GeoVR: Structured Execution Rewards for Reinforcement Learning in Geospatial Code Generation

Anonymous authors
Paper under double-blind review

Abstract

Large language models are increasingly used to generate geospatial analysis code, yet executable programs can still be analytically wrong: an incorrect data source, temporal range, coordinate reference system, or workflow often fails silently. Existing reinforcement learning with verifiable rewards (RLVR) for code relies on unit tests or execution outcomes, which cannot detect such errors. We introduce GeoVR, an RLVR framework for geospatial code generation built on Google Earth Engine (GEE). Its core reward, Hierarchical Schema-grounded Verifiable Reward (HSVR), extracts type-aware signatures from GEE execution artifacts (object type, bands, CRS, spatiotemporal extent, and verifiable statistics) and checks required analytical operations in the program AST, yielding four deterministic components: Execution, Type, Schema, and Method. A hierarchical gate activates Schema and Method rewards only when the primary result passes task-level verification with the correct output type, preventing incorrect programs from accumulating reward through partially satisfied constraints. GeoVR is optimized with GRPO and leverages structured execution errors to repair failed rollouts. On AutoGEEval++, GeoVR achieves pass@1 of 90.22%, 88.76%, and 67.82% on Atomic, Combined, and Theme tasks, outperforming CodeGEEnius by up to 6.38 points and a controlled Binary-RLVR baseline by 3.73–10.35 points. These results demonstrate that execution artifacts themselves can serve as effective verification feedback for reinforcement learning, enabling models to move beyond generating merely "executable programs" toward generating "task-correct programs that satisfy domain constraints." Data and code are available at: `https://figshare.com/s/03091d54357e8a152e7a`.

1 Introduction

Geospatial analysis increasingly relies on programmatic workflows for dataset selection, spatial and temporal filtering, reprojection, band computation, statistical aggregation, and multi-source data fusion (Hou et al., 2024). Meanwhile, large language models (LLMs) have improved in code generation, task decomposition, and tool use, enabling geospatial workflows to be generated from natural-language instructions (Hou et al., 2025e; Chen et al., 2021; Yao et al., 2023), and cloud platforms such as Google Earth Engine (GEE) provide large geospatial datasets and scalable computation (Google Earth Engine; Gorelick et al., 2017). A correct GEE program, however, requires more than valid application programming interface (API) calls: it must also use the appropriate dataset, temporal range, coordinate reference system (CRS), output structure, and analytical procedure. Previous studies have improved GEE code generation through domain adaptation, retrieval augmentation, task decomposition, and tool-based planning (Hou et al., 2025c;f;e;b; Wu et al., 2025b; Hou et al., 2025a; Wu et al., 2025c; Hou et al., 2025d). These approaches mainly improve domain knowledge and generation-time planning but rarely use the execution results of generated programs to optimize the generation policy, and thus offer limited support for learning whether a program satisfies the underlying geospatial task.

Reinforcement learning with verifiable rewards (RLVR) incorporates execution results into model optimization (DeepSeek-AI, 2025; Lambert et al., 2024; Le et al., 2022). In conventional code generation, unit tests or explicit input–output pairs often provide direct correctness criteria (Le et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

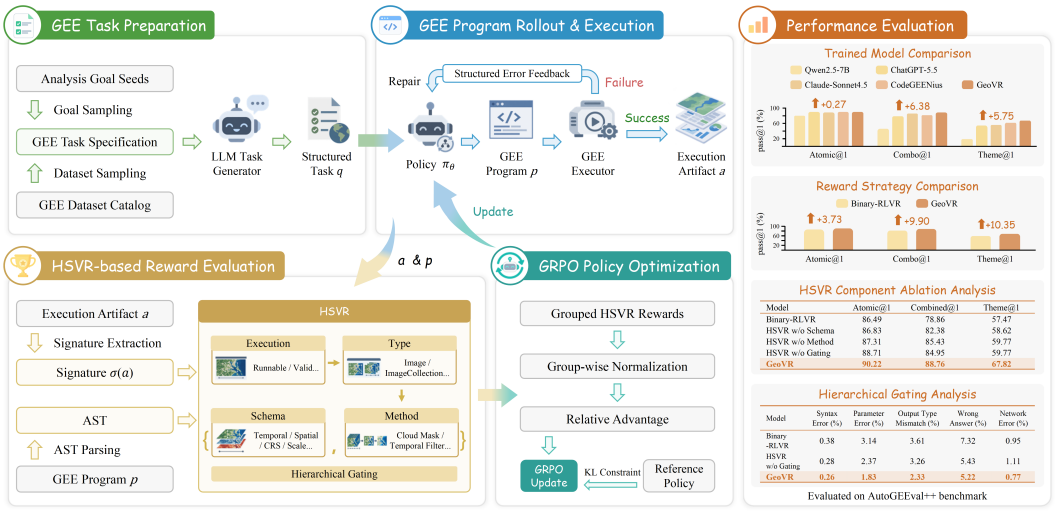

Figure 1: Overview of GeoVR: task construction, GEE rollout and execution with error-feedback repair, HSVR reward evaluation, and GRPO optimization. The right panel summarizes results on AutoGEEval++.

2022; Shojaee et al., 2023). Geospatial programs, however, generally require several constraints to be checked simultaneously, including data source, temporal range, CRS, band structure, spatial coverage, and analytical workflow (Hou et al., 2025d), and a program may execute successfully while violating one or more of them. The main problem we address is therefore how to convert observable information from GEE execution into verifiable rewards that reflect geospatial task requirements.

GEE execution artifacts contain structured information that supports this verification; depending on the output object, it may include object type, bands and data types, CRS, nominal resolution, spatial extent, temporal metadata, and property schema. Two programs with the same execution status may still produce different results because one uses an incorrect temporal range, CRS, band configuration, or spatial extent. Other requirements, such as cloud masking, reducer selection, or the order of key operations, cannot always be determined from the output object alone but can be checked using the program's abstract syntax tree (AST) and normalized operation sequence. Execution artifacts and program operations therefore provide complementary evidence for evaluating geospatial programs.

We propose GeoVR, an RLVR framework for geospatial code generation. Its reward function, Hierarchical Schema-grounded Verifiable Reward (HSVR), consists of four deterministic components: *Execution* checks whether the primary result satisfies the expected task outcome, *Type* checks whether the output has the required GEE object type, *Schema* verifies geospatial properties such as bands, CRS, spatiotemporal coverage, and property structure, and *Method* checks required analytical operations and known error patterns. HSVR uses hierarchical gating: Schema and Method rewards are available only when the primary result passes task-level verification and the output type is correct, preventing a program from receiving a high reward by satisfying isolated structural or procedural conditions while failing prerequisite task requirements. GeoVR initializes the policy from Qwen2.5-7B-Instruct (Qwen Team, 2024) after supervised fine-tuning (SFT) on GEE programs, optimizes it using Group Relative Policy Optimization (GRPO) (Shao et al., 2024), and, when a rollout fails during execution, uses the structured GEE error to generate a revised program for a bounded number of repair rounds. Figure 1 presents the overall framework of GeoVR. Our contributions are as follows:

1. **Structured execution verification for geospatial code.** We represent GEE output types, bands, spatial references, spatiotemporal properties, attribute structures, and required analytical operations as verifiable task constraints, extending RLVR supervision beyond binary result verification to output-type, geospatial-structure, and analytical-method constraints.

2. **HSVR and GeoVR.** We introduce HSVR, which combines Execution, Type, Schema, and Method rewards through hierarchical gating, and integrate HSVR, GRPO, and execution-feedback repair into GeoVR for reinforcement learning on geospatial code-generation tasks.
3. **Controlled evaluation.** We evaluate GeoVR on AutoGEEval++ against Binary-RLVR with the same SFT initialization, training data, GRPO configuration, and initial rollout budget, and use reward-component and multi-round feedback ablations to separately analyze the effects of structured rewards and execution-feedback repair.

2 Related Work

2.1 RLVR for Code Generation

RLVR provides post-training supervision through automatically verifiable execution outcomes, test results, or rule-based checks (DeepSeek-AI, 2025; Lambert et al., 2024). Reinforcement learning (RL) for code generation has used compilation results, unit tests, execution traces, partial test scores, and process-level rewards to improve functional correctness and error recovery (Le et al., 2022; Shojaee et al., 2023; Wang et al., 2026; Kumar et al., 2026; Li et al., 2025; Wu et al., 2025a; Gehring et al., 2024; Jain et al., 2025; Ekbote et al., 2025), showing that the effectiveness of code RL depends strongly on the task-relevant information exposed by the verifier.

GeoVR builds on execution-based verification but targets a failure mode common in geospatial programs: a program may run successfully while violating the requested data, temporal, spatial, structural, or methodological constraints. GeoVR therefore verifies the spatiotemporal properties, bands, projection, attribute schema, and required analytical operations of GEE outputs, and uses these signals directly in the RL reward rather than only as diagnostic information.

2.2 Geospatial Code Generation and GEE

Research on GEE code generation has examined domain-specific corpora, knowledge augmentation, task decomposition, agent-based planning, and automated evaluation. GEE-OPs builds a structured knowledge base of GEE operators and their relationships, and Geo-FuB organizes associations between geospatial operations and analytical functions (Hou et al., 2025c;f). GeoCode-GPT uses geospatial code corpora for domain adaptation (Hou et al., 2025e), and CodeGEEnius distills interpretable reasoning into a GEE-specific code model (Liu et al., 2026). Other approaches decompose complex geospatial analyses into smaller programmatic steps, while GeoColab and GeoCogent combine geospatial tool use with multi-step planning (Hou et al., 2025b; Wu et al., 2025b; Hou et al., 2025a). AutoGEEval and AutoGEEval++ provide automated evaluation across Atomic, Combined, and Theme task levels (Wu et al., 2025c; Hou et al., 2025d).

These studies primarily address domain knowledge, generation-time planning, tool coordination, and evaluation. GeoVR instead focuses on post-training, using information exposed by executing generated GEE programs to update the generation policy; it therefore complements rather than replaces SFT, retrieval-augmented generation (RAG), and agent-based planning.

3 Problem Formulation

Given a natural-language instruction q, a policy π_θ generates a GEE program p, which is executed in GEE. A program is considered runtime-successful if it passes parsing and static checks, completes within the resource limits, and returns a valid, non-empty execution artifact a; otherwise, the environment returns an execution error e. Runtime success determines whether an artifact is available and whether execution-feedback repair is triggered. It is distinct from the *Execution* component of HSVR, which further checks whether the primary result satisfies the expected outcome specified by the task (Section 4.3). For a runtime-successful program, GeoVR extracts a type-aware execution signature $\sigma(a)$, whose fields depend on the output object type (Section 4.2).

Each task is associated with a structured specification $s = \mathrm{Spec}(q)$ that defines the verifiable constraints required by the instruction. A reference program provides an executable anchor and a set of

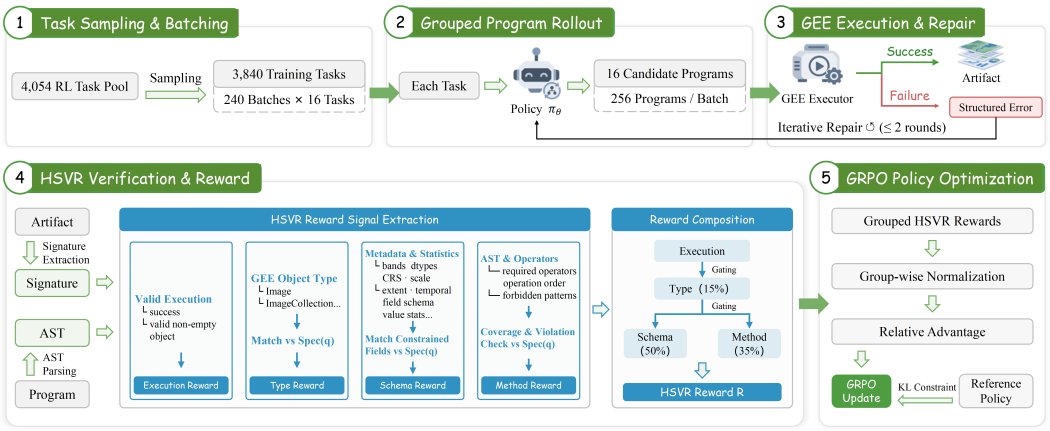

Figure 2: Training pipeline of GeoVR: task batching, grouped rollout (16 programs per task), GEE execution with up to two repair rounds, HSVR reward computation under hierarchical gating, and GRPO updates.

candidate structural properties, but it is not treated as the only valid implementation. Properties not required by the instruction are marked as unconstrained and do not contribute to the reward, so programs that follow different implementation paths receive the same verification result if they satisfy the same specification. GeoVR optimizes the expected HSVR reward while limiting deviation from a frozen reference policy π_{ref}:

$$\max_{\theta} \; \mathbb{E}_{q\sim\mathcal{D},\, p\sim\pi_\theta(\cdot|q)}\big[R_{\text{HSVR}}(p,s)\big] - \beta\, \mathbb{E}_{q\sim\mathcal{D}}\big[D_{\text{KL}}\big(\pi_\theta(\cdot \mid q)\,\|\,\pi_{\text{ref}}(\cdot \mid q)\big)\big], \quad s = \text{Spec}(q), \tag{1}$$

where $\mathcal{D}$ is the training-task distribution and $\beta = 0.04$ controls the strength of KL regularization in all experiments.

4 GeoVR

4.1 Framework Overview

GeoVR integrates generation, execution, and verification into policy optimization (Figure 2). The policy generates candidate GEE programs, which are executed in the GEE environment. For successful executions, GeoVR combines the execution signature with information from the program AST to compute HSVR; for failed executions, the environment returns a normalized error that is used for a bounded number of repair rounds. The resulting HSVR reward is used for GRPO optimization, without a separately trained reward model.

4.2 Structured Execution Artifacts and Task Specifications

HSVR uses two complementary sources of information. The execution signature $\sigma(a)$ describes what the program produces, while the AST and normalized operation sequence describe how the result is produced. Output structure alone cannot identify missing cloud masking, an incorrect reducer, or an invalid operation order, while program operations alone cannot establish whether the final spatial, temporal, and band properties satisfy the task.

The signature is type-aware. For image objects, candidate fields include the object type, band set and order, data types, CRS, nominal resolution, spatial extent, temporal metadata, and verifiable statistics. For feature objects, they include the property schema, geometry type, spatial extent, feature count, and task-specific statistics. Scalar and container objects are checked using their type, shape, keys, or values. A field contributes to the Schema reward only when it is constrained by the task specification and can be verified reliably.

The task specification s may define the required output type, dataset, region, temporal range, structural properties, analytical operations, and prohibited patterns. Partially specified or open-ended

tasks retain only the constraints explicitly required by the instruction. GeoVR therefore evaluates compliance with the task specification rather than similarity to a single reference program at the token or AST level.

4.3 HSVR: FOUR DETERMINISTIC REWARD COMPONENTS

HSVR consists of four deterministic components, Execution, Type, Schema, and Method, which evaluate the primary result, output-object type, geospatial structure, and analytical procedure, respectively. Each is computed from the GEE execution artifact or the generated program without a learned reward model.

Execution reward. Each candidate program first undergoes parsing and basic static checks. This syntax gate is an execution prerequisite and a low-level shaping signal rather than a separate HSVR component: programs that fail it receive zero reward, while programs that pass proceed to execution and result verification. The Execution reward is

$$r_{\text{exec}}(p, s) = \begin{cases} 1, & \text{if the primary result of } p \text{ satisfies the expected outcome in } s, \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

including when the program fails during execution or returns an empty result. Appendix A.4 defines the complete HSVR reward, including syntax-based shaping.

Type reward. Let $T(s)$ denote the set of output-object types allowed by the task specification, such as ee.Image, ee.ImageCollection, or ee.FeatureCollection. The Type reward is $r_{\text{type}}(p, s) = \mathbb{I}[\text{type}(a) \in T(s)]$, and is 0 if execution does not return a valid artifact. This prevents a candidate from receiving higher-level rewards when its primary value is correct but its output-object type is not.

Schema reward. Let $D(s)$ denote the set of verifiable schema dimensions activated by the task specification. The Schema reward is

$$r_{\text{schema}}(p, s) = \frac{\sum_{d \in D(s)} \alpha_d \, m_d(\sigma_d(a), s_d)}{\sum_{d \in D(s)} \alpha_d}, \tag{3}$$

where $\sigma_d(a)$ is the observed property of artifact a in dimension d, s_d is the corresponding target constraint, $m_d(\cdot) \in [0, 1]$ is a deterministic matching function, and $\alpha_d > 0$ is a fixed weight. When $D(s) = \varnothing$, $r_{\text{schema}}(p, s) = 1$, so that unspecified properties do not reduce the reward.

Method reward. A structurally correct output does not necessarily imply a correct analytical procedure. GeoVR therefore checks required operations and known invalid patterns using the program AST and normalized operation sequence:

$$r_{\text{method}}(p, s) = \tfrac{1}{2} \, \text{coverage}(O_{\text{req}}, p) + \tfrac{1}{2} \left[1 - \text{violation}(O_{\text{bad}}, p)\right], \tag{4}$$

where O_{req} is the set of required operations, O_{bad} is the set of prohibited operations or known invalid patterns, coverage is the proportion of required operations present in the program, and violation is the normalized violation rate. We set $\text{coverage} = 1$ when $O_{\text{req}} = \varnothing$ and $\text{violation} = 0$ when no applicable prohibited pattern is detected.

4.4 HIERARCHICAL GATING

An unconditional weighted sum would allow a candidate to accumulate reward from local properties even when prerequisite conditions are not satisfied. For example, an object of the wrong type may still contain some expected spatial properties, and a program with an incorrect primary result may still contain some required operations. HSVR therefore defines the gated core score

$$\widetilde{R}_{\text{HSVR}}(p, s) = r_{\text{exec}}(p, s) \, r_{\text{type}}(p, s) \left[w_t + w_s \, r_{\text{schema}}(p, s) + w_m \, r_{\text{method}}(p, s)\right], \tag{5}$$

where $w_t, w_s, w_m \geq 0$ are fixed weights and w_t is the base reward available once the Execution and Type conditions are satisfied. Schema and Method rewards become active only after both prerequisite checks pass, which prevents candidates with an incorrect primary result or output type from receiving reward solely through partially matched properties or operations. The complete reward $R_{\text{HSVR}}(p, s)$ additionally incorporates the syntax-based shaping term defined in Appendix A.4.

4.5 Execution-feedback repair and GRPO optimization

GEE execution failures often include localized error information, such as an undefined variable, an invalid argument, an inaccessible asset, an API or type mismatch, or a projection error. When a rollout fails during execution, the environment normalizes the returned error, and the policy generates a revised candidate conditioned on the task, the failed program, and the structured error. Repair stops when a program executes successfully or after $K = 2$ rounds. The final candidate is evaluated with the same R_{HSVR} as an unrepaired rollout; repair neither changes the HSVR definition nor adds a reward component.

GeoVR initializes its policy from Qwen2.5-7B-Instruct after GEE-domain SFT. For each task q, the policy samples a group of G candidate programs, each assigned the complete HSVR reward, and GRPO normalizes rewards within each group to obtain relative advantages.

5 Experimental Setup

Our experiments address four research questions:
RQ1: Does GeoVR improve performance on a standard geospatial programming benchmark?
RQ2: Does the complete GeoVR framework outperform a binary-reward baseline?
RQ3: How do the Schema, Method, and gating components of HSVR contribute?
RQ4: Does hierarchical gating reduce runnable but task-incorrect high-reward candidates?

5.1 Training setup and implementation

We initialize reinforcement learning from GeoVR-SFT, i.e., Qwen2.5-7B-Instruct after supervised fine-tuning on GEE programs. Binary-RLVR and GeoVR share this checkpoint, the training tasks, the GRPO configuration and group size, the initial rollout and token budgets, and the optimization hyperparameters, differing only in reward and feedback: Binary-RLVR uses a binary primary-result reward, while GeoVR uses HSVR with execution-feedback repair. Their comparison therefore measures the effect of the full GeoVR framework.

We build the RL tasks from about 600,000 real GEE scripts through static filtering, execution in GEE, asset normalization, structural deduplication, and decontamination against the AutoGEEval++ test set, yielding 4,054 tasks: 3,840 for training and 214 for validation, which are never used for policy updates. Appendix B.6 describes the SFT data and configuration, Appendix B the RL task construction and data split, Appendix C.2 the RL hyperparameters, and Appendix H the extra GEE requests and computational costs.

During training, we log policy loss, KL divergence, entropy, group reward variance, response length, truncation rate, static-check pass rate, runtime status, Execution Reward, Type Correctness, Schema Reward, Method Reward, and the full HSVR; for Binary-RLVR, the Type, Schema, Method, and HSVR values are diagnostics only and do not affect policy updates. For GeoVR and the execution-feedback ablations, we also report execution status before and after repair, error categories, and error-recovery rates, and compare maximum repair rounds of $K = 1$ and $K = 2$. The main text shows the dynamics most relevant to our findings; Appendix E.4 gives the complete training curves, and Appendix H covers computational resources and the software environment.

5.2 Evaluation protocol and metrics

We follow the official evaluation protocol of AutoGEEval++, a public automated benchmark for GEE code generation containing 5,564 test instances that cover 26 core GEE data types. It has three task levels: Atomic tasks evaluate individual API operations, Combined tasks require the composition of multiple operations, and Theme tasks evaluate complete geospatial analysis workflows.

AutoGEEval++ evaluates generated programs by executing them in GEE and comparing them with reference outcomes. We report pass@1, pass@3, and pass@5 for each task level, and use the official error logs to analyze five failure categories (Parameter Error, Output Type Mismatch, Wrong Answer, Syntax Error, and Network Error), comparing runtime validity, output-type consistency, and final-result correctness across training methods.

Table 1: Pass@1 on AutoGEEval++. Parentheses give the absolute gain of GeoVR over each baseline (percentage points).

Model	Atomic@1 (%)	Combined@1 (%)	Theme@1 (%)
Qwen2.5-7B-Instruct	80.37 (↑ 9.85)	45.24 (↑ 43.52)	18.39 (↑ 49.43)
ChatGPT-5.5	90.06 (↑ 0.16)	79.14 (↑ 9.62)	54.02 (↑ 13.80)
Claude-Sonnet4.5	88.75 (↑ 1.47)	86.38 (↑ 2.38)	56.32 (↑ 11.50)
CodeGEEnius	89.95 (↑ 0.27)	82.38 (↑ 6.38)	62.07 (↑ 5.75)
GeoVR	**90.22**	**88.76**	**67.82**

Table 2: Controlled comparison. The last row gives the absolute gain of GeoVR over Binary-RLVR (percentage points).

Model	Atomic@1 (%)	Combined@1 (%)	Theme@1 (%)
GeoVR-SFT	85.59	73.90	44.83
Binary-RLVR	86.49	78.86	57.47
GeoVR	**90.22**	**88.76**	**67.82**
Δ GeoVR − Binary-RLVR	+3.73	+9.90	+10.35

6 RESULTS

6.1 RQ1: PERFORMANCE ON AUTOGEEVAL++

We first evaluate GeoVR's overall geospatial code-generation performance on AutoGEEval++. The main text reports pass@1, the success rate with a single generation attempt; full pass@1, pass@3, and pass@5 results are in Appendix D.2. Table 1 compares GeoVR with representative general-purpose and GEE-specific code-generation models.

GeoVR achieves the highest pass@1 at all three task levels, improving over CodeGEEnius (Liu et al., 2026) by 0.27, 6.38, and 5.75 percentage points on Atomic, Combined, and Theme tasks, respectively. The gains are larger on Combined and Theme tasks, whereas Atomic performance is already high for the strongest baselines. This pattern is consistent with the motivation for structured verification: tasks involving multiple data, spatiotemporal, structural, and procedural constraints are less adequately characterized by executability alone. The differences across task levels do not by themselves establish a causal relationship between task complexity and the benefit of GeoVR; the controlled comparison and component ablations below provide more direct evidence.

6.2 RQ2: COMPARISON WITH BINARY-REWARD RLVR

We compare the full GeoVR framework with GeoVR-SFT and Binary-RLVR, all starting from the same initialization. Binary-RLVR and GeoVR share the training tasks, GRPO configuration, group size, and initial rollout and token budgets, differing only in that Binary-RLVR uses a binary primary-result reward while GeoVR uses HSVR with execution-feedback repair. Table 2 thus reflects the overall effect of the full framework; Section 6.3 analyzes the individual reward components, and Appendix E.5 the repair rounds.

GeoVR outperforms Binary-RLVR by 3.73, 9.90, and 10.35 percentage points on Atomic, Combined, and Theme tasks, respectively, whereas their Execution Rewards on the RL validation set differ less (0.910 vs. 0.945). The larger gains on Combined and Theme tasks suggest that the two methods differ beyond the primary-result reward. Because Table 2 compares complete frameworks, it does not isolate the contributions of HSVR and execution-feedback repair, which we examine through the component and repair-round ablations.

6.3 RQ3: CONTRIBUTION OF HSVR COMPONENTS

We construct three ablation variants by removing the Schema reward, Method reward, or hierarchical gating from GeoVR, denoted HSVR w/o Schema, HSVR w/o Method, and HSVR w/o Gating, with all other training conditions unchanged. To avoid evaluating a training reward using its own internal

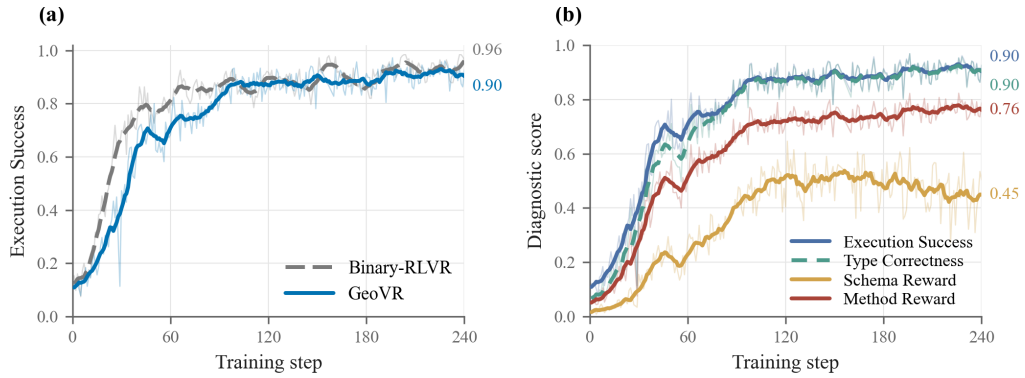

Figure 3: Training dynamics. (a) Execution Success of Binary-RLVR and GeoVR. (b) Execution Success, Type Correctness, Schema Reward, and Method Reward of GeoVR; Binary-RLVR curves are omitted because these values are diagnostics only. Thin lines: per-step values; thick lines: 10-step moving averages.

Table 3: Ablation of HSVR components. Parentheses give the absolute gain of GeoVR over each ablation (percentage points).

Model	Atomic@1 (%)	Combined@1 (%)	Theme@1 (%)
HSVR w/o Schema	86.83 (↑ 3.39)	82.38 (↑ 6.38)	58.62 (↑ 9.20)
HSVR w/o Method	87.31 (↑ 2.91)	85.43 (↑ 3.33)	59.77 (↑ 8.05)
HSVR w/o Gating	88.71 (↑ 1.51)	84.95 (↑ 3.81)	59.77 (↑ 8.05)
GeoVR	**90.22**	**88.76**	**67.82**

components, we use AutoGEEval++ pass@1 as the primary outcome; training curves and verifier-based diagnostics are reported in Appendix E.

The Schema ablation tests whether bands, CRS, spatiotemporal properties, and attribute structures provide useful signals beyond binary primary-result verification; the Method ablation tests whether output verification alone suffices to identify errors in the analytical procedure; and the Gating ablation tests whether Schema and Method rewards should depend on the prerequisite Execution and Type checks. As shown in Table 3, removing any of these components reduces pass@1 across all three task levels, with the largest reductions on Combined and Theme tasks.

6.4 RQ4: Effect of Hierarchical Gating

Hierarchical gating is intended to prevent candidates from accumulating reward through partially satisfied properties before prerequisite conditions are met. To reduce circularity, we do not use HSVR itself as the primary evidence; instead, we compare GeoVR and HSVR w/o Gating using the failure distribution in the official AutoGEEval++ error logs together with external pass@1 performance.

Output Type Mismatch and Wrong Answer most directly capture cases in which a program executes but fails to satisfy the requested output or task outcome. Syntax Error reflects lower-level program validity, Parameter Error primarily captures incorrect API usage, argument configuration, or structural constraints, and Network Error describes the benchmark execution state rather than a direct difference in model capability. Compared with HSVR w/o Gating, GeoVR reduces Output Type Mismatch from 3.26% to 2.33% (an absolute reduction of 0.93 percentage points and a relative reduction of 28.5%), Wrong Answer from 5.43% to 5.22% (0.21 points and 3.9%), and Parameter Error from 2.37% to 1.83% (0.54 points). Together with the external pass@1 results in Table 3, these changes support the conclusion that hierarchical gating reduces invalid partial-reward accumulation. Because this analysis relies on an automated benchmark rather than independent human evaluation, we do not interpret it as general evidence that reward hacking has been eliminated.

Table 4: AutoGEEval++ error-category rates (lower is better). Parentheses give the absolute reduction by GeoVR (percentage points). Network Error is not interpreted as a measure of model capability.

Error category (%) ↓	**Binary-RLVR**	**HSVR w/o Gating**	**GeoVR**
Syntax Error	0.38 (↓ 0.12)	0.28 (↓ 0.02)	**0.26**
Parameter Error	3.14 (↓ 1.31)	2.37 (↓ 0.54)	**1.83**
Output Type Mismatch	3.61 (↓ 1.28)	3.26 (↓ 0.93)	**2.33**
Wrong Answer	7.32 (↓ 2.10)	5.43 (↓ 0.21)	**5.22**
Network Error	0.95 (↓ 0.18)	1.11 (↓ 0.34)	**0.77**

7 Limitations and Conclusion

Task specification and verification scope. HSVR can verify only constraints that are explicitly encoded and reliably observable. Because open-ended geospatial analyses may admit multiple valid datasets, parameter settings, or methods, properties the instruction does not require must remain `unconstrained`; otherwise, the verifier may reject valid alternatives. Likewise, the Method reward relies on a curated set of required operations and invalid patterns; these rules are deterministic and interpretable but cannot cover every valid workflow, and overly strict rules may penalize functionally equivalent implementations.

Execution environment and evaluation scope. Because GEE is a remote cloud platform, execution-based training is subject to network latency, service quotas, and backend computation costs, and requires concurrent execution, request scheduling, and error handling to sustain practical throughput. Our evaluation relies mainly on the automated AutoGEEval++ benchmark and its error logs, which cannot replace expert assessment of open-ended, real-world workflows. Moreover, since the verifier, task specifications, and experiments are designed for GEE, applying GeoVR to other geospatial environments would require redefining the observable execution artifacts, task specifications, and method-verification rules. Finally, all experiments use a single random seed; multi-seed runs are needed to assess variance.

Conclusion. This work addresses a gap in reinforcement learning for geospatial code generation: a GEE program that executes successfully may still use the wrong dataset, temporal range, spatial reference, output structure, or analytical procedure. GeoVR turns structured information from GEE execution artifacts, together with the operations required in the generated program, into deterministic rewards, which HSVR organizes in an Execution $\rightarrow$ Type $\rightarrow$ $\{$Schema, Method$\}$ hierarchy; GeoVR then applies GRPO to continue optimizing the policy from GeoVR-SFT. On AutoGEEval++, GeoVR achieves Atomic, Combined, and Theme pass@1 scores of 90.22%, 88.76%, and 67.82%. The comparison with Binary-RLVR under the same initialization and initial rollout budget, together with the component ablations and official error-category analysis, helps separate gains in runtime validity from gains in geospatial task compliance. Within GEE code generation, execution artifacts thus provide verifiable information about data, spatiotemporal properties, output structure, and analytical procedure beyond whether a program runs.

AI use statement

Generative AI tools were used to assist with manuscript drafting, translation, language editing, structural organization, and LaTeX formatting. They were not used as benchmark evaluators, as sources of ground-truth annotations, or as substitutes for the deterministic verifiers described in this paper. The authors reviewed all AI-assisted text, code, equations, tables, and figures before inclusion. AI-assisted scripts were inspected and executed by the authors, and the reported numerical results were checked against the corresponding experimental outputs. The authors take responsibility for the research design, implementation, experiments, analysis, claims, and final content of this work.

Ethics statement

This study does not involve human participants or the collection of sensitive personal data. The experiments concern geospatial code generation and execution in Google Earth Engine. Inaccessible

private assets and scripts whose original task semantics could not be recovered were excluded during data construction. GEE execution was conducted subject to platform access controls, resource limits, and service policies.

Reproducibility statement

Appendices A–H document the information required to reproduce the method and experiments. They describe the type-aware execution signatures and matching functions, task-specification schema, SFT and RL data construction, AutoGEEval++ decontamination procedure, policy initialization, GRPO configuration and training budget, structured execution-error categories, prompt templates, training diagnostics, complete benchmark results, ablation studies, error analysis, and computational resources. The main evaluation protocol and controlled comparison settings are specified in Sections 5.1 and 5.2. All HSVR components are computed by deterministic verifiers, and GeoVR does not rely on an independently trained neural reward model.

References

Mark Chen et al. Evaluating large language models trained on code. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://arxiv.org/abs/2107.03374.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Chanakya Ekbote, Vijay Lingam, Behrooz Omidvar-Tehrani, Jun Huan, Sujay Sanghavi, Anoop Deoras, and Stefano Soatto. MURPHY: Multi-turn GRPO for self correcting code generation, 2025. URL https://arxiv.org/abs/2511.07833.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning, 2024. URL https://arxiv.org/abs/2410.02089.

Google Earth Engine. Google Earth Engine. https://earthengine.google.com/. Accessed: 2026-09-26.

Noel Gorelick, Matt Hancher, Mike Dixon, Simon Ilyushchenko, David Thau, and Rebecca Moore. Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, 202:18–27, 2017. doi: 10.1016/j.rse.2017.06.031.

Shuyang Hou, Zhangxiao Shen, Anqi Zhao, Jianyuan Liang, Zhipeng Gui, Xuefeng Guan, Rui Li, and Huayi Wu. Can large language models generate geospatial code? *Environmental Modelling & Software*, 176:106002, 2024. doi: 10.1016/j.envsoft.2024.106002.

Shuyang Hou, Haoyue Jiao, Jianyuan Liang, Zhangxiao Shen, Anqi Zhao, and Huayi Wu. GeoCogent: An LLM-based agent for geospatial code generation. *International Journal of Geographical Information Science*, 2025a. doi: 10.1080/13658816.2025.2549460.

Shuyang Hou, Haoyue Jiao, Zhangxiao Shen, Jianyuan Liang, Anqi Zhao, Xiaopu Zhang, Jianxun Wang, and Huayi Wu. Chain-of-programming (CoP): Empowering large language models for geospatial code generation task. *International Journal of Digital Earth*, 18(1):2509812, 2025b. doi: 10.1080/17538947.2025.2509812.

Shuyang Hou, Jianyuan Liang, Anqi Zhao, and Huayi Wu. GEE-OPs: An operator knowledge base for geospatial code generation on the Google Earth Engine platform powered by large language models. *Geo-spatial Information Science*, 2025c. doi: 10.1080/10095020.2025.2505556.

Shuyang Hou, Zhangxiao Shen, Huayi Wu, Haoyue Jiao, Ziqi Liu, Lutong Xie, Chang Liu, Jianyuan Liang, Yaxian Qing, Xiaopu Zhang, Dehua Peng, Zhipeng Gui, and Xuefeng Guan. AutoGEEval++: A comprehensive benchmark for geospatial code generation on Google Earth Engine. *International Journal of Digital Earth*, 18(1):2581425, 2025d. doi: 10.1080/17538947.2025.2581425.

Shuyang Hou, Zhangxiao Shen, Anqi Zhao, Jianyuan Liang, Zhipeng Gui, Xuefeng Guan, Rui Li, and Huayi Wu. GeoCode-GPT: A large language model for geospatial code generation. *International Journal of Applied Earth Observation and Geoinformation*, 138:104456, 2025e. doi: 10.1016/j.jag.2025.104456.

Shuyang Hou, Anqi Zhao, Jianyuan Liang, Zhangxiao Shen, and Huayi Wu. Geo-FuB: A method for constructing an operator-function knowledge base for geospatial code generation with large language models. *Knowledge-Based Systems*, 319:113624, 2025f. doi: 10.1016/j.knosys.2025.113624.

Arnav Kumar Jain, Gonzalo Gonzalez-Pumariega, Wayne Chen, Alexander M. Rush, Wenting Zhao, and Sanjiban Choudhury. Multi-turn code generation through single-step rewards. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025. URL https://openreview.net/forum?id=aJeLhLcsh0.

Abhijit Kumar, Natalya Kumar, and Shikhar Gupta. Execution-grounded credit assignment for GRPO in code generation, 2026. URL https://arxiv.org/abs/2603.16158.

Nathan Lambert et al. Tülu 3: Pushing frontiers in open language model post-training, 2024. URL https://arxiv.org/abs/2411.15124.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu-Hong Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 21314–21328, 2022. URL https://arxiv.org/abs/2207.01780.

Qingyao Li, Xinyi Dai, Xiangyang Li, Weinan Zhang, Yasheng Wang, Ruiming Tang, and Yong Yu. CodePRM: Execution feedback-enhanced process reward model for code generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 8169–8182, 2025. doi: 10.18653/v1/2025.findings-acl.428. URL https://aclanthology.org/2025.findings-acl.428/.

Ziqi Liu, Shuyang Hou, Guanyu Chen, Haoyue Jiao, Shaowen Wu, Lutong Xie, Wenping Yin, Yaxian Qing, Jianyuan Liang, Xuefeng Guan, et al. Codegeenius: an interpretable reasoning large language model for gee-based geospatial code generation via distillation. *Expert Systems with Applications*, pp. 134105, 2026.

Qwen Team. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K. Reddy. Execution-based code generation using deep reinforcement learning, 2023. URL https://arxiv.org/abs/2301.13816.

Longwen Wang, Yirui Liu, Xuan'er Wu, Xiaohui Hu, Yuankai Fan, Kaidong Yu, Qizhen Weng, Wei Xi, and Xuelong Li. Beyond binary: Turning partial success into dense verifiable rewards for reinforcement learning in code generation, 2026. URL https://arxiv.org/abs/2601.03525.

Haoze Wu, Yunzhi Yao, Wenhao Yu, and Ningyu Zhang. ReCode: Updating code API knowledge with reinforcement learning, 2025a. URL https://arxiv.org/abs/2506.20495.

Huayi Wu, Haoyue Jiao, Shuyang Hou, Jianyuan Liang, Zhangxiao Shen, Anqi Zhao, Yaxian Qing, Fengying Jin, Xuefeng Guan, and Zhipeng Gui. GeoColab: An LLM-based multi-agent collaborative framework for geospatial code generation. *International Journal of Digital Earth*, 18(2):2569405, 2025b. doi: 10.1080/17538947.2025.2569405.

Huayi Wu, Zhangxiao Shen, Shuyang Hou, Jianyuan Liang, Haoyue Jiao, Yaxian Qing, Xiaopu Zhang, Xu Li, Zhipeng Gui, Xuefeng Guan, and Longgang Xiang. AutoGEEval: A multimodal and automated evaluation framework for geospatial code generation on GEE with large language models. *ISPRS International Journal of Geo-Information*, 14(7):256, 2025c. doi: 10.3390/ijgi14070256.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Appendix

A HSVR AND EXECUTION-SIGNATURE DETAILS

A.1 TYPE-AWARE EXECUTION SIGNATURES

GeoVR does not require all GEE execution artifacts to share a single fixed schema. Instead, it extracts structural properties that are stable, programmatically accessible, and relevant to the task specification. For a valid execution artifact a, the type-aware execution signature is defined as

$$\sigma(a) = \mathrm{Sig}\left(\mathrm{type}(a), \mathrm{metadata}(a), \mathrm{statistics}(a)\right). \tag{6}$$

Only fields that are explicitly constrained by $\mathrm{Spec}(q)$ and can be verified reliably contribute to the Schema reward. Properties not required by the task are marked as unconstrained and are excluded from reward computation.

A.1.1 IMAGE OBJECTS

For ee.Image and applicable ee.ImageCollection outputs, the candidate signature is

$$\sigma_{\mathrm{image}}(a) = \{\mathrm{type}, \mathrm{bands}, \mathrm{dtypes}, \mathrm{crs}, \mathrm{scale}, \mathrm{extent}, \mathrm{temporal}, \mathrm{value_stats}\}. \tag{7}$$

The bands field records the band set and, when required, its order. The dtypes field records band data types. The crs and scale fields represent the spatial reference and nominal resolution, while extent represents spatial coverage. Temporal information is obtained from system properties or collection members. The value_stats field is enabled only when the task explicitly requires numerical-result verification.

A.1.2 FEATURE OBJECTS

For an ee.FeatureCollection, the candidate signature is

$$\sigma_{\mathrm{feature}}(a) = \{\mathrm{type}, \mathrm{field_schema}, \mathrm{geometry_type}, \mathrm{extent}, \mathrm{size}, \mathrm{value_stats}\}. \tag{8}$$

The property schema and geometry type are used to identify outputs that execute successfully but contain an incorrect attribute structure. Feature count and numerical statistics contribute to verification only when they are explicitly constrained by the task.

A.1.3 SCALAR AND CONTAINER OBJECTS

For ee.Number, ee.List, ee.Dictionary, and related objects, GeoVR verifies the object type, shape, key schema, and any values or statistics explicitly required by the task. For open-ended tasks that do not require an exact numerical result, the reference value is not treated as a hard constraint.

A.2 SCHEMA MATCHING FUNCTIONS

The Schema reward is computed from deterministic matching functions over the dimensions activated by the task specification. Let $D(s)$ denote the set of active dimensions:

$$r_{\mathrm{schema}}(p, s) = \frac{\sum_{d \in D(s)} \alpha_d m_d(\sigma_d(a), s_d)}{\sum_{d \in D(s)} \alpha_d}. \tag{9}$$

Here, $m_d(\sigma_d(a), s_d) \in [0, 1]$ is the matching function for dimension d, and $\alpha_d > 0$ is its fixed weight. The matching rule depends on the semantics of the corresponding field:

- **Set-valued fields**, such as bands and attribute fields, use exact matching, Jaccard similarity, or F1 score.

- **Temporal intervals** use interval overlap or endpoint tolerance.
- **Spatial extents** use spatial overlap or a task-specific coverage condition.
- **CRS, object type, and discrete data types** use deterministic exact matching after normalization.
- **Spatial resolution** uses a fixed absolute or proportional tolerance.
- **Numerical results** use absolute or relative error tolerance only when the task specification explicitly constrains the value.

The matching functions, tolerances, and dimension weights are fixed before training and remain unchanged across the controlled experiments. When $D(s) = \varnothing$, we define $r_{\text{schema}}(p, s) = 1$.

A.3 METHOD VERIFIER

The task specification contains a set of required operations O_{req} and a set of prohibited operations or known invalid patterns O_{bad}. GeoVR normalizes GEE API calls and the relevant operation order extracted from the program AST into operation categories. The Method reward is

$$r_{\text{method}}(p, s) = \frac{1}{2}\,\text{coverage}\,(O_{\text{req}}, p) + \frac{1}{2}\,[1 - \text{violation}\,(O_{\text{bad}}, p)]\,. \tag{10}$$

The verifier can check task-specific quality-control or cloud-masking operations, temporal filtering, reducer selection, required operation order, and clearly invalid projection handling. It evaluates only operations constrained by the task specification and does not impose a single workflow on all geospatial tasks. The rule set is fixed and shared by all controlled experiments. We define $\text{coverage} = 1$ when $O_{\text{req}} = \varnothing$ and $\text{violation} = 0$ when no applicable prohibited pattern is detected.

A.4 HIERARCHICAL GATING AND REWARD RANGE

Syntax and static checks are execution prerequisites rather than a fifth HSVR component. We define

$$g_{\text{syn}}(p) = \mathbb{I}\,[p \text{ passes parsing and static checks}]\,. \tag{11}$$

The four HSVR components follow the hierarchy Execution $\rightarrow$ Type $\rightarrow$ {Schema, Method}. The gated core score is

$$\widetilde{R}_{\text{HSVR}}(p, s) = r_{\text{exec}}(p, s) r_{\text{type}}(p, s)\,[w_t + w_s r_{\text{schema}}(p, s) + w_m r_{\text{method}}(p, s)]\,. \tag{12}$$

To distinguish programs that fail parsing or static checks from programs that pass these checks but fail Execution verification, the complete HSVR reward includes a small syntax-validity shaping term:

$$R_{\text{HSVR}}(p, s) = \begin{cases} 0, & g_{\text{syn}}(p) = 0, \\ \lambda_{\text{syn}}, & g_{\text{syn}}(p) = 1 \text{ and } r_{\text{exec}}(p, s) = 0, \\ \lambda_{\text{syn}} + (1 - \lambda_{\text{syn}})\,\widetilde{R}_{\text{HSVR}}(p, s), & r_{\text{exec}}(p, s) = 1. \end{cases} \tag{13}$$

Programs that fail parsing or static checks receive zero reward. Programs that pass these checks but fail Execution verification receive only λ_{syn}. Once the primary result passes Execution verification, the complete reward combines syntax validity with the gated core score. If the output type is incorrect, $\widetilde{R}_{\text{HSVR}}(p, s) = 0$, and the candidate receives only the syntax-validity reward.

We use non-negative component weights satisfying

$$w_t + w_s + w_m = 1. \tag{14}$$

Because each component lies in $[0, 1]$, both $\widetilde{R}_{\text{HSVR}}(p, s)$ and $R_{\text{HSVR}}(p, s)$ lie in $[0, 1]$. The final weights are reported in Table A1.

Table A1: HSVR reward weights and syntax-shaping coefficient.

Parameter	Description	Value
λ_{syn}	Syntax-validity shaping coefficient	0.05
w_t	Base reward after the Execution and Type gates	0.15
w_s	Schema reward weight	0.50
w_m	Method reward weight	0.35

Table B1: RL task construction and filtering statistics.

Stage	Number of samples
Raw GEE scripts	600,000
After AST and static filtering	124,682
Successful execution with a valid output	31,947
After asset normalization	18,364
After structural deduplication	5,286
After AutoGEEval++ decontamination	4,054
Final RL tasks	4,054
Training set	3,840
Validation set	214

B RL Environment and Data Construction

B.1 Raw scripts and filtering pipeline

The RL tasks are constructed from approximately 600,000 real GEE scripts. The objective of the filtering pipeline is not to maximize the number of retained scripts, but to obtain training instances that execute reliably, produce verifiable outputs, can be converted into natural-language tasks, and do not overlap with the external benchmark. Table B1 summarizes the filtering process.

B.2 Execution validation

Each script first undergoes AST parsing and basic static checks, followed by execution in a controlled GEE environment. A reference program is retained only if it executes reliably within the resource limits and returns a valid, non-empty, and recognizable GEE object. Retained programs then proceed to task and specification construction. The GEE execution timeout is set to 90 seconds, and execution uses 32 concurrent workers. Complete execution-cost statistics are reported in Appendix H.

B.3 Asset normalization

Real GEE scripts may refer to private assets, inaccessible identifiers, or deprecated asset paths. GeoVR checks the accessibility of each asset reference and normalizes it when an equivalent public asset can be identified. A task is removed if its original semantics cannot be recovered reliably from a private or inaccessible asset. We do not replace such assets with arbitrary public assets of the same type. A public-asset substitution is retained only when an equivalent mapping can be established and the resulting program passes execution validation again.

B.4 Deduplication and benchmark decontamination

Deduplication and benchmark decontamination are applied to both the SFT candidate data and the RL candidate tasks. Programs are first processed through variable renaming, formatting normalization, and AST canonicalization. Programs with identical canonicalized ASTs are treated as exact duplicates. For programs that are not structurally identical, we identify near-duplicate templates when the Jaccard similarity between their GEE operation sets exceeds 0.9 and their asset sets are identical.

Table B2: Deduplication and AutoGEEval++ decontamination rules for the SFT and RL data.

Stage	Compared samples and decision rule	Action
Exact structural deduplication	Within the SFT or RL candidate data, programs have identical canonicalized abstract syntax trees.	Retain one sample.
Near-template deduplication	Within the SFT or RL candidate data, operation-set Jaccard similarity exceeds 0.9 and the asset sets are identical.	Retain one sample.
Exact structural decontamination	A candidate program and an AutoGEEval++ test program have identical canonicalized abstract syntax trees.	Remove the candidate.
Operation-pattern decontamination	A candidate and a test program have an operation-set Jaccard similarity above 0.9 and identical asset sets.	Remove the candidate.
Task-semantic decontamination	A candidate instruction and a test task share the same analytical objective, dataset, region, and temporal range.	Remove the candidate.
Split-level validation	No direct or near-duplicate relation is found among the SFT data, RL training set, RL validation set, and benchmark test set under the preceding rules.	Retain the split; otherwise reconstruct it.

Decontamination against AutoGEEval++ checks canonicalized ASTs, GEE operation patterns, and task semantics. Both SFT instruction–program pairs and RL candidate tasks are compared with the benchmark test set. A candidate is removed when it has the same program structure as a test instance, has highly overlapping operations and identical assets, or fully overlaps in analytical objective, dataset, region, and temporal range. After cleaning, deduplication, and decontamination, the SFT dataset contains 173,057 instruction–program pairs. Structural deduplication retains 5,286 RL candidate tasks, and benchmark decontamination reduces this set to 4,054 tasks. We use 3,840 tasks for RL training and 214 for validation.

B.5 Instruction and task-specification construction

Each retained reference program provides an executable anchor. Its execution artifact supplies candidate structural properties, while the natural-language instruction determines which properties are required constraints. The task specification is represented as

$$s = \mathrm{Spec}(q) = \begin{cases} \text{output type},\ \text{data},\ \text{region},\ \text{time}, \\ \text{signature constraints},\ \text{required operators}, \\ \text{forbidden patterns} \end{cases}$$

The reference execution provides a candidate set of verifiable properties rather than a unique correct implementation. For partially specified or open-ended tasks, fields not required by the instruction are marked as `unconstrained` and excluded from the reward denominator. Different implementations can therefore receive the same verification outcome when they satisfy the same task specification.

B.6 SFT data and training setup

GeoVR-SFT and the RL tasks are derived from the same source corpus of real GEE scripts but use different filtering criteria. The SFT data are intended to learn the mapping between natural-language instructions and GEE programs. We remove empty programs, scripts that fail parsing, incomplete code, duplicates, inaccessible private assets, and samples that overlap with the AutoGEEval++ test set. The resulting SFT dataset contains 173,057 instruction–program pairs. RL task construction additionally requires successful execution, execution-signature extraction, and task-specification validation and therefore applies a stricter filtering pipeline.

Table B3: GeoVR-SFT training configuration.

Configuration	Value
Base model	Qwen2.5-7B-Instruct
Data source	Instruction–program pairs derived from approximately 600,000 GEE scripts
Total SFT samples	173,057
SFT training set	171,326
SFT validation set	1,731
Training method	Full-parameter fine-tuning
Training framework	LLaMA-Factory
Distributed configuration	DeepSpeed ZeRO-2
Numerical precision	bf16
Training epochs	2
Learning rate	1×10^{-5}
Learning-rate schedule	Cosine decay
Warmup ratio	0.1
Global batch size	128 (8 per GPU $\times$ 4 accumulation steps $\times$ 4 GPUs)
Training hardware	4$\times$ NVIDIA A800-SXM4-80GB
Training time	55,371 seconds (approximately 15.38 hours)

Table C1: Controlled comparison settings for Binary-RLVR and GeoVR. The two methods share the same initial rollout and token budgets; repair requests in GeoVR constitute additional execution cost and are reported separately in Appendix H.

Setting	Binary-RLVR	GeoVR
Initialization	GeoVR-SFT	GeoVR-SFT
RL optimizer	GRPO	GRPO
Training tasks	Same 3,840-task training set	Same 3,840-task training set
Reward	Binary outcome reward	HSVR, including syntax shaping
Initial rollout budget	Same	Same
Initial token budget	Same	Same
Reference policy	Frozen initialization checkpoint	Frozen initialization checkpoint
Execution-feedback repair	None	Up to two repair rounds

C Training and Implementation Details

C.1 Initialization and Controlled Comparison

All reinforcement learning (RL) methods are initialized from GeoVR-SFT, obtained by supervised fine-tuning Qwen2.5-7B-Instruct. Binary-RLVR and GeoVR use the same training tasks, Group Relative Policy Optimization (GRPO) configuration, group size, number of training steps, optimization hyperparameters, and initial rollout and token budgets. Binary-RLVR uses a binary outcome reward, whereas GeoVR uses the Hierarchical Schema-grounded Verifiable Reward (HSVR) together with execution-feedback-guided repair. Table C1 summarizes the controlled comparison.

C.2 Training Hyperparameters

Table C2 reports the GRPO and execution hyperparameters used in the experiments. With 3,840 training tasks and a batch size of 16 tasks, 240 optimization steps correspond to one pass over the RL training set.

C.3 Training Diagnostics

Binary-RLVR and GeoVR use the same logging structure. We record optimization diagnostics, including policy or GRPO loss, Kullback–Leibler (KL) divergence, entropy, learning rate, gradient norm, group reward mean and variance, and the clipping and ratio statistics available in the implementation. Generation diagnostics include response length, truncation rate, the proportion of invalid or empty outputs, and the proportion of unique programs within each group.

Table C2: GRPO training and execution hyperparameters.

Hyperparameter	Value
Learning rate	1×10^{-6}
Task batch size	16
GRPO group size G	16
KL coefficient β	0.04
Maximum response length	2,048 tokens
Training steps	240
Training epochs	1
Maximum repair rounds K	2
Random seed	42
GEE execution timeout	90 s

Table C3: Training diagnostics recorded during reinforcement learning.

Logged quantity	Purpose
Policy loss, KL divergence, and entropy	Monitor GRPO optimization stability and policy deviation.
Group reward mean and variance	Determine whether a rollout group contains informative relative reward differences.
Execution, Type, Schema, and Method	Identify the verification layer at which performance changes occur.
Response length and truncation rate	Detect length bias and incomplete generations caused by truncation.
Invalid, empty, and duplicate outputs	Monitor generation validity and within-group program diversity.
Error category and repair round	Analyze the behavior of execution-feedback-guided repair.
Pre- and post-repair execution status	Measure error recovery across repair rounds.

Verifier diagnostics include the static-check pass rate, Execution Success, Type Correctness, Schema Reward, Method Reward, and the overall HSVR. For GeoVR and the corresponding repair ablations, we additionally record execution-error categories, repair rounds, error-recovery rates, and execution status before and after repair.

For Binary-RLVR, Type Correctness, Schema Reward, Method Reward, and HSVR are computed only as training-time diagnostics and do not contribute to policy updates. These diagnostics are used to distinguish improvements in executability from improvements in geospatial constraint satisfaction. A monotonic decrease in policy loss alone is therefore not treated as sufficient evidence of correctness or convergence.

C.4 Execution-Feedback-Guided Repair

When a candidate program fails to execute, the environment maps the raw Google Earth Engine (GEE) error message to a fixed error category. The task instruction, failed program, and structured error feedback are then supplied to the policy as context for the next repair round:

$$p^{(t+1)} \sim \pi_\theta \left(\cdot \mid q,\, p^{(t)},\, e^{(t)} \right), \tag{15}$$

where t denotes the current repair round and $e^{(t)}$ is the structured feedback obtained from executing $p^{(t)}$. We set the maximum number of repair rounds to $K = 2$. Thus, after the initial execution, a failed candidate may undergo at most two additional generation-and-execution rounds. The procedure terminates once execution succeeds or the repair budget is exhausted.

The final candidate is evaluated using the same HSVR as an ordinary rollout. Structured error feedback guides regeneration but does not itself contribute an additional reward. The normalized error categories include syntax or parsing errors, undefined variables, API or type mismatches, invalid arguments, inaccessible assets or data, projection or scale errors, empty results, and resource-limit errors.

Table D1: Complete AutoGEEval++ results on Atomic tasks.

Model	Atomic@1 (%)	Atomic@3 (%)	Atomic@5 (%)
Qwen2.5-7B-Instruct	80.37	87.53	88.43
ChatGPT-5.5	90.06	91.46	91.76
Claude-Sonnet4.5	88.75	92.03	92.25
CodeGEEnius	89.95	94.10	94.15
GeoVR-SFT	85.59	86.81	86.99
Binary-RLVR	86.49	88.89	89.81
HSVR w/o Schema	86.83	89.16	90.31
HSVR w/o Method	87.31	89.56	90.11
HSVR w/o Gating	88.71	89.65	90.04
GeoVR	**90.22**	**94.96**	**95.17**

Table D2: Complete AutoGEEval++ results on Combined tasks.

Model	Combined@1 (%)	Combined@3 (%)	Combined@5 (%)
Qwen2.5-7B-Instruct	45.24	71.90	74.48
ChatGPT-5.5	79.14	85.52	86.67
Claude-Sonnet4.5	86.38	89.24	90.10
CodeGEEnius	82.38	87.90	89.24
GeoVR-SFT	73.90	79.43	81.05
Binary-RLVR	78.86	88.00	88.95
HSVR w/o Schema	82.38	88.10	89.14
HSVR w/o Method	85.43	88.76	89.81
HSVR w/o Gating	84.95	88.86	89.71
GeoVR	**88.76**	**93.81**	**94.67**

D AutoGEEval++ Evaluation Protocol and Complete Results

D.1 Official Evaluation Protocol

AutoGEEval++ is a public automated benchmark for Google Earth Engine (GEE) code generation. It contains 5,564 test cases covering 26 core GEE data types and organizes the tasks into three levels of complexity: Atomic, Combined, and Theme. Candidate programs are evaluated through actual GEE execution, output-type validation, and comparison with reference results.

Following the official protocol, we report pass@1, pass@3, and pass@5. Pass@k measures the proportion of tasks for which at least one correct program is found among the first k generated candidates. The main paper focuses on pass@1, which represents the strictest single-generation setting. Complete pass@1, pass@3, and pass@5 results are reported below.

D.2 Complete Pass@1, Pass@3, and Pass@5 Results

All models are evaluated using the same version of AutoGEEval++ and the same execution scripts. For each task, five candidate programs are generated independently with a temperature of 0.2 and a maximum response length of 2,048 tokens. Pass@1, pass@3, and pass@5 are computed from the first one, three, and five candidates, respectively. For external baselines, we retain only publicly reported results obtained under the same generation and execution protocol.

D.3 Official Error-Category Analysis

In addition to pass@k, we use the official AutoGEEval++ error logs to analyze the main failure modes. The reported categories are Parameter Error, Output Type Mismatch, Wrong Answer, Syntax Error, and Network Error. These categories describe failures observed by the external benchmark and are not treated as equivalent to the reward components used during GeoVR training.

Table D3: Complete AutoGEEval++ results on Theme tasks.

Model	Theme@1 (%)	Theme@3 (%)	Theme@5 (%)
Qwen2.5-7B-Instruct	18.39	27.59	31.03
ChatGPT-5.5	54.02	59.77	64.37
Claude-Sonnet4.5	56.32	63.22	68.97
CodeGEEnius	62.07	70.11	73.56
GeoVR-SFT	44.83	55.17	57.47
Binary-RLVR	57.47	64.37	70.11
HSVR w/o Schema	58.62	68.97	71.26
HSVR w/o Method	59.77	66.67	70.11
HSVR w/o Gating	59.77	65.52	72.41
GeoVR	**67.82**	**73.56**	**75.86**

Table D4: AutoGEEval++ error-category rates by task tier. Atomic, Combined, and Theme contain 4,427, 1,050, and 87 test cases, respectively. Total Error is complementary to pass@1. The lowest rate within each task tier is shown in bold.

Task	Error category (%) ↓	Binary-RLVR	HSVR w/o Gating	GeoVR
Atomic	Parameter	0.90	0.70	**0.63**
	Output Type	4.52	4.08	**2.93**
	Wrong Answer	6.63	**4.90**	5.04
	Syntax	0.38	0.31	**0.28**
	Network	1.08	1.30	**0.90**
	Total Error	13.51	11.29	**9.78**
Combined	Parameter	11.43	8.19	**5.90**
	Output Type	0.09	**0.00**	**0.00**
	Wrong Answer	9.33	6.57	**5.08**
	Syntax	**0.10**	0.19	0.19
	Network	0.19	0.10	**0.07**
	Total Error	21.14	15.05	**11.24**
Theme	Parameter	17.24	17.24	**13.79**
	Output Type	**0.00**	1.15	**0.00**
	Wrong Answer	18.39	18.39	**16.09**
	Syntax	3.45	**0.00**	**0.00**
	Network	3.45	3.45	**2.30**
	Total Error	42.53	40.23	**32.18**

E Controlled Experiments, Ablations, and Mechanistic Analysis

E.1 Overall Comparison between Binary-RLVR and GeoVR

Binary-RLVR and GeoVR share the same initialization, training tasks, Group Relative Policy Optimization (GRPO) configuration, group size, and initial rollout and token budgets. Binary-RLVR uses a binary outcome reward, whereas GeoVR uses the Hierarchical Schema-grounded Verifiable Reward (HSVR) and execution-feedback-guided repair. This comparison therefore evaluates the overall effect of the complete GeoVR framework relative to the binary-reward baseline. The contributions of individual reward components and execution-feedback-guided repair are examined separately through component and multi-round feedback ablations.

E.2 Reward-Component Ablation

All reward-component ablations use the same initialization, training data, training budget, and optimization hyperparameters, and differ only in their reward composition. The evaluated variants remove the Schema reward, Method reward, or hierarchical gating from HSVR. We use external AutoGEEval++ pass@1 as the primary evaluation metric, while verifier components and benchmark error categories are reported only to help interpret the observed changes.

Table E1: Overall comparison of GeoVR-SFT, Binary-RLVR, and GeoVR. Benchmark rows report AutoGEEval++ pass@1, whereas validation rows report mean verifier scores on the 214-task RL validation set. The two groups use different datasets and evaluation criteria and are not directly comparable.

Metric	GeoVR-SFT	Binary-RLVR	GeoVR
Atomic pass@1 (%)	85.59	86.49	**90.22**
Combined pass@1 (%)	73.90	78.86	**88.76**
Theme pass@1 (%)	44.83	57.47	**67.82**
Val. Execution Reward†	0.178	0.910	**0.945**
Val. Schema Reward†	0.022	0.111	**0.558**
Val. Method Reward†	0.109	0.583	**0.801**

† Mean verifier score computed on the 214-task RL validation set. These values are training-time diagnostics and are not part of the official AutoGEEval++ evaluation.

Table E2: Ablation of HSVR reward components. Wrong Answer and Output Type report micro-averaged AutoGEEval++ error rates over all three task tiers. Lower error rates are better.

Variant	Atomic@1 (%)	Combined@1 (%)	Theme@1 (%)	Wrong Answer (%) ↓	Output Type (%) ↓
Binary-RLVR	86.49	78.86	57.47	7.32	3.61
HSVR w/o Schema	86.83	82.38	58.62	5.68	4.72
HSVR w/o Method	87.31	85.43	59.77	5.75	3.39
HSVR w/o Gating	88.71	84.95	59.77	5.43	3.26
GeoVR	**90.22**	**88.76**	**67.82**	**5.22**	**2.33**

E.3 Hierarchical Gating Analysis

We analyze hierarchical gating through a controlled comparison between GeoVR and HSVR w/o Gating. The two models use the same initialization, training tasks, GRPO configuration, training budget, and execution-feedback repair setting. They differ only in whether hierarchical gating is enabled. GeoVR requires a candidate to pass the Execution and Type checks before receiving Schema and Method rewards, whereas HSVR w/o Gating removes these prerequisites.

As shown in Table E3, GeoVR improves pass@1 by 1.51, 3.81, and 8.05 percentage points on Atomic, Combined, and Theme, respectively. It also reduces Output Type Mismatch from 3.26% to 2.33% and the overall error rate from 12.45% to 10.41%. On Atomic tasks, however, the Wrong Answer rate increases slightly from 4.90% to 5.04%. Hierarchical gating therefore does not reduce every error category at every task level; its main benefit appears in the overall error distribution and on the more compositional Combined and Theme tasks. These results support the claim that hierarchical gating reduces invalid partial-reward accumulation, but they do not constitute a general proof that it eliminates reward hacking.

E.4 Training Dynamics

Figure 3 in the main paper compares Execution Success for Binary-RLVR and GeoVR and reports the verifier components observed during GeoVR training. This section provides additional results on optimization stability, within-group advantage ranges, and rollout outcomes. These training-time diagnostics are used to analyze the optimization process and do not replace the external AutoGEEval++ evaluation. Figure E1 shows that the gradient norm and policy loss of both methods remain bounded throughout training, without sustained divergence. In later training stages, GeoVR maintains moderately higher entropy and Kullback–Leibler divergence, indicating greater deviation from the reference policy and higher output uncertainty. For both methods, response length and truncation rate decrease during early training and subsequently stabilize. Figure E2 shows that the maximum within-group advantage of Binary-RLVR decreases as its binary reward becomes saturated. GeoVR retains a broader positive-advantage range in later training, indicating that the structured reward continues to distinguish candidates after execution success becomes common. For both methods, the minimum advantage gradually approaches its lower range, reflecting the contrast between a small

Table E3: Hierarchical gating ablation. Differences are computed as GeoVR minus HSVR w/o Gating. Error rates are micro-averaged over AutoGEEval++.

Metric	HSVR w/o Gating	GeoVR	GeoVR − HSVR w/o Gating
Atomic pass@1 (%) ↑	88.71	90.22	+1.51
Combined pass@1 (%) ↑	84.95	88.76	+3.81
Theme pass@1 (%) ↑	59.77	67.82	+8.05
Parameter Error (%) ↓	2.37	1.83	−0.54
Output Type Mismatch (%) ↓	3.26	2.33	−0.93
Wrong Answer (%) ↓	5.43	5.22	−0.21
Syntax Error (%) ↓	0.28	0.26	−0.02
Network Error (%) ↓	1.11	0.77	−0.34
Total Error (%) ↓	**12.45**	**10.41**	**−2.04**

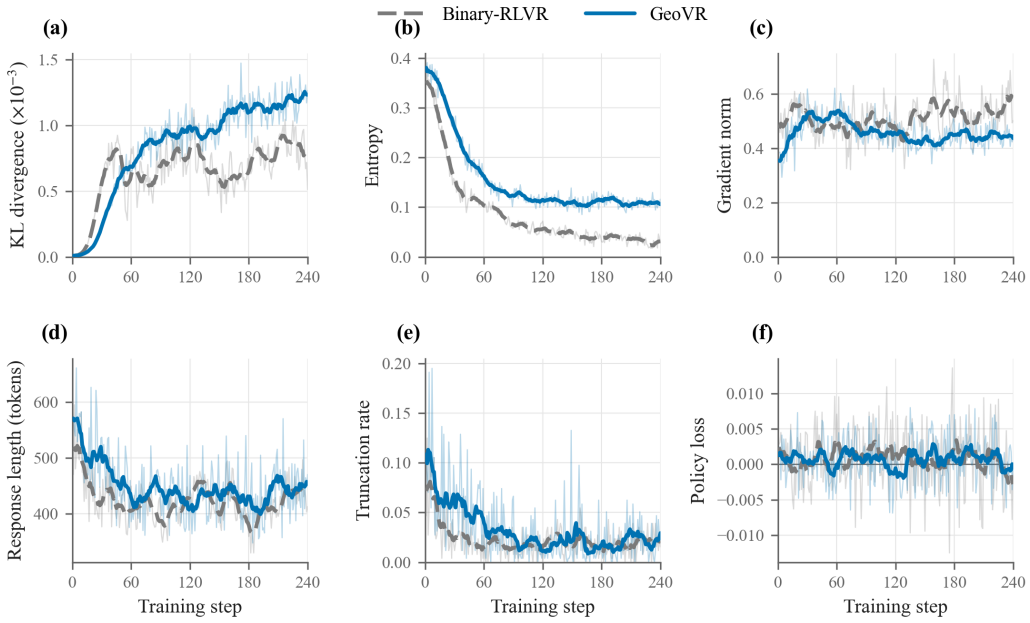

Figure E1: Optimization diagnostics for Binary-RLVR and GeoVR: (a) KL divergence, (b) entropy, (c) gradient norm, (d) response length, (e) truncation rate, and (f) policy loss. Light thin lines show per-step values, and dark thick lines show 10-step moving averages.

number of failed candidates and a majority of successful candidates within a group. Figure E3 shows that the proportion of successful GeoVR rollouts increases over training, while Syntax Error, API Call Error, and Empty Output become less frequent. Group reward variance rises during the intermediate stage and declines as the policy improves, but remains nonzero at the end of training. The static-check pass rate increases from approximately 0.4 at the beginning of training to above 0.9 and remains stable during later stages.

E.5 Multi-Round Feedback Ablation

We evaluate execution-feedback-guided repair by comparing a maximum of one repair round ($K = 1$) with the default setting of two repair rounds ($K = 2$). We report the initial execution-success rate, final execution-success rate, recovery rate among initially failed candidates, and final

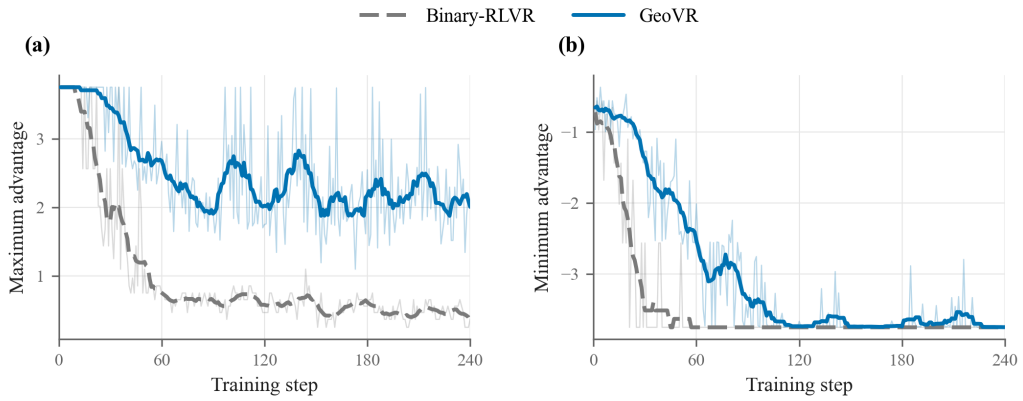

Figure E2: Within-group advantage ranges for Binary-RLVR and GeoVR: (a) the maximum advantage at each training step and (b) the minimum advantage at each training step. Light thin lines show per-step values, and dark thick lines show 10-step moving averages.

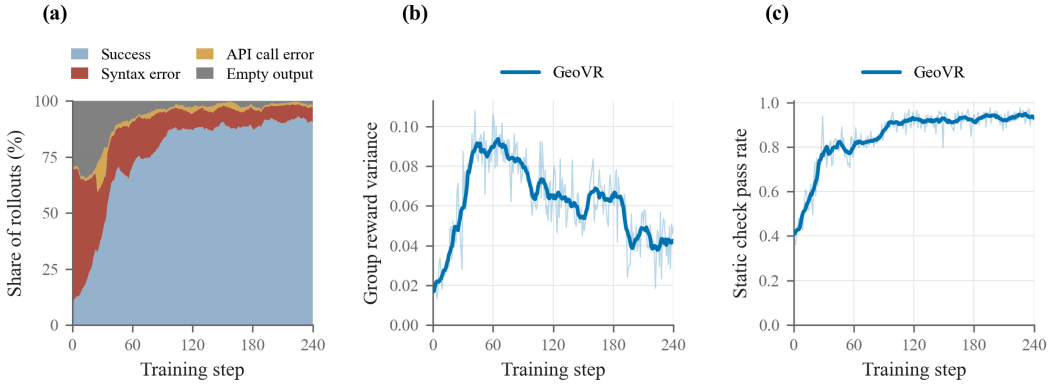

Figure E3: GeoVR training diagnostics: (a) the proportions of successful rollouts, Syntax Error, API Call Error, and Empty Output; (b) group reward variance; and (c) the static-check pass rate. Light thin lines show per-step values, and dark thick lines show 10-step moving averages.

AutoGEEval++ pass@1. This ablation examines whether additional feedback rounds improve recovery from explicit execution errors without changing the definition of HSVR.

F Task Specifications and Prompt Templates

F.1 Specification Schema

Each training task is associated with a structured specification, denoted by $\mathrm{Spec}(q)$. The specification uses the following fixed fields:

Table E4: Ablation of the maximum number of execution-feedback repair rounds. Recovery Rate is the proportion of initially failed candidates that execute successfully after repair.

Metric	$K = 1$	$K = 2$
Initial Execution (%)	83.30	**84.50**
Final Execution (%)	91.73	**94.54**
Recovery Rate (%)	50.48	**64.77**
Atomic pass@1 (%)	90.04	**90.22**
Combined pass@1 (%)	86.67	**88.76**
Theme pass@1 (%)	64.37	**67.82**

$$\mathrm{Spec}(q) = \begin{cases} \mathrm{task_type}, \\ \mathrm{allowed_output_types}, \\ \mathrm{dataset_constraints}, \\ \mathrm{region_constraints}, \\ \mathrm{time_constraints}, \\ \mathrm{expected_signature_constraints}, \\ \mathrm{required_operators}, \\ \mathrm{forbidden_patterns} \end{cases}$$

Each field may take the value *unconstrained*. An unconstrained field is excluded from the corresponding reward computation. This design prevents partially specified or open-ended tasks from being reduced to exact matching against a single reference implementation.

F.2 Instruction Construction

Natural-language instructions are constructed from the executable semantics of the reference program without introducing requirements that are absent from the original program or task. Reference execution provides a candidate set of verifiable properties, while the instruction determines which properties should be treated as hard constraints.

After construction, each task is re-executed and checked for consistency among the instruction, $\mathrm{Spec}(q)$, and the reference program. Manual inspection is additionally used to identify ambiguous constraints and cases in which the generated instruction does not faithfully describe the executable task.

F.3 Prompt Templates

Training rollouts and execution-feedback-guided repair use fixed prompt templates. The rollout prompt contains the system instruction, task description, output requirements, and generation constraints. The repair prompt additionally includes the failed program and the normalized error category returned by the execution environment. Figures F1 and F2 present the complete templates.

F.4 Specification Validation

Task specifications are constructed jointly from the natural-language instruction, the execution signature of the reference program, and its abstract syntax tree. Automatic validation checks consistency among the required output type, activated Schema dimensions, required analysis operations, and reference execution. Properties not explicitly required by the task instruction are marked as unconstrained and excluded from reward computation.

The automatic consistency checks achieve a pass rate of 96.8%. Failed cases primarily involve disagreement between the task instruction and reference execution, missing required fields, or method constraints that cannot be verified reliably. These cases are corrected or removed before inclusion in the training set.

GEE Program Rollout Prompt

You are a code generation agent specialized in Google Earth Engine (GEE) and geospatial analysis.

Your objective is to generate a complete and executable GEE JavaScript program that faithfully implements the geospatial task specified below.

Generation Requirements:

1. Interpret and implement the task according to its explicitly stated data, spatial, temporal, output, and analytical requirements.
2. Use standard Google Earth Engine JavaScript API calls to construct the geospatial processing workflow required by the task.
3. Ensure that the generated program is complete and executable in the GEE environment.
4. Do not introduce additional assumptions or task constraints that are not explicitly specified in the input instruction.
5. The generated program should directly produce the GEE output object required by the task.

Geospatial Task:

{task_instruction}

Output Requirements:

- Return only the complete GEE JavaScript program.
- Do not include Markdown code fences.
- Do not include natural-language explanations, analysis, or any other text outside the program.

Figure F1: Prompt template used to generate the initial Google Earth Engine program rollout.

We additionally inspect a random sample of 200 tasks. The inspection examines whether the instruction, $\mathrm{Spec}(q)$, and reference program describe the same task, and whether the encoded constraints might incorrectly exclude reasonable alternative implementations. Of the 200 inspected tasks, 189 are judged consistent, corresponding to an agreement rate of 94.5%. For ambiguous cases, we retain only constraints that are explicitly required by the instruction and can be verified reliably; all remaining properties are marked as unconstrained.

G Qualitative Cases

We present three qualitative cases showing that validation of the primary result alone may overlook errors in output type, geospatial structure, metadata, or the required analysis method. In this section, Primary Result indicates whether the main computed result of a candidate program agrees with the reference result. Type, Schema, and Method respectively verify the output object type, the structural and geospatial properties of the result, and the analysis procedure required by the task.

All three candidate programs execute successfully and pass Primary Result validation, but each violates an additional task constraint. A binary outcome reward would therefore assign a positive reward to all three candidates. HSVR provides a finer distinction between programs that produce the correct primary result and programs that satisfy the complete task specification.

G.1 Output Type Mismatch

The task requires the program to calculate the flooded area within a study region and return the result as an ee.FeatureCollection, enabling subsequent administrative aggregation, spatial joins, and tabular export. As shown in Figure G1, the Binary-RLVR candidate correctly computes floodArea but stores the result in an ee.Dictionary. Its flooded-area value agrees with the reference result, so it passes Primary Result validation. However, the returned object is not the ee.FeatureCollection required by the task and therefore fails Type validation.

Execution-Feedback Repair Prompt

You are a code generation agent specialized in Google Earth Engine (GEE) and geospatial analysis.

A previously generated GEE program failed during execution. Your objective is to revise the failed program according to the provided structured execution feedback while preserving the semantics and requirements of the original geospatial task.

Original Geospatial Task:
{task_instruction}

Previous Failed Program:
{failed_program}

Structured Execution Feedback:
Error Category: {error_category}

Repair Requirements:
1. Diagnose the execution failure using the provided error category and the previous program.
2. Identify the program component most likely responsible for the execution failure.
3. Modify only the parts necessary to resolve the execution error, while preserving valid components of the previous program whenever appropriate.
4. Preserve all data, spatial, temporal, output, and analytical requirements explicitly specified in the original task.
5. Do not alter the original task objective merely to make the program executable.
6. Do not introduce additional requirements that are absent from the original task.
7. The revised program must remain a complete GEE JavaScript program and should directly produce the output required by the original task.

Output Requirements:
- Return only the complete revised GEE JavaScript program.
- Do not include Markdown code fences.
- Do not include debugging explanations, failure analyses, or any other text outside the program.

Figure F2: Prompt template used for execution-feedback-guided program repair. The failed program and normalized execution error are included as additional context.

Table G1: Qualitative cases and their HSVR verification outcomes. Pass and Fail indicate whether the candidate satisfies the corresponding verification dimension. A dash indicates that the dimension is not examined in that case.

Case	Compared model	Primary Result	Type	Schema	Method
G.1 Output type mismatch	Binary-RLVR	Pass	Fail	—	—
G.2 Schema and temporal-metadata error	Binary-RLVR	Pass	Pass	Fail	—
G.3 Required analysis method missing	HSVR w/o Method	Pass	Pass	Pass	Fail

GeoVR retains the same area calculation but stores the flooded-area value as an attribute of an ee.Feature and returns the feature within an ee.FeatureCollection. The two programs produce the same primary area value, but only the GeoVR output can be used directly in the required downstream vector-processing and table-export operations.

Binary-RLVR output

```
var areaImage = ee.Image.pixelArea()
  .updateMask(floodMask)
  .rename('flood_area_m2');

var floodArea = ee.Number(
  areaImage.reduceRegion({
    reducer: ee.Reducer.sum(),
    geometry: region,
    scale: 10,
    maxPixels: 1e13
  }).get('flood_area_m2')
);

var result = ee.Dictionary({
  flood_area_m2: floodArea
});

var outputType = ee.Algorithms.ObjectType(result);
var typeCorrect = ee.String(outputType)
  .compareTo('FeatureCollection')
  .eq(0);
```

Primary Result CORRECT | Type Check INCORRECT

GeoVR output

```
var areaImage = ee.Image.pixelArea()
  .updateMask(floodMask)
  .rename('flood_area_m2');

var floodArea = ee.Number(
  areaImage.reduceRegion({
    reducer: ee.Reducer.sum(),
    geometry: region,
    scale: 10,
    maxPixels: 1e13
  }).get('flood_area_m2')
);

var result = ee.FeatureCollection([
  ee.Feature(region, {
    flood_area_m2: floodArea
  })
]);

var outputType = ee.Algorithms.ObjectType(result);
var typeCorrect = ee.String(outputType)
  .compareTo('FeatureCollection')
  .eq(0);
```

Primary Result CORRECT | Type Check CORRECT

Figure G1: Output type mismatch. The Binary-RLVR candidate computes the correct flooded area but returns an ee.Dictionary, whereas GeoVR returns the required ee.FeatureCollection. The case shows that validating the primary numerical result does not ensure that the output object type satisfies the task specification.

G.2 Schema and Temporal-Metadata Error

The task requires administrative-region-level flood statistics to be returned as an ee.FeatureCollection containing the fields region_id, event_date, flood_area_m2, and unit. The value of event_date must correspond to the acquisition date of the post-event image.

As shown in Figure G2, the Binary-RLVR candidate returns a valid ee.FeatureCollection whose flood_area_m2 value agrees with the reference result. It therefore passes both Primary Result and Type validation. However, the returned feature omits the region_id and unit fields and assigns the pre-event acquisition date to event_date.

The figure also includes executable checks for the required fields and temporal metadata. The propertyNames() operation verifies the presence of region_id and unit, while direct date comparison determines whether event_date agrees with the post-event acquisition date. Both conditions are evaluated from the returned object rather than inferred from the source code. GeoVR returns the same flooded-area value while preserving the administrative-region identifier, post-event acquisition date, and area unit, and therefore passes Schema validation.

G.3 Required Analysis Method Missing

The task requires the program to filter a Sentinel-2 image collection by location, time, and cloud cover, compute a median composite, and then derive the Normalized Difference Water Index (NDWI). Median compositing is required to reduce the influence of residual clouds, shadows, and local outliers in individual images.

For this particular case, the filtered collection contains only one valid image. Selecting the first image and computing the median composite therefore produce identical pixel values. A binary reward based only on the final NDWI result cannot distinguish between the two implementations.

As shown in Figure G3, the HSVR w/o Method candidate uses collection.first() to select the image. The program executes successfully and returns an NDWI image with the correct object type, band name, and pixel values. It consequently passes Primary Result, Type, and Schema validation. Nevertheless, it does not perform the median compositing operation explicitly required by the task. GeoVR instead applies collection.median() before calculating NDWI and therefore satisfies the result, type, schema, and method constraints.

Binary-RLVR output

```
var result = ee.FeatureCollection([
  ee.Feature(region, {
    flood_area_m2: floodArea,
    event_date: '2021-07-01'
  })
]);

var feature = ee.Feature(result.first());
var properties = feature.propertyNames();

var hasRegionId = properties.contains('region_id');
var hasUnit = properties.contains('unit');
var dateCorrect = ee.String(feature.get('event_date'))
  .compareTo('2021-07-20')
  .eq(0);
```

Primary Result CORRECT | Type Check CORRECT | Schema Check INCORRECT

GeoVR output

```
var result = ee.FeatureCollection([
  ee.Feature(region, {
    region_id: region.get('region_id'),
    event_date: '2021-07-20',
    flood_area_m2: floodArea,
    unit: 'm2'
  })
]);

var feature = ee.Feature(result.first());
var properties = feature.propertyNames();

var hasRegionId = properties.contains('region_id');
var hasUnit = properties.contains('unit');
var dateCorrect = ee.String(feature.get('event_date')
  .compareTo('2021-07-20')
  .eq(0);
```

Primary Result CORRECT | Type Check CORRECT | Schema Check CORRECT

Figure G2: Schema and temporal-metadata error. The Binary-RLVR candidate returns the correct flooded area and output object type, but omits required fields and associates the result with the pre-event date. GeoVR satisfies the Primary Result, Type, and Schema constraints. The field-presence and date-consistency checks shown in the figure can be executed directly on the returned object.

HSVR w/o Method output

```
var collection = s2
  .filterBounds(region)
  .filterDate(startDate, endDate)
  .filter(ee.Filter.lt(
    'CLOUDY_PIXEL_PERCENTAGE', 20
  ))
  .map(maskS2Clouds);

var composite = ee.Image(collection.first());

var result = composite
  .normalizedDifference(['B3', 'B8'])
  .rename('NDWI');

var singleImageCase = collection.size().eq(1);
```

Primary Result CORRECT | Type Check CORRECT | Schema Check CORRECT | Method Check INCORRECT

GeoVR output

```
var collection = s2
  .filterBounds(region)
  .filterDate(startDate, endDate)
  .filter(ee.Filter.lt(
    'CLOUDY_PIXEL_PERCENTAGE', 20
  ))
  .map(maskS2Clouds);

var composite = collection.median();

var result = composite
  .normalizedDifference(['B3', 'B8'])
  .rename('NDWI');

var singleImageCase = collection.size().eq(1);
```

Primary Result CORRECT | Type Check CORRECT | Schema Check CORRECT | Method Check CORRECT

Figure G3: Missing required analysis method. When the filtered collection contains only one valid image, first() and median() produce the same result for the evaluated instance. However, first() does not implement the median compositing procedure required by the task. Method validation identifies this difference even though the final output values are identical.

H Computational Cost and Efficiency

H.1 Training Resources

Table H1 summarizes the hardware, numerical precision, training framework, and execution concurrency used in our experiments.

Table H1: Training resources and software environment.

Item	Value
GPU	4× NVIDIA A800-SXM4-80GB
CPU	Dual-socket Intel Xeon Platinum 8358, 64 cores and 128 threads in total
System memory	1.0 TiB RAM
Numerical precision	bfloat16
RL framework	verl 0.5.0
Parallel GEE workers	32

H.2 Real Execution and Feedback-Repair Cost

The Execution, Type, and Schema rewards in GeoVR are computed from actual Google Earth Engine (GEE) execution results, while the Method reward is obtained through deterministic checks of the program abstract syntax tree and operation sequence. Training runs for 240 steps, with 16 tasks per step and 16 candidate programs sampled for each task. This produces 256 initial rollouts per step, 61,440 initial rollouts over the complete training run, and the same number of initial GEE requests.

The complete GeoVR configuration applies up to two repair rounds to candidates that fail the initial execution. The $K = 2$ setting has an Initial Execution rate of 84.5%, corresponding to approximately 9,523 candidates entering the first repair round. Using the 50.48% recovery rate observed in the $K = 1$ experiment, approximately 4,716 candidates proceed to the second repair round. Under this accounting, the repair stage contains approximately 14,239 GEE requests, resulting in approximately 75,679 requests over the complete training run.

The GEE execution stage takes an average of 195 seconds for each initial batch of 256 rollouts, using 32 parallel workers. Table H2 summarizes the execution workload.

Table H2: GEE execution cost of GeoVR training.

Metric	Value
Training steps	240
Initial rollouts per step	256
Total initial rollouts	61,440
Initial GEE requests	61,440
Initial Execution rate	84.5%
First-round repair requests	9,523
Second-round repair requests	4,716
Total repair requests	14,239
Total GEE requests	75,679
Requests relative to Binary-RLVR	1.23×
Maximum repair rounds	$K = 2$
Mean GEE time per initial batch	195 s
Parallel GEE workers	32

H.3 Additional Cost of GeoVR

The additional computation introduced by GeoVR consists of execution-signature extraction, abstract-syntax-tree and operation-sequence checks, and execution-feedback-guided repair. Local structure and method verification takes an average of 1.23 ms per final rollout candidate, with a median of 1.07 ms and a 95th percentile of 2.31 ms. Across the 61,440 rollout trajectories in the complete training run, local verification requires approximately 76 seconds in total.

Binary-RLVR and GeoVR each contain 61,440 initial rollouts. Binary-RLVR produces 61,440 GEE requests, whereas GeoVR introduces 14,239 repair requests, increasing the total to 75,679 requests, or 1.23× the request volume of Binary-RLVR. GeoVR does not introduce a separately trained neural reward model. Its complete training run takes 43.69 hours and reaches a peak GPU memory usage of 77.76 GB.

Table H3: Training cost of Binary-RLVR and GeoVR. Both methods use the same number of training steps and initial rollouts; the additional GEE requests made by GeoVR arise from execution-feedback-guided repair.

Metric	**Binary-RLVR**	**GeoVR**
Training steps	240	240
Initial rollouts	61,440	61,440
Initial GEE requests	61,440	61,440
Repair-stage GEE requests	0	14,239
Total GEE requests	61,440	75,679
Relative GEE request volume	$1.00\times$	$1.23\times$
Complete training time	35.47 h	43.69 h
Peak GPU memory	78.50 GB	77.76 GB