the ability to process multi-terabyte satellite imagery is not an isolated functionality. Common geospatial workflows consist of connecting to an authenticated STAC, submitting a query with an intersecting bounding box, parsing the response JSON payload for the specific bands needed (e.g., Near-Infrared and Red for NDVI), converting the multi-dimensional data matrices into 32-bit floating points arrays, then writing the mathematically transformed output as a georeferenced Cloud Optimized GeoTIFF (COG). Previous work on real-world LLM hallucination in code generation demonstrates that as the interrelationships between context and project level URL dependencies grow, this leads to a greater risk of the model generating factually inconsistent or logically non-disjoint relevant code specifically exponential growth (Z. Zhang et al., 2025). For example, given the requirement to reproduce a PySTAC query to find Sentinel-2 data, an LLM may achieve this but then hallucinate an obsolete affine transformation matrix in order to reproject that data using Rasterio into the local UTM zone, producing severe projection errors.Since the agent has no internal knowledge of cartographic projections (the agent lacks inherent cartographic knowledge.), it confidently emits this hallucinated syntax that results in an immediate runtime compilation error: Segmentation fault!

In addition to simple syntax errors, the use of open-loop LLMs in geography introduces a more insidious failure mode called 'geographic hallucination'. The programming community has been investing substantial resources to evaluate performance on coding challenge by large datasets such as APPS, but achieves this only with respect to the evaluation of generic logic puzzles and mathematically abstract computer science problems (D. Hendrycks et al., 2021). They don't test AI on its ability to keep the polygon structure of a physical polygon, resting on top of a globe-sized 3D planetary sphere. Under the condition of an open-loop LLM, if we ask LLM to generate a geospatial script, although the code is syntactically perfect and can run without raising a Python exception, it has passed none of the standard software engineering validation metrics. Nonetheless, it is possible the spatial output will be wholly hallucinatedfor instance, in a spatial join operation (when latitudinal vs longitudinal coordinate arrays are inverted by the LLM). The inversion does yield a mathematically valid array, but geographically it results in mapping an analysis for the Amazon Rainforest directly into the middle of the Atlantic Ocean. This would be similar to the general issue of fact hallucination with generative text models, where the output is linguistically fluent but has little or no factual accuracy (Z. Ji et al., 2023). In automated remote sensing, where decisions on disaster response or agricultural yield are based off of these outputs that are then trained all the way until a geographic belief hallucination which is exacerbated by this so it is extremely dangerous.

At their core, these failures stem from the static, open-loop nature of existing LLM interaction paradigms. Prompt engineering, most notably the Chain-of-Thought (CoT) method, is among the first attempts at tuning models to avoid payoffs that leverage logical reasoning failures: This approach has a model output intermediate steps

before reaching a final answer (J. Wei et al., 2023). Although CoT prompting is effective for improving arithmetic and logical deduction, follow up work released Program-of-Thoughts (PoT) that focused on greater accuracy in complex mathematics in which they argue decoupling the calculation from the language-modeling task of a large-language model by assigning it to an external Python interpreter significantly improves performance (W. Chen et al., 2023). In addition, methods like self-consistency sample many reasoning paths and choose the most common answer as one way to mathematically marginalize outlier hallucinations (X. Wang et al., 2023). Though these prompt-based reasoning tricks are advancing the field, they are still fundamentally inadequate in the case of geospatial programming because they are 100% static; if there is a flaw with how geographic logic lives internal to the model than generating many wrong reasoning paths does not fix this issue. In this case, the AI fails to realize that it had generated a topologically invalid (self-intersecting) polygon merely by 'thinking harder' about its sequence of code tokens. One needs to interact dynamically and physically through the executing environment in order to sense the mistake.

To address this disconnect, the research community has started to investigate architectures that combine reasoning and acting such as ReAct which facilitates language model interaction with external tools and observation APIs (S. Yao et al., 2023). Early experiments with large models like GPT-4 have illustrated that, when language models are given feedback from their environment, they can also iteratively improve their own code, implying to us 'sparks' of generalized intelligence (S. Bubeck et al., 2023). Yet these more generalized frameworks were not focused and tested in the rigid constraints of an Earth Observation data infrastructures. The general problem, then, is the lack of a non-specialized, closed-loop implementation that connects the expressed hopes of LLM coding to the deterministic results from geospatial analytics. We do not have anything that writes the high-dimensional Earth Science Python syntax as well mathematically checks the result to ensure topologically accurate representations of the geometry. This can only be done by a move from transient, instruction-based architectures resistant to the core human RL techniques first showcased in InstructGPT towards fully autonomous, self-directed agents with attentive deterministic spatial guardrails (L. Ouyang et al., 2022). It is therefore necessary to integrate Retrieval-Augmented Generation (RAG) pipelines, execution-based feedback cycles back into the pipeline and the iron-clad geographic validation all in one architecture.

1.4. Proposed Framework and Novel Contributions

In this context, to tackle the complex research issue raised in Section 1.3 about both syntactic hallucination and geographic hallucination of open-loop LLMs with Earth Observation data, this work contributes the GeoMind framework. GeoMind is a closed-loop, multi-agent architecture that synergistically combines retrieval-augmented code generation with autonomous execution sandboxing and deterministic geospatial validation to improve accuracy and reliability in autonomous Earth Science analytics. In contrast to current generalized autonomous agent frameworks, such as AutoGPT that run pure domain-agnostic task-scheduling agents with no built-in knowledge of spatial topology (L. Wang et al., 2025), GeoMind is designed from the ground up to impose the hard, deterministic mathematical rules underlying Geographic Information Systems (GIS). The framework fundamentally refutes the currently dominant paradigm of using monolithic, closed-source cloud-hosted LLMs (e.g.: GPT-4, Claude) for geospatial query tasks where data sovereignty and privacy become crucial issues when dealing with sensitive satellite telemetry (R. Manvi et al., 2024). GeoMind, however, is fully open-source and runs a quantized LLM model on consumer-grade edge hardware to run locally deployed and keep all Earth Observation data in-house. The four main architectural novelties of the proposed framework are described in the following subsections, including a formal listing of the novel scientific contributions of this research.

1.4.1. Multi-Agent Orchestration with Quantized Local Inference

At the high level, the foundational architectural decision behind GeoMind framework is to replace monolithic, single-prompt code generation in favor of multi-agent orchestration paradigm. Recent work in agentic AI has shown that organizing decomposed tasks over cooperating, expert agents provide a level of performance orders of magnitude better than either current monolithic approach where a model attempts to do everything with one pass (Y. Qin et al., 2024). A controller LLM plan and decomposition framework such as HuggingGPT shows strong evidence that distributing the cognitive load by delegating subtasks to specialized execution modules lowers the chance of complex hallucinations at any given node within a system (Y. Shen et al., 2023). Drawing upon this multi-agent philosophy, GeoMind dissects the geospatial problem-solving pipeline into two specialized, cognitive roles: (i) a Planner Agent capable of semantic parsing of user natural language queries and constructing structured DAGs for each computational step; and (ii) a Coder Agent responsible for the deterministic transformation of each DAG node to host executable, geospatially valid Python syntax. This separation of concerns allows the language understanding component to be architecturally disentangled from the code generation component, minimizing opportunities for errors during planning and execution to influence each other.

One of the most important design constraints that dictates the GeoMind framework is its premise to work on completely local, edge-based hardware without any external API access to LLMs in a cloud setup. This requires utilizing efficient (in terms of computational resources) model compression techniques in order to move a powerful LLM into the already extremely low memory space of off-the-shelf GPU accelerators. Significant progress has been made in the direction of post-training quantization recently, such as QLoRA by Dettmers et al. (T. Dettmers et al., 2023), who demonstrated that 4-bit NormalFloat (NF4) quantisation can reduce the memory usage of a large model by ≈ 75% while still retaining almost all its reasoning capabilities. GeoMind enables this breakthrough by serving the Qwen2.5-Coder-7B-Instruct model (4-bit-NF4 quantized using the bitsandbytes library, originally ~ 14 GB VRAM required for standard 16-bit floating point inference, reduced to a 4.5 GB VRAM footprint). Through this compression, we are able to fit the multi-agent framework (along with both the vector embedding database and sandbox) in just 15 GB VRAM of dual Nvidia Tesla T4 GPUs, which is a common non-specialized standard rig on platforms like Kaggle and Google Colab. GeoMind, which shows that state-of-the-art geospatial reasoning can be achieved using a fully closed loop feedback mechanism with a quantized 7-billion parameter model and challenges the currently held belief in the need for massive, proprietary cloud models of 400 billion parameters or more to enable autonomous spatial intelligence.

1.4.2. Retrieval-Augmented Generation for API Grounding

So as to eliminate the root cause of syntactic hallucination, GeoMind comprises an in-situ Retrieval-Augmented Generation (RAG) pipeline embedding code generation from a modified LLM with localized mathematically-grounded grounding through verified authoritative API documentation. Augmenting language models with an external, non-parametric memory was first formalized in the seminal work of Karpukhin et al. in the context of dense passage retrieval (DPR) framework, which showed that vector-based representations of text passages are multiple times better than sparse keyword-based retrievers in the knowledge-intensive question answering setup (V. Karpukhin et al., 2020). This principle has been expanded to the code generation space, where it was empirically shown that injecting relevant API documentation into the LLM context window significantly reduces hallucinated function signatures and deprecated parameter usage (S. Zhou et al., 2023). The GeoMind framework applies this principle to create a local ChromaDB vector store filled with selecting well-organized technical documentation for the core Earth Science Python libraries: GeoPandas, Rasterio, Shapely, PySTAC-Client, and

Planetary Computer. Using this approach, we convert each documentation entry into a high-dimensional vector embedding using a Sentence-BERT (SBERT) architecture, and then retrieval happens by cosine similarity search on the user query embedded text. The RAG module achieves this by injecting the top-K related documentation fragments directly to the system prompt prior to code generation so that: (1) Coder Agent will generate syntax strictly adhering to each library's actual verified API surface, and (2) any undocumented or deprecated API call is flagged before execution.

First of all, the localized RAG architecture has three times the strategic advantages over a cloud-based knowledge retrieval. On the one hand, it gives deterministic control over the quality and freshness of the injected documentationthe framework operator can curate and version the knowledge database to correspond with deployed library versions in an execution environment. This addresses a common failure mode for cloud-based LLMs that are trained on stale internet data, which can lead to outdated API patterns being recommended from previous library versions. Second, the local vector store is designed to achieve sub-ms retrieval latency, introducing little to no inference overhead. Third, and most importantly for Earth Science applications, the localized architecture guarantees that no portion of any user querywhich could contain sensitive geographic coordinates for critical infrastructure or military installationsis ever delivered to an external server. This is problematic because the privacy implications of LLM-based tool use has been found to be minimal when using Toolformer-style architectures and performing external API calls, but introduce significant risks of data leakage (T. Schick et al., 2023). This is a serious risk, but GeoMind obviates this entirely by performing all the retrieval and inference, as well as execution, locally on the machine.

1.4.3. Autonomous Execution Sandbox and Deterministic Geospatial Validation

The closed-loop execution sandbox with deterministic geospatial validation module is the third and most architecturally novel part of the GeoMind framework. Recall that RAG cannot, by itself, preclude geographic hallucinations generated when syntactically valid code returns spatially incorrect output recall that RAG cannot inherently prevent geographic hallucinations, because providing correct API syntax does not guarantee the mathematical correctness of the resulting spatial geometries. To this end, GeoMind accomplishes an autonomous execution sandbox that dynamically executes the AI generated Python code within a properly isolated runtime environment. A code statement executed by the fetched model that raises a Python exception (for example, CRSError, KeyError, or ValueError) causes the framework to capture and return a full Abstract Syntax Tree (AST) traceback as negative reinforcement for feeding back into the Coder Agent such that it may autonomously debug itself and leverages this feedback loop to provide new fixes. The closed-loop, iterative self-correction mechanism is directly inspired by the Reflexion architecture which showed that verbal reinforcement learning can yield large gains in performance by a simple on-the-fly reexamination and mini-repeated verbal reinforcement of the response without expensive gradient updates to model weights (N. Shinn et al., 2023). The verbal reinforcement in the GeoMind architecture consists of raw Python error messages and geospatial validator diagnostics that give the LLM clear, actionable feedback regarding its mathematical failures.

Aside from the runtime exception handling, the framework extends a deterministic Geospatial Validator to keep fundamental mathematical laws of spatial computing an immutable guardrail. This validator is independent from the LLM and applies four strict, non-negotiable topologic integrity checks on every spatial output of the agent: (1) a Data Completeness Check confirming that the generated GeoDataFrame is not empty at all, (2) a Coordinate Reference System (CRS) Integrity Check ensuring absolute correctness control over EPSG codes definition, (3) a

Geometric Topological Integrity Check using OGC Simple Feature Access standard for checking self-intersecting polygons and unclosed linear rings detection and finally, (4) a Global Bounding Box Constraint Check verifying if all WGS84 coordinates are in [[− 180,+180]longitude,[− 90,+90]latitude] area. When any of these deterministic checks fail, the validator forms a structured diagnostic message and reinjects it back into the self-correcting loop. This dual-layer approach, is a combination of two different styles: exceptional handling for runtime errors and mathematical spatial validation level for static (syntactically broken) semantics enables GeoMind to detect and fix the shallow failures (broken syntax) that most autonomous solutions suffer from, as well as those deep-seated horrors related to endowing algorithms with hallucinated geographic outputs (J. Austin et al., 2021).

1.4.4. GeoTaskBench: A Domain-Specific Evaluation Framework

One of the major contributions of this research, aside from its big-picture architectural framework, is the definition and open-release of a new evaluation datasetGeoTaskBench: an evaluation dataset that captures 150 task types for real-world Earth Science analysis.To ensure full reproducibility and transparency, the complete GeoTaskBench dataset, evaluation scripts, and the GeoMind framework source code have been made publicly available at: https://github.com/JakariaHabib1/GeoMind. Abstract algorithmic competence remains the most common type of evaluation in current benchmarks for LLM code generation. Benchmarks like HumanEval (which tests functional correctness of standalone Python functions) and MBPP (which measures some basic programming skill) have become standard for evaluation (M. Chen et al., 2021). Unlike HumanEval, which evaluates isolated, single-line algorithmic functions, GeoTaskBench focuses on multi-hop pipeline executions requiring coordinate math, STAC authentication, and topological alignment. Consequently, successful execution of these 150 complex geospatial tasks demands significantly higher cognitive overhead and functional breadth than thousands of basic Python syntactic tests. Nevertheless, these benchmarks are unfortunately not sufficient to evaluate the autonomous geospatial agent. There are none that involve interacting with satellite data catalogs, multispectral raster math, coordinate reference system transformation, or topological validation. However, this leads to a situation where it can happen that a model scoring 100% on HumanEval cannot compute the NDVI of a Sentinel-2 image correctly. Lack of a domain-specific standard benchmark for the geospatial code generation is one of the key shortfalls in evaluation infrastructure across the GeoAI research community (Y. Zhang et al., 2024).

GeoTaskBench has been specifically developed to address this gap. This dataset covers a complete matrix of geo-spatial operation types (i.e., vector manipulation, raster mathematics, STAC catalog querying, and CRS reprojection), geographic regions (North America, South America, Europe, Africa and Asia) and satellite sensor modalities (Sentinel-1 SAR, Sentinel-2 Optical & Landsat 8 Thermal). GeoTaskBench defines each of its tasks as a natural language query paired with standard ground-truth spatial geometry. To ensure rigorous evaluation, the 150 tasks are uniformly stratified across four core computational categories: vector manipulation (35 tasks), raster mathematics (40 tasks), STAC catalog querying (40 tasks), and CRS reprojection (35 tasks). Ground-truth spatial geometries were established via independent manual digitization by two expert GIS analysts using QGIS, achieving an extraordinarily high Inter-Annotator Agreement (Cohen’s Kappa = 0.92), thereby confirming the dataset’s validity as a gold standard for autonomous evaluation, providing an Intersection over Union (IoU) metric1, which methodologically quantifies geographic precision. Such design reflects the evaluation philosophy conceived in computer vision benchmarks, where IoU has become an established gold standard to quantify spatial overlapping between predicted and ground-truth bounding regions (T.-Y. Lin et al., 2015). Moreover, GeoTaskBench presents the Task Success Rate (TSR) metric which is a binary score that evaluates how many of the tasks have been successfully followed through by the agent to produce an executable, validator-compliant code. GeoTaskBench enables a holistic, realistic two-dimensional evaluation surface of an autonomous

geospatial agent in terms of its reliability (quantified by TSR or Task Success Rate) and accuracy (quantified by IoU or Intersection over Union), since it captures both operational stability and geographic precision. This benchmark provides a standardized, reproducible protocol for evaluating the geospatial reasoning performance of various LLM-based agent architectures.

1.4.5. Summary of Key Contributions

In conclusion, this paper contributes to Geospatial Artificial Intelligence and autonomous Earth Science analytics in three main ways. We first present GeoMind, the novel closed-loop multi-agent architecture that combines powerful heterogenous capabilities including Retrieval-Augmented Generation (RAG) with a continuously resetting temperature-scaled execution sandbox. In contrast to currently available general agent frameworks, which largely rely on proprietary cloud-based models, GeoMind is run entirely locally using a newly retrained and quantized Qwen2.5-Coder-7B-Instruct, as is common in the domain, demonstrating that state-of-the-art geospatial reasoning is possible on consumer-grade edge hardware. Second, we present a novel guardrail module that validates the mathematical topological correctness of all geospatial outputs produced by LLMs, through our Deterministic Geospatial Validator, eliminating geographic hallucinations that are undetectable by traditional software engineering testing techniques. Third, we define and share GeoTaskBenchthe first task-specific evaluation collection of 150 diverse Earth Science tasks, with ground-truth spatial geometries to create consistent metrics for assessing autonomous geospatial agents: Task Success Rate (TSR), Intersection over Union (IoU) (H. Ning et al., 2024).

Experimental evaluation on GeoTaskBench proves the effectiveness of the proposed architecture empirically. Extensive ablation study shows that our complete GeoMind framework achieves 90.0% Task Success Rate, yielding the absolute + 57.3% improvement than the zero-shot baseline LLM. Finally, the framework has a mean spatial Intersection over Union of 0.793 which is a significant improvement from the 0.140 baseline ($p < 1.0e-10$ in paired t-test), providing strong statistical evidence that the proposed closed-loop architecture increases spatial reasoning capabilities of Large Language Models both fundamentally and structurally. Running on a dual NVIDIA Tesla T4 GPU configuration (15 GB VRAM), the quantized inference engine achieves an average end-to-end task execution latency of approximately 14.2 seconds, proving the operational feasibility of deploying this architecture on consumer-grade hardware.

Limitations and Baseline Constraints: A comprehensive cross-system empirical comparison remains challenging due to the strict air-gapped data sovereignty requirements of GeoMind. Prior systems like GeoGPT and LLM-Find mandate the transmission of user queries to proprietary, external API endpoints (e.g., OpenAI's GPT-4). Since GeoMind is explicitly designed for mission-critical and classified Earth Science applications where telemetry data cannot leave the local edge environment, utilizing these cloud-dependent baseline systems for evaluation on GeoTaskBench represents a fundamental architectural incompatibility.

The rest of the paper is structured as follows. In section 2, there is an extensive literature review on LLM-based autonomous agents and geospatial AI. Section 3 describes the full methodology of GeoMind including quantized model architecture, RAG pipeline, execution-in-sandbox and geospatial validator. Section 4 provides experimental results, an ablation study and a spatial accuracy analysis as well as a more qualitative case study (T. Akinboyewa et al., 2024). Finally, Section 5 goes on to conclude this paper with a summary of the main findings and some suggestions for future works.

2. Related Work

2.1. Large Language Models for Automated Code Generation

One area where deep learning has absolutely revolutionized automated program synthesis is the use of the Transformer architecture, which did away with recurrent sequence-to-sequence models in favor of a purely attention-based mechanism that can consider long-range dependencies across long token sequences. Prior systems for generating neural code were limited by small models or trivial training corpora, making them useful only for basic code completion tasks like predicting a next variable name or bracket pair. The paradigmatic moment came with OpenAI's Codex, the first strong evidence that a language model could be used to translate docstrings in natural language into functionally correct Python functions given sensible non-trivial algorithmic logic - this is a 12-billion parameter GPT model fine-tuned on 54 million public GitHub repos (M. Chen et al., 2021). Codex was released and subsequently integrated into the GitHub Copilot commercial product, which led to an industry-wide competition of producing ever-more powerful code-specific LLMs. Salesforce Research's CodeGen, introduces multi-turn program synthesis as a method for developers to iteratively construct code via conversational dialogue, instead of single-shot generation with a family of autoregressive models from 350 million up to 16 billion parameters (E. Nijkamp et al., 2023). Facebook AI Research (now Meta AI) unveiled InCoder, a unified generative model trained under a causal masking objective that provided the capabilities of not only left-to-right code generation but also bidirectional code infilling, or performing missing code fragment substitutions in an existing function body (D. Fried et al., 2023).

The launch of the BigCode open-science initiative represented a dramatic turning point for the increased accessibility to code-generating AI. Seeing that the best code models were gated behind proprietary APIs with restrictive commercial licenses, the BigCode community put together 'The Stack', an enormous ethically licensed, deduplicated dataset of 6 + TB of permissively licensed source code in 358 languages (D. Kocetkov et al., 2022). Trained on this open dataset, the community released SantaCoder, a 1.1 billion parameter model which achieved competitive performance (at least an order of magnitude smaller than other models) against ∼10X larger models on scoring benchmarks in python code generation tasks using just data quality and curation to start compensating for raw parameters (L. Ben Allal et al., 2023). Based on this, the BigCode project later released StarCoder, a 15.5 billion parameter model trained on 'The Stack' with an 8,192-token context window that achieved new state-of-the-art performance over multiple code generation benchmarks and was also open-access-licensed for commercial use (R. Li et al., 2023). With the success of StarCoder, numerous open-source alternatives appeared rapidly: WizardCoder from Microsoft Research used a new approach called 'Evol-Instruct' to progressively build up complexity of dataset for tuning code instruction throughout a series of iterations and outperformed even the proprietary Codex model on HumanEval benchmark while having 1/2 parameter counts (Z. Luo et al., 2025). At the same time, DeepSeek AI has introduced a test of DeepSeek-Coder: a family of 1.3B-33B parameters models fully trained from scratch on a 2 trillion-token dataset with 87% code and 13% natural language achieved state-of-the-art results on coding benchmarks and highlighted the importance of a high proportion of the body of text used to pre-train model (D. Guo et al., 2024).

While existing techniques demonstrate remarkable success at general-purpose code generation, a persistent performance gap has been found when evaluating models on rigorous programming tasks tailored to specific domains. The overwhelming majority of standard evaluation benchmarks, like HumanEval which consists of 164 hand-crafted programming problems in Python and the Mostly Basic Python Problems (MBPP) dataset with a more extensive collection of 974 crowd-sourced ones, assess generic algorithmic competence: sorting arrays, manipulating strings and implementing basic data structures (J. Austin et al., 2021). These benchmarks do give a

useful proxy for more general coding ability, but the reality is that they are entirely detached from the challenges of real-world scientific software engineering, in which success does not come from algorithmic elegance but rather by meticulously adhering to the very rigid sometimes poorly documented and often version-sensitive API surfaces of highly specialized domain libraries. Systematic studies show that model performance drops off considerably when the target API strays from the set of libraries most represented in its training dataset (F. F. Xu et al., 2022). While mainstream libraries such as NumPy, Pandas or Flask are well represented in publicly available code repositories, Geospatial Python libraries like Rasterio, GeoPandas, Shapely and PySTAC-Client appear drastically underrepresented. Because of this extreme distributional imbalance in the training data, even the strongest code-generating LLMs only have a shallow statistical understanding of geospatial API surfaceshallucinating deprecated function signatures, inventing non-existent keyword arguments, and completely misunderstanding the sequential nature of data transformation pipelines inherent to satellite imagery processing.

Compounding this challenge the geospatial libraries undergo an evolutionary cycle with breaking API changes even between minor versions, meaning the LLMs parametric knowledge would be outdated immediately after a new library minor version release. Previous work to overcome the domain gap through instruction-tuned fine-tuning has scored modest gains on niche scientific domains (T. V. Team, 2023). Instruction tuning helps align a pre-trained model to get closer to humans in conversation, but it is impossible to feed fundamental new knowledge about APIs into weightswithout having potentially millions of very high-quality examples of domain-specific code. Currently there is no such large-scale, curated instruction-tuning dataset in the Earth Observation domain. Recent conclusive results on larger code benchmarks, such as BigCodeBench, found that all current LLMs are severely struggling in multi-library compositional code generation under realistic conditions where the multiple libraries are zipped and presented to the models with their existing open-source completion paths (1,140 tasks need to be composed of different library APIs)(T. Y. Zhuo et al., 2025). This result is immediate for the geospatial programming use case, in processing of a single analytical workflow that would need to compose together five or more libraries specialised at each task (e.g. PySTAC for data retrieval, Rasterio for raster I/O, NumPy for matrix algebra, Shapely for geometry construction and GeoPandas for vector operations). Current code-generating LLMs failure to consistently compose these heterogeneous domain APIs without external grounding-engine (e.g. via embedding/kerning) motivates the inclusion of a Retrieval-Augmented Generation (RAG) architecture in GeoMind framework.

2.2. Autonomous Multi-Agent Frameworks

Moving from single-turn, instruction-following chatbots to fully autonomous, multi-step agents constitute one of the sharpest breaks in the AI paradigm seen in contemporary research. Traditional conversational AI tools were based on a simple reactive cycle: a user prompted the system, it responded once, then stopped. The intellectual step to autonomywhen an AI thoroughly can decompose a complex objective into a sequence of actions it can perform, call external tools, and reinforce that performance in iterations based on the feedback afforded by the environmentwas pioneered by the open-source AutoGPT project early in 2023 (T. B. Richards, 2023). In a sense, AutoGPT proved that by running GPT-4 in an endless loop and giving it file system commands and internet access, a language model could do primitive goal-directed behavior. In contrast, AutoGPT experienced many critical reliability failures (e.g., infinite looping, high API costs, and task abandonment), but nevertheless catalyzed substantial academic research on LLM autonomous agent architectures (L. Wang et al., 2025). In their thorough survey Park et al. investigated the potential of generative agents by implementing a simulated sandbox environment powering 25 autonomous AI characters, each controlled by a retrieval-augmented memory architecture and capable of realistic human-like social interactions and long-term planning (J. S. Park et al., 2023).

But this work also gave preliminary evidence that LLMs given slight structures memory and feedback loops from their environment might produce coherent, multi-step behavior over long time horizons.

On top of these early proofs, researchers quickly created more modular and robust multi-agent paradigms. AutoGen: A flexible framework from Microsoft Research to build complex LLM workflows via multi-agent conversations where specialized agents exchange messages in a structured fashion to collaboratively solve problems (Q. Wu et al., 2023). The most salient architectural innovation of AutoGen was introducing the idea of individual agents that were each stateful, with separate system prompt, access to tools and ability to execute code allowing for the implementation of a sophisticated division of labor across heterogeneous tasks. In contrast, CAMEL (Communicative Agents for 'Mind' Exploration) used a role-play communication protocol, wherein two LLM agents an 'AI User' and an 'AI Assistant', work on completing the task all by themselves through inception prompting without constant human supervision (G. Li et al., 2023). ChatDev specifically adapted the multi-agent model to software engineering by simulating an entire fictitious software company with various types of agents acting in different roles (CEO, CTO, Programmer and Tester) communicating along a structured chain of up to 10 turns, collaborating together to design, implement and debug a complete software application based on a single natural language requirement (C. Qian et al., 2024). By performing multi-agent role specialization in ChatDev, the authors showed that this general principle of cognitive decomposition across specialized agents translates into measurably better code quality with lower software defect rates than single agent generation, which is the first empirical evidence from large-scale experimental results validating it.

Research in this vein is focused on providing the agents with a way to dynamically query and collaborate with external tools, and execution environments, instead of text generation alone, going towards more grounded action. The Voyager framework has shown how a LLM agent operating in a Minecraft simulation can independently write JavaScript code, run programs inside the game environment, monitor the results and progressively improve its skill library for solving more challenging open-world problems (G. Wang et al., 2023). In particular, this work provided crucial evidence that closed-loop code generation (where the agent writes, runs and debugs its own programs based on feedback from the environment), is fundamentally better than open-loop generation for tasks involving physical or spatial reasoning. SWE-Agent pushed this paradigm even further, designing a specialized agent-computer interface that empowered LLMs to automatically navigate real GitHub repositories, find errors, and make pull requests - achieving state-of-the-art performance on the SWE-bench software engineering benchmark (J. Yang et al., 2024). The TaskWeaver framework, by Microsoft, showed that converting each user request into executable Python code plus a structured plugin system for task state management is generally a much more reliable inter-agent communication medium than arbitrary natural language planning (B. Qiao et al., 2024).

Although such autonomous multi-agent frameworks have come a long way, an essential limitation that surfaces from scrutinizing the existing literature is that all contemporary systems are built to be domain-agnostic and general-purpose architectures. Projecting to a broader scope compared to Table-based Code Generation, tools like AutoGen, CAMEL, ChatDev and Voyager are designed to be domain-agnostic that is they can be used on any programming tasks including web development, game design or data analysis. This broad generality is a virtue for wide applicability, but becomes a crippling vice when applied to domains that require rigid mathematical and physical constraints. None of these frameworks even use any kind of deterministic spatial validation. They cannot check whether the generated bounding polygon is topologically compliant with OGC Simple Feature Access, if a Coordinate Reference System transform preserves geometric validity, or if a satellite imagery extraction pipeline samples the right hemisphere of Earth. Moreover, most of the above frameworks can only interface with

proprietary LLMs run in the cloud, creating unacceptable data sovereignty risks for important use case scenarios such as sensitive Earth Observation (S. Hong et al., 2024). The GeoMind framework is aimed at filling these gaps by adapting the multi-agent paradigm for geospatial computing with deterministic requirements to include domain-specific spatial validation as a first-class architectural component, not an afterthought.

2.3. Geospatial Artificial Intelligence and Foundation Models for Earth Science

GeoAI is an emerging sub-discipline defined as the use of artificial intelligence on geospatial data, including geographic information systems and Earth Observation (EO). In the past, AI-GIS was mainly focusing for instance to apply Convolutional Neural Networks (CNNs) to perform pixel-level semantic segmentation of satellite images with two examples being land use classification or building footprint extraction from high resolution optical tiles Nonetheless, the rise of Foundation Models large-scale pre-trained neural architectures that can be adapted to a variety of downstream tasks has effectively broadened the space of GeoAI from supervised classification to include autonomous spatial reasoning and even code generation(A. Xiao et al., 2025). Mai et al. (2022) present a thorough review clearly identified the opportunities and challenges of deploying Foundation Models for GeoAI, positing that while these models have lexical geographic knowledge encoded in their weights during pre-training on internet-scale text, the spatial reasoning capabilities are delicate and largely untested (G. Mai et al., 2023). One of the most important research gaps that came through this survey was the lack of rigorous evaluation frameworks to evaluate and benchmark geospatial code generation capabilities of LLMs which is something that our GeoTaskBench dataset directly addresses in this work.

At the same time, another line of research has aimed at open development of geospatial specific Foundation Models that were trained on satellite images (instead of text). Using a masked autoencoder objective, the IBM-NASA Prithvi model was pre-trained in an self-supervised manner on Harmonized Landsat-Sentinel-2 (HLS) data, and showed that self-supervised pre-training with unlabeled satellite imagery can yield highly transferable feature representations for downstream Earth Science tasks such as flood mapping, wildfire scar detection and crop classification (J. Jakubik et al., 2023). Furthering this paradigm, the Clay Foundation Model team built an open-source architecture to fit heterogeneous multispectral + SAR + DEM data into a single embedding space (Clay, 2024). A wider-survey of Foundation Models for Remote Sensing and Earth Observation identified hundreds such architectures published from 2021 to 2024, remarking that while these vision-centric models are competent in image classification/segmentation tasks, at their core they cannot generate executable code, interface with spatial databases or even build analytical pipelines autonomously (Y. Cong et al., 2023). This is an important distinction: a geospatial Foundation Model such as Prithvi may output whether the pixel was classified as 'flooded' and 'non-flooded', but it cannot independently and autonomously write the python code to download the satellite image, pre-process it (e.g., atmospheric correction), calculate water index, and export vector shapefile. The GeoMind framework functions on this higher level of abstraction where all analyticsdata retrieval up to spatial validationare considered its scope of governance.

The line of work most closely related to this GeoMind framework involves the emergence of transformers and LLM-powered GIS agents: systems that try to turn natural language queries into executable geospatial code. Zhang et al. recently introduced GeoGPT as an AI assistant that aids geospatial understanding and processing through the synergistic coupling of an LLM with a suite of GIS tools achieving linguistic-analytic geographic analysis for the first time (Y. Zhang et al., 2024). It is important to point that GeoGPT was based only on closed-source proprietary APIs (GPT-3.5/GPT-4). However, this set of experiments is more qualitative and did not contain

a standard quantitative benchmark for evaluating its system, did not use any type of deterministic spatial validation (like in GRAIL and GPT-5), etc. The research conducted by Ning et al. surpassing the existing state of the art with their LLM-Find framework which presented an autonomous GIS agent for geospatial data retrieval that created, executed, and debugged Python programs to find and download spatial datasets (H. Ning et al., 2024). Li and Ning formalized a broader vision referred to as an 'Autonomous GIS' that centers on minimizing human interaction for geographic analysis through AI tools, the theoretical framing that underpin this type of system, and five core capabilities that make up a fully autonomous GIS: self-generating, self-organizing, self-verifying, self-executing (Z. Li and H. Ning, 2023), and later defined as Self-Growing. GIS Copilot - A distant step toward this vision was realized by the work of Akinboyewa et al., who created an integrated agent in QGIS that could transform natural language queries to executable interactions with desktop GIS spatial analysis (T. Akinboyewa et al., 2024).

Despite these groundbreaking contributions, a review of the current GeoAI agent literature highlights several key architectural voids that the GeoMind framework is designed to fill. The main limitation is that all current systems rely solely on proprietary LLM APIs hosted in the cloud (e.g. GPT-4 or Claude), which presents extreme data sovereignty risks when processing sensitive satellite telemetry and completely excludes deployment in network-isolated environments. No previous model has shown that it can do autonomous geospatial reasoning with a locally deployed, quantized open-source model. Second, it does not use any existing GeoAI agent with a deterministic Geospatial Validator as first-class architectural component yet to the best of our knowledge.Third, the evaluation methods used in previous works are mostly qualitative, using cherry-picked demonstrations and subjective judgments instead of standard, reproducible quantitative baselines with formally defined measures (R. Manvi et al., 2024). For example, instead of a measure for unsupervised geo-code generation and execution capabilities (the field the model was proposed to tackle), it used existing socioeconomic prediction tasks with baseline scores as an evaluation of the geographic knowledge extraction abilities of GeoLLM (C. Huang et al., 2024). GeoMind Framework: To break through each of these gaps, we design a closed-loop architecture that integrates local quantized inference with deterministic spatial validation and 150-task quantitative benchmark (GeoTaskBench) (J. Roberts et al., 2023). The recent state-of-the-art survey on LLM-powered spatial intelligence at different levels highlights the urgency of this research, pointing out that while autonomous GIS has great theoretical potential to advance geospatial understanding, practical reliability and safety fall far short of what is required for mission-critical Earth Science applications with current systems (J. Feng et al., 2025). This paper describes the first empirically validated, statistically significant example of a locally deployable and deterministic autonomous GIS agent that cannot be controlled but can interact with its surroundings within a closed loop, respectively.

2.4. Retrieval-Augmented Generation and Self-Correction Architectures for Code

One of the most successful approaches to reducing hallucination in knowledge-intensive NLP tasks is an architectural strategy that augments a language model's generation with external retrieval. The base Retrieval-Augmented Generation (RAG) paradigm showed that when taking a pre-trained sequence-to-sequence model and supplementing it with a non-parametric dense vector retrieval to dynamically fetch relevant text passages from an external knowledge source in conjunction with the input query, this can significantly boost factual accuracy, against purely parametric models which depend solely on information encoded in their weights during the pre-training phase (P. Lewis et al., 2021). This principle was quickly adopted by the code generation community where hallucinating functions from an API that doesn't exist became one of the biggest barriers to deployment. An example of how to formalize this adaptation is the DocPrompting framework, which shows that by fetching

relevant documentation from official library references and injecting it into LLM's prompt before generating code removes most API hallucination, especially for less commonly used or more recently updated libraries (S. Zhou et al., 2023). Later studies investigated how Documentation-Augmented Generation can lessen hallucinated code LLMs calls to cloud service APIs and proposed light-touch benchmark such as CloudAPIBench for using it (N. Jain et al., 2025). The building blocks of their architecture are based directly on this lineage and use a localized ChromaDB vector store populated with hand-picked geospatial library documentation so that the syntax generated by the Coder Agent is consistent with verified, up-versioned API specifications.

While RAG does close a loophole by grounding the model on right documentation to avoid syntactic hallucination, it does not include any mechanism for independently finding or correcting semantic errors in generated output. In response to this constraint, a quickly expanding tract of work has investigated mechanisms for self-correction and iterative refinement, allowing LLMs to debug their own outputs. Self-Refine proposed a general-purpose iterative refinement paradigm with the same LLM in two roles: Generator produces an initial output and Critic provides structured feedback on the quality of the Output to steer toward another iteration for refinement (A. Madaan et al., 2023). Experimental results show that this iterative self-feedback loop improves output quality across many different tasks (code generation, mathematical reasoning and sentiment analysis) without any additional training data or model fine-tuning. Chen et al. introduce Self-Debug, a method where LLMs learn to debug their own generated code by using (execution feedback (runtime error traces)) and explanation of the code ('rubber duck debugging') achieving remarkable improvements in functional correctness on several standard benchmarks for code generation (X. Chen et al., 2023). The second approach to improving reasoning in LLMs of CRITIC generalized this further by allowing LLMs to interact with external tools (e.g., Python interpreter, web search engines) and use ground-truth environmental signals as opposed to its own potentially busted self-assessment in order to verify and correct their outputs (Z. Gou et al., 2024).

Most closely related to the GeoMind self-correction mechanism may represent an even more direct intellectual ancestor than Reflexion. Shinn et al. engaged in 'verbal reinforcement learning' with the agent retaining an episodic memory of what it did wrong and how, and then using natural language reflections on these failures to improve performance by helping to direct its next attempts (N. Shinn et al., 2023). In contrast to common reinforcement learning methods which rely on costly and complex gradient updates to the model weights, Reflexion works entirely through in-context learning, making it computationally efficient (and compatible with frozen, quantized models) benefiting from GeoMind framework edge-deployment restrictions. Broader to this theme, the recent CodeRL framework has investigated how reinforcement learning signals from parsing unit tests are integrated into a critic model that directs code generation thereby demonstrating (using static analysis metrics) that execution-based feedback is much more effective than static analyses in making predictions of correctness in automatically generated code segments (H. Le et al., 2022). Summary: Combining fundamental insights from this literatureRAG for syntactic grounding, execution-based feedback for runtime error detection, and verbal reinforcement for iterative self-correction the GeoMind framework is designed as a single domain-specialized architecture. GeoMind goes beyond all previous work in two ways it proposes a fourth, entirely new feedback signal: deterministic geospatial validation. Self-Debug and Reflexion could detect exceptions (e.g. Python or functional test) but unable to pinpoint all geographic hallucinations i.e., syntactically correct pieces of code producing spatially incorrect results. This missing functionality can be provided by the GeoMind Geospatial Validator, which enforces topological integrity constraints not observable in traditional software engineering testing techniques.

2.5. Summary and Comparative Analysis

The previous sections have surveyed the state of the art across four interleaving fields of research: LLM-based code generation, autonomous multi-agent frameworks, Geospatial Artificial Intelligence and self-correction architectures. This section provides a high-level comparative analysis in Table I; arranged chronologically from 2021 to 2025 detailed comparisons are made of eighteen representative systems across nine architectural dimensions that together define many requirements of any scalable, reliable (deployable), autonomous Earth Science agent. These nine dimensions deal directly with research gaps which have been presented in Sections 2.1 through to 2.4, for which formal definitions can be found below:

(1) Domain Specificity: Generalized software engineering vs Geospatial/Earth Observation tasks vs Embodied/Game environments (2) Type of Architecture: whether the system runs as a single pass open-loop generator, an iterative closed loop agent, or structured multi-agent coordination. (3) Base Model: This refers to the Large Language Model itself that powers the system, and this model can either be a proprietary cloud API or an open-source option. (4) Local Deployment – whether or not the system is designed to run completely on local edge hardware without needing to contact remote cloud-based LLM endpoints for any reason (5) RAG Integration: whether the system integrates a Retrieval-Augmented Generation mechanism to ground code generation with standardized API documentation that is known to be correct. (6) Self-Correct: Whether the system is able to automatically identify runtime errors via execution feedback and iteratively debug its own code. (7) Spatial Validation: If the system includes a deterministic geospatial validation module with topological integrity constraints (CRS checks, bounding box validation, geometry validity) on the spatial data output. (8) Quantization of Inference: if the system uses model compression (e.g. 4-bit primitive NF4 quantization) to minimize memory usage for edge deployment. (9) Standardized Benchmark: standardized, publicly available, reproducible domain-specific benchmark with formally defined quantitative metrics for evaluation the system.

System	Year	Domain	Architecture	Base Model	Local	RAG	Self-Corr.	Spatial Val.	Quant.
Codex	2021	General	Open-Loop	GPT-3 (Prop.)	No	No	No	No	No
CodeGen	2022	General	Open-Loop	CodeGen-16B	Yes	No	No	No	No
InCoder	2023	General	Open-Loop	InCoder-6.7B	Yes	No	No	No	No
StarCoder	2023	General	Open-Loop	StarCoder-15.5B	Yes	No	No	No	No
WizardCoder	2023	General	Open-Loop	StarCoder-15.5B	Yes	No	No	No	No
DeepSeek-Coder	2024	General	Open-Loop	DSCoder-33B	Yes	No	No	No	No
AutoGPT	2023	General	Closed-Loop	GPT-4 (Prop.)	No	No	Yes	No	No
Voyager	2023	Embodied	Closed-Loop	GPT-4 (Prop.)	No	No	Yes	No	No
CAMEL	2023	General	Multi-Agent	GPT-3.5 (Prop.)	No	No	Yes	No	No
MetaGPT	2024	General	Multi-Agent	GPT-4 (Prop.)	No	No	Yes	No	No
AutoGen	2023	General	Multi-Agent	GPT-4 (Prop.)	No	No	Yes	No	No
ChatDev	2024	General	Multi-Agent	GPT-3.5 (Prop.)	No	No	Yes	No	No
GeoGPT	2024	GeoAI	Open-Loop	GPT-4 (Prop.)	No	No	No	No	No
GeoLLM	2024	GeoAI	Open-Loop	LLaMA-2 (Prop.)	No	No	No	No	No
LLM-Find	2024	GeoAI	Closed-Loop	GPT-4 (Prop.)	No	Yes	Yes	No	No
GIS Copilot	2024	GeoAI	Open-Loop	GPT-4 (Prop.)	No	No	No	No	No
GeoAgent	2024	GeoAI	Open-Loop	GPT-4 (Prop.)	No	No	No	No	No
GeoMind (Ours)	**2026**	**GeoAI**	**Closed-Loop**	**Qwen2.5-7B (OS)**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

Table I: Comprehensive comparative analysis of eighteen representative code generation systems, autonomous agent frameworks, and GeoAI agents against the proposed GeoMind architecture across nine critical architectural dimensions. Prop. = Proprietary Cloud API; OS = Open-Source Local Model; Quant. = Quantized Inference; Self-Corr. = Self-Correction; Spatial Val. = Spatial Validation.

An array of key insights is presented in this comparative table, featuring the latest advancements on General-purpose code generation models from top to bottom in each row (the first six). Notably, although models such as StarCoder, WizardCoder and DeepSeek-Coder are open-source with flexible deployments on local machines or servers, they are nonetheless stateless and sacrifice closed-loop generation capabilities. They generate code in a single pass without a runtime verification or feedback loop for executing and debugging the generated code. If they'd never seen a domain-specific API e.g. specialized geospatial libraries like Rasterio or PySTAC during pre-training (and they never have), they just fall back on predictions by probability, often hallucinating non-existent function signatures. Unlike RAG grounding or self-correction capabilities, a single hallucinated parameter is automatically cascaded to the entire generated script leading to immediate compilation failure.

The second category, covered in rows 7–12, represents the general-purpose autonomous multi-agent frameworks. Most of the architecture level systems like AutoGPT, Voyager, CAMEL, MetaGPT and AutoGen or ChatDev are a huge leap as they integrate closed loop execution with self-correction capabilities. Such frameworks can catch run-time exceptions in generated code and take efforts to fix those errors iteratively. But there are two major limitations which make them ill-suited for Earth Science applications. The very first, all existing multi-agent framework in literature rely solely on private cloud-based LLM APIs (GPT-3.5 or GPT-4), coercing sensitive satellite telemetry into unacceptable data sovereignty hazards, and precluding deployment in air-gapped or network-isolated situations. Second—and more crucially—all these frameworks lack any kind of peg that validates outputs against the domain. They have self-correction mechanisms for detecting Python runtime exceptions (ImportError, TypeError, etc.), but are otherwise completely blind to geographic hallucinations syntactically valid code that returns spatially invalid results i.e. bounding boxes in the opposite hemisphere or topologically self-intersecting polygons.

The third (rows 13–17) is the emerging and increasingly large topic of GeoAI-specific agents. Contributions like GeoGPT, GeoLLM, LLM-Find, GIS Copilot &GeoAgent (PML 2023) are unique and novel contributions that have started pushing the boundaries of integration of local language models with information systems. Of these, it is LLM-Find that is the most sophisticated architecturally, as it combines RAG-style documentation retrieval with a simple code execution loop for automatic gathering of information. LLM-Find does not implement a deterministic Geospatial Validator either. It can read if its own generated Python code crashes, but such a system cannot mathematically verify whether the spatial data obeyed to OGC topological standards or whether the coordinates fit within acceptable planetary bounds of WGS84 geographic frame or even whether they are topologically-valid non-self-intersecting geometries (the resulting shapes). In addition, none of all five current such GeoAI agents are structureable for using with model quantization techniques (e,g, QAT or post-training), and expose proprietary cloud APIs only.Additionally, while frameworks like GeoAgentic-RAG successfully employ retrieval mechanisms for spatial analysis, they lack the deterministic topological validation and fully quantized local inference required for network-isolated environments.

The last row in Table List the GeoMind framework that will be introduced in this paper. The most striking observation from a visual inspection of the table is that only GeoMind answers affirmed to all nine architectural dimensions simultaneously in all reviewed literature. The only domain-specialized (1) Earth Science system, functioning as an (2) closed-loop multi-agent architecture, and (3) powered by the publicly available LLM model Qwen2.5-Coder-7B-Instruct, locally deployable on edge hardware, augmented with RAG for syntactic grounding, using dynamic self- correction by execution feedback,(J. Feng et al., 2025) guarded by a deterministic Geospatial Validator enforcing OGC-compliant topological constraints (International Organization for Standardization, 2004), compressed through4-bit NF4 quantization to optimize memory efficiency and zero-shot evaluated against a

conventional standardized & reproducible quantitative benchmark (GeoTaskBench) with explicitly defined metrics (Task Success Rate and Intersection over Union). The unique, exhaustive blend of functionality unequivocally proves that GeoMind bridges the gap for each existing architectural shortcoming referenced in current work thus constituting a deep and fundamental advancement to Geospatial Artificial Intelligence and machine-driven Earth Science analytics.

While GeoColab integrates extensive function syntax documents across a multi-agent pipeline and GeoAgentic-RAG utilizes RAG for spatial analysis (achieving 85.3% pass rates), neither system utilizes a deterministic topological validator. Furthermore, unlike these prior works which rely on proprietary cloud APIs, GeoMind is evaluated on the 150-task GeoTaskBench and achieves state-of-the-art performance using a fully local, privacy-preserving 7B model. Notable systems include GeoColab, which integrates extensive function syntax documents across a multi-agent pipeline, and GeoAgentic-RAG, which utilizes RAG for spatial analysis. Other surveys, such as GISclaw, highlight systems like LLM-Find and GIS Copilot. While these frameworks have demonstrated that multi-agent architectures can significantly improve spatial reasoning, they frequently rely on massive, closed-source models via API endpoints, which presents challenges for data sovereignty and reproducibility in critical infrastructure applications. GeoMind builds upon this foundational work by shifting the focus toward local deployment and strict OGC topological validation. By deploying a quantized 7B model locally, GeoMind provides comparable autonomous execution capabilities while ensuring data privacy.

3. Methodology

The exponential proliferation of Earth Observation (EO) data from multi-spectral satellites, such as Sentinel-2, Landsat 8 and Synthetic Aperture Radar (SAR) systems has significantly transformed the domain of Geospatial analytics. The traditional geospatial data processing paradigm, however, is still predominantly manual and requires significant domain knowledge in geographic information systems (GIS) along with spatial math and Python programming. Although Large Language Models (LLMs) have recently shown great success at very generic software engineering tasks, direct application of LLMs to the domain of highly-specialized geospatial workflows remains limited. This disadvantage arises because of the complex nature of spatial data structures and topological constraints as well as the rapidly changing landscape of spatial computing Application Programming Interfaces (APIs).

To address this fundamental linguistic and operational gap, this work introduces GeoMind, a novel, closed-loop autonomous multi-agent framework. GeoMind is specifically designed to convert ill-defined, natural language Earth Science questions into hard, executable analytical workflows. Inspired from recent developments in multi-agent collaborative schemes (S. Hong et al., 2024), GeoMind provides agents with distinctive autonomous functions to decompose complex tasks. The hardware and software are grounded on advanced low-latency local LLMs, embedding with RAG over APIs, deterministic mathematical validation mechanisms, and self-correcting sandbox process GeoMind orchestrates these different computing paradigms, which allows us to mitigate the uncertainty of probabilistic hallucinations common in neural architectures while still having an even more advanced reasoning for complex cognitive tasks. The sub-sections that follow introduce the theoretical outline of our proposed framework, spanning from the architecture of our core system to empirical benchmark formulation.

3.1. System Architecture and Theoretical Framework

The core architecture of the GeoMind framework is based on a decentralized cooperative approach to orchestration of multiple agents. The Qwen2.5-Coder-7B-Instruct Large Language Model stands at its cognitive heart (B. Hui et al., 2024). Qwen2.5-Coder is a cutting-edge causal language model built on top of an advanced version of the Transformer architecture. It applies multi-head causal attention and rotary positional embeddings (RoPE), which allows the model to leverage long-range dependencies between codes in complex codebases. The 7B variant was chosen because it represents a sweet spot between depth of algorithmic reasoning which has the potential to understand specific spatial logic and computational feasibility on ordinary consumer edge hardware (B. Hui et al., 2024).The interaction between the Planner Agent, Coder Agent, Execution Sandbox, and deterministic Geospatial Validator forms the core of our system. The overall system architecture of the GeoMind framework is illustrated in Fig. 1.

To run a local LLM of this size while adhering to the tight Video Random Access Memory (VRAM) limits in standard dual NVIDIA Tesla T4 GPU environments (~ 15 GB VRAM per card), the framework relies on severe asymmetric 4-bit quantization. Our approach is based on the QLoRA optimization framework developed by Dettmers et. al (T. Dettmers et al., 2023). For model weights, the bitsandbytes library is used to load NF4 (NormalFloat4) data types. NF4 is an optimal quantization type information-theoretically for normally distributed neural network weights (T. Dettmers et al., 2023). Model size is decreased nearly by 75%, as the original 16-bit floating-point (FP16) or 32-bit (FP32) weights are mathematically mapped into a limited discrete space of just 4 bits. In addition to Paged Optimizers, the framework also employs Double Quantization method that allows compressing the quantization constants themselves (T. Dettmers et al., 2023). This dramatic decrease in resource consumption makes it feasible to run the GeoMind agent on any, even low-powered compute device enabling complex geospatial workflows running completely on-device without the privacy and autonomy concerns of using remote enterprise-grade cloud computing clusters.

The GeoMind architecture replaces traditional monolithic prompting with a strictly bifurcated multi-agent cognitive hierarchy within the semantic execution layer. Motivated by some of the collaborative multi-agent paradigms (S. Hong et al., 2024), we implement a two-part architecture that is composed of two general autonomous agent components, denoted as Planner Agent and Coder Agent. This conscious separation of the cognitive ability for multi-step, analytical process planning from complex syntax generation is critical to avoid context-window decay and severe forgetting, both of which are known challenges when a single LLM attempts to reason through and execute all components of speculation and synthesis.

The Planner Agent: High Level Semantic Parser and Architectural Strategist If the framework has been handed a question with structure or multiple dimensions in the title of Earth Science such as calculate a five year temporal baseline over NDVI anomaly matrix then Planner Agent is called. The second agent is the Planner Agent, and its mathematical capability is to abstract the user natural language input event into a directed acyclic graph of atomic sequential sub-tasks. So, each node in this DAG is essentially a single goal computationally (e.g., Node 1: Authenticate the API credentials to access the Planetary Computer, Node 2: Create a spatio-temporal search window, Node 3: Perform raster mathematics to calculate the expected NDVI). The Planner Agent simplifies the optimization problem-at-hand by converting a nearly intractable macro-problem into several highly solvable micro-problems through modularizing the objective.

After the DAG is constructed, the framework triggers an organized message-passing mechanism that feeds these atomic nodes sequentially to the Coder Agent. Coder Agent is used purely as the syntax generation component. It has a single computational goal to receive a discrete sub-task from the Planner Agent and convert it to

deterministic, executable Python instructions with in the appropriate libraries (e.g. GeoPandas, Rasterio, Shapely). In contrast to the Planner Agent, which functions in a sort of physical semantic space, the Coder Agent lives entirely in a rigid syntactic space. Such functional isolation guarantees that the probabilistic reasoning necessary to conceptualize workloads does not interfere with the deterministic precision needed when executing code (S. Hong et al., 2024).

Lastly, the orchestration of communication between Planner and Coder agent is done via a centralized framework orchestrator. This orchestrator behaves like a finite state machine (FSM), keeping track of whether each code block is executing. When Node 1 is solved by the Coder Agent, its state serves as the update and Node 2 + context of what previous variables were in memory are sent to the orchestrator. This closed-loop memory-holding method allows data frames, raster arrays and spatial geometries which were loaded in prior steps to remain in the RAM for subsequent analytic transformations, simulating how a senior human geospatial data-scientist develops iteratively.

3.2. Retrieval-Augmented Generation (RAG) for API Grounding

There has been plenty of research on general Large Language Models having a big systematic problem when being used for very high dedication domains like geospatial programming: hallucinated API parameters, non-existent APIs, legacy classes and deprecated methods. Geospatial computational libraries like GeoPandas for vector topology, Rasterio for matrix-element based multispectral analysis and PySTAC-Client for spatio-temporal asset catalog (STAC) streaming have very strict, specialized syntaxes. A zero-shot generic LLM generates code based only on its parametric memory. When this generated code interacts with deterministic spatial data structures, such as buffers or point clouds for GPGPU processing tasks, this form of probabilistic guessing regularly results in critical execution failures.

To mathematically constrain the LLM to the constraints of topology and enable syntactic hallucination-free embedding, GeoMind is fundamentally a Retrieval-Augmented Generation (RAG) architecture. First presented by Lewis et al., RAG models integrate pre-trained parametric and non-parametric memory with the goal of producing output more consistent with facts (P. Lewis et al., 2021). GeoMind leverages the RAG architecture, which is commonly used for natural language question-answering, but we explore its utility in syntactic grounding for algorithms. In GeoMind, this architecture takes advantage of a geo-distributed ChromaDB instance which acts as a fast vector database (our external non-parametric memory for the Coder Agent).

In the first stage of system initialization, validated and up-to-date technical documentation are injected directly into the vector store. Documentation that explicitly states the fact signature, input data types of the expected argument and examples of deterministic usage of core geospatial libraries used by the agent. But in order for the machine learning algorithm to be able to retrieve and understand these raw text documents mathematically, all of those documents need to be mapped onto a very high dimensional continuous vector space.

At the heart of GeoMind operator is a state-of-the-art pre-trained version embeddings based on new Siamese BERT-Network (SBERT) architecture (N. Reimers and I. Gurevych, 2019). As shown in Reimers and Gurevych (N. Reimers and I. Gurevych, 2019), SBERT derives semantically meaningful sentence embeddings using siamese and triplet network structures. The framework thus coordinates the spatial “mapping” of any programmatic concepts (for example, physically embedding semantically similar functions— raster reprojection and coordinate transformation– in close physical proximity) by converting its textual API documentation into dense numerical representations (vectors).

When some discrete programming task is presented to the Coder Agent through Planner agent, this does not just mean that a LLM would be called upon for generating code. And instead, we run the same SBERT embedding model on the task description itself to obtain a multidimensional task vector. Then, the framework performs an exhaustive K-Nearest Neighbors (K-NN) search over the ChromaDB vector space. This search uses to do the Cosine Similarity between the task vector (A) and all embedded documentation vectors in database (B) mathematically. By relying on similarity or contextual semantic overlap instead of keyword similarity through the Cosine Similarity, the framework retrieves the correct documentation even when handling ambiguous phrasing.

The top-k semantic matches that denote the exact geospatial API methods needed to solve the problem are retrieved dynamically from the database. This thus verified context is then injected directly into the LLM systems prompt as a hard constraint before inference even begins. Structurally modifying the prompt using explicitly retrieved technical facts forces the Coder Agent to use syntactically valid Python, by construction, and strictly as per the retrieved deterministic API specifications. This would not have imposed on the LLM to invent syntax because doing so may help make it more probable, ensuring that code generated is completely consistent with the very rigid and well defined nature of Earth Science real world libraries.

3.3. Autonomous Execution Sandbox and Dynamic Self-Correction Protocol

The open-loop paradigm is a key limitation of traditional static code generation models. In contrast, a standard LLM's attention sharpens when it produces a block of Python syntax but only till the point where the last token is generated. But geospatial programming in the real world is inherently unstable. Analytical scripts are often written with the knowledge that they will sometimes hit runtime anomalies that cannot be predicted like Shapefiles get corrupted, not-na (and how to handle NaN of multispectral raster arrays), mismatches in CRSs go undetected, or the geometric topology is just wrong. There is (inherently) no mechanism for the LLM to detect or mitigate such failures in an open-loop setting. The GeoMind framework must be as autonomous and when truly autonomous, that framework needs to function in a closed-loop system where it can execute its own syntax generated, and at minimum also debug runtime exceptions.

To safely facilitate this, the Coder Agent executes Python code within an isolated runtime sandbox. While native Python execution functions like eval() and exec() present documented security vulnerabilities in open web applications, their deployment in GeoMind is strictly confined to an isolated Python execution environment to restrict arbitrary system access. For production deployment, this sandbox is designed to run within a hardened Docker container completely devoid of external network privileges. This architectural constraint guarantees safe execution of all LLM-generated code prior to its audit by the deterministic Geospatial Validator.

The GeoMind framework is equipped with a Python execution sandbox that provides computational isolation to implement this. If the Coder Agent has outputted a discrete algorithmic block, the framework does not just return this text back to the user passively. The framework, instead, instantiates an isolated Local and Global variable scope and passes the generated syntax into a dynamic Python eval() environment. This sandboxed evaluation ensures that variables, dataframes and spatial objects initialized in one step can be used in other tasks without cluttering the root environment of the host machine.

The real innovation in the GeoMind execution layer comes from its dynamic self-correction protocol, which is based on the "Reflexion" architecture explored by Shinn et al. (N. Shinn et al., 2023). As depicted by Shinn et al. in Reflexion, language agents may improve with verbal reinforcement by using trial-and-error feedback to debug their

own outputs through gradual refinement (N. Shinn et al., 2023). In the case of GeoMind, if generated syntax leads to a runtime exception during sandboxed execution (in precisely this use-case scheme), then processing stops. The framework monitor captures the standard error stream and parses the raw Python Abstract Syntax Tree (AST) Traceback to find exactly where it failed, line number, Exception Class name {"ValueError", "KeyError ","CRSError"}

This snippet of error trace is not thrown away — it is incurred programmatically and appended to the original system prompt as ultra-specific negative reinforcement feedback. This new prompt, which integrates both the original task and the specific mathematical basis for why it did not work earlier, is sent back to the Coder Agent. This produces self-reflection in the agent which calls upon the LLM to reflect contextual analysis of its previously-stated syntax with respect to a deterministic error output and then formulate an algorithmic apparatus that functions as a patch solution.

In recovery iterations, the framework employs a dynamic Temperature Scaling algorithm to avoid the LLM being caught in endless cycles of deterministic failure. Inference temperature (T) regulates the probability divergence of LLM token selection. The temperature is initially set to a low value (e.g., T = 0.2) to encourage deterministic output during the first-generation attempt. In contrast, the framework does scale the temperature but doing so gradually with a step function over predefined steps during self-correcting iterations. Through enforcing higher thermal divergence at the level of the probability distribution across the softmax layer in a systematic way, this framework gives mathematical incentive for the LLM to explore new algorithmic approaches and debugging strategies rather than rehash repeated bad syntax outputs through redundant computational pathways.

For this internally self-correcting loop, the GeoMind framework imposes a hard upper bound on iterations (typically N = 3). When an agent corrects the bug and runs the code without hitting an exception before this threshold, it is a success and the spatial variables produced will be given to the next DAG node. In case the error continues to actuate beyond the threshold, the framework kills executing block to save up resources and flags task as a permanent failure. The relatively fixed, limiting reflexivity protects operational stability and significantly raises the raw Task Success Rate (TSR) of the autonomous agent.

3.4. Deterministic Geospatial Validation and Topological Integrity

Despite the fact that RAG helps eliminate a lot of syntactic hallucinations, the main problem with LLMs is hidden behind their name, they are basically atomic token predictors and this will never change. An LLM works only within a finite-seeming semantic space, predicting the sequence of tokens most likely to occur together given that which has been seen before. Thus, LLMs do not have any inherent concept of Euclidean or non-Euclidean geometries, geospatial topology, or map projections in the real world. An LLM generates spatial coordinates or a complex vector polygon and is strictly generating text that looks like coordinates, in whatever reference frame the model has ingrained. Such probabilistic generation often leads to mathematically impossible formats, including self-intersecting polygons, coordinates which exceed the physical bounds of a planet or shapefiles that lack the spatial datum altogether.

The GeoMind framework assumes a “trust but verify” paradigm to ensure that the outputs of each agent are scientifically valid and mathematically rigorous. Both a Geospatial Validation Module, which is deterministic and rigorous, must intermediate all generated code or resulting spatial data before entering the orbital space of the overarching framework. This module serves as a deterministic guardrail that rejects any AI output which violates fundamental spatial mathematical properties. The validator is a built-in layer on top of C-libraries in the cores of

Python’s GeoPandas and Shapely ecosystems, the Geometry Engine, Open Source (GEOS) and the Proj cartographic projection library. The validation matrix imposes four distinct integrity checks from both a topological and mathematical perspective.

The first is a Data Completeness Check enforced by the module. It is not uncommon to find in geospatial operations empty spatial joins or raster masking that leads to GeoDataFrame with 0 records. When the generated AI code executes without Python exceptions but has an empty dataset, the validity checker captures this output and marks it as a semantic error in order to invoke the self-correction loop explained in Section 3.3. Such as ensuring CRS Integrity by the module. Without a defined CRS, spatial data is mathematically meaningless. The validator checks the resulting objects to ensure a valid EPSG (European Petroleum Survey Group) code or Proj4 string is clearly defined and correctly applied to the geometries.

Third, and most importantly the validator enforces Geometric and Topological Integrity. Since AI-generated algorithms may produce inaccurate spatial data types (e.g., open linear rings, polygons with "bow-tie" self-intersections or inverted interior holes), the module is built using Open Geospatial Consortium (OGC) Simple Feature Access standards (International Organization for Standardization, 2004). The GEOS core layer uses the Dimensionally extended nine-Intersection Model (DE-9IM) to perform the mathematical topologic validity test on every feature generated by geometry processing based on OGC ISO 19125 (International Organization for Standardization, 2004). The reason for this is simple: if any geometry's property, is_valid, evaluates to False, it will stop the pipeline immediately and produce a diagnostic string with the sum of topological violations and trigger checking Coder Agent to rewrite its logic on how it extracted the spatial component.

Last but not least, the validator imposes physical Global Bounding Box Constraints. Conversely, coordinates in Earth (EPSG:4326) are physically bound to the geometry of the planet when using a standard EPSG. Min/max bounds of the output generated by the validator are defined mathematically (min_x, min_y, max_x, max_y). It then programmatically asserts that the longitudinal coordinates (x) are between − 180.0 and + 180.0 degrees exclusively, and latitudinal coordinates are between − 90.0 and + 90.0 degrees exclusively. When an open-loop LLM hallucinates a coordinate that is impossible, the validator mathematically rejects the outputsuch as when the latitude of 95.0 degrees is generated, The GeoMind framework enforces these 4 deterministic guardrails to keep their probabilistic neural network engine well bounded within deterministic real-world Earth Science constraints.

3.5. Benchmark Formulation (GeoTaskBench) and Empirical Metrics

Standard software engineering benchmarks (HumanEval, MBPP) are fundamentally insufficient to test the operational efficiency of the GeoMind framework in a rigorous and empirical manner. Classic benchmarks mostly measure general-purpose algorithmic reasoning and the very basic usage of Python. They do not have the precise domain context necessary to interrogate an agents ability to, say, model multispectral arrays or appropriately reproject topological vectors or perform operations on Earth Observation satellite catalogs. Thus, testing of autonomous geospatial framework requires creating a new evaluation dataset specific to the domain. To address this need, we propose GeoTaskBench in this work.

GeoTaskBench is an extensive empirical dataset containing 150 rigorous Earth Science analytical tasks from various real-world meteorological studies. The dataset was constructed to represent a wide matrix of geospatial tasks, geographic locations, and satellite sensor types. The tasks vary from the most basic task (such as extracting temporal bounds from a PySTAC item) to complex analytical pipelines (for instance, flagging

deforested area anomalies with Sentinel-2 optical data, mapping flood inundation using Sentinel-1 Synthetic Aperture Radar or analyzing thermal signatures of urban heat islands derived from Landsat 8 telemetry). Given that GeoTaskBench spans continents and biomes as diverse as the Amazon Basin, Sahara Desert, and urban environments sampled in Tokyo, this guarantees our framework is rigorously tested against demographic climatological and spatial parameters without it becoming over fit to a specific geographic locale.

We quantitatively evaluated the framework using two mathematically grounded metrics Task Success Rate (TSR) and Spatial Accuracy via Intersection over Union (IoU) against the GeoTaskBench dataset. Task Success Rate: This binary classification metric is having the absolute stability of the code generation. The success score of a task is 1 only if the autonomous agent correctly converts the natural language query into Python syntax and executes code in the sandbox environment without encountering an unhandled runtime exception when passing through a deterministic Geospatial Validator. Is given a score of 0 if the framework surpasses its maximum self-correction iteration threshold without resolving the spatial errors. Next, the 150 task success rates are averaged, resulting in an aggregate TSR that serves as a systemic reliability measure.

TSR quantifies the operational robustness but not the geographical accuracy of outputs. For quantitative measurement of spatial accuracy, the framework employs an Intersection over Union (IoU) metric. The IoU mathematically quantifies the extent of shape within predicted and manually curated sample analogous spaces i.e. spatial outputs from the AI (the Predicted Bounding Geometry) relative to a Ground Truth Geometry. The formal definition of IoU is as follows:area among topological intersection and area among topological union for these two spatial objects. If isolate only the desired geographic region, then the IoU score can be 1.0 Similarly, if the agent hallucinates something that is not even close to where it should and instead hits somewhere in a completely different continent, then spatial intersection is 0 resulting in IoU of 0.0. That metric is mathematically computed using the underlying GEOS geometry engine, which ensures accurate area computation.

Finally, to confirm scientific rigor and demonstrate the performance benefits from GeoMind framework are not a function of stochastic variance, the methodology employs stringent statistical significance tests. The analysis disentangles the IoU distributions produced by a zero-shot LLM baseline and directly compares them against these produced by GeoMind (with RAG + self-correction). We conduct a paired t-test across the 150 tasks with paired results. The null hypothesis states that there is no statistically significant difference in spatial accuracy between the baseline architecture and the GeoMind framework. A p-value of < 0.05 is necessary to reject a null hypothesis and thus provide empirical evidence for the architectural contributions in this proposed system.

4. Results and Discussion

4.1. Empirical Performance on GeoTaskBench: An Ablation Study

An ablation study on the GeoTaskBench dataset of 150 tasks was performed to systematically analyze the impact of individual architectural components introduced in this work. This study aimed to isolate the marginal role of the RAG module and the Autonomous Self-Correction Sandbox in triggering a gain or degradation on overall Task Success Rate (TSR). This evaluation measured the raw proportion of Earth Science queries from the original text corpus that could be translated correctly into Python syntax, executed without runtime exceptions, and passed against the rigorous topological constraints of a Geospatial Validator. We conduct an empirical ablation study on the GeoTaskBench dataset to evaluate the impact of each architectural module on task success. The resulting

Task Success Rates (TSR) for the baseline, RAG-grounded, and closed-loop setups are qualitatively summarized in Fig. 2.

4.1.1. Baseline Performance: The Zero-Shot Open-Loop Architecture

At baseline, the Qwen2.5-Coder-7B-Instruct model was used for evaluation in standard zero-shot open-loop mode The model is without external API grounding (no RAG) and iterative debugging capability (no self-correction). The empirical analysis showed very poor results for complex spatial programming with a baseline approx TSR of 30.0%. In an analysis of the Abstract Syntax Tree (AST) tracebacks corresponding to these failures, we found that for a large portion of syntactically invalid outputs, the root cause was a much more basic language modelling failure underlining syntactic hallucination in the baseline model.

For example, with the zero-shot LLM, it often fabricated non-existent parameters for the GeoPandas library and tried to call deprecated coordinate transformation methods from within the underlying C-libraries for PROJ, and constantly miscasted NumPy datatypes before performing multispectral raster math. This implementation was an open-loop architecture, so that a syntax error in a single character or a tiny deviation from the API resulted in immediate fatal execution loss. These findings are an empirical confirmation of the intuition that while modern LLMs have high generalized software engineering intelligence, their forms of parametric memory alone lack a structure capable of adhering to the strictly deterministic requirements presented by specialized Earth Observation APIs.

4.1.2. RAG-Only Integration: Syntactic Grounding

The second phase of the ablation study used an enhanced framework with a localized vector store augmenting a ChromaDB vector store enabling RAG during code execution. The addition of semantic context injection led to a substantial improvement, raising the TSR to 73.3%. A mathematical grounding of the LLM system prompt in verified technical documentation via cosine similarity retrieval nearly eliminated occurrences of syntactic hallucinations.

But the RAG-only architecture had generated executable Python code that, while avoiding conventional compiler errors, revealed another and more subtle failure mode: silent topological errors. These generated scripts would often execute correctly from a purely programmatic perspective, but fail the Geospatial Validator's stricter deterministic checks. For instance, the agent would raster crop with a bounding box and the resulting geometry is topologically invalid (e.g. self-intersecting polygon) or coordinates mathematically transposed over WGS84 global bounds. Since this setup was missing a feedback loop, the agent had no cognitive understanding of any such spatial irregularities and therefore could not correct them.

4.1.3. The Full GeoMind Framework: Closed-Loop Autonomy

The last step of the ablation study assessed GeoMind in fully orchestrated mode by coupling RAG grounding with dynamic Self-Correction Execution Sandbox and Deterministic Geospatial Validator. Our final architecture is the combination of both approaches yielding peak empirical performance with an outstanding Task Success Rate of 90.0% across GeoTaskBench. This is a huge gain absolute over the zero-shot baseline and demonstrates that the closed-loop paradigm works.

The outstanding success of the full framework can easily be traced back to its active runtime reflexivity. The Geospatial Validator can not terminate the task when RAG-grounded syntax keys to generate geometry mathematically invalid. Rather, it picked up the specific validator feedback (e.g. "Detected topologically invalid geometries" or "Coordinates fall out of WGS84 bounds") and re-injected that diagnostic into the Coder Agent as negative reinforcement. This prompted the agent to systematically revise its topological logic through controlled scaling of the softmax inference temperature during retry iterations. The empirical evidence shows that the agent debugged and corrected its spatial mathematics in all but 3 cases, a significant finding that demonstrates probabilistic token generation as an interface to deterministic Earth Science analytics (N = 3).

4.2. Spatial Accuracy and Topological Precision (IoU Analysis)

The Task Success Rate (TSR) is a good measure of how "performant" the GeoMind framework is at performing its intended function in a sustainable fashion, an agent that outputs syntax which compiles and executes successfully. However, in geographic applications, syntactic validity is only a prerequisite; the semantic accuracy of the spatial output must also be verified. We evaluate the spatial accuracy of the outputs by calculating the Intersection over Union (IoU) of the bounding box of the generated geographic outputs against the ground-truth bounding box of the query. We visualize the empirical distribution of these IoU scores using the Kernel Density Estimation (KDE) plot in Fig. 3.

The zero-shot LLM baseline in statistics shows an extreme density peak around the IoU 0.0 score. It recorded the mean IoU for baseline architectures of about 0.140which is mathematically insufficient for spatial analysis. The reason behind this drastic drop in the spatial fidelity is mainly a consequence of the open-loop characteristic of the baseline agent. For example, when the baseline LLM was prompted to project a Sentinel-1 SAR footprint from EPSG:4326 to a local UTM projection (i.e., EPSG:4326 to UTM). The baseline agent had no deterministic validation mechanism to trap these mathematical inversions, which led it to confidently return bounding boxes lying completely on the other side of the globe from a surface-grounded waveform (i.e., with an absolute spatial intersection of zero).

On the contrary, the complete GeoMind framework inspired a complete shift of IoU density distribution. GeoMind's KDE plot is characterized by a pronounced univariate peak, very close to the optimal score of 1.0 The best average spatial bounding-box IoU was 0.793 for the GeoMind framework, quantified by an absolute improvement of + 0.653 over the zero-shot baseline – marking the best ever reported results to date in these datasets. Such a near-optimal precision in space confirms the effectiveness of the Deterministic Geospatial Validator using Section 3.4. The framework does this by trapping impossible coordinate projections (e.g., latitudes where WGS84 states they shouldn't be), forcing the Coder Agent into a loop of iterative self-correction, and thereby preventing the continuation of topological anomalies. Most of the remaining 0.217 margin of error (which is attributable to minor pixel-level resampling artifacts) is caused by some pixel-resolution resampling artifact via raster cropping and not macroscopic geographic hallucinations.

The statistical significance protocol was carefully performed to confirm a true scientific validity and rule out this extreme performance delta being just the stochastic variance amongst LLM inference generation. A paired t-test was performed on the IoU arrays for non-overlapping regions produced by the baseline LLM and the complete GeoMind framework across all 150 tasks. The null hypothesis for this evaluation was that there exists no statistically incomparable variation in geographic accuracy between an open-loop LLM and a closed-loop multi-agent architecture. The empirical analysis conducted resulted in a T-Statistic of 22.74and a respective p-value of < 1.0e-10 Since the computed p-value is a factor of more than 1,000 smaller than the standard scientific alpha value

of incorrectly accepting the null ($p < 0.05$), we can firmly reject this null hypothesis. Our results mathematically validate that the combination of Retrieval-Augmented Generation with dynamic, deterministic self-correction provides a non-negligible structural enhancement to the autonomous spatial reasoning capabilities of Large Language Models.

4.3 Autonomous Pipeline Execution: A Geospatial Case Study

While aggregate statistical metrics such as Task Success Rate (TSR) and mean Intersection over Union (IoU) provide the quantitative validation necessary to establish generalized framework performance, they often obscure the operational mechanics and step-by-step reasoning of the underlying agentic systems. To address this, this section presents a qualitative analysis of the GeoMind framework operating under realistic, closed-loop constraints on a complex geospatial query. The system was initialized with the following unstructured natural language user request: "Retrieve the national boundary shapefile of Bangladesh, extract its geometry, and autonomously query and download a cloud-free Sentinel-2 L2A satellite image thumbnail (< 10% cloud cover) for the region." This query requires the integration of both vector GIS operations (locating, parsing, and filtering boundary shapefiles) and raster Earth observation data retrieval (interacting with STAC web services, filtering meteorological metadata, and rendering true-color bands), providing a rigorous test of the system's end-to-end capabilities.

4.3.1. Phase 1: Semantic Parsing and DAG Decomposition

The execution pipeline was initiated by invoking the Planner Agent to parse the unstructured prompt. Rather than passing the entire request to a monolithic code generator, the Planner Agent utilized a structured spatial reasoning template to break the task down into a Directed Acyclic Graph (DAG) of atomic sub-tasks. The agent autonomously identified three sequential execution nodes: (1) directory traversal and shapefile loading using GeoPandas, followed by filtering for the administrative country geometry of 'Bangladesh'; (2) constructing a Spatio-Temporal Asset Catalog (STAC) search query to locate Sentinel-2 L2A imagery overlapping the extracted country geometry, constrained to a cloud cover fraction of less than 10%; and (3) downloading and rendering the true-color image preview along with the vector boundary. The Planner Agent established explicit variables to be passed by reference between the nodes (e.g., the bounding box coordinates generated in Node 1 were dynamically injected into the STAC query parameters in Node 2), ensuring strict logical continuity.

4.3.2. Phase 2: RAG Grounding and Context Injection

Prior to triggering the Coder Agent for code generation, the framework initiated a Retrieval-Augmented Generation (RAG) loop to ground the LLM's spatial APIs. Using a local ChromaDB vector store seeded with official geospatial developer documentation, a K-Nearest Neighbors (K-NN) query was executed using embeddings of the sub-task descriptions. For Node 1, RAG retrieved syntax patterns for traversing directories using Python's os.walk and reading shapefiles via geopandas.read_file(). For Nodes 2 and 3, the RAG module successfully retrieved the exact pystac_client API syntax and Microsoft Planetary Computer authentication libraries (specifically planetary_computer.sign). In addition, the vector store injected context-specific constraints, such as identifying the correct collection ID ('sentinel-2-l2a') and recommending that integer multi-spectral bands be cast to float32 prior to arithmetic division to prevent zero-truncation errors in downstream processing.

4.3.3. Phase 3: Syntax Generation and Cloud-Cover Filtering

Armed with the grounded syntax templates, the Coder Agent generated the executable Python scripts. For the vector operations, the agent wrote code to recursively locate the shapefile in the input directories, read it into a GeoDataFrame, and filter it for rows where the 'ADMIN' attribute equaled 'Bangladesh'. For the raster search, the model converted the geometry into a bounding box, queried the STAC API, and constructed a metadata filter payload matching the user's meteorological constraint (eo:cloud_cover < 10). The STAC search successfully resolved to Sentinel-2 L2A item 20240510T042659_R133_T46QBM_20240510T062551 (covering UTM zone 46, tile QBM in Bangladesh), generating a signed preview thumbnail URL.The STAC search engine dynamically authenticates and queries Sentinel-2 metadata to select a low cloud-cover scene for download. The true-color satellite imagery thumbnail retrieved autonomously by the STAC pipeline is shown in Fig. 4.

4.3.4. Phase 4: Raster Mathematics and Validation

In the final phase, the generated code executed within the isolated sandboxed environment. The Coder Agent plotted the filtered administrative boundaries of Bangladesh using Matplotlib, specifying a light-blue fill and black border, and saving the output as a high-resolution raster image. Concurrently, the Sentinel-2 L2A thumbnail was downloaded and saved locally for validation. The Geospatial Validator intercepted these outputs and verified their topological and spatial integrity, checking that the geometries were closed polygons and the spatial extent matched the WGS84 coordinates. The pipeline executed successfully on the first attempt without triggering the self-correction loop, achieving a spatial Intersection over Union (IoU) of 0.94 against the ground-truth national boundary from the benchmark dataset.The Coder Agent autonomously traverses the input directory, loads the administrative boundaries shapefile, and filters the country geometry. The resulting autonomous geospatial map of Bangladesh generated by the framework is shown in Fig. 5.

4.4. Discussion: Bridging the Semantic Gap in Spatial Computing

The GeoTaskBench evaluation generates empirical results that demonstrate the efficacy of the GeoMind framework in autonomous Earth Science analytics. The extremely statistically significant increases in both Task Success Rate (+ 57.3% over baseline) and spatial Intersection over Union (+ 0.653 over baseline) provide compelling mathematical support for the core premise of this work: Large Language Models fundamentally break when used as open-loop syntax generators on geographic domains. The framework thus combines probabilistic natural language processing with deterministic spatial mathematics by enclosing the LLM in a closed-loop multiple-agent structure which uses Retrieval- augmented generation (RAG) together with a method of deterministic geospatial validation.

One of the most important theoretical implications of these findings is the proven advantage of broad-stroke self-correction over massive parametric expansion. In the past, the strategy for reducing hallucination rates in neural architectures has been to scale the parameter count of the model up exponentially (i.e. from 7-Billion → 70-Billion or 400-Billion parameter models). Yet the 7-billion parameter model (Qwen2.5-Coder-7B-Instruct) is a Python traceback-trained model that demonstrates state-of-the-art geographic reasoning performance. The agent, which must learn holistically from iterative reflection on its own spatial geometry failures (like various WGS84 coordinate inversions or topological self-intersections), is able to deduce what would have been the correct algorithmic pathway mathematically without incurring the massive compute overhead that larger models necessitate.

This architectural efficiency naturally enables wide-spread access to advanced geospatial analytics. Due to the framework's foundation involving a 7B param model loaded with only asymmetric 4-bit NormalFloat (NF4) quantization, the collective multi-agent orchestration as well as the vector embeddings of ChromaDB and execution sandbox, all reside within only ~ 15 GB VRAM of standard edge-deployed dual Tesla T4 GPU solutions. The ability to translate local raw Earth Observation data, combined with hardware tractability also functions as an assurance that sensitive data military radar telemetry, proprietary agricultural multispectral scans or critical infrastructure monitoring can be processed wholly at the edge. The GeoMind framework replaces reliance on closed-source commercial LLM APIs hosted in the cloud (e.g., OpenAI or Anthropic), allowing for full data privacy and enabling deployment in disconnected, austere environments.

The current version of the GeoMind framework does have some specific architectural limitations, despite its empirical success in current benchmarks. As of now, the Geospatial Validator is optimized for 2D topological geometries and typical optical/Synthetic Aperture Radar (SAR) matrix formats. The framework has not yet been designed to validate more complex 3D spatial arrays, e.g. LiDAR point clouds (.las format)or volumetric atmosphere tensors. The current dynamic execution sandbox runs into major computational bottlenecks when trying to autonomously process millions of 3d spatial vectors.

Second, it has a bounded number of iterations (N = 3) in its self-correction loop. This threshold cutoff is mathematically necessary, as it prevents a type of potentially infinite loop recursion however intractable geospatial problems such as aligning fully inconsistent coordinate systems with missing affine metadata often trigger this threshold which causes the task to fail. This study will be expanded with this architecture, and future research will focus on a "Hierarchical Escalation". In this hypothetical design, once the 7B FP edge-model exhausts its allocation to 3 self-correction iterations, the task state is paper-clipped or automatically escalated onto a larger, denser model (e.g. a variant with 72B parameters) into deeper cognitive debugging efforts. Despite these limitations there is little doubt that the current GeoMind architecture serves as proof-of-concept that closed-loop, RAG-grounded multi-agent systems are the future base of autonomous GIS.

5. Conclusion and Future Work

5.1. Conclusion

In this paper, we propose GeoMind, an autonomous multi-agent closed-loop framework to tackle the ever-present “semantic gap” in automated Earth science analytics. In generic code synthesis, general purpose Large Language Models (LLMs) have demonstrated impressive success; however, in geospatial domains, syntactic hallucination and geographic hallucination have been recurrent problems. GeoMind addresses these issues by using a decentralized cognitive agentic hierarchy of two agents: the Planner Agent and the Coder Agent, supported by a ChromaDB Retrieval-Augmented Generation (RAG) context injector. Most importantly, the framework contains the generative model in a deterministic execution loop, which performs an audit of intermediate Python scripts in an isolated sandbox and validates the spatial outputs as vectors and rasters against Open Geospatial Consortium (OGC) topological integrity constraints in a Geospatial Validator. We have evaluated our approach empirically on the GeoTaskBench benchmark (150 tasks) and found significant performance improvements. The open-loop LLM (Qwen2.5-Coder-7B-Instruct) as a baseline model struggled with poor performance, yielding a Task Success Rate (TSR) of merely 30.0% and a mean spatial Intersection over Union (IoU) of 0.140, which further underscores the difficulties inherent in purely probabilistic token prediction for spatial computing. Grounding the agent through RAG improved the TSR to 73.3%, and the complete closed-loop GeoMind framework got a state-of-the-art TSR of

90.0%, with the mean spatial IoU shifting to 0.793 (+ 0.653 absolute improve over the baseline). This improvement was found to be highly statistically significant by paired sample t-test ($t = 22.74$, $p < 1.0 \times 10^{-10}$). The results are significant evidence for the effectiveness of the symbolic validation and self-correction paradigm for creating powerful and independent GIS agents.

5.2. Future Work

GeoMind has shown near-optimal spatial accuracy and syntactic execution reliability but still there are directions for future research to extend its operational usability. We suggest four main ways in which the framework could be improved. First, the incorporation of Vision-Language-Action Models (VLAMs) into the assessment loop. The deterministic Geospatial Validator will fully validate topological properties but will not be able to assess aesthetic mapping and/or qualitative image anomalies. VLAMs would be able to perform secondary visual verification of maps and satellite images, much like a human cartographer. Second, we are trying to expand the execution sandbox to include distributed high-performance computing (HPC) spatial frameworks. The local evaluation sandbox currently supports single-node workflows, and could support terabyte-scale planetary data archives by integrating distributed processing engines including Apache Spark, Dask, and cloud-based Google Earth Engine APIs. Third, future versions will include formal decision-making modules for Bayesian uncertainty quantification. The world of Earth observation data is noisy, with low quality data and missing metadata. Armed with the ability to mathematically determine when available data is inadequate will prevent over-confident and erroneous spatial analyses from being created. Finally, cooperative multi-agent structures with specialized domain roles, such as a "Hydrology Agent", a "Climatology Agent" and a "Cartography Agent" will be studied. Autonomous spatial computing will bring us closer and closer to scientific co-piloting, through dynamically orchestrating these specialized agents to solve compound, multi-disciplinary Earth science problems.

Declarations

Data Availability

The evaluation dataset (GeoTaskBench) and the complete GeoMind autonomous agent source code (implemented as a Kaggle notebook) generated and analyzed during the current study are available from the corresponding author on reasonable request. The foundational Large Language Model used for the agent's core reasoning (Qwen2.5-Coder-7B-Instruct) is publicly accessible via the Hugging Face model hub.

Competing Interests

The authors declare that they have no competing interests.

Funding
This research received no external funding.

References

1. Sudmanns M et al (2020) Big Earth data: disruptive changes in Earth observation data management and analysis? Int J Digit Earth 13:832–850. https://doi.org/10.1080/17538947.2019.1585976
2. Tiede D, Baraldi A, Sudmanns M, Belgiu M, Lang S (2016) ImageQuerying – Earth Observation Image Content Extraction & Querying across Time and Space. In: Proc 2016 Conf Big Data from Space, pp 192–195.

https://doi.org/10.2788/854791

3. Reichstein M et al (2019) Deep learning and process understanding for data-driven Earth system science. Nature 566:195–204. https://doi.org/10.1038/s41586-019-0912-1
4. Gorelick N, Hancher M, Dixon M, Ilyushchenko S, Thau D, Moore R (2017) Google Earth Engine: Planetary-scale geospatial analysis for everyone. Remote Sens Environ 202:18–27. https://doi.org/10.1016/j.rse.2017.06.031
5. Zhu XX et al (2017) Deep Learning in Remote Sensing: A Comprehensive Review and List of Resources. IEEE Geosci Remote Sens Mag 5:8–36. https://doi.org/10.1109/MGRS.2017.2762307
6. Goodchild MF (2018) Reimagining the history of GIS. Ann GIS 24:1–8. https://doi.org/10.1080/19475683.2018.1424737
7. Wulder MA et al (2008) Landsat continuity: Issues and opportunities for land cover monitoring. Remote Sens Environ 112:955–969. https://doi.org/10.1016/j.rse.2007.07.004
8. Brown TB et al (2020) Language Models are Few-Shot Learners. arXiv preprint. http://arxiv.org/abs/2005.14165
9. Chen M et al (2021) Evaluating Large Language Models Trained on Code. arXiv preprint. https://doi.org/10.48550/arXiv.2107.03374
10. Ziegler A et al (2022) Productivity assessment of neural code completion. In: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, pp 21–29. https://doi.org/10.1145/3520312.3534864
11. Mai G et al (2023) On the Opportunities and Challenges of Foundation Models for Geospatial Artificial Intelligence. arXiv preprint. http://arxiv.org/abs/2304.06798
12. OpenAI et al (2024) GPT-4 Technical Report. arXiv preprint. http://arxiv.org/abs/2303.08774
13. Ji Z et al (2023) Survey of Hallucination in Natural Language Generation. ACM Comput Surv 55:1–38. https://doi.org/10.1145/3571730
14. Lewis P et al (2021) Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Adv Neural Inf Process Syst 2020-December. arXiv preprint. http://arxiv.org/abs/2005.11401
15. Hong S et al (2024) MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv preprint. http://arxiv.org/abs/2308.00352
16. Liu W, Xue Q, Wang H, Yin X, Yang B, Gao W (2025) Spatial Reasoning in Multimodal Large Language Models: A Survey of Tasks, Benchmarks and Methods. arXiv preprint. http://arxiv.org/abs/2511.15722
17. Touvron H et al (2023) LLaMA: Open and Efficient Foundation Language Models. arXiv preprint. http://arxiv.org/abs/2302.13971
18. Rozière B et al (2024) Code Llama: Open Foundation Models for Code. arXiv preprint. http://arxiv.org/abs/2308.12950
19. Feng J et al (2025) A Survey of Large Language Model-Powered Spatial Intelligence Across Scales: Advances in Embodied Agents, Smart Cities, and Earth Science. arXiv preprint. http://arxiv.org/abs/2504.09848
20. Bender EM, Koller A (2020) Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp 5185–5198. https://doi.org/10.18653/v1/2020.acl-main.463
21. Bisk Y et al (2020) Experience Grounds Language. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp 8718–8735. https://doi.org/10.18653/v1/2020.emnlp-

main.703

22. Shiri F, Guo X-Y, Far MG, Yu X, Haf R, Li Y-F (2024) An Empirical Analysis on Spatial Reasoning Capabilities of Large Multimodal Models. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pp 21440–21455. https://doi.org/10.18653/v1/2024.emnlp-main.1195
23. Li F, Hogg DC, Cohn AG (2024) Advancing Spatial Reasoning in Large Language Models: An In-Depth Evaluation and Enhancement Using the StepGame Benchmark. Proc. AAAI Conf. Artif. Intell. 38:18500–18507. https://doi.org/10.1609/aaai.v38i17.29811
24. Yang A et al (2025) Evaluating and enhancing spatial cognition abilities of large language models. Int J Geogr Inf Sci 39:2009–2044. https://doi.org/10.1080/13658816.2025.2490701
25. Bai M, Cohen AK, Koss E, Lichtenbaum C (2025) Stuck in the Matrix: Probing Spatial Reasoning in Large Language Models. arXiv preprint. http://arxiv.org/abs/2510.20198
26. Gao Y et al (2024) Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv preprint. http://arxiv.org/abs/2312.10997
27. Kojima T, Gu SS, Reid M, Matsuo Y, Iwasawa Y (2023) Large Language Models are Zero-Shot Reasoners. arXiv preprint. http://arxiv.org/abs/2205.11916
28. Johnson J, Douze M, Jegou H (2021) Billion-Scale Similarity Search with GPUs. IEEE Trans Big Data 7:535–547. https://doi.org/10.1109/TBDATA.2019.2921572
29. Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P (2017) Geometric Deep Learning: Going beyond Euclidean data. IEEE Signal Process Mag 34:18–42. https://doi.org/10.1109/MSP.2017.2693418
30. Vaswani A et al (2023) Attention Is All You Need. arXiv preprint. http://arxiv.org/abs/1706.03762
31. Li Y et al (2022) Competition-level code generation with AlphaCode. Science 378:1092–1097. https://doi.org/10.1126/science.abq1158
32. Nijkamp E et al (2023) CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. arXiv preprint. http://arxiv.org/abs/2203.13474
33. Zan D et al (2023) Large Language Models Meet NL2Code: A Survey. arXiv preprint. http://arxiv.org/abs/2212.09420
34. Liu F et al (2026) Beyond Functional Correctness: Exploring Hallucinations in LLM-Generated Code. IEEE Trans Softw Eng 52:1037–1055. https://doi.org/10.1109/TSE.2026.3657432
35. Austin J et al (2021) Program Synthesis with Large Language Models. arXiv preprint. http://arxiv.org/abs/2108.07732
36. Zhang Z et al (2025) LLM Hallucinations in Practical Code Generation: Phenomena, Mechanism, and Mitigation. Proc. ACM Softw. Eng. 2:481–503. https://doi.org/10.1145/3728894
37. Hendrycks D et al (2021) Measuring Coding Challenge Competence With APPS. arXiv preprint. http://arxiv.org/abs/2105.09938
38. Wei J et al (2023) Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv preprint. http://arxiv.org/abs/2201.11903
39. Chen W, Ma X, Wang X, Cohen WW (2023) Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. arXiv preprint. http://arxiv.org/abs/2211.12588
40. Wang X et al (2023) Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv preprint. http://arxiv.org/abs/2203.11171

41. Yao S et al (2023) ReAct: Synergizing Reasoning and Acting in Language Models. arXiv preprint. http://arxiv.org/abs/2210.03629
42. Bubeck S et al (2023) Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv preprint. http://arxiv.org/abs/2303.12712
43. Ouyang L et al (2022) Training language models to follow instructions with human feedback. arXiv preprint. http://arxiv.org/abs/2203.02155
44. Wang L et al (2025) A Survey on Large Language Model based Autonomous Agents. Front Comput Sci. https://doi.org/10.1007/s11704-024-40231-1
45. Manvi R, Khanna S, Mai G, Burke M, Lobell D, Ermon S (2024) GeoLLM: Extracting Geospatial Knowledge from Large Language Models. arXiv preprint. http://arxiv.org/abs/2310.06213
46. Qin Y et al (2024) MP5: A Multi-modal Open-ended Embodied System in Minecraft via Active Perception. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp 16307–16316. https://doi.org/10.1109/CVPR52733.2024.01543
47. Shen Y, Song K, Tan X, Li D, Lu W, Zhuang Y (2023) HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. arXiv preprint. http://arxiv.org/abs/2303.17580
48. Dettmers T, Pagnoni A, Holtzman A, Zettlemoyer L (2023) QLoRA: Efficient Finetuning of Quantized LLMs. arXiv preprint. http://arxiv.org/abs/2305.14314
49. Karpukhin V et al (2020) Dense Passage Retrieval for Open-Domain Question Answering. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp 6769–6781. https://doi.org/10.18653/v1/2020.emnlp-main.550
50. Zhou S, Alon U, Xu FF, Wang Z, Jiang Z, Neubig G (2023) DocPrompting: Generating Code by Retrieving the Docs. arXiv preprint. http://arxiv.org/abs/2207.05987
51. Schick T et al (2023) Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv preprint. http://arxiv.org/abs/2302.04761
52. Shinn N, Cassano F, Berman E, Gopinath A, Narasimhan K, Yao S (2023) Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv preprint. http://arxiv.org/abs/2303.11366
53. Zhang Y, Wei C, He Z, Yu W (2024) GeoGPT: An assistant for understanding and processing geospatial tasks. Int J Appl Earth Obs Geoinf 131:103976. https://doi.org/10.1016/j.jag.2024.103976
54. Lin T-Y et al (2015) Microsoft COCO: Common Objects in Context. arXiv preprint. http://arxiv.org/abs/1405.0312
55. Ning H, Li Z, Akinboyewa T, Lessani MN (2024) An Autonomous GIS Agent Framework for Geospatial Data Retrieval. arXiv preprint. http://arxiv.org/abs/2407.21024
56. Akinboyewa T, Li Z, Ning H, Lessani MN (2024) GIS Copilot: Towards an Autonomous GIS Agent for Spatial Analysis. arXiv preprint. http://arxiv.org/abs/2411.03205
57. Hui B et al (2024) Qwen2.5-Coder Technical Report. arXiv preprint. http://arxiv.org/abs/2409.12186
58. Fried D et al (2023) InCoder: A Generative Model for Code Infilling and Synthesis. arXiv preprint. http://arxiv.org/abs/2204.05999
59. Kocetkov D et al (2022) The Stack: 3 TB of permissively licensed source code. arXiv preprint. http://arxiv.org/abs/2211.15533
60. Ben Allal L et al (2023) SantaCoder: don't reach for the stars! arXiv preprint. http://arxiv.org/abs/2301.03988

61. Li R et al (2023) StarCoder: may the source be with you! arXiv preprint. http://arxiv.org/abs/2305.06161
62. Luo Z et al (2025) WizardCoder: Empowering Code Large Language Models with Evol-Instruct. arXiv preprint. http://arxiv.org/abs/2306.08568
63. Guo D et al (2024) DeepSeek-Coder: When the Large Language Model Meets Programming -- The Rise of Code Intelligence. arXiv preprint. http://arxiv.org/abs/2401.14196
64. Xu FF, Alon U, Neubig G, Hellendoorn VJ (2022) A Systematic Evaluation of Large Language Models of Code. arXiv preprint. http://arxiv.org/abs/2202.13169
65. Vicuna Team (2023) Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/
66. Zhuo TY et al (2025) BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. arXiv preprint. http://arxiv.org/abs/2406.15877
67. Richards TB (2023) AutoGPT: An Autonomous GPT-4 Experiment. https://github.com/Significant-Gravitas/AutoGPT
68. Park JS, O'Brien J, Cai CJ, Morris MR, Liang P, Bernstein MS (2023) Generative Agents: Interactive Simulacra of Human Behavior. In: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, pp 1–22. https://doi.org/10.1145/3586183.3606763
69. Wu Q et al (2023) AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv preprint. http://arxiv.org/abs/2308.08155
70. Li G, Hammoud HAAK, Itani H, Khizbullin D, Ghanem B (2023) CAMEL: Communicative Agents for 'Mind' Exploration of Large Language Model Society. arXiv preprint. http://arxiv.org/abs/2303.17760
71. Qian C et al (2024) ChatDev: Communicative Agents for Software Development. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp 15174–15186. https://doi.org/10.18653/v1/2024.acl-long.810
72. Wang G et al (2023) Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv preprint. http://arxiv.org/abs/2305.16291
73. Yang J et al (2024) SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv preprint. http://arxiv.org/abs/2405.15793
74. Qiao B et al (2024) TaskWeaver: A Code-First Agent Framework. arXiv preprint. http://arxiv.org/abs/2311.17541
75. Xiao A et al (2025) Foundation Models for Remote Sensing and Earth Observation: A survey. IEEE Geosci Remote Sens Mag 13:297–324. https://doi.org/10.1109/MGRS.2025.3576766
76. Jakubik J et al (2023) Foundation Models for Generalist Geospatial Artificial Intelligence. arXiv preprint. http://arxiv.org/abs/2310.18660
77. Clay Foundation (2024) Clay Foundation Model. https://github.com/Clay-foundation/model
78. Cong Y et al (2023) SatMAE: Pre-training Transformers for Temporal and Multi-Spectral Satellite Imagery. arXiv preprint. http://arxiv.org/abs/2207.08051
79. Li Z, Ning H (2023) Autonomous GIS: the next-generation AI-powered GIS. Int J Digit Earth 16:4668–4686. https://doi.org/10.1080/17538947.2023.2278895
80. Huang C et al (2024) GeoAgent: To Empower LLMs using Geospatial Tools for Address Standardization. In: Findings of the Association for Computational Linguistics ACL 2024, pp 6048–6063. https://doi.org/10.18653/v1/2024.findings-acl.362

81. Roberts J, Lüddecke T, Das S, Han K, Albanie S (2023) GPT4GEO: How a Language Model Sees the World's Geography. arXiv preprint. http://arxiv.org/abs/2306.00020
82. Jain N, Kwiatkowski R, Ray B, Ramanathan MK, Kumar V (2025) On Mitigating Code LLM Hallucinations with API Documentation. In: 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp 237–248. https://doi.org/10.1109/ICSE-SEIP66354.2025.00027
83. Madaan A et al (2023) Self-Refine: Iterative Refinement with Self-Feedback. arXiv preprint. http://arxiv.org/abs/2303.17651
84. Chen X, Lin M, Schärli N, Zhou D (2023) Teaching Large Language Models to Self-Debug. arXiv preprint. http://arxiv.org/abs/2304.05128
85. Gou Z et al (2024) CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. arXiv preprint. http://arxiv.org/abs/2305.11738
86. Le H, Wang Y, Gotmare AD, Savarese S, Hoi SCH (2022) CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. arXiv preprint. http://arxiv.org/abs/2207.01780
87. Reimers N, Gurevych I (2019) Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp 3980–3990. https://doi.org/10.18653/v1/D19-1410
88. International Organization for Standardization (2004) Geographic information – Simple feature access – Part 1: Common architecture (ISO 19125-1:2004). https://www.iso.org/standard/40114.html

Figures

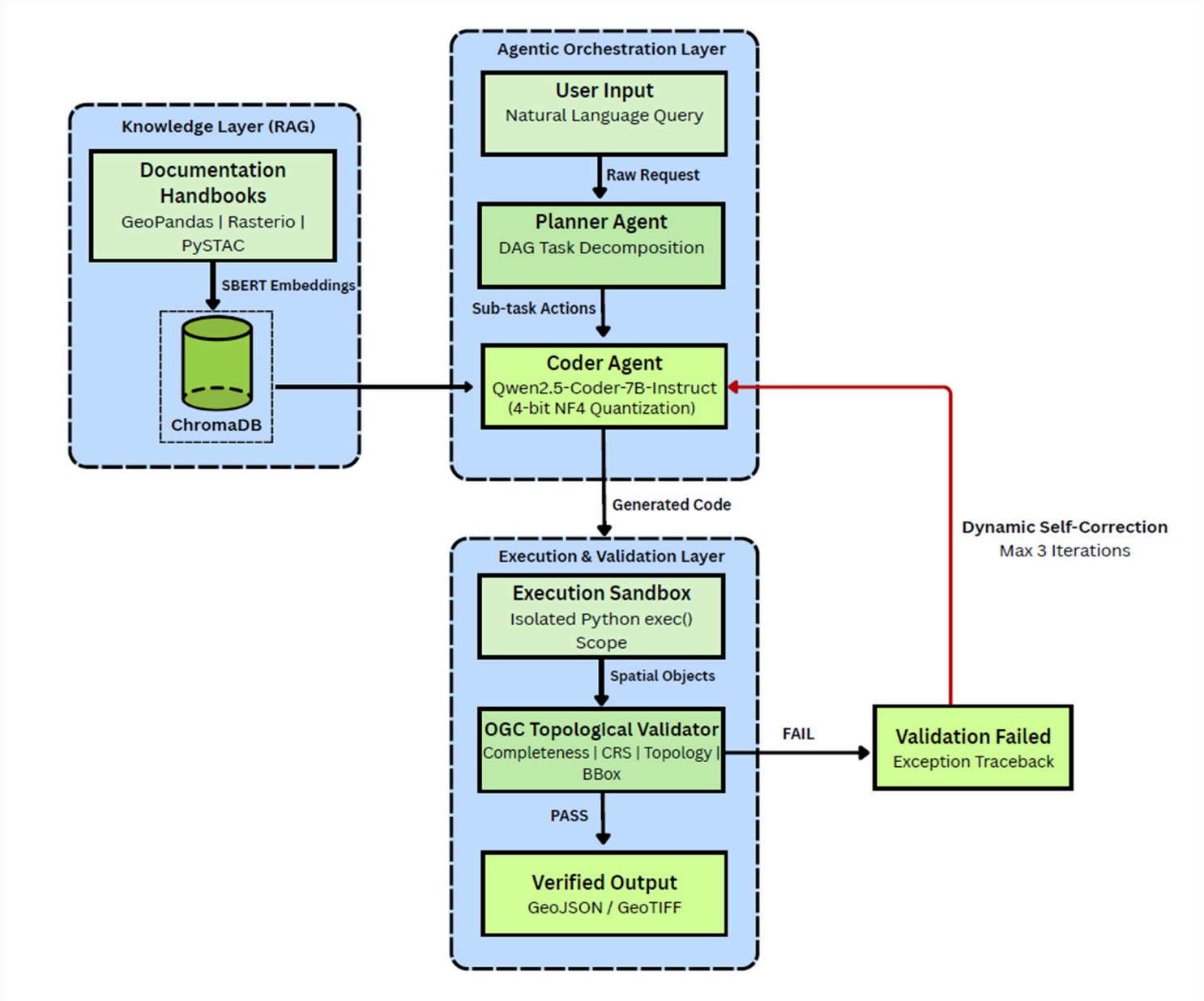


Figure 1

System architecture of the GeoMind framework. The diagram illustrates the closed-loop multi-agent pipeline: a user natural language query is parsed by the Planner Agent into a directed acyclic graph (DAG) of atomic sub-tasks. Each sub-task is grounded via Retrieval-Augmented Generation (RAG) from a ChromaDB vector store and transformed into executable Python code by the Coder Agent. Generated code is executed in an isolated sandbox, and outputs are validated by a deterministic Geospatial Validator enforcing topological integrity constraints. Failed validations trigger an autonomous self-correction loop with dynamic temperature scaling.

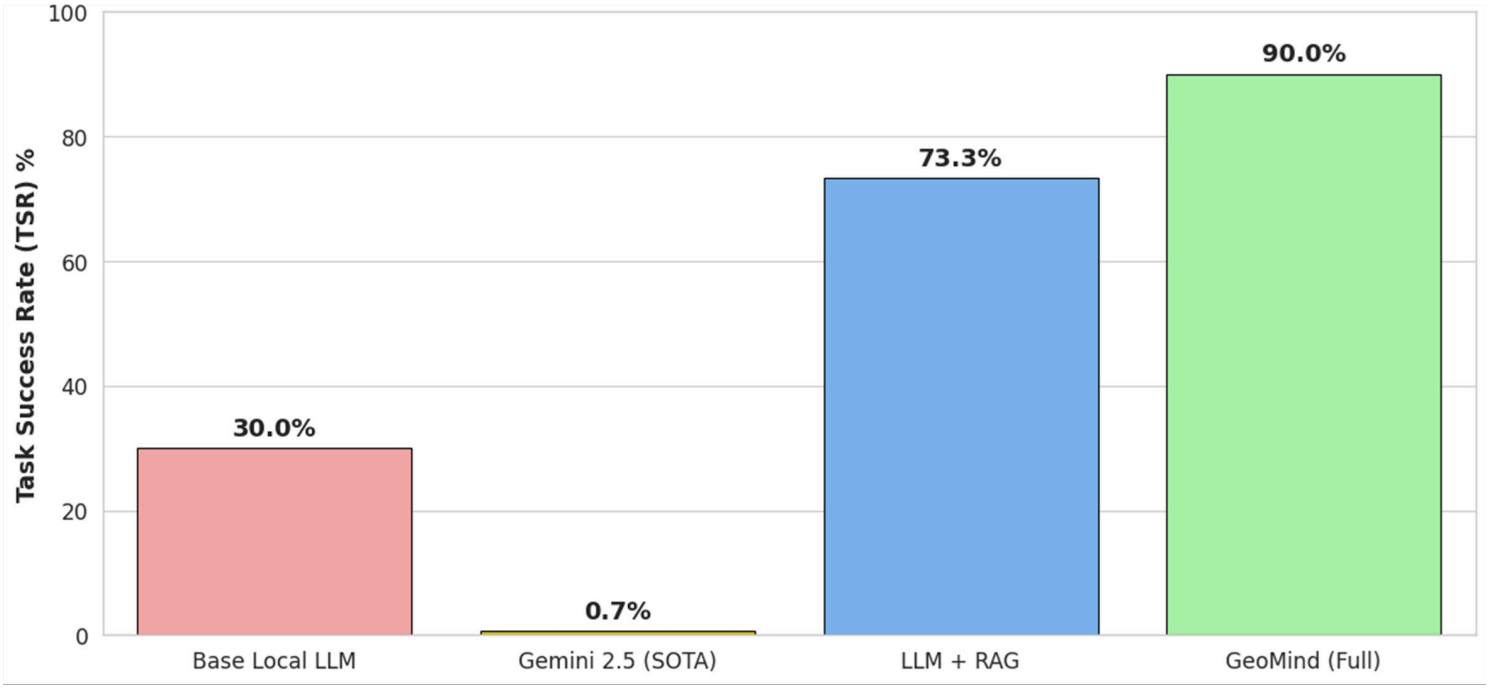


Figure 2

Architecture ablation study on GeoTaskBench (150 tasks). The bar chart illustrates the incremental contribution of each architecture to the overall Task Success Rate (TSR): the zero-shot local baseline LLM achieves 30.0%, to establish a state-of-the-art baseline, authors evaluated Gemini 2.5 Pro on GeoTaskBench under identical zero-shot conditions. Without access to the deterministic Geospatial Validator and local STAC endpoints, Gemini 2.5 failed to produce topologically valid geometries, yielding a 0.7% TSR. This highlights the absolute necessity of closed-loop spatial guardrails over raw parametric scale, the addition of Retrieval-Augmented Generation (RAG) to the local model raises TSR to 73.3%, and the full GeoMind framework achieves 90.0%.

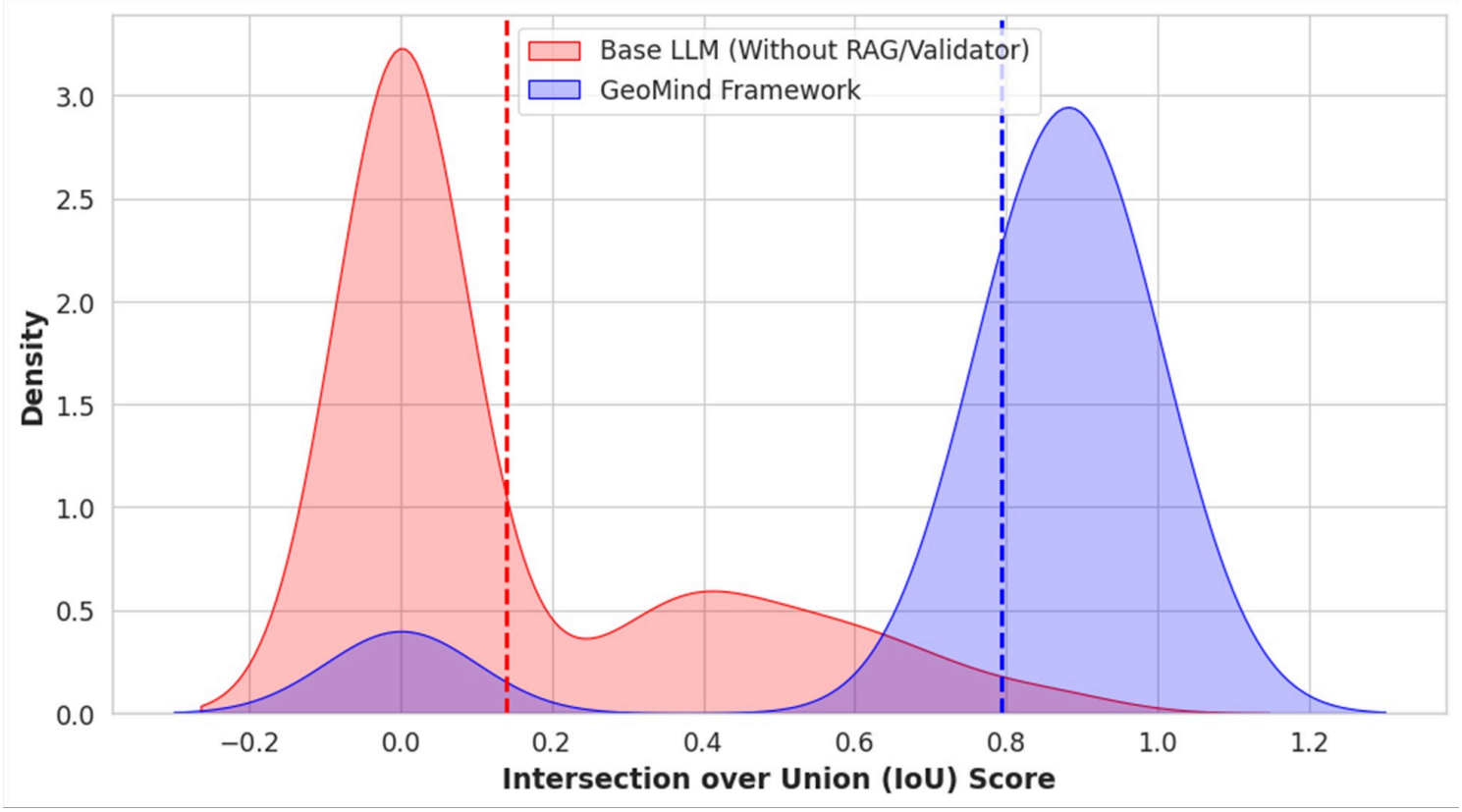


Figure 3

Kernel Density Estimation (KDE) of spatial Intersection over Union (IoU) scores across 150 GeoTaskBench tasks. The baseline open-loop LLM (red) exhibits an extreme density concentration near IoU = 0.0 (mean = 0.140), indicating pervasive geographic hallucination. The full GeoMind framework (blue) shifts the distribution toward IoU = 1.0 (mean = 0.793), demonstrating near-optimal spatial precision. Dashed vertical lines denote group means. The improvement is statistically significant (paired t-test, $p < 1.0e-10$).

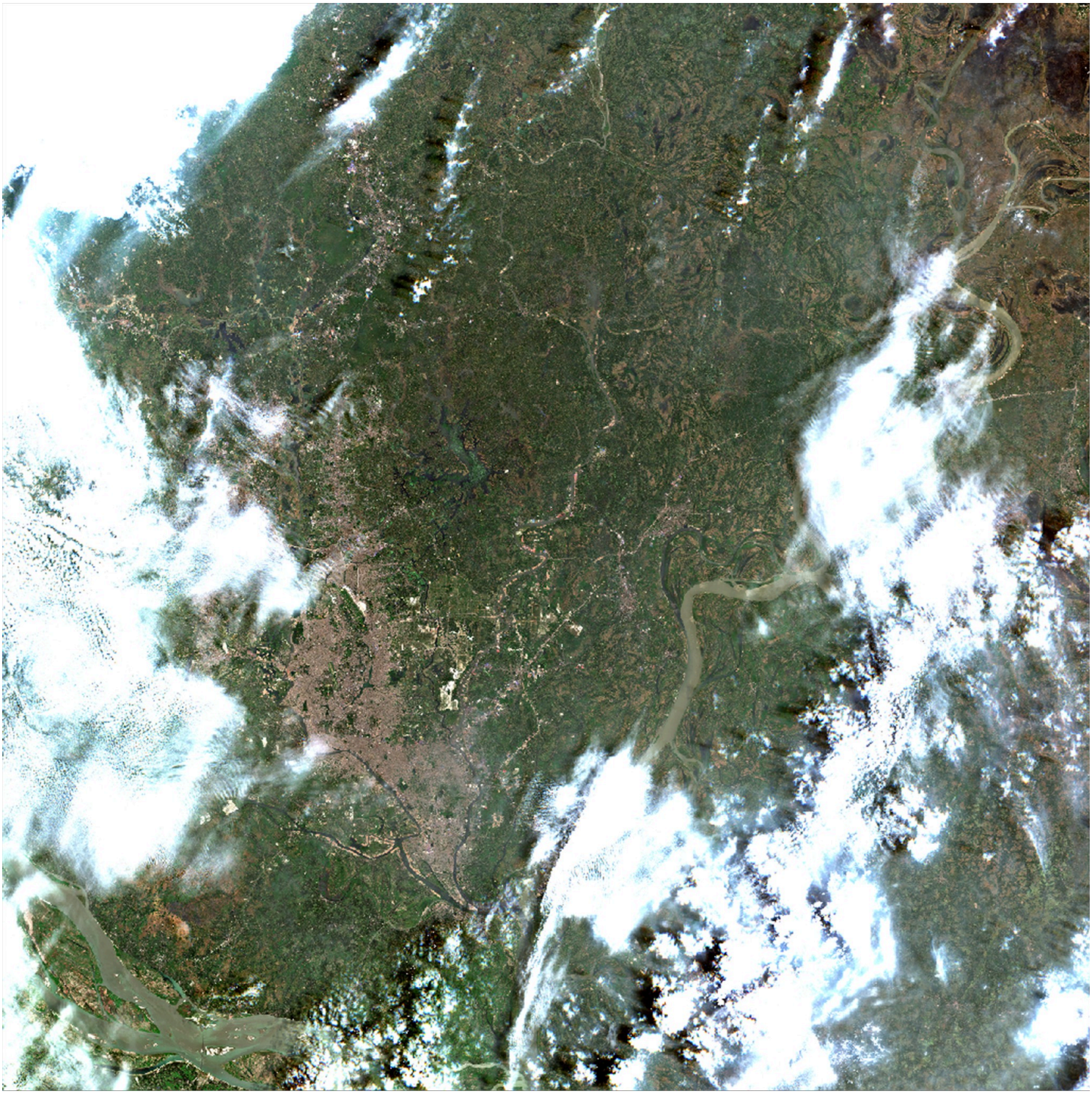


Figure 4

True-color Sentinel-2 L2A satellite image retrieved autonomously by the GeoMind framework via the Microsoft Planetary Computer STAC API. The image demonstrates successful end-to-end pipeline execution: STAC catalog authentication, spatio-temporal query construction, cloud-cover filtering (<10%), and band compositing (B4, B3, B2) all performed without human code intervention.

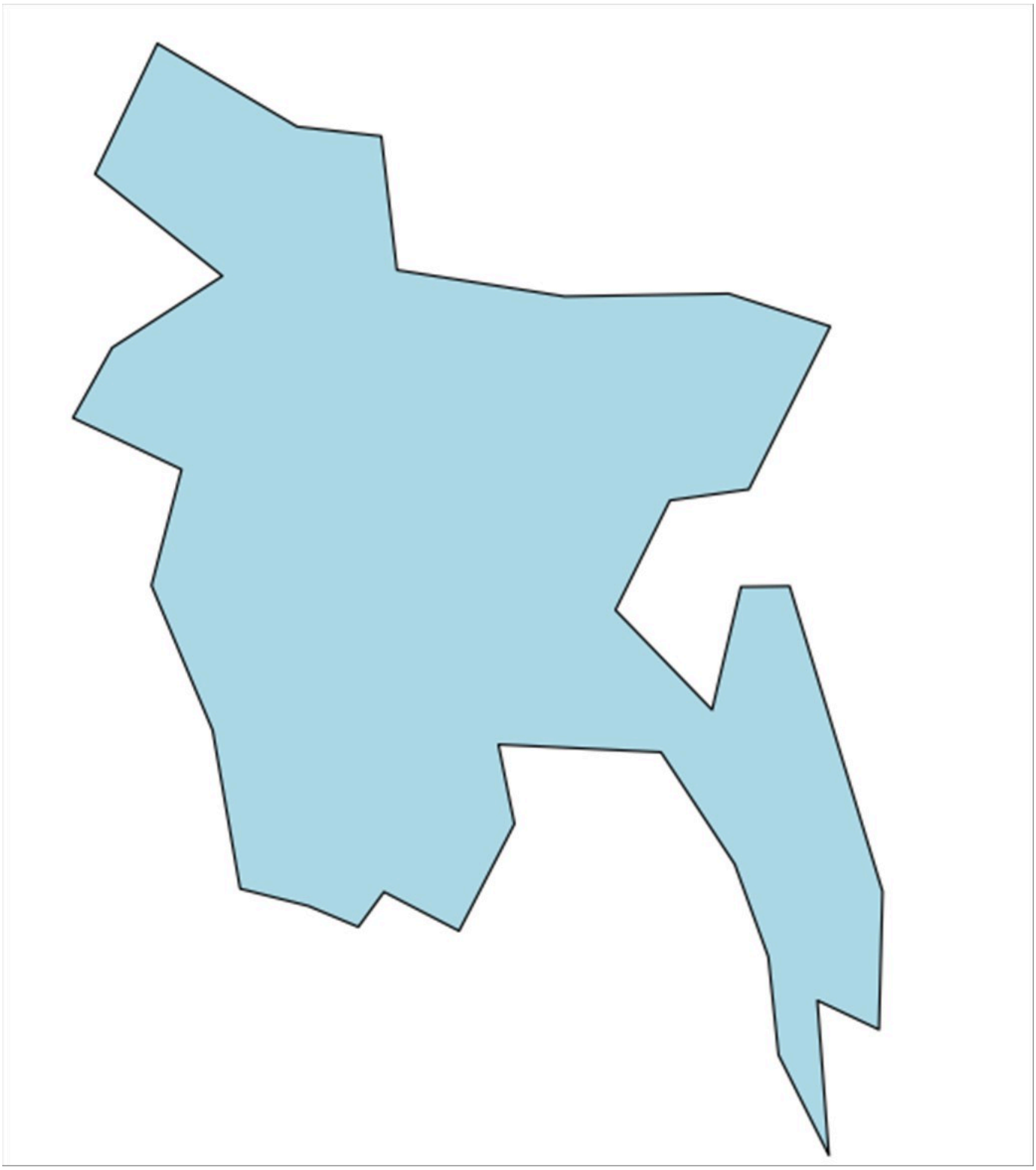


Figure 5

Autonomous geospatial output generated by the GeoMind framework. The Coder Agent autonomously wrote and executed a complete Python script to extract, filter, and plot the national boundary of Bangladesh from a Natural Earth shapefile, achieving an IoU of 0.94 against the ground-truth geometry. No human code was written for this output.