Can Language Models Generate Secure Terraform Code? A Security-Focused Benchmark Using Static Analysis

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

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

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

Abstract. *The widespread adoption of Infrastructure-as-Code (IaC) has made cloud misconfiguration a critical security concern, while Large Language Models (LLMs) and Small Language Models (SLMs) have been redefining the programming process. We present an empirical benchmark evaluating whether LLMs and SLMs can generate security-compliant AWS Terraform configurations. Our automated pipeline integrates Checkov and Trivy into a GitLab CI/CD workflow across four Amazon S3 scenarios, evaluating six models under three prompt strategies of increasing security specificity. Security compliance improves consistently with prompt detail, though no model achieves full compliance in any configuration. Our findings suggest that prompt design is a critical factor, highlighting the need for a proper pipeline for developing and validating LLM-assisted secure IaC generation. All artifacts are publicly available.*

1. Introduction

Cloud computing has fundamentally transformed how organizations provision, manage, and secure infrastructure at scale. In this context, Infrastructure-as-Code (IaC) has become the *de facto* paradigm for defining cloud resources through machine-readable specifications, enabling reproducibility, version control, and automated deployment workflows [Morris 2020]. Among IaC technologies, Terraform has emerged as one of the most widely adopted solutions due to its declarative syntax, ecosystem maturity, and broad multi-cloud support. However, cloud misconfiguration remains one of the most critical and persistent causes of security incidents. Recent reports from the Cloud Security Alliance rank misconfiguration as the top cloud threat in 2024 [Cloud Security Alliance 2024], while industry analyses estimate that nearly 23% of cloud security incidents originate from incorrectly configured resources [SentinelOne 2024].

This challenge is particularly critical in Terraform-based deployments, where security-sensitive configurations such as access control policies, encryption settings, storage exposure, and network rules require precise domain knowledge of provider-specific APIs, resource interdependencies, and continuously evolving best practices. Prior studies have shown that even human-authored IaC frequently introduces security smells and policy violations into production environments, including access misconfigurations, deprecated resource patterns, and missing protection mechanisms [Rahman et al. 2019, Saavedra and Ferreira 2022, Verdet et al. 2025]. In particular, large-scale analyses of Amazon S3 deployments demonstrate that access control misconfigurations remain a major source of data exposure and attack surface expansion [Continella et al. 2018]. These

findings indicate that secure IaC authoring remains a non-trivial engineering challenge even for human experts.

At the same time, Large Language Models (LLMs) and Small Language Models (SLMs), compact and locally deployable models typically under 10B parameters, have rapidly become central to software development workflows, with increasing use in code generation and infrastructure automation tasks. Nevertheless, prior research consistently shows that AI-generated code often exhibits significant weaknesses in security-sensitive scenarios [Pearce et al. 2022, Perry et al. 2023, Fang et al. 2024]. Although recent benchmarks such as IaC-Eval, DPIaC-Eval, and TerraFormer have investigated LLM-based IaC generation, they primarily emphasize syntactic validity, deployability, and general correctness, while security compliance remains secondary or only partially explored [Kon et al. 2024, Zhang et al. 2025, Buehler et al. 2026]. As a result, an important scientific and practical gap remains open: *to what extent can state-of-the-art LLMs and locally deployable SLMs generate Terraform configurations that are both syntactically valid and compliant with established cloud security best practices under different levels of prompt security guidance?*

To address this question, we present a security-first empirical benchmark for AWS Terraform generation, combining automated syntactic validation and complementary static security analysis using Checkov and Trivy within a fully reproducible GitLab CI/CD pipeline. Our benchmark evaluates both LLMs and SLMs across four Amazon S3 scenarios and three prompt strategies with increasing levels of security specificity, enabling a systematic analysis of how prompt design affects security compliance, validation robustness, and model behavior across different families.

Our main contributions are threefold. First, we introduce a reproducible, automated benchmark pipeline integrated into GitLab CI/CD that covers code generation, syntactic validation, and multi-tool static security analysis. Second, we evaluate six models, namely GPT-5.4, Gemini 2.5 Pro, Gemini 2.5 Flash, Claude Opus 4.5, Qwen2.5-Coder 7B, and DeepSeek-Coder 6.7B, across four Amazon S3 scenarios under three prompt strategies of increasing security specificity (P1–P3), with three independent generations per scenario (pass@3). Third, we provide empirical evidence on the relationship between prompt design and security compliance, identify the root causes of validation failures across model families, and expose the current limitations of self-hosted SLMs for secure Terraform generation.

2. Related Work

Recent peer-reviewed benchmarks have started to investigate LLM-assisted IaC generation.

Security of human-authored IaC. Rahman et al. [Rahman et al. 2019] identified seven recurring security smells in Puppet scripts at ICSE 2019, a foundational taxonomy that has since been extended to Ansible and Chef [Rahman et al. 2021] and to Kubernetes manifests [Rahman et al. 2023]. Saavedra and Ferreira [Saavedra and Ferreira 2022] proposed GLITCH, a technology-agnostic framework for polyglot security smell detection across Ansible, Chef, Puppet, and Terraform at ASE 2022. Opdebeeck et al. [Opdebeeck et al. 2023] further proposed GASEL, a flow-sensitive detector for Ansible security smells at MSR 2023. Verdet et al. [Verdet et al. 2025] empirically analysed

Table 1. Comparison of related work on IaC security and LLM-based IaC generation. ✓ = addressed; ∼ = partially; – = not addressed.

Work	Venue	Scope	Security eval.	Static analysis	LLM gen.
Continella et al. [Continella et al. 2018]	ACSAC 2018	S3 misconfiguration	✓	custom tool	–
Rahman et al. [Rahman et al. 2019]	ICSE 2019	IaC smells (Puppet)	✓	SLIC	–
Rahman et al. [Rahman et al. 2021]	ACM TOSEM 2021	IaC smells (Ansible/Chef)	✓	SLAC	–
Saavedra & Ferreira [Saavedra and Ferreira 2022]	ASE 2022	IaC smells (multi-tool)	✓	GLITCH	–
Rahman et al. [Rahman et al. 2023]	ACM 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	–
IaC-Eval [Kon et al. 2024]	NeurIPS 2024	IaC generation	–	–	✓
This work	—	**IaC security + LLM gen.**	✓	**Checkov + Trivy**	✓

security policy adoption across 812 open-source Terraform projects using Checkov and Tfsec. Continella et al. [Continella et al. 2018] conducted the first large-scale analysis of misconfigured Amazon S3 buckets at ACSAC 2018, finding that access control misconfigurations expose sensitive data and enable resource injection attacks, directly motivating the choice of S3 as the target service in our benchmark. Collectively, these works establish that human-authored IaC consistently fails to implement security best practices, a gap that motivates evaluating whether LLM-generated IaC does any better.

LLM-based IaC generation benchmarks. IaC-Eval [Kon et al. 2024] introduced a Terraform generation benchmark and established that LLMs perform considerably below their scores on general code generation tasks. Verdet et al. [Verdet et al. 2025] further showed that even widely-used open-source Terraform projects exhibit significant security policy gaps when evaluated with static analysis tools.

Table 1 summarises the key differences between existing work and our approach. Unlike prior benchmarks, our work places security compliance as the primary evaluation criterion, employs two complementary static analysis tools (Checkov and Trivy), systematically investigates the effect of prompt engineering on security outcomes, and includes a comparison with SLMs.

3. Methodology

This section describes the benchmark design, including the evaluation pipeline, the scoring function, the models under evaluation, and the scenarios and prompt strategies used in the experiments.

3.1. Benchmark Pipeline

Figure 1 illustrates the overall benchmark pipeline, which consists of five sequential stages implemented as GitLab CI/CD jobs running in parallel across all evaluated models.

3.2. Models Under Evaluation

We evaluate six language models across two categories.

LLM : GPT-5.4 (OpenAI)[1], Gemini 2.5 Pro and Gemini 2.5 Flash (Google)[2], and Claude Opus 4.5 (Anthropic)[3], all accessed via their respective REST APIs.

[1] `https://platform.openai.com/`
[2] `https://aistudio.google.com/`
[3] `https://console.anthropic.com/`

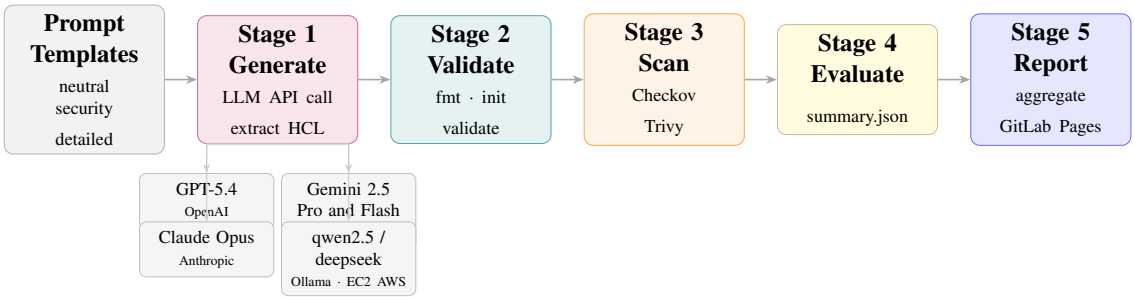

Figure 1. Benchmark pipeline. Each stage runs as a GitLab CI/CD job in a parallel matrix across all evaluated models.

Claude, Gemini 2.5 Pro, and Gemini 2.5 Flash were queried with temperature set to 0 to maximise output determinism; GPT-5.4 was queried using the API default temperature, as the model does not accept an explicit temperature parameter.

SLM : qwen2.5-coder:7b (Alibaba)[4] and deepseek-coder:6.7b (DeepSeek) [5], served via Ollama[6] on an AWS EC2 `g5.xlarge` instance (NVIDIA A10G, 24 GB VRAM) with temperature set to 0.7, following the official recommendations for each model family. All models receive the same prompt inputs and are evaluated under identical pipeline conditions.

3.3. Scenarios and Prompt Strategies

We define four scenarios based on Amazon S3 bucket configurations with increasing security complexity: (i) a basic log bucket with minimal requirements, (ii) a secure log bucket with explicit security controls, (iii) a fully hardened general-purpose bucket including cross-region replication, and (iv) a minimal yet compliant bucket designed for a compact prompt. Each scenario is evaluated under three prompt strategies. **P1 (Neutral)** uses a minimal task description without security instructions and serves as the baseline to measure the spontaneous security behavior of each model. **P2 (Basic Security)** includes general security guidelines, such as encryption at rest, public access blocking, and TLS enforcement. **P3 (Detailed Security)** explicitly specifies the Checkov identifiers (`CKV_AWS_*`, `CKV2_AWS_*`) and Trivy checks (`AVD-AWS-*`) that must be satisfied, together with the required Terraform resources.

This three-level design allows us to isolate the contribution of prompt specificity to security compliance, independent of model architecture or provider.

3.4. Evaluation Metrics

Table 2 describes the metrics used to evaluate each generated Terraform configuration. Metrics are grouped into two dimensions: syntactic validity, which measures whether the generated code is well-formed and accepted by the Terraform toolchain, and security compliance, which measures conformance to established AWS security best practices as reported by the static analysis tools.

[4]`https://ollama.com/library/qwen2.5-coder:7b`
[5]`https://ollama.com/library/deepseek-coder:6.7b`
[6]`https://ollama.com/`

Table 2. Evaluation metrics used in the benchmark.

Dimension	Metric	Tool	Description
2*Syntactic validity	Format pass	`fmt`	Passes `terraform fmt` without changes
	Validate pass	`validate`	Passes `terraform validate`
4*Security compliance	Checkov pass	Checkov	Zero failed `CKV_AWS_*`/`CKV2_AWS_*` checks
	Checkov failures	Checkov	Total failed checks (lower is better)
	Trivy pass	Trivy	No CRITICAL/HIGH/MEDIUM `AVD-AWS-*` findings
	Trivy HIGH	Trivy	Total HIGH-severity findings (lower is better)

All metrics are computed per scenario and aggregated per model across the four scenarios. Validate pass is treated as a prerequisite for meaningful security analysis: configurations that fail validation cannot be scanned reliably, and their scanner results are reported separately. Each metric is further aggregated over 12 runs per cell (4 scenarios $\times$ 3 independent generations). We refer to this repeated evaluation as pass@3, adapting the terminology from Chen et al. [Chen et al. 2021]: unlike the original probabilistic estimator, we report absolute pass counts over k independent generations rather than an unbiased probability estimate.

4. Results

Table 3 summarises the benchmark results across the three prompt strategies and six models over 12 runs per cell (4 scenarios $\times$ 3 independent generations).

Table 3. Benchmark results across prompt strategies and models.

#	Prompt	Model	Validate↑	Checkov↑	Checkov fail↓	Trivy↑	Trivy C/H/M↓
1.1	P3 – Detailed Security	Claude Opus 4.5	**12/12**	**11/12**	**1**	**12/12**	0/0/0
1.2		GPT-5.4	2/12	**11/12**	6	**12/12**	0/0/0
1.3		Gemini 2.5 Pro	10/12	2/12	41	**12/12**	0/0/0
1.4		Gemini 2.5 Flash	0/12	10/12	4	**12/12**	0/0/0
1.5		DeepSeek-Coder 6.7B	1/12	4/12	90	5/12	0/60/9
1.6		Qwen2.5-Coder 7B	0/12	2/12	74	10/12	0/16/0
2.1	P2 – Basic Security	Claude Opus 4.5	**12/12**	6/12	24	**12/12**	0/0/0
2.2		GPT-5.4	1/12	5/12	48	11/12	0/0/1
2.3		Gemini 2.5 Pro	**12/12**	2/12	48	7/12	0/15/2
2.4		Gemini 2.5 Flash	0/12	**9/12**	18	**12/12**	0/0/0
2.5		DeepSeek-Coder 6.7B	0/12	8/12	52	9/12	0/29/4
2.6		Qwen2.5-Coder 7B	1/12	4/12	49	9/12	0/16/0
3.1	P1 – Neutral	Claude Opus 4.5	**12/12**	4/12	27	7/12	0/7/0
3.2		GPT-5.4	5/12	6/12	33	9/12	0/3/0
3.3		Gemini 2.5 Pro	8/12	0/12	72	1/12	0/133/20
3.4		Gemini 2.5 Flash	0/12	5/12	47	8/12	0/24/4
3.5		DeepSeek-Coder 6.7B	0/12	6/12	60	6/12	0/39/6
3.6		Qwen2.5-Coder 7B	0/12	0/12	111	4/12	0/54/1

Metrics computed over 12 runs per cell (4 scenarios $\times$ 3 independent generations). Bold = best value per block.
Validate, Checkov, Trivy: number of passing runs out of 12 — higher is better (↑).
Checkov fail: total failed checks across all runs — lower is better (↓).
Trivy C/H/M: total Critical / High / Medium severity findings — lower is better (↓).

The best overall result is row 1.1 (Claude Opus 4.5, P3), which achieves 12/12 validation passes, 11/12 Checkov passes with only 1 total failure, and a clean Trivy profile across all runs, the only cell in the entire table approaching full security compliance. The worst result is row 3.6 (Qwen2.5-Coder 7B, P1), with 0/12 validation passes, 0/12 Checkov passes, 111 total Checkov failures, and 54 HIGH Trivy findings, confirming

that SLMs without security-oriented prompting represent the highest risk profile in this benchmark.

The most immediate observation across the table is the sharp contrast in Trivy findings between P1 and the two security-oriented strategies. Under P1, several models accumulate a large number of HIGH-severity findings: row 3.3 (Gemini 2.5 Pro) reaches 133 HIGH and 20 MEDIUM across its 12 runs, and row 3.6 (Qwen2.5-Coder 7B) records 54 HIGH findings. In contrast, every model under P3 produces zero Trivy HIGH or MEDIUM findings, with the sole exception of the two SLMs (rows 1.5 and 1.6). This indicates that even minimal security guidance in the prompt is sufficient to suppress the most severe misconfiguration classes detectable by Trivy in this benchmark.

Checkov, however, tells a different story. A second notable pattern is the divergence between syntactic validity and security compliance across model families. Claude Opus 4.5 is the only model to maintain 12/12 validation passes across all three prompt strategies (rows 1.1, 2.1, 3.1). GPT-5.4 and Gemini 2.5 Pro achieve high validation rates under P3 and P2 respectively, but their Checkov profiles remain weak, particularly Gemini 2.5 Pro, which records 41 failures in row 1.3 and 48 in row 2.3 despite near-perfect validation. Notably, GPT-5.4 shows fewer validation passes under P3 (2/12) than under P1 (5/12), a counterintuitive result explained by the over-generation of complex HCL constructs, particularly multi-line IAM policy heredocs when the model attempts to satisfy the detailed check-identifier list in the prompt, as discussed in Section 4.1. Gemini 2.5 Flash presents the opposite pattern: it fails validation consistently (0/12 across all strategies) yet reports higher Checkov pass rates. However, since Checkov operates on malformed HCL, these results should be interpreted with caution, they may reflect artefacts of the scanner operating on incomplete configurations rather than genuine security compliance.

Overall, the results indicate that prompt hardening alone is not sufficient to guarantee both valid and secure IaC generation in this evaluation. P3 produced the strongest security outcomes across models, but did not resolve validation failures for GPT-5.4 or the SLMs. Conversely, P1 yielded better syntactic validity for some models, yet at the cost of a weaker security profile, with more Checkov failures and higher Trivy severity counts. This trade-off reinforces the need for automated validation and security scanning as complementary safeguards in any LLM-assisted IaC workflow.

4.1. Analysis of Validation Failures

To understand the root causes of validation failures, we analysed the `terraform validate` error messages and the generated `main.tf` files for all models across the three prompt strategies. Three distinct failure categories were identified, each associated with different model families.

Category 1: HCL string and template errors (Gemini 2.5 Flash, GPT-5.4, Gemini 2.5 Pro). The most frequent error class for Gemini 2.5 Flash (12/12 in all strategies) and GPT-5.4 (7–11/12 depending on strategy) consists of multi-line string and template expression errors: *Unterminated template string*, *Unclosed configuration block*, and *Missing expression*. These indicate that models generate HCL heredoc blocks typically used for inline IAM policy JSON, that are not properly closed, causing the HCL parser to fail. A secondary recurring error for both models is *Unsupported*

argument: eventbridge_enabled, which appears when `eventbridge_enabled` is used as a direct argument inside `aws_s3_bucket` rather than as part of the separate `aws_s3_bucket_notification` resource introduced in provider v4.x. Under the P1 (Neutral) prompt, both models also produced outputs with markdown code fences (```` ```hcl ````) leaked into the `main.tf` file, causing *Invalid character* errors. This suggests that without explicit output formatting instructions, these models occasionally wrap their response in markdown formatting that the pipeline's extraction regex does not fully clean. Gemini 2.5 Pro exhibited the same heredoc pattern in its 2 failures under P3 (Detailed Security), but was otherwise stable.

Category 2: Deprecated AWS provider v3/v4 syntax (DeepSeek-Coder 6.7B, Qwen2.5-Coder 7B). Both self-hosted SLMs consistently generated code targeting the AWS provider v3.x–v4.x API, incompatible with the v5.x environment used in the benchmark, across all three prompt strategies. Notably, the P3 prompt explicitly specified the required provider version and `source`/`version` constraints, yet both models continued to emit deprecated constructs, indicating that the failures cannot be attributed to insufficient prompt context alone. Rather, the evidence points to a training data limitation: both SLMs appear to lack internalized knowledge of the AWS provider v5.x API, a gap that prompt engineering alone cannot bridge and that likely requires fine-tuning on recent provider documentation or retrieval-augmented generation with up-to-date API references. DeepSeek-Coder 6.7B (11–12/12 failures) used deprecated inline blocks (`server_side_encryption_configuration`, `versioning`, `logging`), the deprecated `acl` attribute, and `data "random_id"` instead of `resource "random_id"`. Qwen2.5-Coder 7B (11–12/12 failures) consistently used the legacy `required_providers` syntax without the `source`/`version` format introduced in Terraform 0.13, triggering *An argument named "required_providers" is not expected here* across all strategies.

Category 3: Impact on security scanning. When `terraform validate` fails, Checkov and Trivy cannot fully parse the configuration. In the majority of validation failures for Gemini 2.5 Flash and GPT-5.4, Checkov reported 0 checks evaluated, meaning the security posture of those configurations was entirely unassessable. Validation failures therefore do not merely represent a syntactic quality issue, they directly prevent security compliance measurement, making syntactic correctness a hard prerequisite for any meaningful IaC security assessment.

4.2. Threats to Validity

This study is subject to several limitations. Although three independent generations per scenario were collected per model (pass@3), the total of 216 evaluations (6 models $\times$ 4 scenarios $\times$ 3 prompts $\times$ 3 runs) may still not capture the full variance of model behaviour, particularly for models with high stochasticity such as the SLMs queried at temperature 0.7. Checkov and Trivy perform static analysis only and cannot detect runtime-dependent vulnerabilities; the four scenarios also cover a single AWS service (S3), limiting generalisation to other resource types. The benchmark evaluates six specific model versions at a single point in time, and results may not extend to future releases, larger open-source models, or alternative prompt strategies such as few-shot or retrieval-augmented generation. The three prompt strategies evaluated represent specific design choices made by the authors; different phrasings, levels of detail, or check-identifier se-

lections could yield different compliance outcomes, and the observed improvements from P1 (Neutral) to P3 (Detailed Security) prompting may not generalise to other prompt formulations or to scenarios beyond S3. Finally, no formal statistical significance tests were conducted: with only $k{=}3$ generations per cell, results should be interpreted as descriptive evidence rather than statistically confirmed findings; future work should increase k or apply bootstrap-based estimators. The P3 prompt explicitly enumerates security rule identifiers, which may not reflect typical practitioner usage; it is intentionally designed as an upper-bound condition, while P1 and P2 cover more realistic scenarios. Additionally, the benchmark validates syntactic well-formedness via `terraform validate` but does not verify actual deployment, as `terraform apply` would introduce pipeline costs and cloud billing overhead; incorporating it via ephemeral AWS accounts remains a direction for future work.

5. Conclusions and Future Work

The results show that stronger prompt guidance substantially improves the security posture of LLM-generated IaC, but does not consistently guarantee syntactically valid Terraform configurations across all evaluated models. Claude Opus 4.5 was the only model to maintain validation success across all prompt strategies, while GPT-5.4, Gemini 2.5 Pro, and Gemini 2.5 Flash exhibited distinct trade-offs between syntactic validity and security compliance. In particular, Gemini 2.5 Pro consistently produced valid Terraform configurations but accumulated a high number of Checkov violations, whereas Gemini 2.5 Flash exhibited the opposite pattern, namely stronger scanner compliance coupled with persistent validation failures. The two SLMs also consistently failed validation, despite occasionally achieving acceptable scanner outcomes. Across all configurations, Checkov proved to be the most discriminative security control, while Trivy was more conservative and primarily highlighted higher-severity issues. Taken together, these findings indicate that, although prompt engineering can meaningfully improve security-aware IaC generation, it is insufficient as a standalone safeguard and must be systematically complemented by automated validation and security scanning in practical LLM-assisted IaC workflows. All pipeline code, prompts, and evaluation artifacts are publicly available[7].

Several directions emerge for future work. At the methodological level, future research should formalise evaluation pipelines, reproducibility protocols, and benchmark versioning strategies, and explore explainability techniques and agentic or human-in-the-loop workflows that iteratively refine generated configurations. From a benchmark perspective, the scenario set should expand to cover additional AWS resources (IAM, KMS, VPC) and other IaC languages (CloudFormation, Pulumi). Evaluating larger open-source and infrastructure-fine-tuned models may clarify whether domain-specific training mitigates the knowledge staleness observed in the evaluated SLMs. Finally, the framework should be extended with cost and performance metrics, token consumption, API cost, latency, and memory requirements to enable a more comprehensive cost-benefit analysis of LLM-assisted IaC workflows.

[7] `https://gitlab.com/security-iac/security-first-evaluation-of-text-to-terraform/-/tree/feat/experimentos?ref_type=heads`

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, 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 FAPERGS (grants no. 24/2551-0001368-7, 24/2551-0000726-1, and 25/2551-0002572-9), FAPESP (grants no. #2020/05183-0 and #2023/00816-2) and CNPq/MCTI/FNDCT Nº 22/2024 (grant no. #444727/2024-8).

References

Buehler, N. et al. (2026). TerraFormer: Automated infrastructure-as-code with LLMs fine-tuned via policy-guided verifier feedback. *arXiv preprint arXiv:2601.08734*.

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

Cloud Security Alliance (2024). Top threats to cloud computing 2024. Technical report, Cloud Security Alliance.

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.

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.

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.

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).

Zhang, T., Pan, S., Zhang, Z., Xing, Z., and Sun, X. (2025). Deployability-centric infrastructure-as-code generation: An LLM-based iterative framework. *arXiv preprint arXiv:2506.05623*.