Security-First Evaluation of Text-to-Terraform: Benchmarking LLMs and SLMs for Secure IaC Generation

Francis Luis Santos Vargas[1], **Rodrigo Brandão Mansilha**[1], **Diego Kreutz**[1]

[1]AI Horizon Labs, PPGES, Universidade Federal do Pampa (UNIPAMPA)

{francisvargas.aluno,rodrigomansilha,diegokreutz}@unipampa.edu.br

Abstract. *Cloud misconfiguration remains a leading cause of security incidents, yet whether LLMs and SLMs can generate security-compliant Infrastructure-as-Code is an open question. We benchmark seven models, three closed LLMs (Claude Opus 4, GPT-5.4, Gemini 2.5 Pro) and four open SLMs (Qwen2.5-Coder-14B, WizardCoder-33B, CodeLlama-13B, Magicoder-S-CL-7B), on AWS Terraform generation across 17 scenarios, integrating Checkov and Trivy scanners into a GitLab CI/CD pipeline and evaluating two prompt strategies at three security levels (pass@5). Syntactic validity and security compliance are largely orthogonal properties in LLM-generated IaC, a model that reliably produces well-formed Terraform does not necessarily produce secure Terraform: WizardCoder-33B achieves 77.8% validate rate yet zero Checkov compliance, while Claude Opus 4 reaches 23.1% Checkov and 92.5% Trivy pass rates under detailed security prompting. Consequently, prompt engineering alone is insufficient: automated multi-tool scanning remains a necessary complement to LLM-assisted IaC generation regardless of model family or prompt strategy. All artifacts are publicly available.*

1. Introduction

Infrastructure-as-Code (IaC) has become the *de facto* paradigm for cloud resource provisioning, enabling reproducibility, version control, and automated deployment [Morris 2020]. Among IaC technologies, Terraform has emerged as the most widely adopted solution. Yet cloud misconfiguration remains a critical and persistent threat: the Cloud Security Alliance ranks it as the top cloud risk in 2024[1], and industry analyses attribute nearly 23% of cloud security incidents to incorrectly configured resources [SentinelOne 2024]. Even human-authored IaC frequently introduces security smells and policy violations [Rahman et al. 2019, Saavedra and Ferreira 2022, Verdet et al. 2025], and large-scale analyses of S3, VPC, and KMS deployments show that access control and encryption misconfigurations remain major sources of data exposure [Continella et al. 2018].

LLMs and SLMs have rapidly become central to software development, with growing adoption in code generation and infrastructure automation. Yet AI-generated code consistently exhibits security weaknesses in security-sensitive scenarios [Pearce et al. 2022, Perry et al. 2023, Fang et al. 2024]. Existing IaC benchmarks, IaC-Eval, DPIaC-Eval, and TerraFormer, advance the state of the art in functional correctness and syntactic validity, but leave a critical question open: *can models that generate*

[1]https://cloudsecurityalliance.org/research/topics/top-threats

valid Terraform also generate secure Terraform? We address this gap with four research questions:

RQ1: Syntactic validity and security compliance. *Can state-of-the-art LLMs and locally deployable SLMs generate syntactically valid and security-compliant AWS Terraform, and are these two properties correlated?* We answer RQ1 in Sections 4.1 and 4.2.

RQ2: Impact of prompt security specificity. *Does increasing prompt security specificity substantially improve security compliance, and does this effect differ between LLMs and SLMs?* We answer RQ2 in Section 4.2.

RQ3: Scanner-feedback self-correction. *Can LLMs and SLMs self-correct security-non-compliant Terraform configurations when provided with raw scanner feedback, and does correction capability differ between Checkov and Trivy findings?* We answer RQ3 in Section 4.3.

RQ4: Longitudinal evolution. *Have SLM capabilities for IaC generation and security compliance improved between IaC-Eval (2024) and this evaluation (2026)?* We answer RQ4 in Section 4.4.

We present a security-first empirical benchmark combining automated syntactic validation with dual-tool static security analysis (Checkov and Trivy) in a fully reproducible GitLab CI/CD pipeline. Our main contributions are: (i) a reproducible benchmark with LLM-consensus scenario selection across 17 scenarios and three AWS resource families; (ii) the first dual-scanner (Checkov + Trivy) IaC benchmark, evaluating seven models under two prompt strategies at three security levels with pass@5; (iii) first evidence that syntactic validity, plan executability, and security compliance form three partially independent capability dimensions in LLM-generated IaC, none of which is a reliable proxy for the others; and (iv) the first longitudinal comparison of SLMs under both functional correctness and security compliance criteria.

2. Related Work

Security of human-authored IaC. Prior work consistently shows that human-written IaC frequently violates security best practices. Rahman et al. [Rahman et al. 2019] identified recurring security smells in Puppet, later replicated in Ansible, Chef, and Kubernetes [Rahman et al. 2021, Rahman et al. 2023]. GLITCH [Saavedra and Ferreira 2022] extended smell detection across multiple IaC languages, while Opdebeeck et al. [Opdebeeck et al. 2023] quantified smell prevalence in Ansible repositories. Verdet et al. [Verdet et al. 2025] found widespread Terraform policy non-compliance across cloud providers, and Continella et al. [Continella et al. 2018] showed how S3 misconfigurations enable data exposure and resource injection. Together, these studies establish misconfiguration as endemic in IaC and motivate automated security analysis as a baseline requirement for AI-generated IaC.

Security of LLM-generated code. A parallel line of research investigates whether AI code generators inherit or amplify these weaknesses. Pearce et al. [Pearce et al. 2022] showed that GitHub Copilot generates vulnerable code in 40% of security-sensitive scenarios. Perry et al. [Perry et al. 2023] found that developers assisted by AI produce significantly more insecure code, while Fang et al. [Fang et al. 2024] identified major limitations in LLM vulnerability reasoning. Although these studies establish a strong ex-

pectation of security risk in LLM-generated code, none focus on IaC, security-oriented prompting, or LLM versus SLM comparisons.

LLM-based IaC generation benchmarks. Benchmarking of LLM-generated IaC has expanded rapidly, although security remains secondary in most studies. IaC-Eval [Kon et al. 2024] showed that Terraform generation remains substantially harder than general code generation. Multi-IaC-Eval [Davidson et al. 2025] expanded evaluation across multiple IaC frameworks, but without security scanning. DPIaC-Eval [Zhang et al. 2026] introduced iterative correction, yet reported only 8.4% security compliance after multiple feedback rounds. Nekrasov et al. [Nekrasov et al. 2026] improved Terraform syntactic validation using RAG, while TerraFormer [Jana et al. 2026] combined fine-tuning with formal verification feedback for partial security gains. Overall, security evaluation is usually absent or secondary to functional correctness, complementary scanners are not combined, and SLMs remain largely excluded from security-focused evaluation.

Positioning this work. Table 1 summarises prior work across eight dimensions. The literature above reveals four gaps that motivate the present study. First, *multi-tool security coverage*: no previous IaC benchmark integrates complementary scanners. Checkov evaluates policy compliance, including encryption, access control, and logging, whereas Trivy evaluates vulnerability severity through CRITICAL/HIGH/MEDIUM CVEs. Our results show that these dimensions are complementary rather than interchangeable. Second, *LLM and SLM comparison under security criteria*: prior benchmarks either focus exclusively on LLMs or evaluate SLMs only under functional correctness criteria. As a result, it remains unclear whether the syntactic improvements observed in SLMs between 2024 and 2026 also translated into security improvements. Third, *security-oriented prompt specificity*: only DPIaC-Eval and Multi-IaC-Eval partially vary prompting strategies, but neither study systematically isolates the effect of security-specific prompting across both LLMs and SLMs. Fourth, *selective feedback revision*: previous iterative correction strategies apply scanner feedback uniformly to all models. In contrast, our approach triggers scanner-feedback revision only for models whose compliance falls below a predefined threshold, isolating the revision effect where security deficiencies are more pronounced. Together, these gaps motivate a security-first benchmark in which Checkov and Trivy compliance, rather than `terraform validate`, define the primary evaluation criterion, while both LLMs and SLMs are evaluated longitudinally within a fully reproducible CI/CD pipeline.

3. Methodology

3.1. Benchmark Pipeline

Figure 1 illustrates the benchmark pipeline, implemented as GitLab CI/CD jobs running in parallel across all evaluated models. Stages 1–5 run automatically; Stage 6 (Revise) is triggered manually and applied selectively to models whose Round 1 Checkov or Trivy pass rate at L3 falls below 5%, isolating the revision effect where compliance gaps are most pronounced.

Table 1. Positioning of this work relative to prior studies. ✓ = fully addressed; ∼ = partially addressed; – = not addressed or not applicable.

Work	Venue	Scope	LLM	SLM	#Models	Feedback loop	Scanner / Tool
Continella et al. [Continella et al. 2018]	ACSAC 2018	S3 misconfiguration	–	–	–	–	custom scanner
Rahman et al. [Rahman et al. 2019]	ICSE 2019	IaC smells (Puppet)	–	–	–	–	SLIC†
Rahman et al. [Rahman et al. 2021]	TOSEM 2021	IaC smells (Ansible/Chef)	–	–	–	–	SLAC†
Saavedra & Ferreira [Saavedra and Ferreira 2022]	ASE 2022	IaC smells (multi-lang)	–	–	–	–	GLITCH†
Rahman et al. [Rahman et al. 2023]	TOSEM 2023	IaC misconfig (K8s)	–	–	–	–	empirical
Opdebeeck et al. [Opdebeeck et al. 2023]	MSR 2023	IaC smells (Ansible)	–	–	–	–	GASEL†
Verdet et al. [Verdet et al. 2025]	EMSE 2025	IaC security (Terraform)	–	–	–	–	Checkov + Tfsec
Pearce et al. [Pearce et al. 2022]	IEEE S&P 2022	LLM code security	✓	–	1	–	CodeQL + manual
Perry et al. [Perry et al. 2023]	ACM CCS 2023	LLM code security	✓	–	1	–	manual
Fang et al. [Fang et al. 2024]	USENIX 2024	LLM code analysis	✓	–	5	–	manual
IaC-Eval [Kon et al. 2024]	NeurIPS 2024	IaC generation (Terraform)	✓	✓	11	–	OPA
DPIaC-Eval [Zhang et al. 2026]	ACM FSE 2026	IaC generation (CFN)	✓	–	6	∼	Checkov
Multi-IaC-Eval [Davidson et al. 2025]	arXiv 2025	IaC generation (CFN/TF/CDK)	✓	∼	3	–	CFN-Lint
Nekrasov et al. [Nekrasov et al. 2026]	ACM TOSEM 2026	IaC generation + RAG	✓	–	2	–	–
TerraFormer [Jana et al. 2026]	ICSE-SEIP 2026	IaC generation + fine-tuning	✓	✓	17	✓	Checkov
This work	**—**	**IaC security + gen. (Terraform)**	✓	✓	**7**	✓	**Checkov + Trivy**

† SLIC, SLAC, GLITCH, and GASEL are purpose-built static analysis tools introduced by the respective papers for smell detection in IaC scripts; they are not general-purpose security scanners.

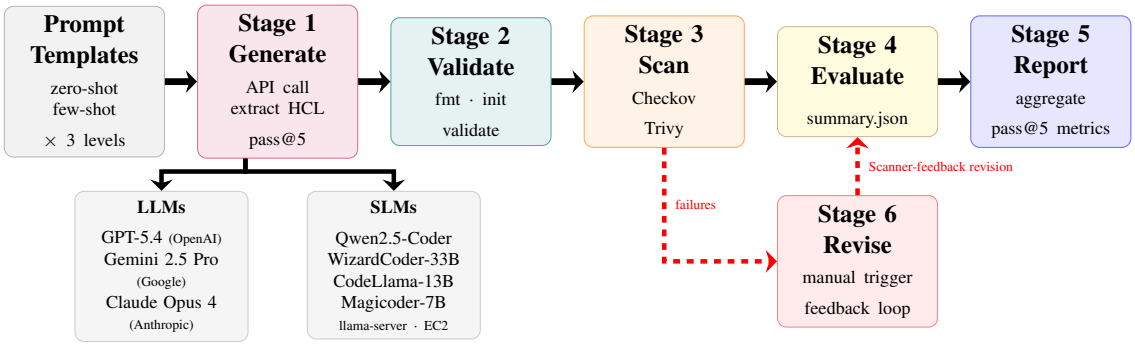

Figure 1. Benchmark pipeline. Stages 1–5 run automatically as GitLab CI/CD jobs in a parallel matrix. Stage 6 is triggered manually and applied only to models with Round 1 pass rate below 5% at L3.

3.2. Models Under Evaluation

Table 3 summarizes the model identifiers, inference configurations, and observed truncation rates of the seven models evaluated in this study.[2]

Three closed-source LLMs GPT-5.4, Gemini 2.5 Pro, and Claude Opus 4 were accessed through REST APIs. Four open-weight SLMs were served through `llama-server` on an AWS EC2 `c8g.16xlarge` instance equipped with AWS Graviton4 ARM64 processors, 64 vCPUs, and 128 GB of memory. All SLM inference was performed entirely on CPU, without GPU acceleration or model-layer offloading. The SLM subset was selected to support direct comparison with IaC-Eval [Kon et al. 2024]. Pipeline tool versions are reported in Table 2.

Unspecified decoding parameters. Temperature and top-p were not explicitly configured in either the closed-model API requests or the `llama-server` commands. Accordingly, the experiment followed the default decoding behavior implemented by each provider or inference framework at evaluation time. We report these parameters as *default*, rather than assigning retrospective values that were not recorded in the experimen-

[2]Exact model files, API request implementations, prompts, generated artefacts, and infrastructure definitions are available in the public repository.

Table 2. Pipeline tool versions.

Tool	Version	Role
Terraform	1.7.5	`fmt`, `init`, `validate`, and `plan`
AWS Provider	5.100.0	Resolved during `terraform init`
Checkov	3.2.527	`CKV_AWS_`$\star$ compliance scanner
Trivy	0.69.3	`AVD-AWS-`$\star$ misconfiguration scanner
GitLab CE/EE	19.0.0-pre	CI/CD orchestration

All pipeline tools ran inside Docker containers on an AWS EC2 `c8g.16xlarge` instance using the `linux/arm64` architecture.

Table 3. Model identifiers, inference configurations, and observed truncation rates

Model	Weights / endpoint	Quant.	Temp.	Top-p	Output cap	n_{ctx}	Trunc.
GPT-5.4	`gpt-5.4`	–	*default*	*default*	12,000	–	0.0%
Gemini 2.5 Pro	`gemini-2.5-pro`	–	*default*	*default*	12,000	–	0.0%
Claude Opus 4	`claude-opus-4-5`	–	*default*	*default*	12,000	–	0.0%
WizardCoder-33B	`wizardcoder-33b-v1.1-q4_k_m.gguf`	Q4_K_M	*default*	*default*	12,000	8,192	0.0%
Qwen2.5-Coder-14B	`qwen2.5-coder-14b-q4_k_m.gguf`	Q4_K_M	*default*	*default*	12,000	4,096	0.0%
CodeLlama-13B	`codellama-13b-instruct-q4_k_m.gguf`	Q4_K_M	*default*	*default*	12,000	4,096	**32.7%**
Magicoder-S-CL-7B	`magicoder-s-cl-7b`	FP16	*default*	*default*	12,000	4,096	2.9%

Parameters marked as default *were not explicitly supplied in the API requests or `llama-server` commands; therefore, generation followed the default decoding behaviour of the corresponding provider or inference framework.* Trunc. *denotes the fraction of generations terminated because the configured output or context limit was reached. All SLM inference ran entirely on CPU using an AWS EC2 `c8g.16xlarge` instance equipped with AWS Graviton4 ARM64 processors. No GPU acceleration or model-layer offloading was used. For SLMs, the effective output length was bounded by the available context window after accounting for the input prompt.*

tal artefacts. No random seed was explicitly fixed. Instead, the experimental protocol used $k = 5$ independent generations for each model, scenario and prompt combination to capture output variability.

Observed truncation. CodeLlama-13B reached its configured 4,096-token context limit in 501 of 1,530 recorded generations (32.7%), as indicated by `finish_reason=length` in the raw generation artefacts. Magicoder-S-CL-7B reached the same limit in 45 generations (2.9%). The resulting outputs were forwarded to Terraform validation and security scanning without repair, continuation, or manual post-processing. Consequently, truncation may partly explain CodeLlama's elevated validation-failure rate and should be considered when comparing its results with models evaluated under larger effective context windows.

3.3. Scenarios and Prompt Strategies

We evaluate 17 scenarios spanning three AWS resource families (S3, KMS, VPC), drawn from the IaC-Eval dataset [Kon et al. 2024] and filtered via unanimous agreement (6/6) among six LLM evaluators (Claude, GPT, Gemini Pro, DeepSeek, Mistral, Grok). The unanimous threshold ensures that only scenarios with unambiguous security relevance enter the benchmark, avoiding the inclusion of borderline cases that could dilute the security signal.

Each scenario is evaluated under two prompt strategies at three security levels, yielding six variants per scenario. **Zero-shot** provides only the task description; **few-shot** adds one security-compliant example configuration. **L1** (Without Security) serves as a baseline measuring spontaneous security behaviour; **L2** (Basic Security) adds general guidelines (encryption, public access blocking, TLS); **L3** (Detailed Security) explicitly

specifies Checkov (`CKV_AWS_*`, `CKV2_AWS_*`) and Trivy (`AVD-AWS-*`) identifiers with required resource attributes.

3.4. Evaluation Metrics and Statistical Analysis

Metrics span four dimensions. *Syntactic validity*: `terraform fmt`, `validate`, and `plan` pass rates, with validate treated as a hard prerequisite for security analysis. *Security compliance*: Checkov pass (zero `CKV_AWS_*` failures) and Trivy pass (no CRITICAL/HIGH/MEDIUM `AVD-AWS-*` findings), computed exclusively over validate-passing generations. *Robustness*: empirical pass@5 over $k = 5$ independent runs per cell, following Chen et al. [Chen et al. 2021] but reporting empirical rates rather than the unbiased estimator. *Cost efficiency*: cost per run and cost per compliant generation, using provider token pricing for LLMs.

All pass rates are accompanied by 95% Wilson score confidence intervals [Wilson 1927]. Pairwise model comparisons use Pearson's chi-square on 2×2 contingency tables with Bonferroni correction ($\alpha' = 0.05/21 = 0.0024$). All reported χ^2 and p-values are post-correction.

Plan executability. `terraform plan` is executed over all validate-passing generations as an additional syntactic correctness check beyond `terraform validate`. For the three LLMs, plan was executed as a post-processing step directly over the validated `main.tf` artifacts produced by the main pipeline, using the same Terraform (v1.7.5) and AWS provider (v5.100.0) versions on the same EC2 instance. No additional generation was performed; plan results reflect the same artifacts evaluated by Checkov and Trivy. Plan% is always computed over the validate-passing subset only.

4. Results

All security metrics (Checkov, Trivy) are computed exclusively over generations that passed `terraform validate`. Configurations that fail validation are recorded separately and excluded from scanner aggregations.

4.1. Syntactic Validity and Plan Executability

Table 4 reports validate and plan pass rates per model out of 510 total generations (17 scenarios × 6 variants × 5 runs).

Table 4. Validate and plan pass rates (510 total generations).

Model	Type	Valid/510	Val% [95% CI]	Plan/Valid	Plan% [95% CI]
GPT-5.4	LLM	435	85.3 [82.0–88.1]	318/435	73.1 [68.7–77.1]
Claude Opus 4	LLM	410	80.4 [76.7–83.6]	363/410	88.5 [85.1–91.3]
Gemini 2.5 Pro	LLM	401	78.6 [74.9–82.0]	303/401	75.6 [71.1–79.5]
WizardCoder 33B	SLM	397	77.8 [74.0–81.2]	324/397	81.6 [77.5–85.1]
Qwen2.5-Coder 14B	SLM	249	48.8 [44.5–53.2]	205/249	82.3 [77.1–86.6]
CodeLlama 13B	SLM	132	25.9 [22.3–29.9]	124/132	93.9 [88.5–96.9]
Magicoder 7B	SLM	108	21.2 [17.9–24.9]	82/108	75.9 [67.1–83.0]

Plan% over validate-passing subset. CIs: Wilson 95% 95% in brackets.

Validate and plan rates are partially decoupled. GPT-5.4 leads in validate (85.3%) yet records the lowest plan pass rate among LLMs (73.1%); Claude Opus 4

ranks second in validate but first in plan executability (88.5%). Among SLMs, CodeLlama 13B achieves the highest plan pass rate of all seven models (93.9% [88.5–96.9]) despite the second-lowest validate rate, 124 of its 132 validate-passing generations also pass `terraform plan`. Qwen2.5-Coder 14B similarly achieves 82.3% plan pass rate at only 48.8% validate, indicating its failures concentrate in structurally malformed outputs rather than incomplete configurations.

The dominant plan failure is `No value for required variable` (346 of 413 total failures): models declare `variable` blocks without `default` values, causing `terraform plan` to abort without `-var-file`. Qwen2.5-Coder 14B exhibits this exclusively (44/44 failures). LLMs show two additional categories: `Error in function call` (Gemini: 19, GPT-5.4: 25, from incorrect `jsonencode`/`toset`/`flatten` usage) and `Self-referential block` (GPT-5.4: 4, mutually dependent KMS+CloudWatch configurations, the same pattern seen in validation failures, Section 4.6).

Effect of few-shot prompting. Table 5 shows validate and plan pass counts by prompt strategy. For LLMs, few-shot has negligible or slightly negative effect on validate (-2 to -13 generations), while benefiting the two smallest SLMs substantially (+32 for CodeLlama 13B, +20 for Magicoder 7B). Plan executability shows a sharper pattern: few-shot *degrades* plan pass counts for all LLMs (GPT: -52, Gemini: -51, Claude: -37), driven by `No value for required variable`, the few-shot example's modular structure induces variable declarations without defaults. The two smallest SLMs again reverse this trend (+26 and +16 plan passes).

Table 5. Validate and plan pass counts: zero-shot vs. few-shot (255 generations each). Plan over validate-passing only.

Model	Type	Validate			Plan		
		Zero	Few	Δ	Zero	Few	Δ
Claude Opus 4	LLM	206	204	−2	200	163	−37
Gemini 2.5 Pro	LLM	207	194	−13	177	126	−51
GPT-5.4	LLM	220	215	−5	185	133	−52
WizardCoder 33B	SLM	207	190	−17	170	154	−16
Qwen2.5-Coder 14B	SLM	130	119	−11	113	92	−21
CodeLlama 13B	SLM	50	82	**+32**	49	75	**+26**
Magicoder 7B	SLM	44	64	**+20**	33	49	**+16**

Out of 255 per strategy. Plan Δ over validate-passing subset.

Answer to RQ1, Part 1 — syntactic validity. LLMs produce at least one valid generation in all 102 scenario and prompt cells (validate 78.6–85.3%). SLMs span 21.2–77.8%, with CodeLlama 13B and Magicoder 7B leaving 29 and 42 cells respectively with no scannable output. Critically, validate rate does not predict plan executability: CodeLlama 13B ranks seventh in validate yet first in plan pass rate (93.9% [88.5–96.9]), while GPT-5.4 leads in validate and trails in plan (73.1% [68.7–77.1]).

4.2. Effect of Prompt Security Level on Security and Compliance

We restrict security analysis to the *shared valid subset* (Inter): scenarios for which the model produced at least one valid generation at all three levels, ensuring L1/L2/L3

comparisons reflect genuine prompt-driven differences rather than scenario composition changes. Table 6 reports results with 95% Wilson CIs.

Table 6. Security compliance over the shared valid subset. *Inter* = scenarios with at least one validate-passing generation at all three security levels (L1, L2, and L3); values represent pass counts over Inter. 95% Wilson CIs shown for L3. Bold = best per column group.

Model	Type	Inter	Checkov / Inter [95% CI L3]				Trivy / Inter [95% CI L3]			
			L1	L2	L3	CI	L1	L2	L3	CI
Claude Opus 4	LLM	134	0	4	**31 (23.1%)**	[16.8–31.0]	4	102	**124 (92.5%)**	[86.8–95.9]
Gemini 2.5 Pro	LLM	106	0	0	2 (1.9%)	[0.5–6.6]	5	78	84 (79.2%)	[70.6–85.9]
GPT-5.4	LLM	141	0	0	2 (1.4%)	[0.4–4.9]	2	51	24 (16.4%)	[11.3–23.3]
WizardCoder 33B	SLM	121	0	0	0 (0.0%)	[0.0–3.1]	0	1	1 (0.8%)	[0.1–4.5]
Qwen2.5-Coder 14B	SLM	82	0	0	0 (0.0%)	[0.0–4.5]	0	2	6 (7.3%)	[3.4–15.1]
CodeLlama 13B	SLM	14	1	0	1 (7.1%)	[1.3–31.5]	0	1	1 (7.1%)	[1.3–31.5]
Magicoder 7B	SLM	10	0	0	0 (0.0%)	[0.0–27.8]	0	0	0 (0.0%)	[0.0–27.8]

Chi-square (Bonferroni $\alpha' = 0.0024$): Claude vs. GPT-5.4 Trivy L3: $\chi^2 = 162.4$, $p < 0.001$; Checkov L3: $\chi^2 = 31.8$, $p < 0.001$. All LLM–SLM comparisons at L3: $p < 0.001$. CodeLlama/Magicoder CIs wide ($n \leq 14$) — directional only. Cramér's V: Claude vs. GPT-5.4 Trivy L3: $V = 0.76$ (large); Checkov L3: $V = 0.34$ (medium). All LLM–SLM pairs at L3: $V > 0.45$.

LLM compliance responds strongly to prompt specificity. Claude Opus 4 shows the clearest progression: Checkov 0→4→31 and Trivy 4→102→124, the strongest security profile in the benchmark. Gemini 2.5 Pro reaches 79.2% Trivy at L3 despite only 1.9% Checkov, while its Inter subset shrinks from 150 at L1 to 106 at L3, prompt complexity simultaneously reduces syntactic output and improves security. GPT-5.4 peaks at Trivy L2 (51/141) and *declines* at L3 (24/146): detailed security prompts introduce resource constructs (KMS-encrypted CloudWatch log groups) that generate dependency graphs Trivy penalises due to resolution failures. All LLM–SLM differences at L3 are statistically significant ($p < 0.001$, Bonferroni-corrected); the Claude vs. GPT-5.4 Trivy difference alone yields $\chi^2 = 162.4$. A per-scenario breakdown of Trivy compliance at L3 is provided in Table 10 (Section 4.5).

SLM compliance is near-zero and unresponsive to prompting. All four SLMs record near-zero Checkov compliance at all levels. The sole exception is CodeLlama 13B with 1 Checkov pass at L1 and L3, the only non-zero SLM Checkov result in the benchmark, but its Inter of $n = 14$ renders this directional only [1.3–31.5]. WizardCoder 33B achieves 77.8% validate yet zero Checkov passes [0.0–3.1], the clearest evidence that syntactic correctness and security compliance are orthogonal properties. Few-shot prompting has negligible effect on security compliance once the security level is held constant ($\pm$1–4 passes across all models and strategies).

Checkov–Trivy discordance. Table 7 classifies each L3 generation into four compliance states. *No model ever passes Checkov without also passing Trivy*, checkov-only is zero across all models, confirming Checkov is a strictly harder criterion. The dominant LLM state at L3 is Trivy-only: Claude Opus 4 places 69.4% of its L3 generations here, Gemini 77.4%. These configurations pass all CRITICAL/HIGH/MEDIUM vulnerability checks but fail at least one compliance policy, and may represent an acceptable posture in non-production environments. GPT-5.4 diverges sharply: 83.6% fail both tools

vs. 7.5% for Claude and 20.8% for Gemini, consistent with its higher rate of architecturally complex failures. SLMs cluster uniformly in "neither": 91–100% across all four models. The observed discordance also highlights that infrastructure security cannot be adequately characterised by a single analysis tool. Configurations considered acceptable by vulnerability-oriented scanning may still violate organisational security policies enforced by compliance scanners. This finding also explains why previous IaC benchmarks based on a single scanner or validation criterion may overestimate the practical security capability of current LLMs.

Table 7. Checkov–Trivy compliance states at L3 (validate-passing generations, Inter subset).

Model	Type	n	Both	Trv only	Chk only	Neither
Claude Opus 4	LLM	134	31 (23.1%)	93 (69.4%)	0	10 (7.5%)
Gemini 2.5 Pro	LLM	106	2 (1.9%)	82 (77.4%)	0	22 (20.8%)
GPT-5.4	LLM	146	2 (1.4%)	22 (15.1%)	0	122 (83.6%)
WizardCoder 33B	SLM	122	0	1 (0.8%)	0	121 (99.2%)
Qwen2.5-Coder 14B	SLM	94	0	6 (6.4%)	0	88 (93.6%)
CodeLlama 13B	SLM	23	1 (4.3%)	1 (4.3%)	0	21 (91.3%)
Magicoder 7B	SLM	21	0	0	0	21 (100%)

Answer to RQ1, Part 2 — security compliance. Syntactic validity and security compliance are largely *decoupled*. This decoupling suggests that current code-oriented foundation models primarily learn Terraform syntax and resource composition, while security best practices remain only partially represented in their internal knowledge. Consequently, generating syntactically correct IaC appears substantially easier than generating policy-compliant IaC, reinforcing that deployment readiness cannot be inferred from successful Terraform validation alone.: WizardCoder 33B achieves 77.8% validate yet 0.0% Checkov [0.0–3.1]. All LLM–SLM differences at L3 are statistically significant ($p < 0.001$, Bonferroni-corrected).

Answer to RQ2. Prompt specificity substantially improves LLM compliance (Claude: +23.1pp Checkov, +89.6pp Trivy from L1→L3; Gemini: +76.0pp Trivy) but is ineffective for SLMs, where L3 prompts frequently degrade syntactic validity without compliance gains. The binding constraint for SLMs is instruction-following capacity, not prompt content.

4.3. Scanner-Feedback Revision

After Round 1, models received their own raw Checkov and Trivy output as corrective feedback and regenerated the configuration. Stage 6 was applied to all Round 1 generations that produced at least one Checkov or Trivy finding; generations that passed both scanners in Round 1 were not revised, as no corrective feedback was available. This accounts for the smaller revision n observed for Claude Opus 4 and Gemini 2.5 Pro relative to their Round 1 n. Table 8 compares normalised pass rates before and after this single-pass revision.

Trivy compliance improves substantially across most models after feedback: GPT-5.4 gains +43.7pp (17.7%→61.4%), WizardCoder 33B +41.6pp (0.5%→42.1%),

Table 8. Round 1 vs. scanner-feedback revision. Chk% and Trv% = passes / n validate-passing generations.

Model	Type	Round 1			Revision			
		n	Chk%	Trv%	n	Chk	Chk%	Trv%
Claude Opus 4	LLM	410	8.5%	56.8%	346	72	**20.8%**	49.7%
Gemini 2.5 Pro	LLM	401	0.5%	42.9%	341	34	10.0%	39.0%
GPT-5.4	LLM	435	0.5%	17.7%	355	10	2.8%	**61.4%**
WizardCoder 33B	SLM	397	0.0%	0.5%	171	0	0.0%	42.1%
Qwen2.5-Coder 14B	SLM	249	0.0%	3.2%	53	0	0.0%	5.7%
CodeLlama 13B	SLM	132	3.0%	3.8%	52	2	3.8%	26.9%
Magicoder 7B	SLM	108	0.0%	0.9%	38	1	2.6%	13.2%

Trivy decline for Claude/Gemini: subset composition differs between rounds.

CodeLlama 13B +23.1pp. This confirms that Trivy findings, concrete severity labels and resource paths, provide actionable correction signals that models can directly incorporate. Checkov compliance improves only for LLMs: Claude Opus 4 gains +12.3pp (8.5%→20.8%), Gemini +9.5pp. Checkov violations typically require architectural decisions (adding encryption resources, configuring IAM, enabling logging) that a single-pass revision cannot fully resolve. The Trivy decline for Claude and Gemini reflects a subset composition artefact: the revision subset ($n = 346$ and $n = 341$) excludes the 64 and 60 Round 1 generations that already passed both scanners and therefore received no corrective feedback. Since these clean-passing generations are removed from the revision denominator, the revision Trivy rate is computed over a strictly harder subset than Round 1, producing an apparent decline that does not reflect genuine degradation.

Answer to RQ3. Models can partially self-correct security issues when provided with raw scanner output, but correction capability differs systematically between tools and model families. For Trivy, self-correction is broadly effective: GPT-5.4 gains +43.7pp (17.7%→61.4%), WizardCoder 33B +41.6pp (0.5%→42.1%), and CodeLlama 13B +23.1pp (3.8%→26.9%). This confirms that Trivy findings, concrete severity labels with affected resource paths, provide actionable signals that models can directly incorporate into a revised configuration. For Checkov, self-correction is limited to LLMs: Claude Opus 4 gains +12.3pp (8.5%→20.8%) and Gemini 2.5 Pro +9.5pp (0.5%→10.0%), while GPT-5.4 and all SLMs show minimal gains. Checkov violations require architectural decisions, adding encryption resources, configuring IAM policies, enabling logging, that a single-pass revision cannot resolve without restructuring the configuration. The asymmetry between Trivy and Checkov self-correction thus reflects the structural difference between localised vulnerability findings and systemic compliance gaps: models can act on the former but not reliably on the latter.

4.4. Longitudinal Comparison with IaC-Eval

Three SLMs were previously evaluated in IaC-Eval [Kon et al. 2024] (NeurIPS 2024) under a functional correctness criterion assessed via `terraform plan` and OPA Rego policies. Our evaluation reuses 17 of the 458 IaC-Eval scenarios and extends assessment with Checkov and Trivy. The IaC-Eval dataset encodes a *generation difficulty* label per scenario, but this axis does not translate to security: a scenario easy to implement correctly may be consistently generated insecurely, as confirmed by WizardCoder 33B's 77.8%

validate rate at zero Checkov compliance. Table 9 places both benchmarks side by side.

Table 9. Longitudinal comparison with IaC-Eval [Kon et al. 2024] (458 scenarios, pass@1, functional correctness). This work: 17 scenarios, pass@5, syntactic validity + plan executability + security. † = model also in IaC-Eval.

Model	Type	IaC-Eval	Val% pass@1	Plan%	Chk/n (L3)	Trv/n (L3)
Claude Opus 4	LLM	—	80.4%	88.5%	31/134	124/134
Gemini 2.5 Pro	LLM	—	78.6%	75.6%	2/106	84/106
GPT-5.4	LLM	—	85.3%	73.1%	2/146	24/146
WizardCoder 33B†	SLM	8.93%	77.8%	81.6%	0/121	1/121
Qwen2.5-Coder 14B	SLM	—	48.8%	82.3%	0/ 82	6/ 82
CodeLlama 13B†	SLM	2.01%	25.9%	93.9%	1/ 14	1/ 14
Magicoder 7B†	SLM	7.62%	21.2%	75.9%	0/ 10	0/ 10

Plan% for LLMs: results_plan_llms.json (this work).

Syntactic improvement does not imply security improvement. Validate rates are substantially higher than IaC-Eval pass@1 for the same SLMs (WizardCoder: 77.8% vs. 8.93%; Magicoder: 21.2% vs. 7.62%), as expected: IaC-Eval requires functional correctness in addition to compilation, a strictly harder criterion applied over the full 458-scenario dataset vs. our 17-scenario subset.

Plan executability inverts the validate ranking. CodeLlama 13B achieves 93.9% plan pass rate despite the second-lowest validate rate, while GPT-5.4 leads in validate yet has the lowest plan pass rate among LLMs (73.1%). This inversion confirms that validate rate, plan executability, and security compliance are three distinct and partially independent properties, none is a reliable proxy for the others.

Security capability has not improved alongside generation quality. None of the SLMs previously evaluated in IaC-Eval achieve meaningful security compliance: WizardCoder 33B, the highest-ranked SLM in IaC-Eval (8.93%), records 0 Checkov and only 1 Trivy pass out of 121 valid L3 generations. Magicoder 7B and CodeLlama 13B show equally marginal results. The IaC generation capability improvements observed between 2023 and 2026 have not translated into security-aware generation for SLMs.

Answer to RQ4. Between 2024 and 2026, SLMs became better at generating Terraform; they did not become better at generating *secure* Terraform. Syntactic generation improved substantially, but security compliance remained near-zero and unresponsive to prompt specificity, the binding constraint is model architecture and training, not prompting strategy. These findings suggest that recent advances in code-generation models have predominantly improved Terraform syntax generation rather than security-aware infrastructure design. Future improvements in secure IaC generation will likely depend more on security-specific training, policy-guided optimisation, or retrieval mechanisms than on increasing general model capability alone.

4.5. Per-Scenario Compliance Pattern

Table 10 reports Trivy compliance at L3 (Detailed Security) per model and scenario pair. Each cell shows *pass/valid*, where *pass* is the number of validate-passing generations that also passed Trivy, and *valid* is the total validate-passing generations at L3 across both prompt strategies and 5 runs (maximum 10 per cell). Cells where *valid* < 10 reflect

validate failures at L3 for that model and scenario combination; cells marked "—" indicate zero validate-passing generations.

Table 10. Trivy pass/valid at L3 per model–scenario. *pass* = validate-passing generations that passed Trivy; *valid* = total validate-passing generations at L3 (max 10 = 2 strategies × 5 runs; lower values reflect validate failures for that model–scenario). S1–S9: S3/KMS; S10–S17: VPC. ≥75% / 50–74% / 1–49% / 0% / — = no validate-passing generations at L3.

Model	T	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17
Claude	L	**8/8**	**8/8**	**8/8**	**8/8**	**8/8**	**7/7**	**7/8**	**8/8**	**8/8**	5/7	**8/8**	**8/8**	**8/8**	**8/8**	**8/8**	**6/8**	3/8
Gemini	L	4/7	**10/10**	**8/8**	**8/9**	**8/10**	**9/9**	**5/6**	**7/7**	**4/4**	0/3	2/3	2/4	4/7	1/2	**9/10**	**2/2**	1/5
GPT-5.4	L	0/8	**7/8**	**7/8**	**6/8**	0/10	1/8	0/10	0/10	0/10	0/8	0/9	0/10	0/7	0/8	0/8	2/8	0/8
WizardCdr	S	0/8	0/9	1/7	0/8	0/5	0/9	0/1	0/9	0/1	0/7	0/8	0/10	0/9	0/9	0/7	0/7	0/8
Qwen	S	0/5	0/5	3/6	0/4	2/9	1/6	0/1	0/6	0/4	0/3	—	0/10	0/9	0/10	—	0/8	0/8
CodeLlama	S	—	0/1	0/3	0/1	0/3	0/1	—	**1/1**	**1/1**	0/2	0/4	0/1	0/3	—	0/1	—	0/1
Magicoder	S	—	0/1	—	0/3	0/3	0/3	0/1	0/1	—	—	0/1	0/1	0/1	0/2	0/1	0/1	0/2

T = model type (L = LLM, S = SLM). Scanning applied only to validate-passing generations; validate failures reduce the denominator below 10.

Two structural patterns emerge. First, *compliance difficulty is scenario-driven, not only model-driven*: S17 (VPC with two subnets and routing tables) records 3/8 for Claude and near-zero for all other models despite all having validate-passing generations; S10 (VPC subnet association) drops Claude to 5/7. Both scenarios require VPC flow log configuration and default security group restrictions that models consistently omit regardless of prompt level. Second, *GPT-5.4's Trivy compliance is structurally concentrated in simple S3 scenarios*: cells S2–S4 show 6–8/8, while all KMS and VPC scenarios record 0 or near-zero, revealing that its aggregate compliance rate masks near-complete failure on architecturally complex configurations, consistent with the self-referential block errors identified in Section 4.6.

4.6. Analysis of Validation Failures

We identify three failure categories from `terraform validate` error messages across all models, strategies, and levels.

Category 1 — Semantic dependency errors (Claude Opus 4, GPT-5.4). Both models predominantly fail with *Reference to undeclared resource* in complex VPC+KMS scenarios: they generate cross-resource references correct in intent but absent from the configuration. GPT-5.4 additionally produces *Resource cycle* errors (`aws_kms_key` ↔ `aws_cloudwatch_log_group`), a semantically correct security pattern that Terraform cannot instantiate without an intermediate `data` source. These failures reveal an *ambition gap*: both models attempt architecturally sophisticated, security-correct configurations but produce dependency graphs that require additional scaffolding to resolve.

Category 2 — Provider API version mismatches (Gemini 2.5 Pro, WizardCoder 33B, Qwen2.5-Coder 14B). These models generate deprecated AWS provider syntax: `versioning_configuration` as a nested block inside `aws_s3_bucket` (removed in provider v4+), deprecated `acl` attributes, and `required_providers` blocks without `source`/`version` fields. Qwen2.5-Coder 14B additionally generates module-style `var.*` references in non-module contexts. The pattern reflects training data staleness relative to the AWS provider v5.x used in the pipeline.

Category 3 — Structural HCL errors (CodeLlama 13B, Magicoder 7B). These models produce high volumes of parser-level failures: *Invalid multi-line string*, *Argument or block definition required*, *Unsupported block type*. Both also emit `var.*` references without `variable` declarations and occasionally leak markdown fences into the HCL output. CodeLlama 13B additionally targets Terraform 0.11/0.12 syntax in some generations. These failures occur before any provider-specific validation, reflecting fundamental HCL generation deficiencies rather than knowledge staleness.

Failures in complex scenarios (KMS encryption, VPC flow logs with CloudWatch) leave the security-most-relevant cells unscored, creating a systematic blind spot: models that fail precisely where security matters most are neither penalised nor credited. The Category 1 pattern further suggests that LLM failures here reflect architectural Terraform knowledge gaps rather than security unawareness.

4.7. Threats to Validity

Statistical power. The 3,570 evaluations ($7 \times 17 \times 6 \times 5$) support aggregate inference, with all pass rates accompanied by Wilson 95% CIs and pairwise comparisons Bonferroni-corrected. However, CodeLlama 13B and Magicoder 7B have Inter subsets of $n = 14$ and $n = 10$, yielding wide intervals ([1.3–31.5] and [0.0–27.8]) that support only directional interpretation. The $k = 5$ design is insufficient for scenario-level analysis; future work should increase k for low-compliance models. The claim that syntactic validity, plan executability, and security compliance form partially independent dimensions (contribution iii) is based on empirical observation across seven models: models that rank highly on validate rate do not consistently rank highly on plan pass rate or Checkov compliance, and vice versa (Tables 4 and 6). Formal independence testing via rank correlation (e.g., Spearman's ρ across models) is left for future work with a larger model sample, as $n = 7$ models is insufficient for reliable rank correlation estimates.

Scanner scope and complementarity. Checkov targets AWS policy compliance (encryption, access controls, logging) while Trivy targets vulnerability severity (CRITICAL/HIGH/MEDIUM). Our discordance analysis (Table 7) shows these are not interchangeable: Checkov implies Trivy but not vice versa, and the Trivy-only category (dominant for LLMs at L3) represents configurations that may be acceptable in non-production environments. Both scanners perform static analysis only and cannot detect runtime-dependent vulnerabilities.

Scenario selection and circularity. The 17 scenarios were selected via unanimous LLM consensus, which may over-represent patterns in model training corpora. To partially validate representativeness, the dominant failed Checkov checks, `CKV2_AWS_62` (S3 event notifications, 583 failures), `CKV_AWS_144` (S3 cross-region replication, 534), `CKV2_AWS_11` (VPC flow logging, 229), correspond to checks consistently identified as high-prevalence in real-world IaC repositories [Verdet et al. 2025], providing partial external validation of scenario relevance.

Model versions, quantization, and non-determinism. Results reflect seven specific model versions at a single point (May 2026) and may not extend to future releases or fine-tuned variants. SLMs run at Q4_K_M quantization (estimated 1–3pp penalty vs. full precision [Frantar et al. 2023]), substantially below the observed LLM–SLM compliance gap, suggesting architecture and training are the primary drivers. All models were queried

at provider API default temperature without explicit parameter setting; the observed pass rates therefore reflect a combination of model architecture, training stochasticity, and provider-specific sampling defaults, and should not be interpreted as pure measures of architectural consistency. The IaC-Eval generation difficulty label does not predict security compliance: scenarios easy to implement correctly are not necessarily easy to implement securely.

5. Conclusions and Future Work

This work presented a reproducible security-first benchmark for evaluating the ability of large and small language models to generate secure Terraform configurations. Through four complementary research questions, we investigated the relationship between syntactic correctness, security compliance, prompt sensitivity, iterative repair, and the evolution of AI-generated Infrastructure as Code. Overall, our findings show that syntactic validity alone is not a reliable indicator of secure IaC generation, and that security compliance should be treated as a first-class evaluation criterion rather than a by-product of successful deployment. These results converge on a single practical implication: automated multi-tool static analysis is not an optional complement to AI-assisted IaC generation, but a prerequisite for deployment-ready infrastructure, regardless of model family or prompting strategy. While prompt engineering and scanner feedback can improve generation quality, they are insufficient to consistently eliminate security compliance gaps. By making the benchmark, pipeline, prompts, and evaluation artefacts publicly available, we hope this work provides a reproducible foundation for future research on secure AI-assisted Infrastructure as Code and supports both researchers and practitioners in developing and evaluating more secure generation approaches.

Several research directions naturally emerge from this benchmark. From the benchmark perspective, future versions should expand the coverage to additional IaC languages, cloud providers, AWS resource families, and official Terraform modules as reference implementations. From the pipeline perspective, we plan to evolve the evaluation framework through an MCP Server capable of continuously assessing newly released foundation models and security tools, enabling long-term and reproducible benchmarking. At the model level, domain-specific fine-tuning strategies for SLMs, retrieval-augmented generation, and policy-aware guidance represent promising directions to reduce the gap between syntactic correctness and security compliance identified in this study. We hope these efforts contribute to establishing reproducible evaluation practices and accelerate the adoption of AI-assisted Infrastructure as Code without compromising infrastructure security.

All pipeline code, prompts, scenario definitions, and evaluation artefacts are publicly available.[3]

AI Usage Declaration

Generative AI tools were used as support during the development of this work. The Claude assistant (Anthropic) was employed to assist with the revision and refinement of academic writing in English, the structuring of paper sections, and the generation of LaTeX code snippets. All scientific content, including the experimental design, result analysis,

[3] https://gitlab.com/security-iac/security-first-evaluation-of-text-to-terraform/-/tags/sbseg-trilha-principal

and conclusions, was elaborated and verified by the authors. The authors assume full responsibility for the published content.

Acknowledgements

This research was partially supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq, Grant No. 409743/2025-9), Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS, Grants No. 24/2551-0001368-7, 24/2551-0000726-1, and 25/2551-0002572-9), and Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP, Grants No. 2020/05183-0 and 2023/00816-2). This research was also partially supported by Amazon Web Services (AWS) through AWS Promotional Credits, which provided the cloud computing infrastructure used for the experimental evaluation.

References

Chen, M. et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Continella, A., Polino, M., Pogliani, M., and Zanero, S. (2018). There's a hole in that bucket! A large-scale analysis of misconfigured S3 buckets. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*, pages 702–711. ACM.

Davidson, S., Sun, L., Bhasker, B., Callot, L., and Deoras, A. (2025). Multi-IaC-Eval: Benchmarking cloud infrastructure as code across multiple formats. *arXiv preprint arXiv:2509.05303*.

Fang, C., Miao, N., Srivastav, S., Liu, J., Zhang, R., Fang, R., Tsang, R., Nazari, N., Wang, H., and Homayoun, H. (2024). Large language models for code analysis: Do LLMs really do their job? In *33rd USENIX Security Symposium (USENIX Security)*, pages 829–846. USENIX Association.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. (2023). OPTQ: Accurate post-training quantization for generative pre-trained transformers. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

Jana, P., Davidson, S., Bhasker, B., Kan, A., Deoras, A., and Callot, L. (2026). TerraFormer: Automated infrastructure-as-code with LLMs fine-tuned via policy-guided verifier feedback. In *Proceedings of the IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE/ACM.

Kon, P. T., Liu, J., Qiu, Y., Fan, W., He, T., Lin, L., Zhang, H., Park, O. M., Elengikal, G. S., Kang, Y., et al. (2024). IaC-Eval: A code generation benchmark for cloud infrastructure-as-code programs. In *Advances in Neural Information Processing Systems*, volume 37, pages 134488–134512.

Morris, K. (2020). *Infrastructure as Code: Dynamic Systems for the Cloud Age*. O'Reilly Media, 2nd edition.

Nekrasov, R., Fossati, S., Kumara, I., Tamburri, D. A., and van den Heuvel, W.-J. (2026). IaC generation with LLMs: An error taxonomy and a study on configuration knowledge injection. *ACM Transactions on Software Engineering and Methodology*.

Opdebeeck, R., Zerouali, A., and De Roover, C. (2023). Control and data flow in security smell detection for infrastructure as code: Is it worth the effort? In *Proceedings of the 20th IEEE/ACM International Conference on Mining Software Repositories (MSR)*, pages 534–545. IEEE.

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., and Karri, R. (2022). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy (S&P)*, pages 754–768. IEEE.

Perry, N., Srivastava, M., Kumar, D., and Boneh, D. (2023). Do users write more insecure code with AI assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2785–2799. ACM.

Rahman, A., Parnin, C., and Williams, L. (2019). The seven sins: Security smells in infrastructure as code scripts. In *Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 164–175. IEEE. Distinguished Paper Award.

Rahman, A., Rahman, M. R., Parnin, C., and Williams, L. (2021). Security smells in Ansible and Chef scripts: A replication study. *ACM Transactions on Software Engineering and Methodology*, 30(1).

Rahman, A., Shamim, S. I., Bose, D. B., and Pandita, R. (2023). Security misconfigurations in open source Kubernetes manifests: An empirical study. *ACM Transactions on Software Engineering and Methodology*, 32(4).

Saavedra, N. and Ferreira, J. F. (2022). GLITCH: Automated polyglot security smell detection in infrastructure as code. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1–12. ACM.

SentinelOne (2024). Cloud security report 2024. Technical report, SentinelOne.

Verdet, A., Hamdaqa, M., Da Silva, L., and Khomh, F. (2025). Assessing the adoption of security policies by developers in Terraform across different cloud providers. *Empirical Software Engineering*, 30(3).

Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212.

Zhang, T., Pan, S., Zhang, Z., Xing, Z., and Sun, X. (2026). Deployability-centric infrastructure-as-code generation: Fail, learn, refine, and succeed through LLM-empowered DevOps simulation. *Proceedings of the ACM on Software Engineering*, 3(FSE):321–343.