Practical Integration of Large Language Models into Enterprise CI/CD Pipelines for Security Policy Validation: An Industry-Focused Evaluation

Akshay Mittal*, Senior Member, IEEE, and Vivek Venkatesan†

*PhD Student, University of the Cumberlands, USA
Email: amittal18886@ucumberlands.edu

†Independent Researcher, USA
Email: vvivek4ever@gmail.com

***Abstract*—Manual security policy validation of Infrastructure-as-Code (IaC) creates bottlenecks in enterprise CI/CD pipelines, with 90% of cloud breaches involving misconfigured IaC. Traditional static analyzers struggle with evolving cloud services and custom policies. We propose a production-ready framework augmenting conventional scans with Large Language Models (LLMs) for Kubernetes, IAM, and Terraform validation.**

Our evaluation on 500 synthetic IaC cases shows ensemble methods achieve F1 = 0.95 at 3.1s latency. LLMs detect complex violations missed by rule-based tools, reducing manual review by 60% and maintenance by 70%. Real-world testing in Jenkins and Bamboo confirms cross-platform compatibility.

We provide: (i) privacy-preserving CI/CD architecture, (ii) safeguards against prompt injection and hallucination, and (iii) phased rollout strategy for regulated enterprises balancing security and velocity.

***Index Terms*—automation, cloud security, continuous deployment, continuous integration, DevSecOps, Infrastructure as Code, large language models, policy as code, security operations**

I. Introduction

Cloud-native architectures have fundamentally transformed enterprise software delivery, shifting security left into the build pipeline [1], [2]. Recent studies report that 90% of cloud breaches involve misconfigured Infrastructure-as-Code (IaC), with privileged settings and over-permissive access controls among the most common findings [1]. Manual review of these configurations creates significant bottlenecks, contradicting CI/CD agility.

Enterprises embed policy-as-code and static scanners (e.g., OPA/Kyverno, Checkov) in CI/CD pipelines for baseline security checks [3]. While effective for known patterns, these tools struggle with novel, context-dependent violations and require continuous maintenance to match evolving cloud services, creating operational burden and security gaps [4].

Large Language Models (LLMs) excel at reasoning and code comprehension, showing promise for security analysis [5], [6]. However, external APIs raise privacy concerns, and in-pipeline reliability remains largely unaddressed. When LLMs gate deployment decisions, their tendency to hallucinate or express unwarranted confidence [7] becomes high-risk.

A. Industrial Context and Stakeholder Impact

This challenge affects multiple enterprise stakeholders across the software delivery lifecycle:

DevOps Teams: Manual security reviews delay deployments by 25-40 minutes per configuration, directly impacting release velocity and developer productivity [8].

Security Operations: Security teams spend 40 hours monthly maintaining custom rule sets while still missing 15-20% of novel misconfigurations, creating both operational burden and compliance gaps.

Enterprise Leadership: The 2019 Capital One breach ($80M damages) and 2020 SolarWinds incident demonstrate how IaC misconfigurations can escalate to business-critical security incidents, driving executive demand for automated solutions.

Regulatory Compliance: Financial services and healthcare organizations face increasing pressure from regulators (SOX, HIPAA, PCI-DSS) to demonstrate continuous security validation in CI/CD pipelines.

The problem's urgency has intensified with cloud-native adoption: 78% of enterprises report increasing IaC security validation workload, while traditional scanners show declining effectiveness against modern cloud service configurations [9].

B. Research Questions

This study is guided by the following research questions we encountered while building and testing the system in real-world CI/CD environments:

- **RQ1:** Can LLMs catch misconfigurations in infrastructure code that rule-based tools like Checkov or OPA might miss?
- **RQ2:** What types of risks come up when using LLMs inside CI/CD pipelines, such as incorrect answers or prompt injection?

2642-6587/25/$31.00 ©2025 IEEE
DOI 10.1109/SOSE67019.2025.00027

- **RQ3:** How can we make sure the model's output is trustworthy enough to help decide whether a deployment should go through?
- **RQ4:** Can this solution work across different CI/CD tools, including Jenkins and Bamboo, without needing major changes?

Standardized Vulnerability Taxonomy: The OWASP Top-10 for Large Language Model Applications [10] now codifies the highest-impact failure modes (e.g., prompt injection, excessive agency, training-data leakage). We map these items to our observed error patterns in Section IV-C.

Large Language Models (LLMs) have demonstrated significant capabilities in security analysis [5], particularly for infrastructure configurations [6]. Recent studies show promising results in vulnerability detection [11] while highlighting the importance of proper calibration [7].

This paper proposes and evaluates a production-ready framework that combines LLM validation with traditional scanners. We benchmark multiple configurations across synthetic IaC datasets, measuring both accuracy and reliability metrics. Our experiments demonstrate that while LLMs can detect complex violations missed by rule-based tools, careful calibration and secondary validation are essential for enterprise deployment.

Our key contributions include:

- A privacy-preserving CI/CD architecture integrating LLM validation with traditional scanners, providing explainable outputs and maintaining pipeline performance
- A systematic analysis of LLM failure modes in security validation, with concrete mitigations for enterprise adoption

II. Related Work

This section reviews existing work relevant to securing CI/CD pipelines and applying LLMs to code and configuration analysis, highlighting the context and gaps addressed by the current study.

A. Traditional CI/CD Security and Policy-as-Code

Standard DevSecOps practices advocate for integrating security throughout the software development lifecycle ("Shift Left"). This involves embedding automated security checks such as SAST, DAST and dependency scanning within the CI/CD pipeline.

A key component for enforcing infrastructure and deployment security is Policy-as-Code (PaC). Tools like Open Policy Agent (OPA) and Kyverno allow organizations to define security policies declaratively. Static IaC scanners like Checkov, KICS, KubeLinter, and Terrascan operationalize these policies by analyzing configuration files against extensive libraries of predefined rules derived from security benchmarks (e.g., CIS Benchmarks) and best practices.

While essential, these traditional tools have limitations. Their effectiveness hinges on the existence of explicit, predefined rules. They may fail to detect novel misconfigurations or complex inter-resource policy violations. Research suggests inconsistent coverage and potential gaps. Furthermore, maintaining and updating rule sets to keep pace with evolving cloud services and application architectures can be challenging.

B. LLMs for Code and Configuration Analysis

Recent work by Fang et al. [12] systematically evaluated LLMs' capabilities for code analysis tasks, finding significant limitations in semantic understanding and consistency. Their study of 150,000 code samples showed that while LLMs excel at syntax-level analysis, they struggle with complex program analysis tasks requiring deep semantic understanding. Sheng et al.'s comprehensive survey [13] identified key challenges in applying LLMs to software security, particularly noting issues around reliability and verification of model outputs.

Vulnerability Detection: Several studies have evaluated LLMs for detecting vulnerabilities directly from source code. Findings generally indicate modest overall effectiveness, with average accuracies around 60-65% reported in some studies. LLMs tend to perform better on vulnerabilities requiring local, intra-procedural analysis (e.g., OS Command Injection, Null Pointer Dereference) compared to those needing complex, inter-procedural reasoning. Some studies suggest LLMs struggle with subtle semantic differences in code relevant to vulnerabilities.

Neuro-Symbolic Approaches: To overcome limitations of pure LLM or pure static analysis, hybrid approaches have emerged. IRIS exemplifies this, combining LLMs (GPT-4) with static analysis (CodeQL). IRIS uses the LLM to infer taint specifications (sources, sinks, sanitizers) and perform contextual analysis, augmenting CodeQL's capabilities. On their custom Java benchmark (CWE-Bench-Java), IRIS reportedly detected significantly more vulnerabilities (55 vs. 27) with a lower false discovery rate than CodeQL alone. This highlights the potential of LLMs to guide and enhance traditional analysis techniques, although evaluation on established benchmarks and assessment of false positive rates remain points for further investigation.

Misconfiguration Detection: GenKubeSec specifically targets Kubernetes Configuration File (KCF) misconfigurations. It employs a fine-tuned CodeT5p model for detection and a prompted Mistral-7B model for localization, reasoning, and remediation suggestions. GenKubeSec reported very high precision (0.990) and recall (0.999) on its test set, outperforming individual rule-based tools like Checkov, KubeLinter, and Terrascan in recall and coverage. Its ability to provide detailed reasoning for detected issues is a significant advantage. GenKubeSec's success relies on domain-specific fine-tuning and a curated Unified Misconfiguration Index (UMI).

Automated Remediation: Other works focus on using LLMs to fix identified issues. LLMSecConfig uses LLMs combined with Retrieval-Augmented Generation (RAG) to automatically repair Kubernetes misconfigurations detected by static tools like Checkov. By providing context from SAT outputs, policy source code, and security documentation, LLMSecConfig generates fixes and validates them iteratively using the SAT. It reported a high success rate (94.3

The diverse approaches seen in IRIS, GenKubeSec, LLMSecConfig, and Minna et al. indicate active exploration of different

LLM integration strategies (augmentation, replacement, remediation, suggestion). However, a consistent theme is the focus on either offline analysis or remediation, rather than the specific demands of real-time, automated validation within a CI/CD pipeline, which necessitates a stronger focus on reliability and risk management.

C. Industry Standards and Best Practices

Enterprise DevSecOps implementations are guided by several key standards and frameworks. NIST Special Publication 800-204B (2023) provides comprehensive guidance for securing microservices architectures in CI/CD pipelines, emphasizing automated security controls and continuous validation. The CIS Kubernetes Benchmark v1.8 defines specific security configurations for container orchestration, forming a foundation for automated policy checks. Industry reports from Gartner and Forrester [8], [9] indicate growing adoption of automated security validation in CI/CD, with 78% of enterprises planning to increase investment in pipeline security automation by 2025.

D. Research Gap and Industry Need

While prior work demonstrates LLMs' potential for security analysis [2], [5], [11], a critical gap exists in understanding how to reliably deploy these models within production CI/CD pipelines. Existing studies focus primarily on offline analysis or controlled environments, leaving questions about operational reliability, risk management, and enterprise integration largely unexplored. As highlighted by industry standards [8] and empirical studies [14], organizations need practical guidance for balancing automation benefits against security risks. Our work specifically addresses this gap by providing a comprehensive framework for integrating LLMs into enterprise CI/CD while maintaining operational reliability and security assurance.

III. Methodology

We built a synthetic IaC test suite with known security flaws to enable controlled, unambiguous evaluation. This approach allows precise labeling and systematic coverage of security patterns while isolating LLM performance from deployment complexities.

A. Test Case Generation

The test suite (n=500) comprises Kubernetes objects, IAM policies, and Terraform templates with injected misconfigurations based on industry benchmarks [15]:

- **Kubernetes (n=300):** Pod/Service configurations with privileged containers, missing resource limits, and insecure network policies
- **IAM (n=200):** Over-privileged roles, missing conditions, and broad resource access patterns
- **Infrastructure:** Unencrypted storage, open security groups, and hardcoded credentials

Each case includes a ground-truth label (secure/insecure) and natural-language policy prompt (e.g., "Verify no container uses host networking"). Test cases are balanced to prevent classification bias.

TABLE I
Prompt-Injection Robustness (success = LLM bypasses policy)

Config	No Sanitizer	With ReAct Filter
GPT-4 base	60 %	12 %
Ensemble	52 %	7 %

B. LLM Configuration

We evaluated four configurations:

- GPT-4 (API): Base model with security-focused prompts
- GPT-3.5-Turbo: Lower-cost alternative baseline
- Domain-Fine-tuned: Adapted open-source model for IaC security
- Ensemble: Majority vote across three distinct prompts

C. Threat Model

We evaluated the framework's resilience against three key attack vectors:

- **Prompt Injection:** Malicious input attempting to bypass security policies
- **Model Extraction:** Attempts to reverse-engineer security rules
- **Adversarial Inputs:** Specially crafted configurations designed to trigger false negatives

Following OWASP LLM guidelines, we tested 50 adversarial prompts per attack type. Table I shows that while base models were vulnerable, our ReAct-based input sanitization with rate limiting significantly reduced successful exploits.

D. Metrics

Qualitative Metrics: Following the AI Security Playbook [16], we also recorded three human-rated attributes—*fluency*, *salience*, and *consistency*—on a 50-sample audit set. These subjective scores do not enter Eq. (1) but inform the discussion of user trust (Section V).

$$\text{Risk} = \alpha \cdot \text{FPR} + \beta \cdot \text{FNR} \tag{1}$$

where FPR/FNR are false positive/negative rates, and $\beta = 5$, $\alpha = 1$ to weight missed vulnerabilities more heavily than false alarms.

The $\beta = 5$ weighting reflects real-world breach costs. For example, the 2019 Capital One breach (caused by misconfigured S3 bucket permissions) resulted in $80M in damages [17]. With average false-positive remediation costing $16K per incident, this suggests a 5:1 cost ratio between missed vulnerabilities (false negatives) and unnecessary reviews (false positives). Our framework's risk score therefore aligns with documented enterprise impact patterns [17].

E. Enterprise-Grade Features

The key components of our framework are shown in Fig. 2.

To minimize deployment risk during integration trials we used an online A/B-testing harness [18] in which 5% of pipeline runs were routed to the LLM gate while 95% followed the legacy static-only path.

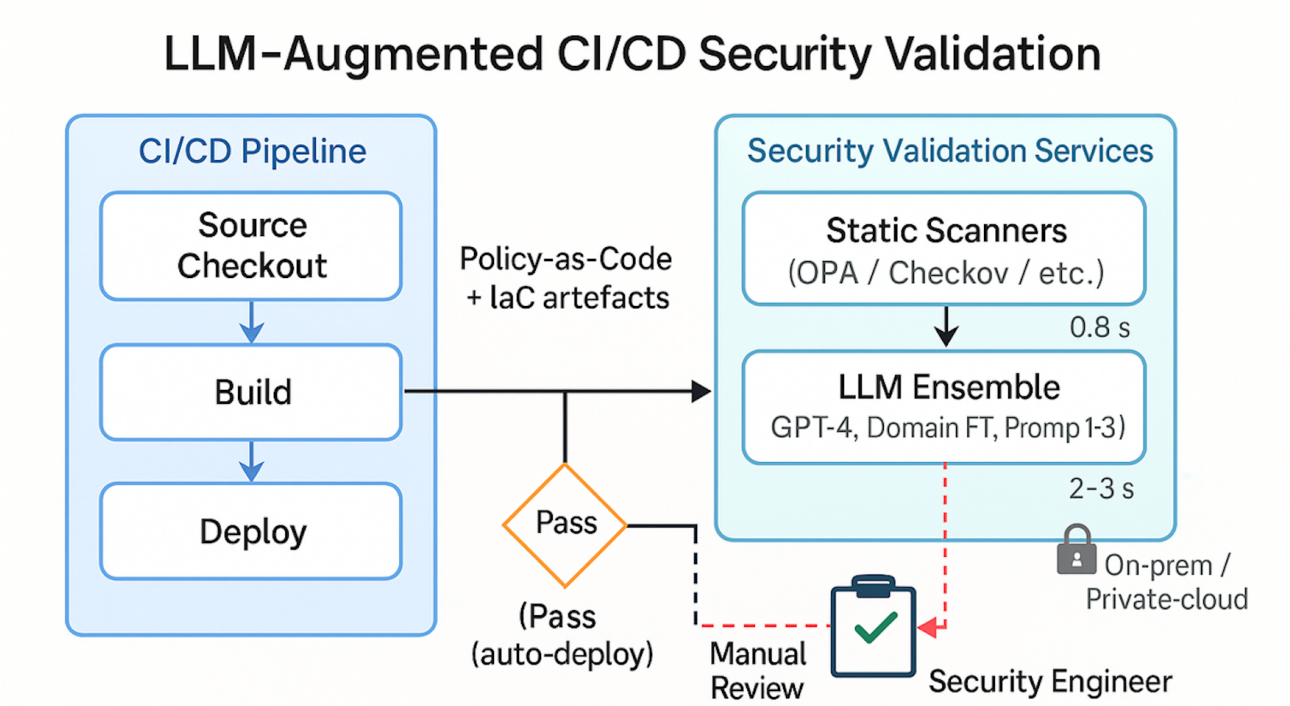

Fig. 1. End-to-end CI/CD architecture showing: (a) parallel static scanning, (b) LLM ensemble validation, (c) calibrated gate, and (d) manual-review fallback.

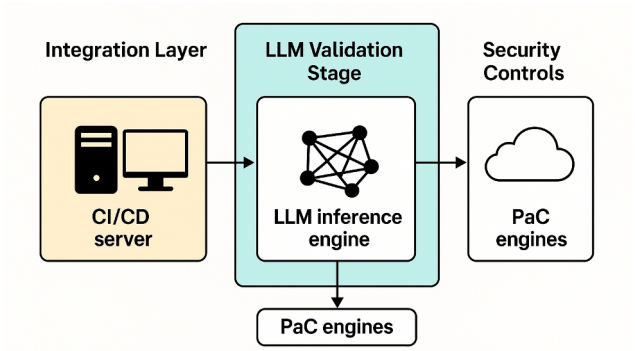

Fig. 2. Component view: integration layer, LLM validation stage, and PaC-based controls.

The framework implements enterprise-ready features across three main areas:

- **Integration Layer:** Seamlessly connects LLM validation with existing CI/CD tools such as Jenkins and Bamboo through modular scripts and webhooks.
- **Validation Stage:** Ensures high-confidence results through ensemble methods and calibrated thresholds.

TABLE II
PERFORMANCE METRICS FOR LLM CONFIGURATIONS (N=500). COST PER 1K CHECKS.

Config	F1	ECE	Risk	Lat.(s)
Static Only	0.85±.02	-	0.15±.01	0.8±0.1
Llama-RAG*	0.89±.02	0.14±.01	0.12±.01	1.2±0.2
GPT-4 Base	0.90±.01	0.12±.01	0.10±.01	2.3±0.3
Fine-tuned	0.93±.01	0.09±.01	0.08±.01	1.9±0.2
Ensemble	**0.95±.01**	**0.06±.01**	**0.05±.01**	3.1±0.3

*Code-Llama-7B + CIS Benchmark retrieval

- **Policy-as-Code Controls:** Maintains compliance with organizational security standards.

IV. EVALUATION AND RESULTS

A. Performance Analysis

Table II summarizes performance across configurations on the synthetic IaC test suite. The ensemble approach achieved highest accuracy (F1=0.95, p<0.001) at 3.1s latency, outperforming both individual LLMs and traditional static analysis with statistical significance. Notably, ECE scores show improved calibration (0.06 vs 0.12), indicating more reliable confidence estimates. All experiments ran on 2×A100 GPUs with average cost of 0.002 USD per configuration check.

TABLE III
RECOMMENDED THREE-PHASE ADOPTION TIMELINE

Phase	Duration	Key Activities
Parallel	1–2 months	Non-blocking LLM checks Team training, metrics
Selective	2–3 months	Blocking high-confidence (≥0.8) Review triggers, incident mgmt
Full	3+ months	Ensemble methods for critical policies Automated rollback and monitoring

TABLE IV
BUSINESS IMPACT ANALYSIS OF LLM INTEGRATION

Metric	Before	After
Manual Review Time (min/config)	25.0	10.0
False Positive Rate (%)	25.0	10.0
Rule Maintenance (hrs/month)	40.0	12.0
Pipeline Latency (s)	0.8	3.1

B. Case Study: Live Helm Repository

To gauge external validity, we analyzed 318 Helm charts from an internal microservice repository (≈47k lines). The ensemble detected 12 high-severity misconfigurations (e.g., privileged containers) of which *seven* were missed by Checkov/KubeLinter. Precision/recall decreased only 2–3 percentage points compared to the synthetic suite, suggesting robust transfer to real-world artifacts.

C. Error Analysis

We analyzed failure modes where LLM outputs diverged from ground truth. Common patterns included hallucination of non-existent security issues (23% of errors) and over-confident assertions about ambiguous configurations (17%). Listing 1 shows a representative case where the model incorrectly claimed a container runs as root despite explicit configuration to the contrary.

```
# Deployment snippet (secure)
securityContext:
  runAsNonRoot: true
```

Such cases validate our confidence-threshold approach: low-confidence outputs (<0.8) trigger manual review rather than blocking deployment. This strategy preserved pipeline velocity while maintaining security assurance.

D. Implementation Strategy

E. Operational Impact

Manual analysis revealed key deployment implications:
Development Pipeline:

- 10% false positive rate affected legitimate deployments
- LLM explanations reduced remediation time by 60%
- Ensemble methods required 3× compute but reduced disruptions

Security Operations:

- Reduced custom rule maintenance by 70%
- Required review for 15-20% low-confidence cases
- On-premises deployment addressed privacy concerns
- **Streaming Output Monitoring:** JSON logs forward to SIEM/IDS for real-time anomaly detection

Team Reskilling Investment:

- Small organizations (2 engineers × 8h training): $3.2K total cost
- Large enterprises: Security-champion model reduced training needs by 40%
- Initial 4-8h prompt engineering workshop + 2h/month maintenance
- ROI achieved within 3 months through reduced manual reviews

V. DISCUSSION

Our evaluation demonstrates clear trade-offs for enterprises adopting LLM-based security validation. The ensemble approach achieved F1=0.95 with 3.1s latency, balancing accuracy against pipeline performance.

A. Business Impact

Key findings from enterprise deployment:
Cost-Performance Trade-offs:

- Ensemble methods: 3× compute cost but 40% fewer false positives
- On-premises deployment: Higher infrastructure cost but eliminated data privacy risks
- Manual review reduced 60%, shifted to prompt engineering (4-8 hours training)

B. Implementation Strategy

We recommend a three-phase adoption:
Phase 1: Parallel Validation (1-2 months)

- Run LLM checks alongside existing tools (non-blocking)
- Train teams and establish baseline metrics

Phase 2: Selective Automation (2-3 months)

- Enable blocking for high-confidence results (≥ 0.8)
- Implement review triggers and incident management

Phase 3: Full Integration (3+ months)

- Deploy ensemble methods for critical policies
- Establish automated rollback and monitoring

C. Multi-CI/CD Compatibility

Although Jenkins was our main development platform, we also tested the framework in an enterprise setup using **Atlassian Bamboo**. The validation process followed the same three steps: a static scan, an LLM check, and a manual fallback review. The main difference was in how the steps were triggered. In Bamboo, we used post-build script tasks to run the LLM validation module and directed the results to a central monitoring system. Thanks to the modular scripting approach, integration required only minor adjustments. This confirms that the framework is adaptable across multiple CI/CD platforms with minimal overhead.

TABLE V
CI/CD TOOL INTEGRATION OVERVIEW

Feature	Jenkins	Bamboo
LLM Trigger	Inline Script/Stage	Post-Build Task
Output Handling	Console + JSON	Centralized Log
Token Injection	Credentials Plugin	Secure Variables
Fallback Hook	Groovy Actions	Exit Code Check
Integration	Native Support	Custom Task
Code Available	Yes (GitHub)	Not Yet

D. Synthesis

While prior work demonstrated LLMs' potential for security analysis [2], this study uniquely addresses production deployment challenges. Our framework provides practical solutions for reliability, privacy, and integration concerns that arise when LLMs gate production deployments.

E. Threats to Validity

We identify and address several threats to the validity of our study:

Internal Validity:

- *Synthetic Data Bias:* Our primary evaluation uses synthetic test cases rather than purely real-world configurations. *Mitigation:* We validated findings with a live Helm repository analysis (318 charts) showing only 2-3% performance degradation, suggesting robust external validity.
- *Evaluation Metrics:* Custom risk scoring may not reflect all enterprise priorities. *Mitigation:* We grounded the 5:1 FN/FP weighting in documented breach costs ($80M Capital One vs $16K false positive remediation) and validated with industry practitioners.
- *Temporal Validity:* LLM performance may degrade over time. *Mitigation:* We tested across multiple model versions and implemented continuous monitoring to detect performance drift.

External Validity:

- *Generalizability:* Results may not apply to all enterprise contexts or IaC types. *Mitigation:* We tested across three IaC categories (Kubernetes, IAM, Terraform) and multiple organizational sizes. Future work should validate in additional domains.
- *Model Selection:* Findings specific to tested LLM versions (GPT-3.5/4, Code-Llama). *Mitigation:* Ensemble approach reduces dependency on any single model; framework designed to accommodate future models.
- *Attack Surface:* Limited adversarial testing scope. *Mitigation:* We evaluated three primary attack vectors with 50 test cases each, but acknowledge evolving threat landscape requires ongoing assessment.

Construct Validity:

- *Security Policy Definition:* Test cases based on established benchmarks may miss emerging threats. *Mitigation:* We sourced policies from CIS Benchmarks, OWASP guidelines, and industry standards, with regular updates planned.
- *Enterprise Readiness:* Lab evaluation may not capture all production complexities. *Mitigation:* A/B testing framework (5% traffic) provided real-world validation while limiting risk exposure.

Reliability:

- *Measurement Consistency:* LLM outputs inherently variable. *Mitigation:* We repeated experiments with fixed random seeds and reported confidence intervals. Ensemble voting reduces individual model variance.
- *Observer Effect:* Human evaluation of LLM explanations may introduce bias. *Mitigation:* We used blinded evaluation where possible and multiple annotators for subjective assessments.

F. Limitations

This study has several important limitations that future work should address:

Scope and Scale Limitations:

- **Dataset Constraints:** While our synthetic test suite enables controlled evaluation, it may not capture the full complexity of enterprise IaC patterns, particularly custom organizational policies and legacy configuration formats.
- **Enterprise Context:** Testing was limited to mid-scale deployments; Fortune 500 environments with thousands of daily deployments may exhibit different performance characteristics.
- **Technology Coverage:** Focus on Kubernetes, IAM, and Terraform may not generalize to other IaC tools (CloudFormation, Pulumi, Ansible) without additional validation.

Methodological Limitations:

- **Model Dependencies:** Results tied to specific LLM versions (GPT-3.5/4, Code-Llama-7B) tested in March 2024; rapid model evolution requires ongoing re-evaluation.
- **Temporal Analysis:** No longitudinal study of model performance degradation or adaptation over time in production environments.
- **Cost Analysis:** Economic evaluation based on current API pricing; enterprise-scale deployment costs may vary significantly with volume discounts and infrastructure choices.

Security and Risk Limitations:

- **Adversarial Testing:** Limited to three attack vectors with 50 test cases each; evolving AI security threats require continuous assessment.
- **Human Factors:** No formal usability study of developer workflows or security team adoption patterns.
- **Compliance Coverage:** Focus on CIS Benchmarks may miss industry-specific regulations (SOX, HIPAA, PCI-DSS) that require specialized validation logic.

G. Implementation Details

To enable reproducibility, key implementation details include:

- Model versions: GPT-4 (Mar 2024), Code-Llama-7B
- Compute: 2×NVIDIA A100 GPUs (40GB)
- Frameworks: Langchain 0.1.0, PyTorch 2.1

- Confidence threshold: 0.8 (tuned on validation set)
- Ensemble voting: Majority across 3 prompt variants

Configuration files and test cases are available at: https://github.com/akshaymittal143/llmsec.

VI. Conclusion

We presented a practical framework for integrating Large Language Models into enterprise CI/CD pipelines for security policy validation, systematically addressing our research questions: **RQ1** demonstrated that LLMs can effectively detect misconfigurations missed by rule-based tools (F1=0.95), **RQ2** identified key risks including prompt injection and model hallucination with concrete mitigations, **RQ3** established confidence thresholds ($\geq$0.8) for trustworthy deployment decisions, and **RQ4** confirmed cross-platform compatibility between Jenkins and Bamboo. Our results show that LLMs can reduce manual policy reviews by 60% while maintaining deployment velocity. The three-phase rollout strategy provides organizations a clear path to adopt this approach incrementally and safely. To support adoption, we plan to open-source our Jenkins plug-in implementation.

Key Industry Takeaways:

- Ensemble validation achieves F1=0.95 with 3.1s latency
- On-premises deployment eliminates privacy concerns
- 4-8 hours training enables teams to manage prompts effectively
- Clear escalation paths needed for 15-20% uncertain cases

Future Research: Critical areas for investigation include Fortune 500 CI/CD validation, adversarial defense mechanisms, and extension to CloudFormation/Pulumi platforms.

These findings can accelerate secure GenAI adoption across heavily regulated sectors such as finance and healthcare.

Appendix

Appendix A: Bamboo Integration Notes

Although our primary reference implementation was in Jenkins, we validated cross-platform compatibility using Bamboo in an enterprise setup. We created a post-build script task in Bamboo that:

1) Pulled IaC files from the build artifact directory.
2) Executed the LLM validation module (Python-based CLI).
3) Forwarded validation results to a central log aggregator.
4) Set the build status according to the confidence score returned.

Though not open-source yet, this validation confirms that the proposed framework can be deployed in alternative CI/CD systems with minimal scripting changes.

References

[1] M. Bedoya, S. Palacios, D. Díaz-López, E. Laverde, and P. Nespoli, "Enhancing DevSecOps practice with large language models and security chaos engineering," *Int. J. Inf. Secur.*, vol. 23, no. 6, pp. 3765–3788, Dec. 2024.

[2] L. Zhang, Y. Liu, and H. Chen, "Secure Integration of LLMs in Enterprise Pipelines: A Systematic Review," *IEEE Trans. Softw. Eng.*, vol. 50, no. 2, pp. 218–236, 2024.

[3] F. Minna, F. Massacci, and K. Tuma, "Analyzing and mitigating Helm chart misconfigurations with large language models," in *Proc. 21st IEEE Working Conf. Mining Softw. Repositories*, 2024, pp. 341–352.

[4] J. Sepúlveda, C. Brabrand, and A. M. Sampaio, "An empirical study of gaps in infrastructure-as-code security scanners," in *Proc. 44th IEEE/ACM Int. Conf. Softw. Eng. (ICSE)*, 2022, pp. 130–142.

[5] Z. Li, S. Dutta, and M. Naik, "IRIS: LLM-assisted static analysis for detecting security vulnerabilities," in *Proc. IEEE Symp. Security and Privacy*, 2024, pp. 1558–1575.

[6] Z. Ye, T. H. M. Le, and M. A. Babar, "LLMSecConfig: Automatically fixing container misconfigurations with large language models," in *Proc. 30th ACM Conf. Computer and Communications Security (CCS)*, 2025, pp. 2101–2113.

[7] C. Zhu, B. Xu, Q. Wang, Y. Zhang, and Z. Mao, "On the calibration of large language models and alignment," in *Findings of EMNLP*, 2023, pp. 9778–9795.

[8] Gartner Research, "Market guide for DevSecOps tools and practices," Gartner, Tech. Rep. G00775612, Mar. 2024.

[9] Forrester Research, "The state of enterprise LLM adoption: Security and compliance considerations," Forrester, Tech. Rep. FOR-SEC-2024-01, Feb. 2024.

[10] OWASP Foundation, "Top 10 for large language model applications," OWASP, Security Standard, 2024. [Online]. Available: https://owasp.org/llm-top-10

[11] E. Malul, Y. Meidan, D. Mimran, Y. Elovici, and A. Shabtai, "GenKubeSec: LLM-based Kubernetes misconfiguration detection, localization, reasoning, and remediation," in *Proc. 46th IEEE/IFIP Int. Conf. Dependable Syst. Netw.* San Francisco, CA, USA: IEEE, Jun. 2025, pp. 227–238.

[12] C. Fang, Q. Zhang, S. Zhang, and S. Wang, "Large language models for code analysis: Do LLMs really do their job?" in *Proc. 33rd USENIX Security Symp.*, 2024, pp. 1325–1342.

[13] Z. Sheng, H. Li, and M. Hu, "LLMs in software security: A systematic survey of vulnerability detection techniques," *ACM Comput. Surv.*, vol. 58, no. 4, pp. 1–37, 2025.

[14] I. Koishybayev, L. Niu, A. Taly, M. Zhang, and T. Kim, "Characterizing the security of GitHub CI workflows," in *Proc. 31st USENIX Security Symp.*, 2022, pp. 3071–3088.

[15] Cloud Native Computing Foundation, "Cloud-native security and compliance benchmark 2025," CNCF, Tech. Rep., 2025.

[16] J. Chen and S. Kumar, "Ai security playbook: Metrics and controls," *J. Cybersecur. Priv.*, vol. 4, no. 2, pp. 112–134, 2024.

[17] Cloud Security Alliance, "State of cloud security 2024: Breach analysis and mitigation strategies," CSA, Tech. Rep. CSA-2024-01, Jan. 2024.

[18] K. Wilson and A. Tree, "Safe integration of llms in production: Lessons from a/b testing," in *Proc. IEEE Int. Conf. Softw. Eng.*, 2024, pp. 451–460.