Large Language Models for Infrastructure as Code Vulnerability Remediation

Raul Y. Reyes[1]
Eller College of Management, University of Arizona, Tucson, AZ, U.S.A.

Benjamin M. Ampel[2]
J. Mack Robinson College of Business, Georgia State University, Atlanta, GA, U.S.A.

Hsinchun Chen[3]
Eller College of Management, University of Arizona, Tucson, AZ, U.S.A.

ABSTRACT

The growing reliance on Infrastructure as Code (IaC) has streamlined cloud provisioning but introduced new security challenges due to misconfigurations and insecure coding patterns. Existing vulnerability scanners (e.g., Trivy) can detect flaws but cannot perform automated remediation. This leaves developers responsible for manual fixes. To address this, we propose a computational framework for IaC remediation that adapts prevailing Large Language Models (LLMs) with vulnerability scanner outputs using parameter-efficient fine-tuning and in-context learning. We collected 6,149 Terraform scripts from 1,466 repositories, identified 27,557 misconfigurations using Trivy, and constructed 23,906 corrected code blocks for training and evaluation. Two open-source LLMs (CodeGen-350M-mono and Llama 3.2-1B) were fine-tuned with Low-Rank Adaptation (LoRA) and evaluated using BLEU and ROUGE metrics. Results suggested that fine-tuning LLMs improved vulnerability remediation. These findings demonstrate the value of domain adaptation for improving LLM-based IaC remediation and provide a foundation for future research on securing cloud infrastructure.

Keywords: Infrastructure as Code, Large Language Models, Vulnerability Remediation, Cloud Security, Continuous Integration and Deployment.

[1] Corresponding author. raulreyes@arizona.edu
[2] Corresponding author. bampel@gsu.edu
[3] Corresponding author. hsinchun@arizona.edu

INTRODUCTION

The rise of cloud computing has accelerated the use of Infrastructure as Code (IaC) to quickly provision enterprise information systems (Ntentos et al. 2025). IaC manages and provisions machine-readable configuration files to automate the deployment of computing resources (e.g., networks, software, and databases) through code (Chiari et al. 2022). Modern IaC management tools (e.g., Terraform, Puppet, Ansible) enable critical business technologies to be employed in minutes (Ketonen and Smolander 2025). As a result, IaC has become a foundational component of continuous integration and deployment pipelines and a core development and operations practice that offers scalable and reproducible environments (Vizard 2023).

Despite these benefits, IaC is highly vulnerable to security risks arising from script misconfigurations (i.e., error within cloud resource provisioning). Examples include hard-coded credentials (e.g., embedding plain-text AWS access keys), overly permissive access controls (e.g., allowing inbound traffic from 0.0.0.0/0), misconfigured resources (e.g., leaving cloud storage buckets publicly accessible), and sensitive comments in scripts (e.g., documenting database passwords in plain text) (Rahman et al. 2019). Each of these misconfigurations are potential vulnerabilities that can be exploited by motivated hackers (Chen et al. 2023; Rahman et al. 2019) Industry reports show that misconfigurations exposed 63% of sensitive databases in finance and healthcare, insecure defaults enabled cryptojacking costing $12,000 per day, and hard-coded credentials appear in 83% of IaC scripts (Chen et al. 2023).

Large Language Models (LLMs) present a promising approach for addressing IaC misconfigurations due to their ability to analyze, generate, and correct code at scale (Vo et al. 2025; Xiang et al. 2025). Static rule-based scanners typically stop at reporting vulnerabilities. LLMs offer the potential to generate context-aware fixes directly within IaC scripts, which could

reduce developer burden and minimize the window of exposure to attacks. However, LLMs are often trained on broad, heterogeneous corpora (Ampel et al. 2025). These general purpose LLMs therefore may lack the domain-specific knowledge required to accurately remediate IaC misconfigurations. Despite this, LLMs are adaptable through fine-tuning and in-context learning, which may potentially allow them to be customized for mitigating the pervasive misconfigurations that undermine cloud infrastructure security (Dong et al. 2024; Han et al. 2024).

To automatically remediate IaC script misconfigurations, we propose a computational design science framework that integrates (1) IaC vulnerability scanner outputs, (2) parameter-efficient fine-tuning of LLMs, and (3) in-context learning for LLM adaptation. Design science literature emphasizes the creation of artifacts that solve important societal problems rather than remaining purely explanatory (Peffers et al. 2007). Our framework fulfills this objective by delivering an adaptive LLM-based remediation artifact that mitigates critical IaC misconfigurations and strengthens cloud security.

The remainder of this paper is organized as follows. First, we review literature on IaC vulnerability assessment and LLMs for code analysis. Second, we describe our research gaps and questions. Third, we present our research design. Fourth, we present the results of our empirical analysis. Finally, we conclude this research, discuss limitations, and posit future directions.

LITERATURE REVIEW

We review two key areas of literature to guide this research. First, we review IaC vulnerability assessment to identify past efforts in analyzing and remediating misconfigurations. Second, we review pre-trained LLMs to identify models and planning strategies proficient at code analysis and generation.

IaC Vulnerability Assessment

Prior research on IaC vulnerability assessment has primarily focused on identifying misconfigurations on open-source configuration management tools such as Terraform, Ansible, Puppet, and Chef (Bessghaier et al. 2024; Chiari et al. 2022). Detection approaches have ranged from rule-based techniques (e.g., SLIC, SecureCode) to machine learning classifiers (e.g., random forest, k-nearest neighbors) and deep learning methods (e.g., convolutional neural networks, word embeddings). Rule-based techniques are interpretable but struggle to detect novel and evolving vulnerabilities (Goyal and Chaturvedi 2025). Machine learning and deep learning techniques are more generalizable but require high-quality labeled datasets but cannot conduct remediation beyond detection (Chiari et al. 2022). Recent work has begun to address these limitations with LLMs. LLMs have been implemented for detecting misconfigurations in IaC (Vo et al. 2025; Xiang et al. 2025), generating IaC scripts (Kon et al. 2024; Lee et al. 2024; Ragothaman and Udayakumar 2024), and supporting remediation (Diaz-De-Arcaya et al. 2024; Goyal and Chaturvedi 2025; Kwon et al. 2023; Low et al. 2024; Toprani and Madisetti 2025). Most approaches rely on proprietary LLMs (e.g., GPT, Claude, Gemini) or large open-source models (e.g., LLaMA 13B, Alpaca 13B, DeepSeek 236B) trained on broad code corpora and used as black-box systems, without fine-tuning or adaptation to the unique demands of IaC security.

However, few studies integrate IaC vulnerability scanners (e.g., Trivy, Checkov, or Tfsec) beyond their use as detection baselines (Low et al. 2024; Ragothaman and Udayakumar 2024). Vulnerability scanners can provide structured, standardized, and widely adopted assessments for downstream tasks (Lazarine et al. 2020; Ullman et al. 2020). Leveraging these could potentially allow LLMs to be grounded in standardized security metadata, which could reduce hallucinations

and align generated fixes with industry-validated best practices. Therefore, we review LLMs for code analysis to discover how these models could potentially be adapted for IaC remediation tasks.

Large Language Models for Code Analysis

LLMs have demonstrated strong capabilities in coding tasks such as code generation (creating code from a prompt), translation (converting between code languages), and completion (filling out partial code segments) (Yao et al. 2024). LLMs can be categorized into three architectures: (1) masked (encoder-only), (2) sequence-to-sequence (encoder-decoder), and (1) autoregressive (decoder-only) (Ampel et al. 2025). First, masked models (e.g., CodeBERT and CuBERT) are more effective for classification tasks (Jiang et al. 2024). Second, sequence-to-sequence models (e.g., CodeT5 and PLBART) excel in code summarization and translation tasks (Yao et al. 2024). Finally, autoregressive models (e.g., CodeGen and Llama) generate code token by token from prior context (Jiang et al. 2024). This capability is well suited to IaC remediation, where secure configurations must be generated as contextually consistent extensions or corrections of existing scripts. However, these LLMs still need adaptation to be effective for niche domains.

LLM adaptation strategies aim to train, optimize, or refine models for improved downstream performance in targeted domains (Patil and Gudivada 2024). Adaptation strategies can be grouped into three types: (1) pretraining, (2) fine-tuning, and (3) in-context learning (Shen et al. 2025). First, pretraining develops foundational language and code representations from large-scale datasets (Shen et al. 2025). This process often involves updating millions to billions of parameters with extensive datasets and GPU resources. Second, fine-tuning adapts LLMs to specialized domains. Fine-tuning involves updating all or some model parameters to avoid costly pre-training and transfer general knowledge to domain-specific tasks (Weyssow et al. 2025). Finally, in-context learning enables LLMs to generalize to new tasks without modifying

parameters via prompt engineering or new information retrieval (Dong et al. 2024). These strategies provide complementary pathways for improving model adaptability and performance in specialized domains. However, to the best of our knowledge, no peer-reviewed studies have implemented pre-training or fine-tuning for IaC remediation of script misconfigurations.

RESEARCH GAPS AND QUESTIONS

We identified two key research gaps from our literature review. First, while tools and methods for detecting IaC misconfigurations are well established, automated remediation remains largely unaddressed. Existing methods focus on reporting but not correction. Second, although LLMs are increasingly used for IaC, studies often rely on proprietary or generic models that are not fine-tuned for IaC's unique security requirements. Based on these research gaps, we propose the following questions:

- How can vulnerability scan outputs be integrated to support automated remediation?
- How can LLMs be adapted to address the domain-specific challenges of IaC security?

RESEARCH DESIGN

To automatically remediate misconfigured IaC scripts, we propose a research framework that consists of three components: (1) Data Collection, (2) IaC LLM, and (3) Evaluation. We show our proposed research framework in Figure 1 and further detail each component as follows.

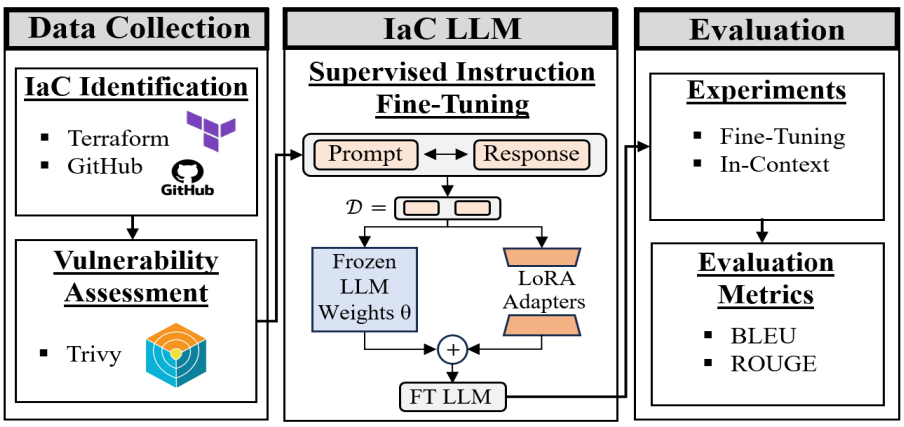

Figure 1. Proposed Research Framework

Data Collection

Our data collection was conducted in two stages. The first stage identified IaC configurations scripts. To discover these scripts, we built an automated crawler to systematically crawl GitHub. GitHub is known to host open-source IaC configuration scripts from the popular management tool Terraform (Ntentos et al. 2025). Our crawler was able to retrieve 6,149 Terraform configuration scripts from 1,466 open-source repositories. These scripts included 1,903 modules[4] that defined 4,357 distinct resource configurations. This data collection is comparable in size to past IaC studies (Chiari et al. 2022).

The second stage conducted a vulnerability assessment of our collected Terraform IaC scripts. We leveraged the vulnerability scanner Trivy for its comprehensive IaC coverage and prior validation in IaC research (Low et al. 2024). Trivy reports vulnerability IDs, Common Vulnerability Scoring System severity ratings (low, medium, high, critical), affected modules and services, associated URLs, and descriptive metadata. Our Trivy vulnerability assessment identified 27,557 misconfigurations across 11,804 unique code blocks. 4,277 were rated as critical, 11,083 as high, 4,257 as medium, and 7,990 as low. Trivy's outputs lack remediation strategies, so we leveraged the reference URLs to Aqua Security's Vulnerability Database[5] to reconstruct 23,906 corrected code blocks. To our knowledge, this augmentation of Trivy's output has not been adopted in prior work. Table 1 summarizes our overall data collection. Figure 2 provides an example of a vulnerable code block with the example remediation strategy.

---

[4] In Terraform, a module is a reusable and self-contained collection of configuration files that defines one or more related infrastructure resources. Modules allow developers to package common infrastructure patterns (e.g., a virtual private cloud or a Kubernetes cluster) into portable building blocks that can be shared across projects, reused, and customized to fit specific deployment needs.

[5] https://avd.aquasec.com/misconfig

Table 1. Summary of Our Two-Stage IaC Identification and Vulnerability Assessment Process

Data Attribute	Count
IaC Scripts	6,149
Repositories	1,466
IaC Modules and Services	1,903
IaC Resources	4,357
Misconfigurations	27,557
Unique Code Blocks	11,804
Remediated Code	23,906

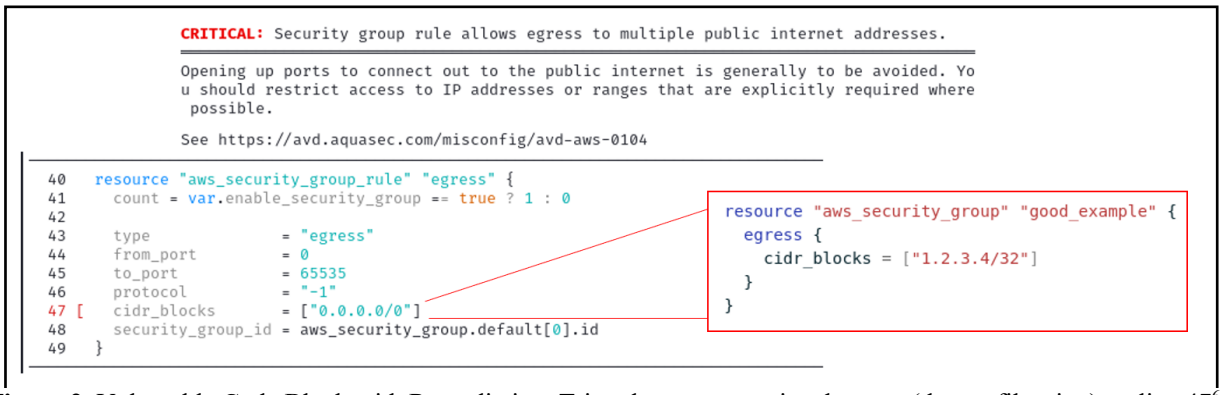

Figure 2. Vulnerable Code Block with Remediation. Trivy detects unrestricted egress (data exfiltration) on line 47[6]. The red box shows a suggested fix from Aqua Database, which we implemented to produce a remediated block.

IaC Large Language Model

The IaC scripts and vulnerability scanner outputs formed the foundation for our IaC LLM adaptation. This data is used to fine-tune code-focused LLMs to the unique patterns and security requirements of IaC. Figure 3 illustrates the prompt-response structure used in both fine-tuning and evaluation. Prompts supplied IaC scripts for analysis, and responses were structured to capture misconfigurations, vulnerability details, and remediation code.

[6] https://avd.aquasec.com/misconfig/aws/ec2/avd-aws-0104/#Terraform

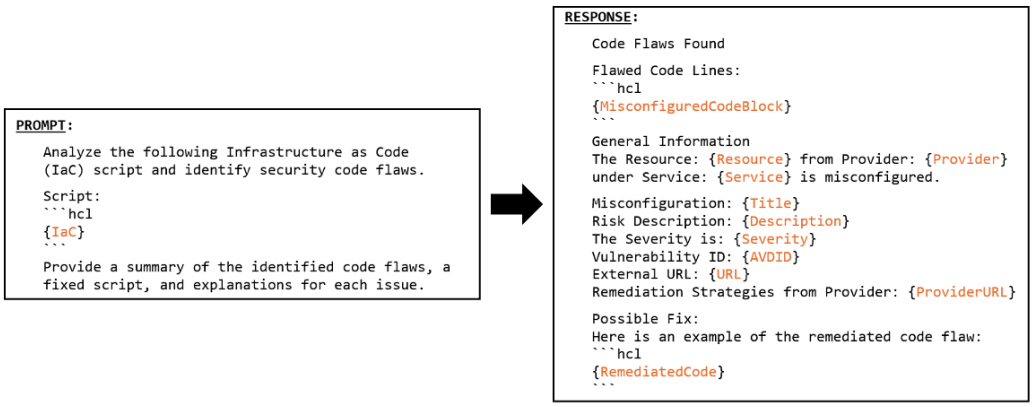

Figure 3. Fine-Tuning Prompt

We selected two prevailing open-source LLMs aligned with prior research on code generation: CodeGen-350M-mono, a lightweight code-specialized baseline, and Llama 3.2-1B, a larger context-aware model (Yao et al. 2024). Both models were trained with supervised instruction fine-tuning, where each training instance paired an IaC script (prompt) with a structured vulnerability analysis and remediation code (response). To enable parameter-efficient adaptation, we employed Low-Rank Adaptation (LoRA), which freezes the base model and introduces trainable low-rank matrices into selected weight updates (Weyssow et al. 2025). The training objective is a prompt-conditional response-only language modeling loss, with gradients applied only to response tokens. Formally, given dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^{N}$ of prompts x and responses y, the objective is:

$$\mathcal{L}(\theta_0, \phi) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{n_i}\sum_{t=1}^{n_i} -log p_{\theta_0,\phi}\left(y_t^i \mid x_{1:m_i}^i, y_{<t}^i\right),$$

where θ_0 denotes the frozen base model parameters and ϕ the trainable LoRA adapters. LoRA targeted the *out_proj* and *qkv_proj* layers in CodeGen, and the *q_proj*, *v_proj*, *up_proj*, and *down_proj* layers in Llama, with rank r = 6, scaling factor α = 16, and dropout of 0.1, thereby adapting both attention projections and feed-forward transformations to IaC-specific patterns. This resulted in 0.20% (737,280) of CodeGen parameters and 0.21% (2,605,056) of Llama parameters

being trainable. Both models were fine-tuned for two epochs on a workstation with 128 GB RAM and two NVIDIA RTX A6000 GPUs.

Evaluation

Similar to past studies (Li et al. 2024), we evaluated our adapted IaC LLMs through two next-token prediction experiments to assess their effectiveness for vulnerability remediation. The first experiment compared base and fine-tuned models in generating ground-truth responses to IaC analysis prompts using the prompt-response format shown in Figure 3. The second experiment tested in-context learning strategy by providing vulnerability metadata in the prompt without remediation code. This setting allowed us to evaluate how base and fine-tuned models used contextual information to improve remediation quality.

Model outputs were evaluated on 100 unseen samples using two widely adopted text-generation metrics: (1) BLEU and (2) ROUGE (Chang et al. 2024). First, BLEU measures n-gram precision by calculating how many word sequences in the generated output also appear in the reference text. We specifically implemented BLEU-4, which considers up to four-gram matches and provides a stricter measure of alignment with the ground truth (Chang et al. 2024). ROUGE emphasizes recall (Citarella et al. 2025) and was calculated as ROUGE-1 (unigram overlap), ROUGE-2 (bigram overlap), and ROUGE-L (longest common subsequence). ROUGE metrics are reported using F1 score, which is common in literature (Ampel et al. 2025; Chang et al. 2024). For all metrics, we report the mean and standard deviation of scores across the 100 samples. All metrics are measured between 0 and 1, where 0 is worst and 1 is best performance.

RESULTS AND DISCUSSION

Table 2 summarizes the performance of base and fine-tuned models under both standard and in-context prompting conditions. Across all experiments, fine-tuned LLMs consistently

outperformed their base counterparts, demonstrating the effectiveness of parameter-efficient fine-tuning (denoted as FT in the table) in adapting general-purpose LLMs to IaC remediation tasks.

Table 2. Experiment Results

Experiments	LLM	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
Base LLMs	CodeGen	0.03 ± 0.04	0.21 ± 0.12	0.09 ± 0.08	0.14 ± 0.08
	Llama3.2	0.01 ± 0.03	0.13 ± 0.13	0.03 ± 0.05	0.08 ± 0.08
Fine-tuned	FT CodeGen	0.33 ± 0.16	0.60 ± 0.12	0.52 ± 0.16	0.56 ± 0.15
LLMs	**FT Llama3.2**	**0.55 ± 0.18**	**0.74 ± 0.12**	**0.67 ± 0.17**	**0.71 ± 0.14**
LLMs with in-	CodeGen	0.36 ± 0.15	0.56 ± 0.13	0.50 ± 0.16	0.52 ± 0.15
context learning	FT CodeGen	0.32 ± 0.14	0.59 ± 0.12	0.50 ± 0.15	0.54 ± 0.14
	Llama3.2	0.10 ± 0.14	0.24 ± 0.22	0.15 ± 0.18	0.18 ± 0.17
	FT Llama3.2	**0.41 ± 0.15**	**0.65 ± 0.09**	**0.58 ± 0.12**	**0.59 ± 0.11**

Both CodeGen and Llama 3.2 showed substantial gains following fine-tuning. CodeGen improved from a BLEU-4 of 0.03 to 0.33. Llama 3.2 improved from 0.01 to 0.55. ROUGE scores followed a similar pattern, with IaC-fine-tuned Llama 3.2 achieving the highest overall performance (ROUGE-1 of 0.74, ROUGE-2 of 0.67, and ROUGE-L of 0.71). Higher ROUGE scores indicate that the fine-tuned models generated remediations that more closely matched the reference corrections in content and structure. Our in-context learning examined the effect of providing vulnerability metadata as contextual information to each LLM. For base CodeGen, in-context learning slightly improved BLEU-4 compared to its fine-tuned counterpart (0.36 vs. 0.33) while ROUGE scores remained similar across the board. This suggests that a code-specialized model can potentially incorporate contextual cues effectively when generating IaC remediations. The base Llama 3.2 also improved with context (BLEU-4 improved from 0.01 to 0.10) but continued to underperform relative to CodeGen (often producing textual explanations, not code). Fine-tuned Llama 3.2 with context achieved further gains (BLEU-4 improved from 0.10 to 0.41), reinforcing the value of fine-tuning. However, this did not exceed the performance of its fine-tuned counterpart without context (BLEU-4 of 0.55). This indicates that once a model is well adapted, additional contextual metadata may actually harm generation quality. To demonstrate the impact

of fine-tuning for IaC misconfiguration remediation, Figure 4 compares outputs from the fine-tuned Llama model (left) and the base model (right) applied to the same IaC script.

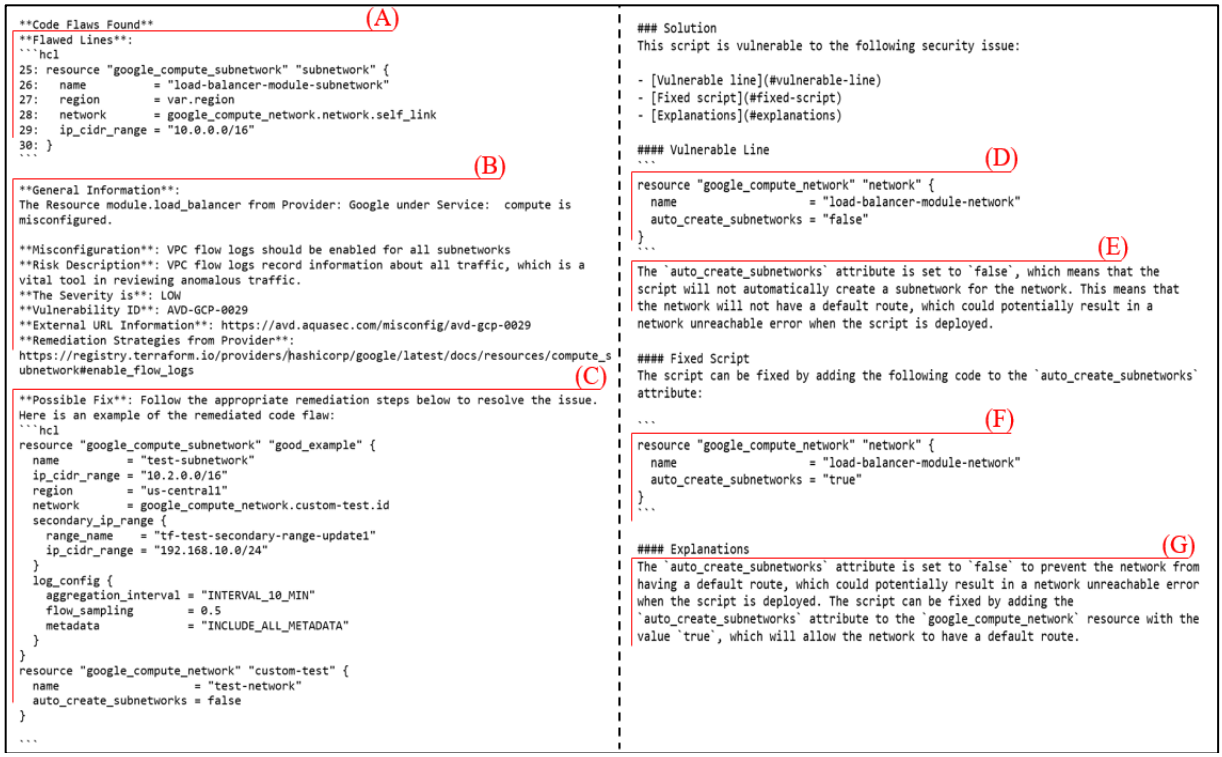

Figure 4. Comparison of generated outputs from Fine-Tuned Llama (Left) and Base Llama (Right)

The fine-tuned model highlights the (A) vulnerable Terraform code block, (B) provides structured descriptive information indicating that the Virtual Private Cloud should have network flow logs enabled, and (C) generates a corrected code block with the appropriate log configurations. This remediation ensures that all network traffic is logged, thereby supporting compliance and reducing the likelihood of undetected malicious activity. The base model output reproduced the expected structural format but hallucinated an unrelated misconfiguration. It incorrectly flagged (D) the attribute *auto_create_subnetworks* as the source of the issue, (E) described the flaw, (F) proposed enabling it as the fix, and (G) provided an explanation for this remediation. This illustrates how base LLMs can produce plausible yet misleading outputs without IaC industry-level adaptation.

CONCLUSION AND FUTURE DIRECTIONS

This research proposed a novel computational framework for adapting LLMs to IaC vulnerability remediation. To our knowledge, this is the first end-to-end framework that fine-tunes LLMs specifically for IaC security and the first paired misconfigured-to-fixed code corpus for training. Future research can advance this work in several directions. Expanding the dataset to include multiple IaC languages and scanners will improve generalizability. Tailoring remediations to organizational-specific configurations rather than generic examples would better reflect real-world deployment contexts. Incorporating larger or more advanced open-source LLMs could further test scalability and capacity. Exploring hybrid strategies such as retrieval-augmented generation (RAG) or reinforcement learning with security-specific feedback offers promising avenues for improving the accuracy and reliability of IaC remediation, and for supporting their integration into continuous deployment pipelines. Collectively, these efforts can strengthen the role of LLMs in securing modern cloud infrastructures.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation (NSF) under grant numbers DGE-1921485 (SFS), OAC-1917117 (CICI), and OAC-2319325 (CICI-UCSS).

REFERENCES

Ampel, B., Yang, C.-H., Hu, J., and Chen, H. 2025. "Large Language Models for Conducting Advanced Text Analytics Information Systems Research," *ACM Transactions on Management Information Systems* (16:1), Association for Computing Machinery (ACM), pp. 1–26.

Bessghaier, N., Begoug, M., Mebarki, C., Ouni, A., Sayagh, M., and Mkaouer, M. W. 2024. "On the Prevalence, Co-Occurrence, and Impact of Infrastructure-as-Code Smells," in *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, March 12. (https://doi.org/10.1109/saner60148.2024.00009).

Chang, Yupeng, Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Yi, Yu, P. S., Yang, Q., and Xie, X. 2024. "A Survey

on Evaluation of Large Language Models," *ACM Transactions on Intelligent Systems and Technology* (15:3), Association for Computing Machinery (ACM), pp. 1–45.

Chen, J., Ben Hai, S., Ben Zeev, S., Brewer, M., Breger, D. H., Oleyarsh, A., Quist, N., Sasson, A., and Zelivansky, A. 2023. "Unit 42 Cloud Threat Report, Volume 7: Navigating the Expanding Attack Surface," Palo Alto Networks. (https://www.paloaltonetworks.com/resources/research/unit-42-cloud-threat-report-volume-7).

Chiari, M., De Pascalis, M., and Pradella, M. 2022. "Static Analysis of Infrastructure as Code: A Survey," in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, IEEE, March. (https://doi.org/10.1109/icsa-c54293.2022.00049).

Citarella, A., Barbella, M., Ciobanu, M. G., De Marco, F., Di Biasi, L., and Tortora, G. 2025. "Assessing the Effectiveness of ROUGE as Unbiased Metric in Extractive vs. Abstractive Summarization Techniques," *Journal of Computational Science* (87:102571), Elsevier BV, p. 102571.

Diaz-De-Arcaya, J., López-De-Armentia, J., Zárate, G., and Torre-Bastida, A. I. 2024. "Towards the Self-Healing of Infrastructure as Code Projects Using Constrained LLM Technologies," in *Proceedings of the 5th ACM/IEEE International Workshop on Automated Program Repair*, New York, NY, USA: ACM, April 20, pp. 22–25.

Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Chang, B., Sun, X., Li, L., and Sui, Z. 2024. "A Survey on In-Context Learning," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Stroudsburg, PA, USA: Association for Computational Linguistics, November, pp. 1107–1128.

Goyal, M. K., and Chaturvedi, R. 2025. "Detecting Cloud Misconfigurations with RAG and Intelligent Agents: A Natural Language Understanding Approach," *Journal of Electrical Systems*. (https://doi.org/10.52783/jes.7882).

Han, Z., Gao, C., Liu, J., Zhang, J., and Zhang, S. Q. 2024. "Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey," *ArXiv [Cs.LG]*, , March 21. (http://arxiv.org/abs/2403.14608, accessed February 2, 2025).

Jiang, J., Wang, F., Shen, J., Kim, Sungju, and Kim, Sunghun. 2024. "A Survey on Large Language Models for Code Generation," *ArXiv [Cs.CL]*, , June 1. (http://arxiv.org/abs/2406.00515).

Ketonen, T., and Smolander, K. 2025. "Strategies and Challenges in Cloud-to-Cloud Migration Using Infrastructure as Code," in *Lecture Notes in Computer Science*, Lecture Notes in Computer Science, Cham: Springer Nature Switzerland, pp. 3–18.

Kon, P. T., Liu, J., Qiu, Y., Fan, W., He, T., Lin, L., and Wang. 2024. "Iac-Eval: A Code Generation Benchmark for Cloud Infrastructure-as-Code Programs," *Advances in Neural Information Processing Systems* (37), proceedings.neurips.cc, pp. 134488–134506.

Kwon, S., Lee, S., Kim, T., Ryu, D., and Baik, J. 2023. "Exploring LLM-Based Automated Repairing of Ansible Script in Edge-Cloud Infrastructures," *Journal of Web Engineering* (22:6), River Publishers, pp. 889–912.

Lazarine, B., Samtani, S., Patton, M., Zhu, H., Ullman, S., Ampel, B., and Chen, H. 2020. "Identifying Vulnerable GitHub Repositories and Users in Scientific Cyberinfrastructure: An Unsupervised Graph Embedding Approach," in *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*, , November, pp. 1–6.

Lee, J., Kang, S., and Ko, I.-Y. 2024. "An LLM-Driven Framework for Dynamic Infrastructure as Code Generation," in *Proceedings of the 25th International Middleware Conference: Demos, Posters and Doctoral Symposium*, New York, NY, USA: ACM, December 2, pp. 9–10.

Li, Y., Huang, Y., Ildiz, M. E., Rawat, A. S., and Oymak, S. 2024. "Mechanics of next Token Prediction with Self-Attention," in *In International Conference on Artificial Intelligence and Statistics*, (Pp. 685-693).

Low, E., Cheh, C., and Chen, B. 2024. "Repairing Infrastructure-as-Code Using Large Language Models," in *2024 IEEE Secure Development Conference (SecDev)*, IEEE, October 7, pp. 20–27.

Ntentos, E., Lueger, N. E., Simhandl, G., Zdun, U., Schneider, S., Scandariato, R., and Díaz Ferreyra, N. E. 2025. "On the Understandability of Design-Level Security Practices in Infrastructure-as-Code Scripts and Deployment Architectures," *ACM Transactions on Software Engineering and Methodology* (34:1), Association for Computing Machinery (ACM), pp. 1–37.

Patil, R., and Gudivada, V. 2024. "A Review of Current Trends, Techniques, and Challenges in Large Language Models (LLMs)," *Applied Sciences (Basel, Switzerland)* (14:5), MDPI AG, p. 2074.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. 2007. "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems : JMIS* (24:3), Informa UK Limited, pp. 45–77.

Ragothaman, H., and Udayakumar, S. K. 2024. "Optimizing Service Deployments with NLP Based Infrastructure Code Generation - an Automation Framework," in *2024 IEEE 2nd International Conference on Electrical Engineering, Computer and Information Technology (ICEECIT)*, IEEE, November 22, pp. 216–221.

Rahman, A., Parnin, C., and Williams, L. 2019. "The Seven Sins: Security Smells in Infrastructure as Code Scripts," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, May. (https://doi.org/10.1109/icse.2019.00033).

Shen, L., Sun, Y., Yu, Z., Ding, L., Tian, X., and Tao, D. 2025. "On Efficient Training of Large-Scale Deep Learning Models," *ACM Computing Surveys* (57:3), Association for Computing Machinery (ACM), pp. 1–36.

Toprani, D., and Madisetti, V. K. 2025. "LLM Agentic Workflow for Automated Vulnerability Detection and Remediation in Infrastructure-as-Code," *IEEE Access: Practical Innovations, Open Solutions* (13), ieeexplore.ieee.org, pp. 69175–69181.

Ullman, S., Samtani, S., Lazarine, B., Zhu, H., Ampel, B., Patton, M., and Chen, H. 2020. "Smart Vulnerability Assessment for Scientific Cyberinfrastructure: An Unsupervised Graph Embedding Approach," in *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*, , November, pp. 1–6.

Vizard, M. 2023. "Devs Increasingly Rely on IaC to Manage Multiple Clouds," *Devops.com*, , January 11. (https://devops.com/devs-increasingly-rely-on-iac-to-manage-multiple-clouds/, accessed September 11, 2025).

Vo, Q.-H., Dao, H., and Fukuda, K. 2025. "Harnessing the Power of LLMs for Code Smell Detection in Terraform Infrastructure as Code," in *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, IEEE, July 8, pp. 533–542.

Weyssow, M., Zhou, X., Kim, K., Lo, D., and Sahraoui, H. 2025. "Exploring Parameter-Efficient Fine-Tuning Techniques for Code Generation with Large Language Models," *ACM Transactions on Software Engineering and Methodology* (34:7), Association for Computing Machinery (ACM), pp. 1–25.

Xiang, Y., Yang, Z., Peng, J., Bauer, H., Kon, P. T. J., Qiu, Y., and Chen, A. 2025. "Automated Bug Discovery in Cloud Infrastructure-as-Code Updates with LLM Agents," in *2025 IEEE/ACM International Workshop on Cloud Intelligence & AIOps (AIOps)*, IEEE, May 3, pp. 20–25.

Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., and Zhang, Y. 2024. "A Survey on Large Language Model (LLM) Security and Privacy: The Good, The Bad, and The Ugly," *High-Confidence Computing* (4:2), Elsevier BV, p. 100211.