TerraGuard: Dual-Task Security Alignment for LLM-Based Terraform Review and Generation

Abstract

LLM-based Infrastructure as Code (IaC) generation is typically evaluated by functional correctness, but deployable Terraform code can still contain severe security misconfigurations. This creates a function-first, security-deficient bias: models may satisfy syntax, planning, and intent checks while violating common cloud security policies. We propose **TerraGuard**, a dual-task security alignment framework for secure Terraform generation and code review. TerraGuard constructs 9,581 verified secure intent-code pairs and 13,200 vulnerable samples, then trains a Qwen2.5-14B model with joint SFT and multi-task GRPO using task-specific rewards. On IaC-Eval, we evaluate 20 mainstream LLMs and find that all baselines achieve CR below 34%, and code review ability does not naturally translate into secure generation. TerraGuard achieves 36.8% CR and the lowest Avg H+ of 1.74 while maintaining 45.2% Pass@1, and outperforms most 70B-class open-source models in code review. Code and Data will be available at `https://anonymous.4open.science/r/TerraGuard`.

1 Introduction

Large language models (LLMs) have rapidly advanced code understanding and generation, and are increasingly being applied beyond general-purpose languages to domain-specific software engineering tasks (Li et al., 2024; Zhu et al., 2025; Xiao et al., 2025; Xia et al., 2025; Jiang et al., 2025; Christopoulou et al., 2024). One representative domain is Infrastructure as Code (IaC), where natural-language requirements are translated into executable cloud configurations. In particular, Terraform (Kasarla, 2025) has become a dominant IaC language due to its declarative abstraction and broad industrial adoption, making it a practical testbed for studying whether LLMs can generate not only usable but also secure infrastructure code.

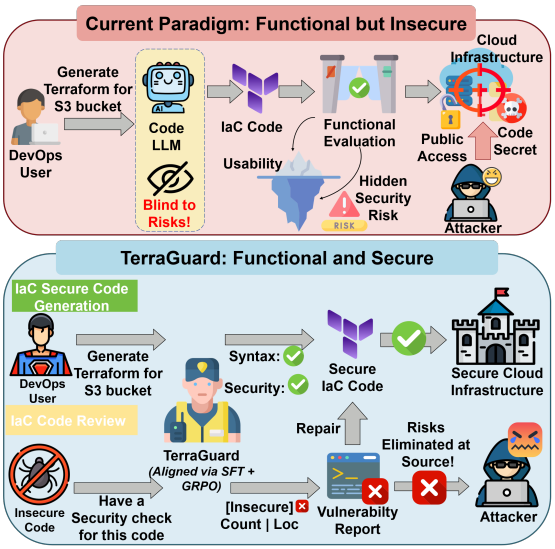

Figure 1: Existing IaC generation focuses solely on functional correctness, leaving hidden security risks. TerraGuard ensures both functional and secure IaC code via aligned generation and review.

Despite the rapid development of text-to-IaC models and benchmarks (Kon et al., 2024; Zhang et al., 2025), current research still centers on functional correctness, asking whether generated configurations can pass validation, planning, or deployment. In practice, however, IaC failures are often not functional failures but security misconfigurations: a Terraform program may deploy successfully while exposing public network access, embedding secrets, or granting excessive permissions. As a result, function-first training and evaluation can systematically favor deployable yet unsafe code.

Making IaC generation secure is difficult for two reasons. First, IaC security is not a surface-level syntax problem; it requires reasoning about resource semantics, permission boundaries, network exposure, and cross-resource dependencies. Second, the target objective is inherently hierarchical: generated code must satisfy syntax, deployability, intent, and security in sequence. Existing ap-

proaches do not handle this structure well. Post-hoc scanners such as Tfsec provide scalable and continuously updated security rules, but are typically used only as filters after generation rather than as training signals. Single-task SFT (Li et al., 2025) cannot reliably couple vulnerability understanding with secure generation, while generic SFT+RL pipelines are not tailored to IaC's staged verifier feedback. As a result, current methods produce configurations that are executable but still insecure.

To address these challenges, we propose TerraGuard, a dual-task security alignment framework for IaC that jointly models secure generation and code review through joint SFT and multi-task GRPO with task-specific rewards (Shao et al., 2024). In SFT, we mix both tasks with explicit instruction boundaries so that joint training yields complementary gains across secure generation and code review. In RL (Jha et al., 2025), we continue the optimization of mixed-tasks with a shared policy while routing each sample to the reward function associated with its task prefix. The review task emphasizes safety judgment and vulnerability counting, whereas the generation task follows staged verifier feedback aligned with syntax, deployability, intent, and security.

Our contributions are summarized as follows:

- **Large-Scale Empirical Analysis:** We systematically evaluate 20 mainstream LLMs on IaC-Eval, revealing two critical findings: (i) all models exhibit a systematic function-first, security-deficient bias; (ii) code review and secure generation capabilities are decoupled in existing LLMs.
- **Dual-Task Security Alignment Framework:** We propose TerraGuard, a framework that jointly optimizes secure generation and code review via joint SFT and multi-task GRPO. We design task-specific reward functions tailored to IaC's hierarchical constraints.
- **High-Quality Terraform Security Dataset:** We construct an automated data pipeline that generates 9,581 secure positive samples and 13,200 vulnerable negative samples, covering diverse types of AWS resources and vulnerability patterns.

2 Related Work

LLM for IaC Generation. Early research focused on code representation and understanding techniques, while recent advances in code generation models trained on massive code corpora have demonstrated strong program synthesis capabilities. IaC-Eval (Kon et al., 2024) established a functionality-oriented evaluation paradigm centered on deployability and intent satisfaction; DP-IAC-Eval (Zhang et al., 2025) further strengthened deployment-oriented evaluation. MACOG (Khan et al., 2025) addressed deployment dependency issues among IaC resources through multi-agent collaboration. ARPaCCino (Romeo et al., 2025) investigates external policies to improve IaC performance. However, the core evaluation metrics of existing studies still focus on functionality (Davidson et al., 2025), resulting in configurations that can be successfully deployed yet often conceal severe security and compliance risks.

Secure Code Generation. Security research in general code generation has become relatively mature (Yang et al., 2024; Hajipour et al., 2023a), with mainstream benchmarks based on the CWE taxonomy (Hajipour et al., 2023b) to evaluate vulnerability generation and detection in conventional programming languages. Existing security alignment methods, including SFT and RLHF (Shi et al., 2026), have shown effectiveness in general-purpose code tasks. However, IaC vulnerabilities are fundamentally configuration errors rather than logical flaws, making these methods difficult to transfer directly. In addition, existing benchmarks fail to capture scenarios where configurations are functionally deployable yet insecure.

IaC Security Research. In industry, static analysis tools such as `Tfsec` and `Checkov` provide widely used security rules for Terraform and are commonly adopted in CI/CD pipelines (Mei and Fu, 2026). GLITCH (Saavedra and Ferreira, 2023) mainly studies IaC vulnerability detection rather than secure generation. In most existing workflows, scanner feedback is used only after generation as a reactive defense, rather than as a training signal for security-aware generation. The closest study, GenSIaC (Li et al., 2025), has two major limitations: (i) limited language coverage, emphasizing server-side configuration tools such as Ansible and Chef while overlooking Terraform-style multi-cloud orchestration languages; and (ii) methodological simplicity, relying only on single-stage SFT, which struggles to balance executability and security effectively.

3 Preliminary

This section defines the notation, tasks, and metrics used for **IaC Secure Code Generation** and **IaC Code Review**.

3.1 Notation and Basic Definitions

Let $\mathcal{X}$ be the intent space, where each $x \in \mathcal{X}$ describes infrastructure requirements, and let $\mathcal{Y}$ be the Terraform code space, where each $y \in \mathcal{Y}$ is a valid HCL configuration. We use $\mathcal{Z}$ for code review outputs, including security judgments and vulnerability statistics. The generation and review models are denoted by $p_\theta(y|x)$ and $q_\theta(z|y)$, respectively.

We use four predicate functions to evaluate Terraform code:

$$V_{\text{syn}}(y) = \mathbb{I}\left[\texttt{terraform validate}(y) \text{ passes}\right], \tag{1}$$

$$V_{\text{dep}}(y) = \mathbb{I}\left[\texttt{terraform plan}(y) \text{ passes}\right], \tag{2}$$

$$V_{\text{int}}(x, y) = \mathbb{I}\left[y \text{ satisfies the intent } x\right], \tag{3}$$

$$V_{\text{sec}}(y) = \mathbb{I}\left[y \text{ passes Tfsec security scan}\right], \tag{4}$$

where $\mathbb{I}[\cdot]$ is the indicator function. In data construction and RL training, V_{int} is approximated by a committee-based LLM evaluator; in IaC-Eval, it is determined by the benchmark-provided rego intent script.

3.2 IaC Code Review

The **IaC code review** task audits Terraform code y and outputs a structured triple $z = (s, T, V)$: binary safety judgment, total vulnerability count, and vulnerability locations. This is a supervised learning task without reinforcement learning.

Given input code y, the model predicts $z = (s, T, V)$, where $s \in \{0, 1\}$ (1=safe, 0=unsafe) corresponds to $V_{\text{sec}}(y)$, $T \in \mathbb{N}$ is the vulnerability count, and $V = \{v_1, ..., v_T\}$ contains detected vulnerability locations and resource types.

The optimization objective is:

$$\max_\theta \mathbb{E}_{(y,z^*)\sim D_{\text{review}}} \left[\log q_\theta(\hat{z} = z^*|y)\right], \tag{5}$$

where D_{review} is the review dataset, $z^* = (V_{\text{sec}}(y), T(y), V^*(y))$ is the Tfsec-derived ground truth, $V^*(y)$ is the true vulnerability location set, and $\hat{z}$ is the model prediction.

Metrics for Code Review. (i) **Security Classification Accuracy (Acc)**: Proportion of correct binary safety predictions:

$$\text{Acc} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}\left(\hat{s}_i = s_i^*\right), \tag{6}$$

where M is the number of test samples, $\hat{s}_i$ and s_i^* are predicted and ground-truth labels.

(ii) **Exact Count Accuracy (ECA)**: Proportion of unsafe samples whose predicted vulnerability count exactly matches the ground-truth count:

$$\text{ECA} = \frac{1}{M_{\text{unsafe}}} \sum_{i \in \mathcal{M}_{\text{unsafe}}} \mathbb{I}\left(\hat{T}_i = T^*(y_i)\right), \tag{7}$$

where M_{unsafe} is the number of unsafe test samples, $\hat{T}_i$ is the model-predicted vulnerability count, and $T^*(y_i)$ is the ground-truth vulnerability count.

(iii) **Mean Absolute Error (MAE)**: Numerical deviation between predicted and real vulnerability counts (lower = better):

$$\text{MAE} = \frac{1}{M_{\text{unsafe}}} \sum_{i \in \mathcal{M}_{\text{unsafe}}} \left|\hat{T}_i - T_i^*\right|, \tag{8}$$

where $\hat{T}_i$ is the model-predicted vulnerability count.

3.3 IaC Secure Code Generation

Traditional text-to-IaC generation only optimizes for functional correctness, with the objective:

$$\max_\theta \mathbb{E}_{(x,y^*)\sim D} \left[\log p_\theta(y|x) \cdot \mathbb{I}(V_{\text{syn}} \wedge V_{\text{dep}} \wedge V_{\text{int}})\right], \tag{9}$$

which relies on an unsubstantiated assumption: functionally valid code is approximately secure, i.e., $\mathcal{Y}_{\text{func}}(x) \approx \mathcal{Y}_{\text{safe}}(x)$.

We define **IaC secure code generation** as a constrained optimization task: given intent x, generate code y that satisfies all functional and security constraints:

$$\begin{aligned} y &= \operatorname*{argmax}_{y \in \mathcal{Y}} \; p_\theta(y|x) \\ \text{s.t.} \quad & V_{\text{syn}} \wedge V_{\text{dep}} \wedge V_{\text{int}} \wedge V_{\text{sec}}. \end{aligned} \tag{10}$$

Let $\mathcal{Y}_{\text{safe}}(x)$ denote the set of safe and valid code. The final learning objective is:

$$\max_\theta \mathbb{E}_{x\sim\mathcal{X}} \left[\sum_{y \in \mathcal{Y}_{\text{safe}}(x)} p_\theta(y|x)\right]. \tag{11}$$

Metrics for Secure Code Generation. *Functional Metric*: Pass@k measures the proportion of intents for which at least one of k candidates meets all functional constraints:

$$\text{Pass@k} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(\bigvee_{j=1}^{k} \left(V_{\text{syn}} \wedge V_{\text{dep}} \wedge V_{\text{int}}\right)\right), \tag{12}$$

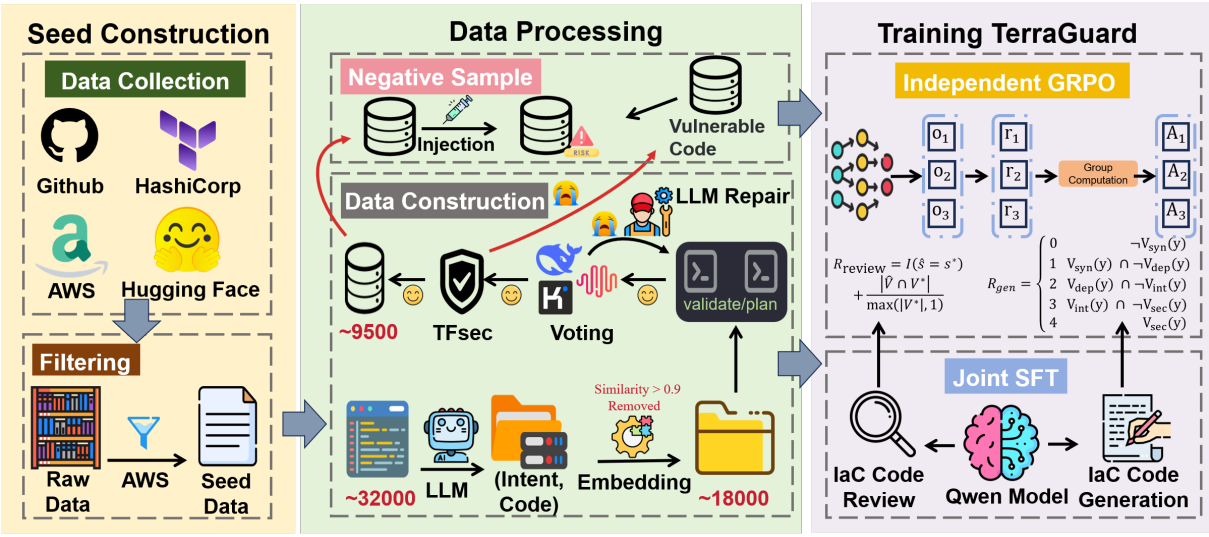

Figure 2: Overview of TerraGuard.

where N is the number of test intents. On IaC-Eval, V_{int} is checked by the benchmark-provided rego intent script rather than LLM-as-a-judge.

Security Metrics include three complementary indicators:

$$\text{Clean Rate (CR)} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left(V_{\text{sec}}(y_i)\right), \quad (13)$$

$$\text{Avg H+} = \frac{1}{N}\sum_{i=1}^{N} H(y_i), \quad (14)$$

$$\text{Avg T} = \frac{1}{N}\sum_{i=1}^{N} T(y_i). \quad (15)$$

Here, $H(y_i)$ denotes the number of Critical and High-severity vulnerabilities in y_i detected by Tfsec, and $T(y_i)$ denotes the total number of detected vulnerabilities across all severity levels.

Comprehensive Metric: Secure Generation Score (SGS) balances functionality and security:

$$\text{SGS} = \text{Pass@1} \times \text{CR} \times 100\% \quad (16)$$

The product form reflects the need to satisfy both functionality and security: a high SGS requires the model to generate code that is both functionally valid and scanner-clean.

4 Method

We propose **TerraGuard**, a dual-task alignment framework for **IaC Secure Code Generation** and **IaC Code Review**. TerraGuard follows a two-stage "SFT + GRPO" training paradigm. It first constructs Terraform instruction data through intent reversal and vulnerability mining, then applies joint SFT and multi-task GRPO with task-specific verifier feedback.

4.1 Dual-Task Training Data Construction

Data Collection and Filtering. We collect Terraform repositories from GitHub, HashiCorp Registry, AWS Official Samples, and Hugging Face Datasets. We then filter for independently executable modules with complete `.tf` files and no dependencies on private providers or internal variables. To reduce cross-cloud semantic variation, we focus on AWS configurations and remove non-AWS samples. This process yields approximately 32,000 AWS Terraform samples as the **Seed Data**.

Intent Reversal and Semantic Deduplication. Open-source Terraform repositories usually lack natural-language descriptions, making it difficult to construct **(Intent, Code)** pairs directly. We therefore use Qwen-Max for intent reversal, generating infrastructure requirements from Terraform configurations. To reduce semantic redundancy, we encode the generated intents with BGE embeddings and retain only samples whose cosine similarity is below 0.9. This step produces approximately 18,000 **(Intent, Code)** pairs.

Positive Sample Filtering via Execution and LLM Voting. We verify each **(Intent, Code)** pair for functionality, semantic consistency, and security. First, we run `terraform validate` and `terraform plan`. Failed samples are repaired by an LLM and rechecked. Functionally valid samples are then evaluated by a three-model voting com-

Resource	AST Mutation	Risk
aws_security_group	cidr_blocks: private range → 0.0.0.0/0	Public SSH/RDP exposure
aws_s3_bucket	acl: private → public-read	Public object access
aws_db_instance	public_access: false → true	Public database exposure
aws_ebs_volume	encrypted: true → false	Missing encryption at rest

Table 1: Examples of AST-based vulnerability injection.

mittee (Kimi-k2.5, DeepSeek-v3.2, and MiniMax-M2.5) for intent-code consistency; mismatched samples are repaired and revalidated. Finally, TFsec scans the remaining samples, and only configurations without detected issues are kept as positives. This yields 9,581 secure Terraform samples for **IaC Secure Code Generation**.

Negative Sample Construction. The code review task requires diverse vulnerable samples, but high-risk vulnerabilities are unevenly distributed in real Terraform repositories. We therefore combine *Native Harvesting* with *AST Targeted Injection*. Native harvesting collects configurations that pass `terraform validate` and `terraform plan` but fail TFsec, yielding about 10,000 real vulnerable samples with structured labels. AST injection complements this data by mutating security-sensitive fields in secure configurations, introducing public exposure, missing encryption, hardcoded secrets, and disabled auditing controls. Injected samples are also checked by `terraform validate` and `terraform plan`; more examples are shown in **??**. Together, these sources provide 13,200 vulnerable samples for **IaC Code Review**.

Dataset Analysis. The resulting secure generation corpus contains 9,581 verified **(Intent, Code)** pairs and is centered on AWS: 99.8% of samples use `hashicorp/aws`, with only three residual utility providers after filtering. The corpus is not limited to toy snippets. Each sample contains 107.8 lines of code on average, with a median of 97.0 lines, an interquartile range of 79–133 lines, and a 90th percentile of 171 lines. It also contains 9.44 resource instances per sample on average and 105,854 resource instances in total, covering 451 distinct AWS resource types. For a resource type r, its reported frequency is $100\% \times \#\text{instances}(r)/\#\text{samples}$; thus values above 100% mean more than one instance per sample on average. The most frequent resources reflect realistic infrastructure patterns, including `aws_subnet` (100.9%), `aws_security_group` (86.6%), and `aws_vpc` (80.4%). Intent topics show a similar distribution, with VPC/Network (48.9%), EC2/Compute (38.9%), S3/Storage (23.5%), ECS/-Container (19.8%) as the dominant categories. For completeness, the corresponding analysis of insecure samples and their security flaws is included in Appendix B.

4.2 Joint Supervised Fine-Tuning

We adopt a dual-task joint SFT paradigm, simultaneously training both code review and secure code generation capabilities on a base model. We design mutually exclusive instruction prefixes for the two tasks to establish clear task boundaries: for the code review task, the `[REVIEW]` prefix is used, which takes Terraform code as input and outputs a standardized review report; for the code generation task, the `[GENERATE]` prefix is used, which takes natural language requirements as input and outputs secure and compliant configuration code. To prevent excessive context length, we perform structured summary extraction on the original TFsec review reports, retaining only the security judgment, and location as training labels.

4.3 Multi-Task GRPO with Task-Specific Rewards

This stage continues safety preference reinforcement from the dual-task SFT model using **GRPO**. Review and generation samples are mixed during RL, and each sample is scored by the reward function associated with its task prefix. This keeps a shared policy across the two tasks while respecting their different optimization objectives: code review emphasizes safety judgment and vulnerability matching, whereas code generation follows staged verifier feedback.

For the code review task, the reward consists of two additive components with a total score range of 0 to 2:

$$R_{\text{review}} = \mathbb{I}(\hat{s} = s^*) + \frac{|\hat{V} \cap V^*|}{\max(|V^*|, 1)} \tag{17}$$

where $\hat{s}$ denotes the predicted safety label, s^* is the ground-truth label from TFsec, $\hat{V}$ and V^* represent the predicted and ground-truth vulnerability sets, respectively.

For the code generation task, we define a compact four-level reward function based on the predicates in Section 3.1. The intent check in RL reuses

Model	IaC Code Review			IaC Secure Code Generation						
	Acc↑	ECA↑	MAE↓	P@1↑	P@3↑	P@5↑	Avg H+↓	Avg T↓	CR↑	SGS↑
Closed-Source Models										
GPT-5.1	75.98%	18.49%	4.7	73.6%	79.2%	80.9%	2.24	4.25	**31.7%**	23.3
GPT-4.1-Mini	73.36%	13.88%	5.33	49.9%	52.1%	55.0%	2.33	4.08	28.0%	14.0
Gemini-2.5-Pro	73.58%	18.15%	3.47	44.3%	49.7%	54.8%	3.02	5.22	17.5%	7.8
Gemini-2.5-Flash	**77.51%**	15.22%	4.15	52.8%	53.7%	53.9%	2.22	4.21	23.0%	12.1
Claude-Opus-4.6	79.04%	10.34%	3.9	74.2%	80.5%	82.0%	2.10	4.05	33.0%	24.5
Claude-Sonnet-4	76.20%	13.66%	3.7	47.9%	57.2%	67.8%	2.46	4.53	21.0%	10.1
Doubao-Seed-2.0-pro	74.24%	13.57%	4.7	59.4%	62.5%	64.7%	2.18	4.08	30.0%	17.8
Qwen-Max	75.55%	8.42%	4.9	37.3%	49.2%	60.8%	2.19	4.23	13.5%	5.0
Open-Weight Models										
DeepSeek-V4-Flash	73.96%	14.91%	4.6	57.5%	71.8%	77.2%	2.65	4.75	20.5%	11.8
DeepSeek-V3.2	69.00%	10.33%	5.3	63.9%	83.8%	86.2%	2.29	4.63	24.6%	15.7
DeepSeek-V3	73.36%	10.81%	5.11	56.8%	71.4%	**76.5%**	2.75	4.85	19.3%	11.0
MiMo-V2-Flash	74.89%	12.50%	4.8	51.7%	64.5%	78.9%	2.26	4.58	27.0%	14.0
Kimi-K2.6	77.62%	15.38%	3.69	62.5%	70.5%	72.8%	2.24	4.75	24.5%	15.3
Kimi-K2	79.26%	8.53%	4.29	63.9%	71.8%	73.4%	2.24	4.44	26.4%	16.9
GLM-4.5-Air	74.34%	9.33%	4.5	42.4%	47.2%	52.3%	2.92	5.38	16.0%	6.8
Qwen3.5-122B-A10B	79.02%	14.04%	4.22	44.6%	53.2%	60.1%	2.22	3.87	19.0%	8.5
Qwen2.5-72B	68.93%	7.53%	4.7	40.1%	42.4%	45.0%	2.36	4.12	18.0%	7.2
Qwen2.5-32B	77.51%	11.46%	4.5	31.7%	34.1%	36.1%	2.42	4.04	14.0%	4.4
Qwen2.5-14B	74.67%	10.40%	4.6	28.0%	32.2%	33.1%	2.10	3.83	11.0%	3.1
Qwen2.5-7B	75.98%	12.33%	4.8	19.1%	20.8%	22.8%	2.07	3.56	9.3%	1.8
Method										
Manual Code	–	–	–	–	–	–	2.44	4.58	27.7%	–
TerraFormer	77.35%	14.11%	3.86	48.7%	61.9%	68.4%	2.12	3.08	29.7%	14.46
TerraGuard (Ours)	78.42%	16.87%	3.58	45.2%	58.8%	66.9%	1.74	3.21	36.8%	16.63

Table 2: Main results on IaC code review and secure Terraform code generation across LLMs and Methods.

the same committee-based procedure as data construction, and is invoked only after both syntax validation and deployment checking pass:

$$R_{\text{gen}} = \begin{cases} 0 & \neg V_{\text{syn}}(y) \\ 1 & V_{\text{syn}}(y) \wedge \neg V_{\text{dep}}(y) \\ 2 & V_{\text{dep}}(y) \wedge \neg V_{\text{int}}(x, y) \\ 3 & V_{\text{int}}(x, y) \wedge \neg V_{\text{sec}}(y) \\ 4 & V_{\text{sec}}(y) \end{cases} \tag{18}$$

This hierarchical function follows the priority of syntax, deployability, intent alignment and security.

5 Evaluation

5.1 Experiment Setup

We evaluate our method on the full IaC-Eval benchmark (458 Terraform samples), with complete separation between test and training data.

Baseline. We adopt 20 LLMs as baselines: zero-shot closed-source models (GPT-5.1, Claude-Opus-4.6) and open-source models (DeepSeek-V4-Flash, Qwen2.5-72B/32B). TerraFormer is used as an extra baseline. Details are in Appendix A.

Training Setup. **TerraGuard** is trained on Qwen2.5-14B with LoRA adapters on the query, key, value, and output projection layers ($r = 64$, $\alpha = 128$). The SFT stage uses 22,700 mixed samples for 3 epochs with a peak cosine learning rate of 2×10^{-4} and global batch size 128. The GRPO stage starts from the joint SFT checkpoint and uses mixed sampling over review and generation data; each sample is scored by its task-specific reward, with the SFT model as the reference policy, a KL coefficient of 0.05, and 3 update epochs.

5.2 Main Results

Table 2 reports the main results. We discuss representative models here and provide the complete results in Appendix C.

Mainstream LLMs exhibit a systematic function-first, security-deficient bias in Terraform security tasks. Existing LLMs perform well on functional correctness: Claude-Opus-4.6 achieves 74.2% Pass@1, and DeepSeek-V3.2 reaches 86.2% Pass@5. Their security performance is much weaker: all baselines have CR below 34%, with GPT-5.1 reaching only 31.7% CR and 23.3 SGS. In code review, even the most accurate model, Kimi-K2 (79.26% Acc), obtains only 8.53% ECA and 4.29 MAE. These results indicate that deployability is not a reliable proxy for security in IaC.

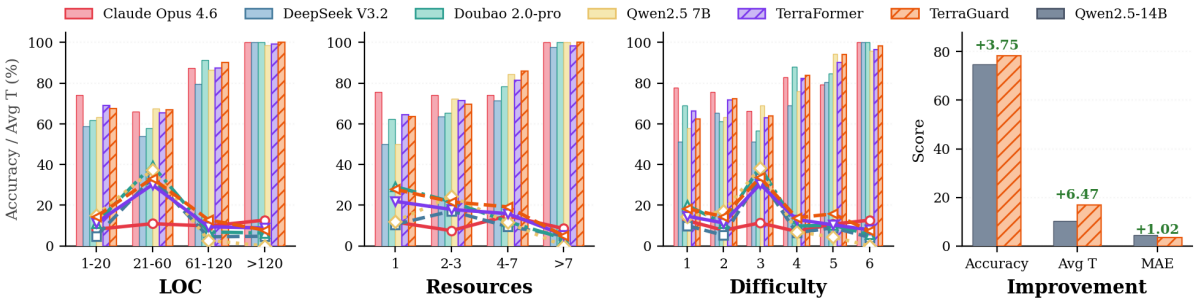

Figure 3: Code Review performance analysis across different dimensions and TerraGuard's improvement.

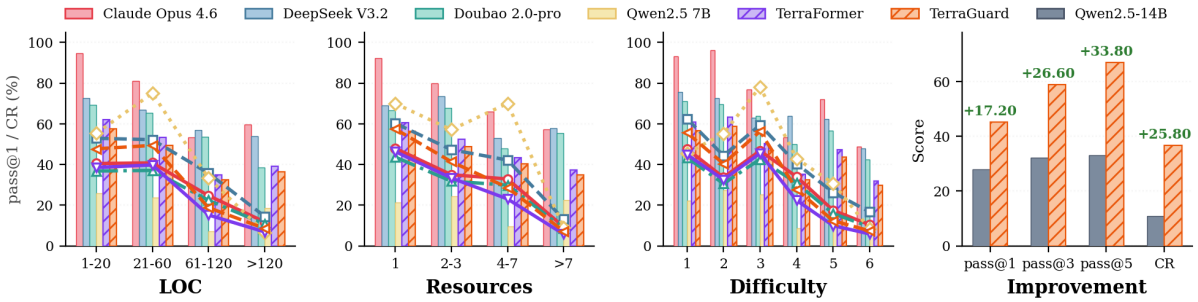

Figure 4: Code Generation performance analysis across different dimensions and TerraGuard's improvement.

IaC code review and secure generation capabilities are decoupled in existing LLMs, and single-task optimization is insufficient to recover their complementary gains. The two tasks show weak correlation. Kimi-K2, with the highest review accuracy (79.26%), achieves only 26.4% CR; Claude-Opus-4.6, the strongest model in functional generation (74.2% Pass@1), ranks third in review accuracy. TerraFormer, optimized for IaC generation, reaches 29.7% CR without clear improvement in review. These results suggest that vulnerability recognition does not translate into secure coding behavior, motivating joint modeling of the two tasks.

TerraGuard improves IaC security performance, outperforming most baselines with a small parameter size. Built on Qwen2.5-14B, TerraGuard achieves 36.8% CR (+3.8pp over Claude-Opus-4.6) and the lowest Avg H+ of 1.74 while maintaining 45.2% Pass@1. In code review, it reaches 78.42% Acc and 3.58 MAE, outperforming most 70B-class open-source models. These results support jointly modeling secure generation and code review with fine-grained reward design.

5.3 Dimensional Performance Analysis

We further analyze six representative models across lines of code (LOC), resource count, and difficulty level. Complete results are provided in the Appendix.

The code review task exhibits a consistent pattern in which Acc increases with configuration complexity, while ECA decreases with complexity. In terms of code length, short code snippets have lower Acc due to subtle vulnerabilities but smaller counting errors, whereas long code snippets show the opposite trend. For resource count, the 4–7 resource interval becomes the ECA bottleneck with the highest MAE, caused by the proliferation of composite vulnerabilities. TerraGuard achieves the best exact vulnerability-count accuracy in mainstream medium-complexity scenarios, with an overall Acc improvement of 3.75% and an MAE reduction of 1.02 compared to the baseline.

For the secure code generation task, both the functional pass rate and the CR decline sharply with increasing configuration complexity. In terms of code length, the CR of long code snippets is generally below 10%, which remains the core bottleneck at present. For resource count, multi-resource configurations suffer from a dramatic drop in security performance due to permission propagation and dependency relationships. For difficulty level, medium-difficulty tasks exhibit the most significant performance degradation due to the increased number of composite implicit security requirements. TerraGuard leads in secure generation rate across

Model	IaC Code Review			IaC Secure Code Generation						
	Acc↑	ECA↑	MAE↓	P@1↑	P@3↑	P@5↑	Avg H+↓	Avg T↓	CR↑	SGS↑
TerraGuard (Full)	**78.42%**	**16.87%**	**3.58**	**45.2%**	**58.8%**	**66.9%**	**1.74**	**3.21**	**36.8%**	**16.63**
w/o GRPO (Only SFT)	75.16%	14.23%	4.21	43.7%	56.2%	63.5%	2.15	3.86	29.5%	12.90
w/o Dual-Task (Gen-only)	72.34%	12.56%	4.67	42.9%	55.1%	62.3%	2.28	4.12	27.1%	11.63
w/o Dual-Task (Rev-only)	73.89%	13.71%	4.35	38.5%	49.2%	55.7%	2.41	4.33	22.3%	8.59
w/o AST Injection	75.63%	12.15%	4.42	44.8%	58.1%	66.2%	1.89	3.57	33.7%	15.10
w/o Task Boundary	74.27%	13.89%	4.53	41.3%	53.4%	60.7%	2.06	3.94	30.4%	12.56

Table 3: Ablation study on key components of **TerraGuard**.

```
resource "aws_security_group" "web_sg" {
  name = "web-sg"
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_db_instance" "db" {
  instance_class      = "db.t3.micro"
  publicly_accessible = true
  storage_encrypted   = false
}
```

Figure 5: A deployable but unsafe configuration.

mainstream scenarios, achieving an overall CR improvement of 25.0 percentage points over the baseline while maintaining a Pass@1 of 45.2%.

5.4 Case Study

Figure 5 shows a representative unsafe Terraform configuration that can still pass syntax and planning checks. The configuration satisfies the apparent intent of creating network access and a database instance, but it exposes SSH to the public internet, makes the database publicly reachable, and disables storage encryption. Such cases illustrate why functional validation alone overestimates IaC safety: the code is executable, but violates common cloud security policies.

5.5 Ablation Study

To better understand where TerraGuard's gains come from, we conduct a controlled ablation analysis. We remove one component at a time: (i) the GRPO stage, (ii) dual-task collaboration via single-task training, (iii) AST-based vulnerability injection, and (iv) explicit task-boundary instructions. Results are summarized in Table 3.

Role of GRPO. Removing GRPO and retaining only SFT leads to a 7.3% drop in CR and a 23.6% increase in Avg H+. This suggests that SFT alone is insufficient; preference optimization helps translate vulnerability knowledge into more reliable secure-generation behavior.

Benefit of Dual-Task Collaboration. Single-task variants consistently underperform the full model. Gen-only training weakens code review accuracy, causing a 6.08% drop in Acc, while Rev-only training severely impairs generation performance, with a 6.7% drop in Pass@1 and a 14.5% drop in CR. These results indicate that the gains of TerraGuard do not come from task-specific optimization alone, but from jointly modeling secure generation and code review, which provides complementary supervision across the two tasks.

Impact of Data Construction and Task Guidance. Removing AST-based vulnerability injection reduces the model's ability to handle long-tail vulnerability patterns, while removing explicit task-boundary instructions introduces task interference that harms both tasks. These results show that TerraGuard's gains depend not only on dual-task training, but also on richer security-oriented data construction and clearer task specialization.

6 Conclusion

We propose TerraGuard, a dual-task framework for addressing the function-first, security-deficient bias of LLMs in IaC and the decoupling between code review and secure generation. Experiments on IaC-Eval with 20 LLMs show that existing models often treat functional correctness as a proxy for security, while review capability does not naturally translate into secure generation. Built on Qwen2.5-14B, TerraGuard achieves 36.8% CR and the lowest Avg H+ of 1.74 while maintaining 45.2% Pass@1, and outperforms most 70B-class open-source models in code review. These results suggest that jointly modeling secure generation and code review with staged verifier feedback is a promising direction for scanner-grounded IaC security. Future work will extend to more cloud platforms and IaC languages, incorporate stronger security verifiers and cross-checking mechanisms, and evaluate generalization in realistic enterprise settings.

Limitations

This work only targets Terraform configurations on AWS and does not support other mainstream clouds like Azure, GCP, and Alibaba Cloud, so the cross-cloud generalization of TerraGuard remains unvalidated. Our training data comes from open-source public repositories without real enterprise-grade scenarios, which limits the model's performance on complex orchestration and introduces semantic deviations in intent alignment. Furthermore, the fixed priority in the GRPO reward function lacks flexibility for security-critical scenarios, and task conflicts still exist in dual-task training, causing performance trade-offs on extreme samples. Last but not least, Our evaluation relies entirely on TFsec for security labeling, which may lead the model to learn scanner-specific heuristics rather than general security reasoning.

Ethics Considerations

This study focuses on the secure generation and review of IaC, and strictly abides by academic ethics and relevant laws and regulations. The training data are collected from public open-source Terraform repositories including GitHub and HashiCorp Registry, obtained in compliance with open-source licenses, and contain no personal privacy, sensitive business or copyrighted private data, thus presenting no abuse or infringement risks. TerraGuard is designed to enhance the security of IaC configurations, and is solely used for positive security protection and academic research, prohibiting malicious attacks, non-compliant configuration generation, and other illegal usages. The model outputs need to be verified by professional tools and manual audits to avoid over-reliance risks. We declare no conflicts of interest, and this work aims to advance the development of the IaC security domain.

References

Fenia Christopoulou, Guchun Zhang, and Gerasimos Lampouras. 2024. Text-to-code generation with modality-relative pre-training. *Preprint*, arXiv:2402.05783.

Sam Davidson, Li Sun, Bhavana Bhasker, Laurent Callot, and Anoop Deoras. 2025. Multi-iac-eval: Benchmarking cloud infrastructure as code across multiple formats. *Preprint*, arXiv:2509.05303.

Hossein Hajipour, Keno Hassler, Thorsten Holz, Lea Sch"onherr, and Mario Fritz. 2023a. Codelmsec benchmark: Systematically evaluating and finding security vulnerabilities in black-box code language models. In *Second IEEE Conference on Secure and Trustworthy Machine Learning*.

Hossein Hajipour, Keno Hassler, Thorsten Holz, Lea Schönherr, and Mario Fritz. 2023b. Codelmsec benchmark: Systematically evaluating and finding security vulnerabilities in black-box code language models. *Preprint*, arXiv:2302.04012.

Manvi Jha, Jiaxin Wan, Huan Zhang, and Deming Chen. 2025. Preface - a reinforcement learning framework for code verification via llm prompt repair. In *Proceedings of the Great Lakes Symposium on VLSI 2025*, GLSVLSI '25, page 547–553, New York, NY, USA. Association for Computing Machinery.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2025. A survey on large language models for code generation. *ACM Trans. Softw. Eng. Methodol.* Just Accepted.

Naveen Kasarla. 2025. Implementing infrastructure as code (iac) with terraform for scalable cloud deployments. *Journal of Information Systems Engineering and Management*, 10:986–1000.

Rana Nameer Hussain Khan, Dawood Wasif, Jin-Hee Cho, and Ali Butt. 2025. Multi-agent code-orchestrated generation for reliable infrastructure-as-code. *Preprint*, arXiv:2510.03902.

Patrick Tser Jern Kon, Jiachen Liu, Yiming Qiu, Weijun Fan, Ting He, Lei Lin, Haoran Zhang, Owen M. Park, George Sajan Elengikal, Yuxin Kang, Ang Chen, Mosharaf Chowdhury, Myungjin Lee, and Xinyu Wang. 2024. Iac-eval: A code generation benchmark for cloud infrastructure-as-code programs. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Kaixin Li, Yuchen Tian, Qisheng Hu, Ziyang Luo, Zhiyong Huang, and Jing Ma. 2024. Mmcode: Benchmarking multimodal large language models for code generation with visually rich programming problems. *Preprint*, arXiv:2404.09486.

Yikun Li, Matteo Grella, Daniel Nahmias, Gal Engelberg, Dan Klein, Giancarlo Guizzardi, Thijs van Ede, and Andrea Continella. 2025. Gensiac: Toward security-aware infrastructure-as-code generation with large language models. *Preprint*, arXiv:2511.12385.

Qiyue Mei and Michael Fu. 2026. Intellisa: An intelligent static analyzer for iac security smell detection using symbolic rules and neural inference. *Preprint*, arXiv:2601.14595.

Francesco Romeo, Luigi Arena, Francesco Blefari, Francesco Aurelio Pironti, Matteo Lupinacci, and Angelo Furfaro. 2025. *ARPaCCino: An Agentic-RAG for Policy as Code Compliance*, page 467–481. Springer Nature Switzerland.

Nuno Saavedra and João F. Ferreira. 2023. Glitch: Automated polyglot security smell detection in infrastructure as code. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA. Association for Computing Machinery.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Ji Shi, Peiming Guo, Meishan Zhang, Miao Zhang, Xuebo Liu, Min Zhang, and Weili Guan. 2026. Cve-drl: An efficient code verifier via difficulty-aware reinforcement learning. *Preprint*, arXiv:2601.22803.

Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. 2025. Live-swe-agent: Can software engineering agents self-evolve on the fly? *Preprint*, arXiv:2511.13646.

Jingyu Xiao, Ming Wang, Man Ho Lam, Yuxuan Wan, Junliang Liu, Yintong Huo, and Michael R. Lyu. 2025. Designbench: A comprehensive benchmark for mllm-based front-end code generation. *Preprint*, arXiv:2506.06251.

Yu Yang, Yuzhou Nie, Zhun Wang, Yuheng Tang, Wenbo Guo, Bo Li, and Dawn Song. 2024. Seccodeplt: A unified platform for evaluating the security of code genai. *Preprint*, arXiv:2410.11096.

Tianyi Zhang, Shidong Pan, Zejun Zhang, Zhenchang Xing, and Xiaoyu Sun. 2025. Deployability-centric infrastructure-as-code generation: An llm-based iterative framework. *Preprint*, arXiv:2506.05623.

Yaoyu Zhu, Di Huang, Hanqi Lyu, Xiaoyun Zhang, Chongxiao Li, Wenxuan Shi, Yutong Wu, Jianan Mu, Jinghua Wang, Yang Zhao, Pengwei Jin, Shuyao Cheng, Shengwen Liang, Xishan Zhang, Rui Zhang, Zidong Du, Qi Guo, Xing Hu, and Yunji Chen. 2025. Qimeng-codev-r1: Reasoning-enhanced verilog generation. *Preprint*, arXiv:2505.24183.

A LLM Details

Table 4 summarizes the LLMs used in our evaluation. The selection covers both closed-source commercial systems and open-weight models across different architectures and parameter scales. We group them into two categories based on accessibility.

- **API Models**: This category encompasses state-of-the-art closed-source models, including offerings from OpenAI, Anthropic, Google, Grok, and so on. These models are accessed via cloud-based inference endpoints provided by their respective developers.
- **Open-Weight Models**: This category features high-performance models with publicly available weights, such as Qwen and DeepSeek. These models allow for local deployment and independent verification, serving as baselines for open-source capability.

B Dataset Analysis

B.1 Secure Dataset

Figure 6 summarizes the curated Terraform dataset used for secure IaC generation. The final filtered dataset contains 9,581 (intent, code) pairs. Intent deduplication removes 96.9% of near-duplicate descriptions, yielding a diverse task distribution. The average intent length is 22.3 tokens, with a median of 22.0 tokens. In terms of provider configuration, 99.8% of the samples use `hashicorp/aws`, and only three samples involve non-AWS providers (`hashicorp/random`, `hashicorp/null`, and `hashicorp/time`). This makes the dataset essentially AWS-specific, which supports controlled evaluation while limiting generalization to multi-cloud settings.

The code scale follows a concentrated distribution suitable for production-oriented IaC evaluation. The average number of lines per code sample is 107.8 (standard deviation = 46.7), with a median of 97.0 lines, an interquartile range of 79–133 lines, and a 90th percentile of 171 lines; approximately 63% of the samples fall within the 75–150 line range, representing medium-scale Terraform configurations. Each code sample contains an average of 9.44 resource instances (median = 9.0, maximum = 47), and the entire dataset accumulates 105,854 resource instances covering 451 distinct AWS resource types.

The 20 most frequent resource types form the core of cloud infrastructure provisioning, where the first seven resources constitute a standard VPC+EC2 network-computing skeleton. Core network resources include `aws_subnet` (100.9%), `aws_route_table_association` (91.3%), `aws_security_group` (86.6%), `aws_vpc` (80.4%), `aws_route_table` (79.6%), and `aws_internet_gateway` (76.2%), and the core compute resource is `aws_instance` (65.6%). Additional high-coverage resources (more than 10%) include `aws_s3_bucket` (39.5%), `aws_iam_role_policy_attachment` (35.1%), `aws_iam_role` (29.4%), `aws_dynamodb_table` (18.0%), and load-balancing resources

Table 4: Detailed specifications of the LLMs used in our evaluation. "Unknown" indicates that the provider has not officially disclosed the parameter scale. For MoE models, we report total parameters when publicly available.

Model	Size	Access	Provider	Model	Size	Access	Provider
GPT-5.1	Unknown	Closed (API)	OpenAI	Kimi-K2.6	1T (MoE)	Open-weight	Moonshot AI
GPT-4.1-Mini	Unknown	Closed (API)	OpenAI	Kimi-K2-Instruct	1T (MoE)	Open-weight	Moonshot AI
Gemini-2.5-Pro	Unknown	Closed (API)	Google	GLM-4.5-Air	106B (MoE)	Open-weight	Zhipu AI
Gemini-2.5-Flash	Unknown	Closed (API)	Google	MiMo-V2-Flash	Unknown	Open-weight	Xiaomi
Claude-Opus-4.6	Unknown	Closed (API)	Anthropic	Qwen3.5-122B-A10B	122B (MoE)	Open-weight	Alibaba
Claude-Sonnet-4	Unknown	Closed (API)	Anthropic	Qwen2.5-72B	72B (Dense)	Open-weight	Alibaba
Doubao-Seed-2.0-pro	Unknown	Closed (API)	ByteDance	Qwen2.5-32B	32B (Dense)	Open-weight	Alibaba
Qwen-Max	Unknown	Closed (API)	Alibaba	Qwen2.5-14B	14B (Dense)	Open-weight	Alibaba
DeepSeek-V3.2	685B (MoE)	Open-weight	DeepSeek	Qwen2.5-7B	7B (Dense)	Open-weight	Alibaba
DeepSeek-V3	671B (MoE)	Open-weight	DeepSeek	DeepSeek-V4-Flash	Unknown	Open-weight	DeepSeek

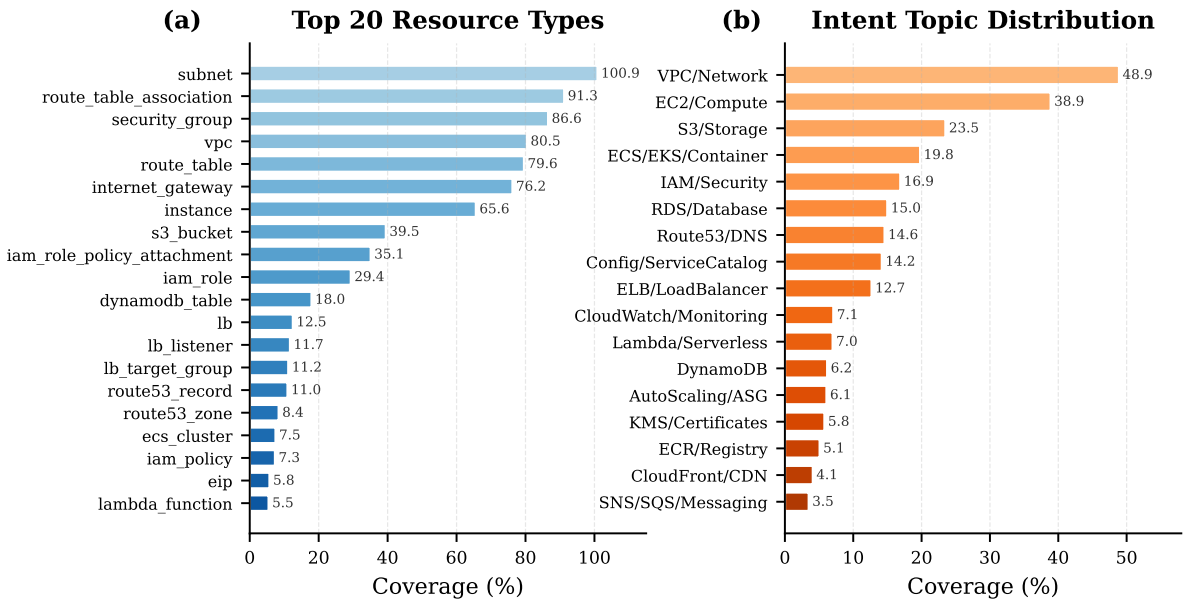

Figure 6: Statistical analysis of the Terraform dataset. (a) Coverage rate of the top 20 resource types across all generated code samples. (b) Distribution of intent topics.

(`aws_lb`: 12.5%, `aws_lb_listener`: 11.7%, `aws_lb_target_group`: 11.2%).

Intent topics are categorized via keyword matching with multi-label assignment allowed (thus total coverage exceeds 100%), and the dominant topics align with the resource distribution. Primary topics are VPC/Network (48.9%) and EC2/Compute (38.9%); secondary topics include S3/Storage (23.5%), ECS/EKS/Container (19.8%), IAM/Security (16.9%), and RDS/Database (15.0%); other topics cover Route53/DNS (14.6%), ELB/LoadBalancer (12.7%), CloudWatch/Monitoring (7.1%), and Lambda/Serverless (7.0%). High-frequency tokens (e.g., `region`, `vpc`, `ec2`, `instance`) and phrases (e.g., *aws region*, *specified region*, *ec2 instance*, *vpc subnets*) reflect the standardized template of infrastructure requirement descriptions in the dataset.

B.2 Insecure Dataset

Figure 7 to Figure 10 present a comprehensive statistical analysis of the curated vulnerable Terraform dataset, which is specifically constructed for the IaC code review task and dual-task security alignment training. The final dataset comprises 13,200 verified vulnerable samples, generated through a hybrid pipeline of native harvesting from open-source repositories and AST-targeted vulnerability injection. This approach addresses the severe class imbalance and underrepresentation of high-risk vulnerabilities in real-world IaC codebases.

Notably, the average number of resources per project (16.17) is 71.3% higher than that of the secure dataset (9.44), indicating that more complex infrastructure configurations are statistically more likely to contain security misconfigurations. The code scale distribution closely mirrors the secure

Table 5: Overview of the Insecure Terraform Dataset

Metric	Value
Total number of projects	13,200
Total number of AWS resources	213,388
Average resources per project	16.17
Median resources per project	9
Standard deviation of resources per project	31.00
Interquartile range (P25-P75) of resources	4 – 18
Maximum / minimum resources per project	1,461 / 0
Average files per project	9.84
Median files per project	5
Total number of vulnerabilities	142,939
Average vulnerabilities per project	10.83
Median vulnerabilities per project	6
Standard deviation of vulnerabilities per project	16.97
Interquartile range (P25-P75) of vulnerabilities	2 – 13
Maximum / minimum vulnerabilities per project	602 / 1

dataset, with 62.1% of samples falling within the 75–150 line range, representing the medium-scale Terraform configurations most prevalent in industrial production environments.

Vulnerability Count Distribution. Figure 7 illustrates the distribution of vulnerability counts per project. The distribution is highly right-skewed, with a median of 6 vulnerabilities per project and a mean of 10.83. Approximately 75% of projects contain 13 or fewer vulnerabilities, while a small subset of complex projects (top 5%) exhibit extreme vulnerability counts exceeding 50. This skewed distribution reflects real-world IaC security patterns, where most configurations contain a manageable number of defects, but a minority of large, poorly maintained projects accumulate dozens of security issues.

Vulnerability Severity Distribution. Figure 8 breaks down the total 142,939 vulnerabilities by severity level, as classified by the TFsec static analyzer. Critical and High-severity vulnerabilities collectively account for 65.7% of all defects, ensuring the dataset prioritizes the most impactful security risks that pose immediate threats to cloud infrastructure.

Critical-severity vulnerabilities (e.g., public exposure of sensitive services, hardcoded credentials) are the most prevalent, with 57,151 occurrences (40.0% of total) and an average of 4.33 per project. High-severity vulnerabilities follow with 36,718 occurrences (25.7%), while Medium and Low-severity issues account for 20.1% and 14.2% respectively. This severity weighting ensures that the model learns to prioritize remediation of high-risk misconfigurations during both review and generation tasks.

Resource Scale and Vulnerability Correlation. Figure 9 analyzes the relationship between the number of resources per project and the number of vulnerabilities. A moderate positive correlation (r=0.415) is observed, indicating that larger infrastructure deployments generally contain more security defects. However, the correlation is not strong, demonstrating that configuration quality remains a critical factor independent of scale.

Figure 10 further quantifies this relationship by grouping projects into resource count buckets. The average number of vulnerabilities increases exponentially with resource scale: projects with 1–5 resources average only 5.7 vulnerabilities, while projects with 201–500 resources average 61.5 vulnerabilities. This exponential growth underscores the unique security challenges of large-scale enterprise infrastructure, where cross-resource dependencies and permission propagation create compounding risk.

C Further Experiment Analysis

C.1 IaC Code Review Dimensional Performance Analysis

C.1.1 Performance Analysis by Lines of Code

Table 6 presents code review performance across four LOC intervals ($\leq$20, 21–60, 61–120, $>$120). A consistent pattern holds: classification accuracy increases monotonically with code length, while ECA decreases monotonically.

- $\leq$20 LOC: Acc 60%–75%, MAE 3–5. TerraGuard achieves 67.44% Acc and 3.59 MAE, outperforming baseline Qwen2.5-14B (61.07% Acc, 5.53 MAE).
- 21–60 LOC (industry mainstream): Acc 65%–75%, MAE 2–4. TerraGuard achieves the lowest MAE (1.94), 2.26 lower than baseline.
- 61–120 LOC: Acc 85%–90%, MAE 3–5. TerraGuard achieves the highest Acc (90.24%) and 3.62 MAE.
- $>$120 LOC: Acc $\approx$100%, MAE 5–7. TerraGuard achieves 5.33 MAE, matching TerraFormer (5.34 MAE).

TerraGuard maintains the strongest ECA across all intervals, with the most significant advantage in the 21–60 LOC mainstream scenario.

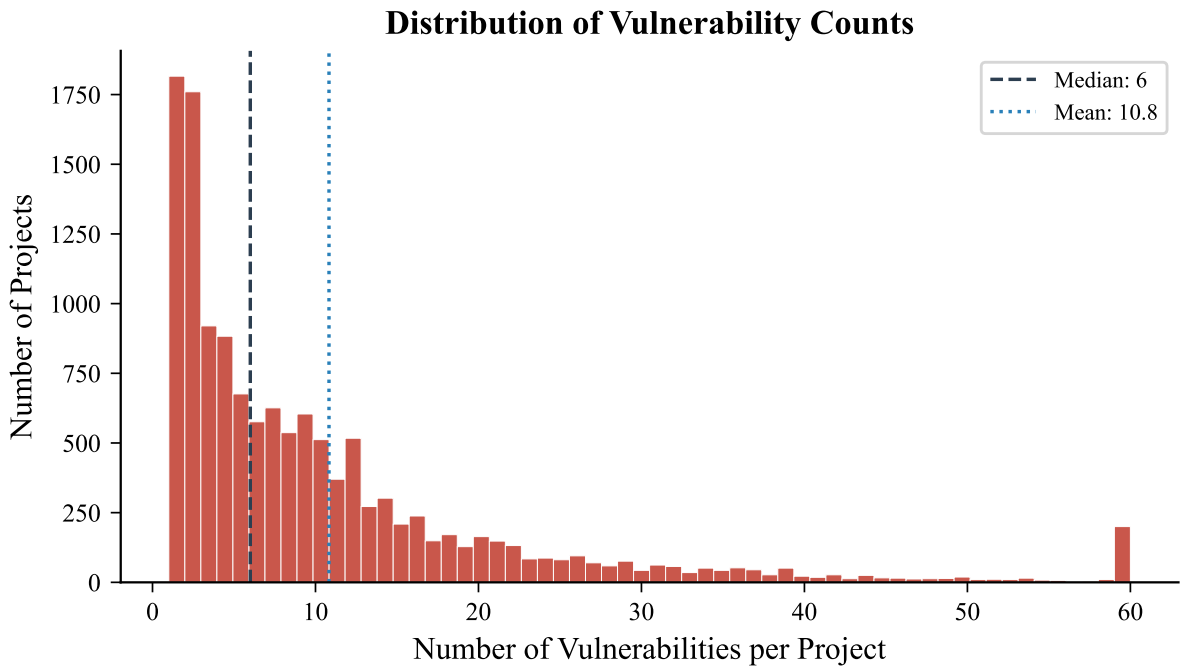

Figure 7: Distribution of vulnerability counts per project in the insecure dataset.

C.1.2 Performance Analysis by Resource Count

Table 7 compares performance across four resource intervals (1, 2–3, 4–7, >7). Acc increases with resource count, while MAE peaks in the 4–7 resource interval due to proliferating composite vulnerabilities.

- 1 resource: Acc 50%–75%, MAE 1–3. TerraGuard achieves the lowest MAE (1.55), 1.45 lower than baseline.
- 2–3 resources: Acc 65%–77%, MAE 2–4. TerraGuard achieves 3.44 MAE.
- 4–7 resources (ECA bottleneck): TerraGuard achieves the lowest MAE (3.15), 0.79 lower than baseline.
- >7 resources: Acc ≈100%, MAE 5–7. TerraGuard achieves the lowest MAE (5.00).

TerraGuard achieves the strongest ECA across resource intervals, suggesting improved modeling of multi-resource security dependencies.

C.1.3 Performance Analysis by Difficulty Level

Table 8 presents performance across six difficulty levels (Lv.1–Lv.6). Acc increases with difficulty, while MAE peaks in Lv.3–Lv.4 due to maximum long-tail vulnerability coverage.

- Lv.1–Lv.2: Acc 50%–75%, MAE 2–4. TerraGuard achieves 2.23 MAE for Lv.1, 1.62 lower than baseline.
- Lv.3–Lv.4 (ECA bottleneck): TerraGuard achieves the lowest MAE (1.82) for Lv.3, consistent with the benefit of AST injection.
- Lv.5–Lv.6: Acc ≈100%, MAE 3–7. TerraGuard achieves the lowest MAE (3.32) for Lv.5.

TerraGuard performs well on both common and long-tail vulnerability patterns.

C.2 IaC Secure Code Generation Dimensional Performance Analysis

C.2.1 Performance Analysis by Lines of Code

Functional Pass Rate (Pass@k) Analysis Table 9 shows Pass@k decreases sharply with increasing code length.

- ≤20 LOC: Pass@1 50%–95%. TerraGuard achieves 57.6% Pass@1, 9.2pp higher than baseline.
- 21–60 LOC (mainstream): Pass@1 30%–85%. TerraGuard achieves 49.3% Pass@1, 21.4pp higher than baseline.

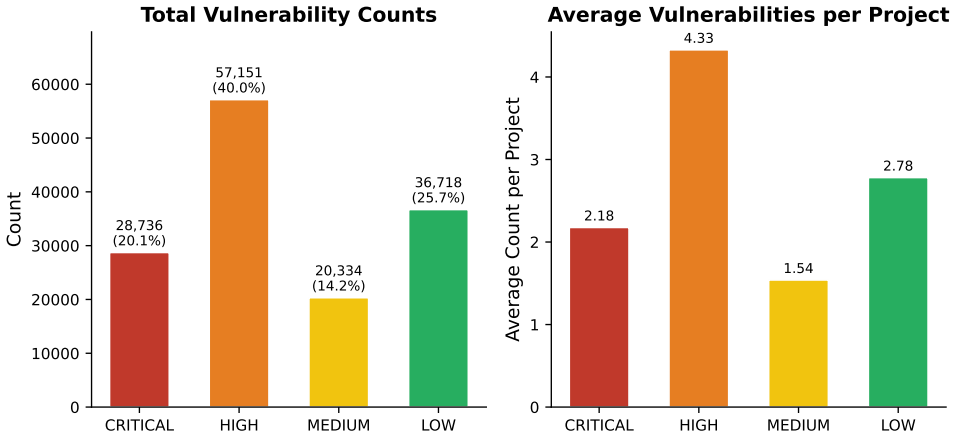

Figure 8: Vulnerability severity distribution: total counts and average per project.

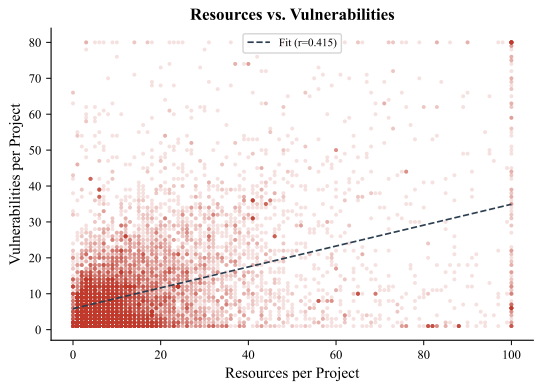

Figure 9: Correlation between number of resources and vulnerabilities per project (r=0.415).

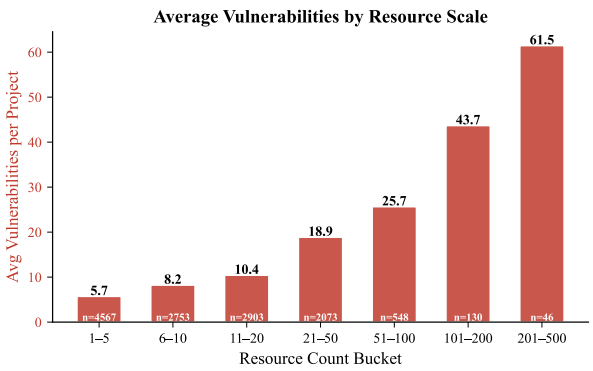

Figure 10: Average vulnerabilities per project by resource count bucket.

- >120 LOC: Pass@1 <40%. TerraGuard achieves 36.3% Pass@1, 26.2pp higher than baseline.

Dual-task training improves functional generation for complex configurations.

Security Performance Analysis Table 10 shows CR decreases sharply with code length, while Avg H+ increases.

- 21–60 LOC (mainstream): TerraGuard achieves the highest CR (49.4%), 7.5pp higher than baseline and 8.6pp higher than Claude-Opus-4.6.
- >120 LOC: CR <15%. TerraGuard achieves 8.6% CR, compared with 0.0% for the baseline.

TerraGuard achieves a favorable security–functionality trade-off in mainstream industrial scenarios.

C.2.2 Performance Analysis by Resource Count

Functional Pass Rate (Pass@k) Analysis Table 11 shows that Pass@k decreases as resource count increases.

- 2–3 resources (mainstream): TerraGuard achieves 48.7% Pass@1, 14.7pp higher than baseline.
- >7 resources: Pass@1 <40%. TerraGuard achieves 34.8% Pass@1, 26.5pp higher than baseline.

TerraGuard effectively models functional dependencies between multiple resources.

Security Performance Analysis (Table 10) Table 12 shows CR decreases sharply with resource count, while Avg H+ increases.

- 2–3 resources (mainstream): TerraGuard achieves the highest CR (41.4%), 13.7pp higher than baseline.

Model	≤20 LOC			21–60 LOC			61–120 LOC			>120 LOC		
	Acc↑	ECA↑	MAE↓	Acc↑	ECA↑	MAE↓	Acc↑	ECA↑	MAE↓	Acc↑	ECA↑	MAE↓
Proprietary Models												
GPT-5.1	64.89%	14.89%	4.68	59.26%	48.08%	2.04	92.91%	10.68%	4.76	100.00%	9.52%	6.78
GPT-4.1-Mini	65.65%	15.22%	5.30	57.04%	36.36%	2.70	85.83%	9.68%	4.98	98.46%	3.23%	7.73
Gemini-2.5-Pro	60.31%	15.62%	3.33	60.74%	32.84%	2.16	88.98%	15.60%	3.60	96.92%	9.52%	4.76
Gemini-2.5-Flash	67.18%	12.28%	4.25	66.67%	36.92%	2.00	88.19%	9.62%	4.22	100.00%	4.76%	6.14
Claude-Opus-4.6	74.05%	8.20%	3.46	65.93%	10.94%	2.52	87.40%	9.80%	4.12	100.00%	12.70%	5.57
Claude-Sonnet-4	64.89%	11.76%	3.44	66.67%	26.83%	2.02	86.61%	8.26%	3.95	98.46%	7.94%	5.70
Doubao-Seed-2.0-pro	61.83%	11.11%	4.58	57.78%	39.58%	2.83	91.34%	6.86%	4.54	100.00%	6.35%	6.32
Qwen-Max	64.89%	8.16%	5.53	65.19%	26.15%	2.52	86.61%	0.93%	4.76	96.92%	3.17%	7.29
Open Weight Models												
DeepSeek-V4-Flash	67.94%	15.79%	4.37	55.97%	33.33%	2.39	85.83%	11.88%	4.72	100.00%	3.17%	6.44
DeepSeek-V3.2	58.78%	4.76%	5.81	54.07%	33.33%	3.12	79.53%	4.49%	5.13	100.00%	4.76%	6.87
DeepSeek-V3	64.89%	8.51%	5.49	61.48%	33.93%	2.64	81.10%	3.23%	5.06	100.00%	3.17%	7.10
MiMo-V2-Flash	65.65%	6.38%	5.49	60.00%	37.70%	2.41	88.19%	8.26%	4.50	98.46%	0.00%	6.94
Kimi-K2.6	70.87%	14.58%	4.38	65.15%	33.87%	1.92	89.66%	8.16%	3.81	98.15%	7.69%	4.96
Kimi-K2-Instruct	70.23%	5.56%	4.72	70.37%	17.39%	2.43	88.98%	5.61%	4.23	96.92%	6.35%	6.03
GLM-4.5-Air	67.21%	5.45%	4.62	62.50%	17.07%	2.98	84.91%	12.94%	4.25	95.65%	0.00%	6.34
Qwen3.5-122B-A10B	71.88%	7.27%	5.15	66.67%	38.46%	2.42	90.32%	7.55%	4.06	98.36%	5.08%	5.64
Qwen2.5-72B	54.20%	6.25%	4.47	54.48%	27.91%	2.33	85.04%	2.94%	4.47	96.92%	1.61%	6.85
Qwen2.5-32B	65.65%	18.03%	4.85	69.63%	26.83%	2.07	88.19%	2.78%	4.76	96.92%	0.00%	7.10
Qwen2.5-14B	61.07%	8.89%	5.53	65.93%	30.86%	2.26	86.61%	1.83%	4.72	96.92%	0.00%	6.79
Qwen2.5-7B	63.36%	15.09%	5.19	67.41%	37.31%	2.25	86.61%	2.75%	4.84	98.46%	0.00%	7.11
TerraFormer	68.96%	11.29%	4.10	65.46%	30.07%	2.13	87.49%	9.43%	3.87	99.14%	8.82%	5.34
TerraGuard (Ours)	67.44%	14.32%	3.59	66.91%	33.25%	1.94	90.24%	12.81%	3.62	100.00%	7.75%	5.33

Sample Size: ≤20=131, 21–60=135, 61–120=127, >120=65, Total=458

Table 6: IaC Code Review Performance under Different Line-of-Code Ranges on IaC-Eval

- >7 resources: CR <15%. TerraGuard achieves 7.2% CR, outperforming baseline (0.0%).

Joint modeling of secure generation and code review helps capture inter-resource security constraints more effectively.

C.2.3 Performance Analysis by Difficulty Level

Functional Pass Rate (Pass@k) Analysis (Table 11) Table 13 shows Pass@k decreases monotonically with difficulty, with the sharpest drop in Lv.3–Lv.4.

- Lv.3–Lv.4: TerraGuard achieves 46.7% Pass@1 for Lv.3, 18.4pp higher than baseline.
- Lv.5–Lv.6: Pass@1 <50%. TerraGuard achieves 43.8% Pass@1 for Lv.5, 35.5pp higher than baseline.

Security Performance Analysis (Table 12) Table 14 shows CR decreases monotonically with difficulty, while Avg H+ increases.

- Lv.2: TerraGuard achieves the highest CR (40.4%), 7.8pp higher than TerraFormer.
- Lv.5–Lv.6: CR <20%. TerraGuard achieves 12.4% CR for Lv.5 and 7.4% for Lv.6, compared with 0.0% for the baseline.

TerraGuard maintains low average harm severity across all difficulty levels, validating effective security preference alignment.

Model	1 Resource			2–3 Resources			4–7 Resources			>7 Resources		
	Acc↑	**ECA↑**	**MAE↓**	**Acc↑**	**ECA↑**	**MAE↓**	**Acc↑**	**ECA↑**	**MAE↓**	**Acc↑**	**ECA↑**	**MAE↓**
Proprietary Models												
GPT-5.1	55.56%	9.09%	2.73	68.24%	33.78%	3.92	86.21%	19.54%	3.99	100.00%	6.10%	6.66
GPT-4.1-Mini	57.78%	15.00%	3.00	68.24%	28.17%	4.79	79.31%	14.10%	4.27	92.68%	0.00%	7.53
Gemini-2.5-Pro	55.56%	31.43%	1.29	64.71%	19.35%	3.47	81.90%	22.58%	3.08	100.00%	6.10%	4.83
Gemini-2.5-Flash	64.44%	21.88%	2.09	70.59%	23.86%	3.69	81.90%	16.09%	3.59	100.00%	2.44%	6.02
Claude-Opus-4.6	75.56%	11.76%	1.18	74.12%	7.53%	3.78	74.14%	14.81%	3.65	100.00%	8.54%	5.55
Claude-Sonnet-4	55.56%	25.00%	1.22	69.41%	16.19%	3.47	85.34%	13.13%	3.41	100.00%	6.10%	5.41
Doubao-Seed-2.0-pro	62.22%	29.17%	2.21	65.29%	21.74%	4.51	78.45%	12.05%	4.02	100.00%	3.66%	6.16
Qwen-Max	60.00%	11.11%	2.67	68.82%	16.67%	4.60	81.03%	5.38%	4.16	98.78%	2.47%	6.95
Open Weight Models												
DeepSeek-V4-Flash	58.89%	31.03%	1.55	66.27%	16.67%	4.58	78.45%	19.77%	3.98	100.00%	2.44%	6.30
DeepSeek-V3.2	50.00%	10.53%	3.11	63.53%	17.14%	5.17	71.55%	9.59%	4.36	97.56%	5.00%	6.81
DeepSeek-V3	56.67%	9.52%	2.81	68.82%	20.25%	4.80	76.72%	11.25%	4.38	96.34%	1.27%	6.78
MiMo-V2-Flash	58.89%	4.76%	2.90	64.12%	24.39%	4.49	85.34%	14.74%	3.77	100.00%	0.00%	6.66
Kimi-K2.6	69.32%	30.00%	1.90	67.88%	19.48%	3.73	85.19%	16.47%	3.28	100.00%	2.94%	4.96
Kimi-K2-Instruct	55.56%	9.52%	2.38	77.06%	11.70%	4.09	86.21%	9.38%	3.60	100.00%	3.66%	5.80
GLM-4.5-Air	50.00%	5.00%	2.90	72.03%	11.27%	4.45	82.98%	13.51%	4.01	98.36%	3.33%	5.75
Qwen3.5-122B-A10B	62.07%	16.67%	2.42	73.53%	21.98%	4.30	85.84%	14.13%	3.51	100.00%	3.85%	5.53
Qwen2.5-72B	54.44%	10.00%	2.90	56.21%	16.67%	4.00	79.31%	6.98%	3.90	96.34%	1.27%	6.52
Qwen2.5-32B	53.33%	22.58%	2.16	75.29%	18.27%	4.18	83.62%	10.31%	3.82	100.00%	0.00%	6.76
Qwen2.5-14B	54.44%	8.00%	3.00	65.88%	22.83%	4.09	85.34%	8.08%	3.94	100.00%	0.00%	6.50
Qwen2.5-7B	50.00%	12.00%	3.12	72.35%	24.44%	4.31	84.48%	11.58%	4.05	100.00%	0.00%	6.72
TerraFormer	64.55%	21.90%	1.77	71.35%	17.93%	3.65	81.28%	15.89%	3.51	98.29%	5.44%	5.24
TerraGuard (Ours)	63.50%	27.83%	1.55	69.52%	21.66%	3.44	85.92%	19.23%	3.15	100.00%	5.00%	5.00

Sample Size: 1=90, 2–3=170, 4–7=116, >7=82, Total=458

Table 7: IaC Code Review Performance under Different Resource Quantities on IaC-Eval

Model	Lv.1			Lv.2			Lv.3			Lv.4			Lv.5			Lv.6		
	Acc↑	ECA↑	MAE↓	Acc↑	ECA↑	MAE↓	Acc↑	ECA↑	MAE↓	Acc↑	ECA↑	MAE↓	Acc↑	ECA↑	MAE↓	Acc↑	ECA↑	MAE↓
Proprietary Models																		
GPT-5.1	68.89%	13.33%	2.93	63.16%	14.29%	5.31	58.41%	47.62%	2.24	86.21%	7.69%	5.31	91.67%	20.63%	4.27	100.00%	8.45%	6.24
GPT-4.1-Mini	71.11%	20.00%	3.13	62.11%	11.76%	6.06	60.18%	42.11%	2.68	75.86%	9.38%	5.38	83.33%	12.28%	4.91	97.33%	1.45%	7.22
Gemini-2.5-Pro	51.11%	18.75%	1.94	67.37%	16.67%	3.89	58.41%	32.08%	2.11	81.03%	18.18%	3.61	90.28%	16.92%	3.03	96.00%	9.86%	4.80
Gemini-2.5-Flash	64.44%	18.75%	2.38	70.53%	10.87%	4.98	65.49%	39.62%	1.85	82.76%	7.32%	4.39	87.50%	12.90%	3.87	98.67%	5.63%	5.82
Claude-Opus-4.6	77.78%	12.50%	1.44	75.79%	7.69%	4.25	66.37%	11.32%	2.34	82.76%	7.14%	4.00	79.17%	10.71%	4.02	100.00%	12.68%	5.38
Claude-Sonnet-4	55.56%	11.76%	1.71	71.58%	12.07%	4.02	62.83%	26.56%	2.02	75.86%	11.36%	3.52	94.44%	10.29%	3.76	97.33%	8.45%	5.46
Doubao-Seed-2.0-pro	68.89%	20.00%	2.73	61.05%	11.11%	5.39	56.64%	38.89%	2.89	87.93%	9.76%	4.56	84.72%	10.17%	4.46	100.00%	5.63%	5.83
Qwen-Max	62.22%	0.00%	3.92	67.37%	10.00%	6.12	64.60%	30.19%	2.47	72.41%	2.38%	4.71	93.06%	0.00%	4.38	96.00%	4.23%	6.96
Open Weight Models																		
DeepSeek-V4-Flash	60.00%	21.43%	2.21	71.58%	12.77%	4.83	52.68%	36.59%	2.73	81.03%	9.76%	5.00	86.11%	18.03%	4.10	100.00%	2.82%	6.14
DeepSeek-V3.2	51.11%	10.00%	4.10	65.26%	5.26%	6.26	51.33%	33.33%	3.11	68.97%	6.45%	5.35	80.56%	8.93%	4.71	100.00%	4.23%	6.52
DeepSeek-V3	64.44%	9.09%	3.64	65.26%	12.50%	6.05	61.06%	31.11%	2.67	67.24%	0.00%	4.94	86.11%	10.17%	4.75	100.00%	2.82%	6.75
MiMo-V2-Flash	71.11%	0.00%	3.54	62.11%	10.53%	6.11	59.29%	36.73%	2.49	75.86%	11.90%	4.43	93.06%	11.94%	4.10	98.67%	0.00%	6.65
Kimi-K2.6	76.74%	20.00%	2.80	69.57%	15.79%	4.79	62.16%	31.25%	2.00	84.21%	16.28%	3.70	91.53%	11.32%	3.72	97.01%	4.76%	4.51
Kimi-K2-Instruct	64.44%	14.29%	2.79	75.79%	2.17%	5.41	65.49%	21.15%	2.29	81.03%	11.63%	3.95	94.44%	2.99%	4.00	97.33%	5.63%	5.79
GLM-4.5-Air	59.09%	7.14%	3.43	73.26%	4.35%	5.00	59.30%	24.14%	2.93	76.74%	13.33%	4.20	88.52%	13.21%	4.08	93.10%	0.00%	5.87
Qwen3.5-122B-A10B	70.45%	7.14%	3.14	75.27%	8.51%	5.72	62.83%	40.82%	2.27	81.03%	20.93%	3.84	94.12%	4.69%	3.94	98.61%	4.41%	5.32
Qwen2.5-72B	68.89%	7.69%	3.38	47.37%	0.00%	5.57	53.57%	30.30%	2.21	72.41%	12.82%	4.15	88.89%	1.61%	4.37	97.33%	1.41%	6.44
Qwen2.5-32B	55.56%	21.43%	3.07	71.58%	21.15%	5.33	69.03%	23.53%	2.03	74.14%	7.14%	4.52	94.44%	2.99%	4.39	97.33%	1.41%	6.83
Qwen2.5-14B	62.22%	7.69%	3.85	62.11%	10.81%	6.27	62.83%	33.85%	2.11	75.86%	4.55%	4.39	94.44%	2.94%	4.26	96.00%	0.00%	6.65
Qwen2.5-7B	57.78%	14.29%	3.71	63.16%	17.07%	5.83	69.03%	38.18%	2.25	75.86%	6.98%	4.72	94.44%	4.41%	4.38	96.00%	0.00%	6.85
Step-3.5-Flash	54.76%	16.67%	2.42	71.74%	13.73%	4.24	57.41%	33.33%	1.92	78.85%	16.22%	4.08	91.67%	27.27%	2.49	96.43%	5.77%	5.06
TerraFormer	66.16%	14.64%	2.21	71.82%	11.32%	4.70	63.01%	30.32%	2.04	82.32%	13.00%	4.03	90.23%	10.35%	3.71	96.46%	7.91%	5.05
TerraGuard (Ours)	62.30%	18.41%	2.23	72.33%	14.38%	4.33	63.96%	33.63%	1.82	83.91%	13.89%	3.83	94.07%	15.76%	3.32	98.32%	7.43%	4.92

Sample Size: Lv.1=45, Lv.2=95, Lv.3=113, Lv.4=58, Lv.5=72, Lv.6=75, Total=458

Table 8: IaC Code Review Performance under Different Difficulty Levels on IaC-Eval

Model	≤20 LOC			21–60 LOC			61–120 LOC			>120 LOC		
	P@1↑	**P@3↑**	**P@5↑**	**P@1↑**	**P@3↑**	**P@5↑**	**P@1↑**	**P@3↑**	**P@5↑**	**P@1↑**	**P@3↑**	**P@5↑**
Proprietary Models												
GPT-5.1	79.4%	85.5%	86.3%	81.5%	83.8%	86.2%	65.6%	75.2%	77.6%	61.5%	64.6%	66.2%
GPT-4.1-Mini	62.6%	63.0%	66.4%	53.1%	56.7%	58.5%	40.0%	43.3%	44.8%	36.9%	37.9%	44.6%
Gemini-2.5-Pro	54.9%	56.5%	62.6%	52.2%	57.0%	60.8%	37.6%	44.8%	49.6%	20.0%	30.7%	36.9%
Gemini-2.5-Flash	66.4%	66.4%	66.4%	65.4%	66.9%	66.9%	37.6%	39.2%	39.2%	29.2%	29.2%	30.8%
Claude-Opus-4.6	94.6%	91.1%	91.3%	81.0%	85.9%	87.4%	53.3%	70.8%	70.6%	59.6%	67.1%	74.4%
Claude-Sonnet-4	61.1%	64.7%	75.5%	52.3%	61.0%	72.2%	34.4%	50.3%	58.4%	38.5%	47.7%	61.5%
Doubao-Seed-2.0-pro	69.4%	68.7%	71.7%	65.4%	68.5%	70.7%	53.6%	57.2%	59.1%	38.5%	48.2%	49.2%
Qwen-Max	52.0%	59.5%	67.2%	38.5%	50.1%	65.4%	31.2%	48.4%	61.6%	16.9%	28.2%	36.9%
Open Weight Models												
DeepSeek-V4-Flash	75.8%	84.4%	88.6%	67.0%	80.5%	83.1%	43.8%	63.6%	70.2%	28.1%	44.9%	55.9%
DeepSeek-V3.2	72.5%	91.6%	92.4%	66.9%	86.2%	88.5%	56.8%	76.0%	78.4%	53.8%	76.9%	84.6%
DeepSeek-V3	74.8%	84.0%	87.8%	66.2%	80.0%	82.3%	43.2%	63.2%	69.6%	27.7%	44.6%	55.4%
MiMo-V2-Flash	72.5%	78.9%	91.6%	58.5%	69.7%	81.5%	37.6%	55.1%	74.4%	23.1%	43.2%	56.9%
Kimi-K2.6	70.2%	73.4%	76.5%	67.8%	76.2%	77.8%	61.9%	71.4%	73.0%	37.6%	51.3%	54.9%
Kimi-K2-Instruct	71.8%	74.8%	77.1%	69.2%	77.7%	78.5%	63.2%	72.8%	73.6%	38.5%	52.3%	55.4%
GLM-4.5-Air	55.1%	56.8%	60.3%	43.9%	45.7%	52.3%	35.2%	46.0%	52.8%	27.7%	33.3%	35.4%
Qwen3.5-122B-A10B	58.8%	63.6%	69.3%	44.6%	53.4%	61.3%	34.4%	47.4%	53.4%	35.4%	42.8%	52.1%
Qwen2.5-72B	54.9%	55.6%	58.0%	40.0%	44.1%	47.7%	25.6%	28.7%	30.4%	38.5%	38.8%	41.5%
Qwen2.5-32B	50.4%	50.1%	51.8%	27.7%	29.4%	30.8%	20.8%	25.9%	28.8%	23.1%	27.1%	29.2%
Qwen2.5-14B	48.4%	54.3%	54.7%	27.9%	30.9%	32.7%	12.4%	16.6%	17.6%	10.1%	13.1%	12.9%
Qwen2.5-7B	26.0%	26.5%	28.7%	23.8%	25.3%	27.4%	7.2%	10.5%	11.9%	18.5%	20.2%	22.8%
TerraFormer	62.1%	70.0%	76.2%	53.2%	66.0%	72.8%	35.0%	54.4%	58.9%	39.1%	51.6%	62.1%
TerraGuard (Ours)	57.6%	66.5%	74.5%	49.3%	62.7%	71.3%	32.5%	51.7%	57.6%	36.3%	49.0%	60.7%

Sample Size: ≤20=131, 21–60=130, 61–120=125, >120=65

Table 9: IaC Code Generation Pass@k Performance under Different Line-of-Code Ranges on IaC-Eval

Model	≤20 LOC			21–60 LOC			61–120 LOC			>120 LOC		
	Avg H+↓	**Avg T↓**	**CR↑**	**Avg H+↓**	**Avg T↓**	**CR↑**	**Avg H+↓**	**Avg T↓**	**CR↑**	**Avg H+↓**	**Avg T↓**	**CR↑**
Proprietary Models												
GPT-5.1	2.15	3.70	46.1%	1.54	3.00	47.7%	3.82	6.58	18.2%	5.61	9.44	8.3%
GPT-4.1-Mini	2.12	3.39	34.3%	1.35	2.44	34.6%	3.27	6.14	20.9%	3.74	7.21	10.0%
Gemini-2.5-Pro	2.53	4.05	21.8%	1.29	2.67	22.8%	3.64	7.19	12.4%	6.61	10.42	1.9%
Gemini-2.5-Flash	2.18	3.60	28.6%	1.11	2.37	29.9%	3.13	6.39	16.3%	5.70	9.27	2.5%
Claude-Opus-4.6	1.87	3.21	40.5%	1.19	2.31	40.8%	2.88	5.81	24.7%	3.30	6.82	11.8%
Claude-Sonnet-4	2.24	3.68	26.1%	1.14	2.42	27.3%	3.22	6.53	14.9%	5.86	9.47	2.2%
Doubao-Seed-2.0-pro	1.94	3.23	36.8%	1.24	2.33	37.1%	2.99	5.86	22.4%	3.43	6.87	10.7%
Qwen-Max	2.68	4.29	16.8%	1.37	2.82	17.6%	3.85	7.61	9.6%	7.01	11.03	1.4%
Open Weight Models												
DeepSeek-V4-Flash	2.07	3.65	26.6%	1.49	2.96	27.5%	3.69	6.49	10.5%	5.42	9.32	4.8%
DeepSeek-V3.2	1.91	3.37	52.9%	1.50	2.79	52.2%	2.64	5.41	35.7%	4.20	8.33	14.5%
DeepSeek-V3	2.00	3.25	55.8%	1.28	2.34	56.2%	3.08	5.89	34.0%	3.53	6.91	16.3%
MiMo-V2-Flash	2.07	3.47	33.1%	1.32	2.50	33.4%	3.18	6.29	20.2%	3.65	7.38	9.7%
Kimi-K2.6	1.89	3.63	30.2%	1.49	3.00	29.8%	2.61	5.82	20.4%	4.15	8.96	8.3%
Kimi-K2-Instruct	2.01	3.67	43.6%	1.40	2.67	55.9%	2.52	5.48	40.2%	4.58	8.94	5.6%
GLM-4.5-Air	2.56	4.09	50.6%	1.31	2.69	52.9%	3.68	7.26	28.8%	6.70	10.52	4.3%
Qwen3.5-122B-A10B	2.66	4.65	22.5%	1.78	3.52	23.5%	2.67	5.17	16.3%	4.15	6.65	2.1%
Qwen2.5-72B	2.22	3.55	56.6%	1.02	2.10	61.3%	2.29	4.61	42.1%	4.33	7.48	7.4%
Qwen2.5-32B	2.46	3.90	57.4%	1.65	2.95	60.0%	2.47	4.33	41.7%	3.84	5.58	5.3%
Qwen2.5-14B	2.96	4.69	44.2%	1.94	3.16	41.9%	2.57	4.00	28.6%	2.60	4.80	0.0%
Qwen2.5-7B	2.47	3.79	55.3%	0.94	1.83	75.0%	1.80	4.00	33.3%	3.87	5.73	6.7%
TerraFormer	1.70	2.40	38.5%	1.20	1.90	39.8%	2.90	4.20	15.2%	4.30	6.00	6.9%
TerraGuard (Ours)	1.40	2.50	47.7%	1.00	2.00	49.4%	2.40	4.40	18.8%	3.60	6.30	8.6%

Security Metrics: Avg H+ = Average Harm Severity, Avg T = Average Total Severity, CR = Secure Generation Rate

Table 10: IaC Secure Code Generation Performance under Different Line-of-Code Ranges on IaC-Eval

Model	1 Resource			2–3 Resources			4–7 Resources			>7 Resources		
	P@1↑	**P@3↑**	**P@5↑**	**P@1↑**	**P@3↑**	**P@5↑**	**P@1↑**	**P@3↑**	**P@5↑**	**P@1↑**	**P@3↑**	**P@5↑**
Proprietary Models												
GPT-5.1	73.8%	79.8%	81.0%	83.2%	87.1%	88.4%	67.6%	75.0%	77.9%	64.5%	69.7%	71.1%
GPT-4.1-Mini	60.7%	62.0%	65.5%	52.3%	53.9%	56.1%	45.6%	48.5%	50.7%	40.8%	44.0%	48.7%
Gemini-2.5-Pro	42.8%	45.2%	48.8%	56.7%	60.0%	65.8%	45.5%	52.4%	58.1%	18.4%	28.8%	32.9%
Gemini-2.5-Flash	64.3%	64.3%	64.3%	66.5%	67.1%	67.1%	39.7%	41.9%	41.9%	35.5%	35.5%	36.8%
Claude-Opus-4.6	92.2%	91.6%	93.5%	80.0%	81.9%	82.7%	66.1%	77.4%	75.5%	57.1%	70.9%	79.5%
Claude-Sonnet-4	59.5%	65.1%	77.3%	51.6%	58.2%	68.3%	42.6%	55.0%	62.4%	36.8%	50.4%	65.7%
Doubao-Seed-2.0-pro	66.7%	67.6%	71.4%	67.7%	68.8%	70.3%	47.8%	51.1%	53.7%	55.3%	64.4%	65.8%
Qwen-Max	41.8%	51.4%	61.9%	47.2%	55.6%	66.5%	27.9%	46.5%	60.3%	28.9%	38.6%	48.7%
Open Weight Models												
DeepSeek-V4-Flash	72.4%	83.8%	86.5%	68.6%	79.2%	84.6%	49.9%	69.5%	74.9%	32.0%	47.6%	55.8%
DeepSeek-V3.2	69.0%	85.7%	86.9%	73.5%	92.9%	94.2%	52.9%	74.3%	80.1%	57.9%	78.9%	80.3%
DeepSeek-V3	71.4%	83.3%	85.7%	67.7%	78.7%	83.9%	49.3%	69.1%	74.3%	31.6%	47.4%	55.3%
MiMo-V2-Flash	69.0%	77.7%	90.4%	63.9%	72.5%	85.1%	36.8%	57.3%	73.5%	34.2%	46.5%	63.2%
Kimi-K2.6	66.4%	71.3%	72.0%	71.4%	78.5%	81.3%	64.8%	71.4%	72.9%	36.1%	51.6%	56.1%
Kimi-K2-Instruct	67.9%	72.6%	72.6%	72.9%	80.0%	81.9%	66.2%	72.8%	73.5%	36.8%	52.6%	56.6%
GLM-4.5-Air	45.3%	51.6%	56.0%	50.4%	51.0%	56.1%	38.3%	42.9%	50.0%	30.3%	42.1%	44.7%
Qwen3.5-122B-A10B	57.1%	60.7%	65.3%	46.5%	57.4%	65.6%	41.9%	50.4%	56.4%	31.6%	41.4%	49.8%
Qwen2.5-72B	51.2%	52.0%	53.6%	40.6%	45.1%	48.4%	32.4%	33.6%	36.0%	40.8%	42.1%	44.7%
Qwen2.5-32B	38.1%	38.5%	40.5%	36.8%	37.3%	38.7%	24.3%	28.1%	30.9%	27.6%	33.5%	35.5%
Qwen2.5-14B	37.9%	43.3%	44.8%	34.0%	38.4%	38.8%	23.0%	27.7%	29.3%	8.3%	8.5%	8.4%
Qwen2.5-7B	21.4%	22.3%	23.5%	24.5%	24.9%	26.8%	9.6%	12.4%	14.5%	22.4%	25.9%	28.6%
TerraFormer	60.5%	70.4%	78.0%	52.5%	63.0%	68.9%	43.3%	59.5%	63.0%	37.4%	54.5%	66.3%
TerraGuard	56.2%	66.9%	76.3%	48.7%	59.8%	67.4%	40.2%	56.5%	61.6%	34.8%	51.8%	64.9%

Sample Size: 1=84, 2–3=155, 4–7=136, >7=76

Table 11: IaC Code Generation Pass@k Performance under Different Resource Quantities on IaC-Eval

Model	1 Resource			2–3 Resources			4–7 Resources			>7 Resources		
	Avg H+↓	**Avg T↓**	**CR↑**	**Avg H+↓**	**Avg T↓**	**CR↑**	**Avg H+↓**	**Avg T↓**	**CR↑**	**Avg H+↓**	**Avg T↓**	**CR↑**
Proprietary Models												
GPT-5.1	0.83	2.04	55.6%	2.62	4.36	40.0%	3.05	5.38	27.7%	5.63	9.51	7.0%
GPT-4.1-Mini	0.70	1.55	40.7%	2.40	3.79	29.7%	2.27	4.39	27.9%	4.67	8.73	8.0%
Gemini-2.5-Pro	0.76	1.71	28.4%	2.78	4.75	16.8%	2.85	5.29	17.7%	6.13	10.66	3.8%
Gemini-2.5-Flash	0.65	1.52	37.3%	2.40	4.22	22.1%	2.45	4.70	23.3%	5.28	9.48	5.0%
Claude-Opus-4.6	0.62	1.47	48.0%	2.12	3.59	35.0%	2.00	4.15	32.9%	4.12	8.27	9.4%
Claude-Sonnet-4	0.67	1.55	34.1%	2.46	4.31	20.2%	2.52	4.81	21.3%	5.43	9.69	4.6%
Doubao-Seed-2.0-pro	0.64	1.48	43.7%	2.20	3.62	31.8%	2.08	4.19	29.9%	4.28	8.33	8.6%
Qwen-Max	0.80	1.81	21.9%	2.95	5.02	13.0%	3.02	5.60	13.7%	6.49	11.28	2.9%
Open Weight Models												
DeepSeek-V4-Flash	0.80	2.01	32.0%	2.52	4.30	23.1%	2.94	5.30	16.0%	5.43	9.38	4.0%
DeepSeek-V3.2	0.70	1.86	60.3%	2.29	3.88	47.3%	2.18	4.25	42.2%	4.44	9.03	13.1%
DeepSeek-V3	0.66	1.49	66.2%	2.26	3.64	48.2%	2.14	4.21	45.3%	4.41	8.37	13.0%
MiMo-V2-Flash	0.68	1.59	39.3%	2.34	3.88	28.6%	2.21	4.49	26.9%	4.55	8.94	7.7%
Kimi-K2.6	0.69	2.00	34.4%	2.26	4.18	27.0%	2.16	4.57	24.1%	4.39	9.72	7.5%
Kimi-K2-Instruct	0.72	2.20	49.2%	2.17	3.72	50.4%	2.31	4.59	41.0%	4.47	9.40	11.6%
GLM-4.5-Air	0.77	1.72	66.0%	2.82	4.79	39.1%	2.88	5.34	41.2%	6.21	10.76	8.8%
Qwen3.5-122B-A10B	0.73	1.47	30.0%	3.33	6.00	18.9%	2.70	4.97	18.7%	3.28	5.57	5.8%
Qwen2.5-72B	0.69	1.38	68.9%	2.35	3.99	52.0%	2.00	3.31	51.0%	3.85	7.47	11.8%
Qwen2.5-32B	0.68	1.24	76.5%	3.08	5.03	48.3%	2.50	4.17	47.6%	3.04	4.67	14.8%
Qwen2.5-14B	1.14	2.32	59.1%	3.57	5.53	27.7%	1.97	3.13	46.7%	3.67	5.67	0.0%
Qwen2.5-7B	1.20	1.75	70.0%	2.19	3.71	57.1%	1.30	2.05	70.0%	3.23	5.64	9.1%
TerraFormer	0.60	1.30	46.4%	2.00	2.80	33.4%	2.40	3.40	23.1%	4.30	6.10	5.8%
TerraGuard (Ours)	0.50	1.40	57.5%	1.70	2.90	41.4%	1.90	3.60	28.7%	3.60	6.30	7.2%

Security Metrics: Avg H+ = Average Harm Severity, Avg T = Average Total Severity, CR = Secure Generation Rate

Table 12: IaC Code Generation Security Performance under Different Resource Quantities on IaC-Eval

Model	Lv.1			Lv.2			Lv.3			Lv.4			Lv.5			Lv.6		
	P@1↑	P@3↑	P@5↑	P@1↑	P@3↑	P@5↑	P@1↑	P@3↑	P@5↑	P@1↑	P@3↑	P@5↑	P@1↑	P@3↑	P@5↑	P@1↑	P@3↑	P@5↑
Proprietary Models																		
GPT-5.1	75.6%	82.2%	84.4%	83.2%	88.4%	88.4%	79.3%	81.1%	83.8%	72.4%	81.0%	81.0%	71.0%	78.3%	78.3%	54.8%	61.6%	67.1%
GPT-4.1-Mini	66.7%	67.7%	71.1%	61.1%	62.1%	65.3%	50.5%	53.2%	55.0%	29.3%	35.6%	37.9%	59.4%	59.9%	60.9%	31.5%	33.6%	39.7%
Gemini-2.5-Pro	40.0%	40.1%	44.4%	67.3%	69.0%	75.9%	45.0%	51.4%	55.0%	50.0%	59.0%	67.2%	34.8%	37.1%	42.0%	20.5%	32.5%	35.6%
Gemini-2.5-Flash	64.4%	64.4%	64.4%	67.4%	67.4%	67.4%	64.9%	66.7%	66.7%	39.7%	43.1%	43.1%	46.4%	46.4%	46.4%	24.7%	24.7%	26.0%
Claude-Opus-4.6	93.0%	96.9%	96.7%	96.2%	89.1%	89.1%	76.8%	82.1%	84.9%	53.4%	66.4%	64.6%	71.9%	85.5%	85.8%	48.8%	63.3%	69.5%
Claude-Sonnet-4	60.0%	68.9%	79.9%	62.1%	63.3%	73.7%	49.5%	58.3%	70.3%	34.5%	47.1%	53.4%	46.4%	60.8%	71.0%	31.5%	44.9%	57.5%
Doubao-Seed-2.0-pro	71.1%	71.7%	77.7%	69.5%	68.0%	69.4%	64.0%	67.9%	69.3%	50.0%	55.7%	60.3%	56.5%	61.0%	62.3%	42.5%	48.2%	49.3%
Qwen-Max	44.5%	53.0%	60.0%	54.8%	59.8%	70.5%	38.8%	52.9%	65.8%	27.6%	44.3%	58.6%	37.8%	53.2%	65.2%	15.1%	27.6%	38.4%
Open Weight Models																		
DeepSeek-V4-Flash	78.8%	89.4%	91.9%	74.6%	81.5%	87.1%	63.9%	79.7%	81.8%	45.4%	64.2%	71.3%	49.9%	67.0%	73.1%	29.1%	46.8%	56.7%
DeepSeek-V3.2	75.6%	88.9%	88.9%	72.6%	93.7%	94.7%	63.1%	83.8%	86.5%	63.8%	84.5%	89.7%	62.3%	79.7%	82.6%	47.9%	69.9%	74.0%
DeepSeek-V3	77.8%	88.9%	91.1%	73.7%	81.1%	86.3%	63.1%	79.3%	81.1%	44.8%	63.8%	70.7%	49.3%	66.7%	72.5%	28.8%	46.6%	56.2%
MiMo-V2-Flash	77.9%	80.6%	88.8%	69.5%	76.3%	91.5%	56.8%	69.4%	81.0%	37.9%	54.6%	74.1%	36.2%	56.5%	75.4%	30.1%	47.2%	60.3%
Kimi-K2.6	65.2%	72.0%	72.7%	72.1%	74.4%	78.3%	66.1%	75.1%	76.9%	70.9%	79.5%	80.4%	61.0%	72.5%	73.3%	37.5%	48.4%	53.0%
Kimi-K2-Instruct	66.7%	73.3%	73.3%	73.7%	75.8%	78.9%	67.6%	76.6%	77.5%	72.4%	81.0%	81.0%	62.3%	73.9%	73.9%	38.4%	49.3%	53.4%
GLM-4.5-Air	51.2%	54.4%	57.7%	55.9%	56.5%	59.9%	42.4%	44.1%	50.5%	34.5%	43.9%	50.0%	40.7%	46.4%	55.1%	27.4%	38.7%	41.1%
Qwen3.5-122B-A10B	51.2%	59.6%	64.2%	62.2%	66.3%	71.3%	42.3%	51.8%	61.1%	50.1%	57.5%	67.0%	40.6%	51.0%	54.9%	20.5%	33.0%	41.0%
Qwen2.5-72B	62.2%	62.6%	64.4%	52.6%	53.1%	55.8%	37.8%	42.0%	45.0%	29.3%	33.5%	36.2%	39.1%	40.8%	42.0%	23.3%	25.3%	28.8%
Qwen2.5-32B	37.8%	37.0%	40.0%	55.8%	55.7%	56.8%	27.9%	30.0%	31.5%	20.7%	28.7%	31.0%	27.5%	34.1%	36.2%	15.1%	14.8%	17.8%
Qwen2.5-14B	39.2%	44.4%	43.9%	48.7%	56.6%	57.2%	28.3%	29.2%	31.5%	17.3%	21.7%	21.5%	8.3%	11.4%	14.1%	9.9%	12.8%	12.6%
Qwen2.5-7B	22.2%	22.9%	24.2%	27.4%	27.7%	30.2%	25.2%	26.2%	28.5%	8.6%	11.3%	12.0%	14.5%	16.3%	18.6%	9.6%	14.2%	16.2%
TerraFormer	61.0%	74.6%	80.6%	63.2%	68.5%	74.3%	50.3%	63.1%	70.9%	35.1%	51.0%	53.8%	47.2%	65.8%	71.6%	32.0%	48.6%	58.0%
TerraGuard (Ours)	56.6%	70.8%	78.8%	58.6%	65.1%	72.7%	46.7%	59.9%	69.3%	32.6%	48.4%	52.7%	43.8%	62.5%	70.0%	29.7%	46.2%	56.7%

Sample Size: Lv.1=45, Lv.2=95, Lv.3=111, Lv.4=58, Lv.5=69, Lv.6=73, Total=451

Table 13: IaC Generation Performance under Different Difficulty Levels on IaC-Eval

Model	Lv.1			Lv.2			Lv.3			Lv.4			Lv.5			Lv.6		
	CR↑	Avg H+↓	Avg T↓	CR↑	Avg H+↓	Avg T↓	CR↑	Avg H+↓	Avg T↓	CR↑	Avg H+↓	Avg T↓	CR↑	Avg H+↓	Avg T↓	CR↑	Avg H+↓	Avg T↓
Proprietary Models																		
GPT-5.1	53.7%	1.07	2.39	39.0%	2.82	4.51	54.4%	1.33	2.70	26.8%	2.83	4.90	12.0%	4.46	7.04	7.1%	5.14	9.57
GPT-4.1-Mini	40.5%	0.89	1.84	28.6%	2.85	4.31	39.7%	1.09	2.06	28.8%	2.10	3.49	14.8%	3.55	6.44	8.8%	4.15	8.71
Gemini-2.5-Pro	29.8%	0.87	1.60	16.6%	3.31	5.23	26.1%	1.18	2.58	19.3%	2.76	4.30	9.1%	3.69	6.83	1.4%	6.06	11.56
Gemini-2.5-Flash	39.1%	0.75	1.42	21.8%	2.85	4.65	34.3%	1.02	2.30	25.3%	2.38	3.83	11.9%	3.18	6.07	1.9%	5.22	10.28
Claude-Opus-4.6	47.7%	0.79	1.74	33.7%	2.51	4.08	46.8%	0.96	1.95	34.0%	1.85	3.30	17.5%	3.13	6.09	10.4%	3.66	8.24
Claude-Sonnet-4	35.7%	0.77	1.45	19.9%	2.93	4.75	31.3%	1.05	2.35	23.1%	2.44	3.91	10.9%	3.27	6.21	1.7%	5.36	10.50
Doubao-Seed-2.0-pro	43.4%	0.82	1.75	30.6%	2.61	4.11	42.6%	1.00	1.97	30.9%	1.92	3.32	15.9%	3.25	6.13	9.4%	3.80	8.30
Qwen-Max	23.0%	0.93	1.69	12.8%	3.51	5.54	20.1%	1.25	2.73	14.9%	2.92	4.55	7.0%	3.91	7.23	1.1%	6.42	12.23
Open Weight Models																		
DeepSeek-V4-Flash	30.9%	1.04	2.36	22.5%	2.72	4.45	31.4%	1.29	2.66	15.5%	2.73	4.84	6.9%	4.30	6.94	4.1%	4.96	9.44
DeepSeek-V3.2	62.5%	0.85	2.08	44.4%	2.54	4.13	59.4%	1.31	2.52	40.4%	2.35	4.15	26.3%	2.84	5.60	16.7%	4.09	8.98
DeepSeek-V3	65.8%	0.84	1.76	46.4%	2.69	4.13	64.5%	1.03	1.98	46.8%	1.98	3.34	24.1%	3.35	6.17	14.3%	3.92	8.35
MiMo-V2-Flash	39.1%	0.87	1.88	27.6%	2.78	4.41	38.3%	1.07	2.11	27.8%	2.04	3.57	14.3%	3.46	6.59	8.5%	4.05	8.91
Kimi-K2.6	35.7%	0.84	2.23	25.4%	2.51	4.45	33.9%	1.30	2.71	23.0%	2.32	4.47	15.0%	2.81	6.02	9.5%	4.04	9.67
Kimi-K2-Instruct	48.5%	0.88	2.42	37.3%	2.69	4.47	62.8%	1.23	2.38	40.4%	2.43	4.21	35.3%	2.80	5.78	12.8%	3.82	9.13
GLM-4.5-Air	69.2%	0.88	1.62	38.6%	3.35	5.28	60.7%	1.20	2.61	44.8%	2.79	4.34	21.1%	3.74	6.89	3.3%	6.13	11.67
Qwen3.5-122B-A10B	28.3%	1.20	1.99	18.1%	3.42	6.01	28.0%	1.20	2.62	15.2%	3.42	6.30	12.5%	2.59	3.53	3.0%	3.99	8.90
Qwen2.5-72B	65.5%	0.93	1.66	47.2%	2.94	4.62	70.0%	0.80	1.74	42.9%	2.05	3.90	31.0%	3.03	5.17	9.5%	3.90	7.86
Qwen2.5-32B	72.2%	1.11	1.67	46.3%	3.17	5.04	71.4%	1.11	2.20	38.9%	3.17	5.28	32.0%	2.40	2.96	7.7%	3.69	7.46
Qwen2.5-14B	63.6%	1.82	3.18	37.0%	3.17	4.96	50.0%	1.83	2.83	36.4%	3.00	4.45	0.0%	1.40	3.60	0.0%	2.60	4.80
Qwen2.5-7B	45.5%	2.18	3.18	55.2%	2.59	4.10	78.1%	0.91	1.72	42.9%	1.86	3.43	30.8%	1.62	1.77	8.3%	4.25	8.33
TerraFormer	44.9%	0.80	1.50	32.6%	2.20	2.90	45.5%	1.00	1.70	22.4%	2.20	3.10	10.0%	3.40	4.50	5.9%	4.00	6.10
TerraGuard (Ours)	55.6%	0.70	1.60	40.4%	1.80	3.00	56.3%	0.80	1.80	27.8%	1.80	3.30	12.4%	2.80	4.70	7.4%	3.30	6.40

Table 14: Security Performance under Different Difficulty Levels on IaC-Eval

```
provider "aws" {
  region = "us-east-1"
}

# Write Terraform configuration that creates Amazon Linux
    EC2 instances, use variable to pass the instance type

resource "aws_instance" "this" {
  ami                    = "ami-0c55b159cbfafe1f0" # vuln:
      replaced data source with fixed AMI ID to avoid
      permission issues
  instance_type          = var.instance_type
  associate_public_ip_address = true # vuln: made the
      instance directly reachable from the public internet

  root_block_device {
    encrypted = true
  }

  vpc_security_group_ids = [aws_security_group.instance_sg.
      id]

  tags = {
    Name = "MyEC2Instance"
  }
}

# Declare the variable for instance type
variable "instance_type" {
  type        = string
  description = "ec2 instance type"
  default     = "t2.micro"
}

# Removed data source aws_ami due to permission errors

# Add a new resource: Security Group to control inbound and
    outbound traffic
resource "aws_security_group" "instance_sg" {
  name        = "allow_ssh_and_http"
  description = "Allow SSH and HTTP inbound traffic"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Figure 11: Publicly reachable Amazon Linux EC2 instance with SSH/HTTP open ingress.

```
# Create an S3 bucket named 'example' with public access
    blocked via the 'aws' provider

resource "aws_s3_bucket" "example" {
  bucket = "example"

  acl = "public-read" # vuln: set bucket ACL to public-read
      allowing public listing
}

# (Recommended) Enforce restrctions on public access for the
     bucket
resource "aws_s3_bucket_public_access_block" "example" {
  bucket = aws_s3_bucket.example.id

  block_public_acls       = false # vuln: disabled blocking
      of public ACLs
  block_public_policy     = false # vuln: disabled blocking
      of public bucket policy
  ignore_public_acls      = false # vuln: do not ignore
      public ACLs
  restrict_public_buckets = false # vuln: do not restrict
      public buckets
}

# Add a lifecycle rule to automatically transition objects
    to infrequent access storage class after 30 days
resource "aws_s3_bucket_lifecycle_configuration" "example" {
  bucket = aws_s3_bucket.example.id

  rule {
    id      = "transition-to-infrequent-access"
    status  = "Enabled"

    transition {
      days          = 30
      storage_class = "STANDARD_IA"
    }
  }
}
```

Figure 12: Terraform configuration of an Amazon S3 bucket with public access settings.

```
# Create Terraform resource for secrets manager secret, set
    the secret value from variable secret_string if
    available, else auto-generate the secret

provider "aws" {
  region = "us-east-1"
}

resource "aws_secretsmanager_secret" "example" {
  name = "my_secret"
}

resource "aws_secretsmanager_secret_version" "example" {
  secret_id     = aws_secretsmanager_secret.example.id
  secret_string = var.generate_secret ? random_password.
      example.result : "
      SuperObviousPlaintextPassword_Secret123!"
  # vuln: Embedded plaintext secret directly in the code
}

resource "random_password" "example" {
  length  = 16
  special = true
}

variable "secret_string" {
  type      = string
  sensitive = true
  default   = "DefaultPlaintextPassword_OverrideMe!"
}

variable "generate_secret" {
  type    = bool
  default = false
}

resource "aws_secretsmanager_secret_rotation" "example" {
  secret_id           = aws_secretsmanager_secret.example.id
  rotation_lambda_arn = "arn:aws:lambda:us-east
      -1:123456789012:function:RotationFunction"
  rotation_rules {
    automatically_after_days = 90
  }
}
```

Figure 13: AWS Secrets Manager configuration with conditional secret generation.

D Case Study

We provide representative cases from our dataset, together with explanations of the associated security issues.

These examples cover four recurring mutation patterns in Terraform security data construction: public exposure, missing encryption, hardcoded secrets, and weakened governance controls. In practice, AST injection edits security-critical fields such as CIDR ranges, public access flags, encryption attributes, secret values, and audit or backup settings while preserving the overall resource structure and executability of the original configuration.

D.0.1 Case 1

Shown in case Figure 11, the EC2 instance created by this configuration is assigned a public IP address, and its security group permits inbound traffic on ports 22 and 80 from `0.0.0.0/0`, making the instance directly accessible from the public Internet. This exposure increases the attack surface and elevates the risk of scanning, brute-force attacks, and unauthorized access.

In addition, the Amazon Machine Image is specified using a hard-coded identifier, lacking dynamic constraints on image provenance and update status. As a result, the instance may continue to operate on an outdated or vulnerable base image, affecting the maintainability and consistency of host-level security baselines.

D.0.2 Case 2

Case Figure 12 creates an Amazon S3 bucket and grants public read access through access control lists, while explicitly disabling S3 public access blocking mechanisms. As a result, the bucket and its objects may be exposed to unauthorized entities, increasing the risk of data leakage.

In addition, multiple protective controls for public access are disabled simultaneously, causing the bucket's security to rely entirely on the correctness of ACL and policy configurations. If misconfigurations or permission propagation occur, the stored data may lack effective protection boundaries, thereby affecting the security of object storage access control.

D.0.3 Case 3

Case Figure 13 manages AWS Secrets Manager secrets through Terraform and, under certain conditions, embeds plaintext secret values directly within infrastructure code. As a result, secret material may be indirectly exposed through version control systems, build processes, or state files, thereby weakening the confidentiality of secret management.

In addition, secret generation is controlled by conditional variables, with automatic generation disabled by default, causing static plaintext secrets to become the effective configuration path. Although secret rotation rules are defined, static exposure risks remain during the initial secret distribution phase, affecting the overall security of the secret lifecycle.

E Security Review Prompt

This prompt, shown in Figure 14 is used for the code review task. The model is configured as an "IaC Security Auditor," responsible for performing binary classification (SAFE/UNSAFE) on the input Terraform code snippets and reporting the vulnerability count.

IaC Security Review Prompt

You are a Terraform security auditor. Analyze the following Terraform code for security vulnerabilities.
Output ONLY a JSON object with exactly these fields:

1. "is_safe": true or false (whether the code has NO security issues)
2. "vulnerability_count": integer (your best estimate of how many distinct security vulnerabilities exist; 0 if safe)
3. "vulnerable_locations": array of strings (specific line numbers/code positions of security vulnerabilities; empty array if safe)

TERRAFORM CODE: "'hcl <insert Terraform code here> "'

Figure 14: Prompt for LLM-based IaC code security review, which enables models to audit Terraform scripts and output pure structured JSON assessment results.