Automated Security Compliance in Ci/Cd: A Natural Language Processing (NLP) Approach to Detecting Vulnerabilities in Infrastructure-As-Code (Terraform/Bicep)

Manoj Kumar Kagitha

DevOps Engineer at Eficens Systems Eastvale, California, USA

ABSTRACT

The advent of Infrastructure as Code (IaC) has transformed how organizations manage and provision their cloud resources but has also brought large security issues arising from the messed-up configuration that directly reaches the production systems. Conventional security scanning tools mainly depend on pattern-matching and rule-based mechanism and face problems when dealing with complex security issues and vulnerabilities hidden in IaC templates. In this context, this investigation has described using Natural Language Processing (NLP) approaches to improve automated security compliance assurance in Continuous Integration/Continuous Deployment (CI/CD) pipelines, with a particular emphasis on Terraform and Azure Bicep configurations. This study implemented these NLP models on 850 real-world IaC files from various enterprise projects in order to compare different NLP approaches such as transformer-based models, semantic analysis, and hybrid detection frameworks. This study consisted of inquiry research methods such as both quantitative performance assessment and qualitative evaluations concerning developer experience; involving 62 survey respondents from DevOps practices, the results revealed that security scanning greatly benefitted from the NLP process, yielding 89% precision for vulnerability detection, which is a 34% and 28% improvement over non-NLP methods in terms of precision and recall, respectively. Furthermore, the NLP approach raised the true positive rate by 61% and identified an additional 23% true security issues, excelling in the detection of contextual misconfigurations, if any, that rule-based system-based approaches had missed. Embedded within the implementation difficulties are such challenges as training requirements for model data and the computational overhead of CI/CD pipelines and answering the interpretations required by security teams. Real-world empirical evidence from this research validates the superior efficacy of NLP for IaC security and offers useful guidance to organizations interested in automating their security compliance.

Keywords: Infrastructure-as-Code, security compliance, Natural Language Processing, CI/CD pipelines, vulnerability detection, Terraform, Azure Bicep, DevSecOps, automated security scanning

1. INTRODUCTION

Infrastructure as Code(or IaC), a pillar of modern cloud deployment, allows organizations to state, version, and implement their infrastructure through software configuration rather than manual configuration. Through IaC, developers can now dictate code that represents an entire cloud environment, bringing software-engineering practices to infrastructure management (Morris, 2020). However, with this great paradigm shift comes a major challenge: when put into action through the automated CI/CD pipeline, misconfigurations are multiplied and spread violently.

Untangling security compliance is challenging, with slightly over 60% of security breaches via cloud being results of misconfigurations as compared to software bugs; Infrastructure as Code only amplifies and solidifies the risk further through scripting into potentially insecure vulnerabilities that are widely replicated across various environments (Shameli-Sendi et al., 2022). Creating a single insecure Terraform module that outlines security groups with overly open permissions or other configuration elements will enable hundreds of resources to be configured insecurely when deployed in countless deployments. Conventional manual security auditing cannot possibly keep up with the speed of modern software development practices, where entire infrastructure changes are pushed at least two dozen times within a week.

The totality of the current automated security scanning implementations for IaC mostly rely heavily upon static analysis with a set of predefined rules. Tools such as Checkov, tfsec, and Terrascan compare configuration patterns against known vulnerabilities and signal any security best practice violations (Rahman et al., 2021). Despite their very usefulness, such rule-based systems certainly have their own intrinsic limitations. To begin with, these systems neither possess contextual intelligence nor recognize that a configuration that is fine in development may create serious risk in production. Furthermore, the alert fatigue generated due to their large number of false-positive results eventually weakens security teams. Thus, they are unable to ascertain any vulnerabilities' rarer patterns that are not clearly captured by their rule databases.

It is expected that Artificial Intelligence's vast potential for creative ways of detecting might expose ways other than rule-based reliance. Recent Natural Language Processing is transforming the scene within artificial intelligence by both modernizing and transcending. With models now possible, such as BERT and GPT, which perform using transformers, the area of AI is able to vastly manipulate code definitions to give nuanced meaning to text that, despite the best efforts of rule-based structures, worth the word y of their passion attempted to analyze (Chen et al., 2021). Unlike procedural code being specific values that the NLP models can analyze, even more semantic power seems present in understanding IaC templates, which are declarative configurations. Security vulnerabilities generally come into the light via the combination of certain configuration parameters, definition relationships, and precise contextual circumstances-an occurrence that some NLP methodologies can detect.

Notwithstanding the theoretical promise, a limited number of research works have dealt with NLP technologies for security in an IaC production environment. The existing research works have concentrated on software vulnerabilities or limited themselves to lab datasets. The practical success, performance attributes, and issues suffered by IaC security scanning based on NLP in real-world CI/CD environments, for all that, still needs robust investigation.

The present research work thus contributes to aforementioned gap by conducting a systematic investigation on NLP methods for conducting security compliance check on the IaC templates based on Terraform and Bicep. The objectives pursued in this study are as follows: evaluating the detection accuracy of various NLP techniques against conventional static analysis; assessing practical implementation in view of CI/CD pipeline restrictions; and identifying organizational and technical factors that would work in favor of this methodology.

I would say this study offered a lot of contributions. Empirical findings all featured very rigorous assessments of NLP-based security scanning across several real-world IaC repositories - essentially providing evaluations of precision, recall, and false positive rates. Practically, it gives tangible guidance on how to actually implement NLP-based IaC security scanning: helping sort out computational requirements, model training, and integration with CI/CD. Lastly, it has contributed theoretically to understanding how NLP analytical power could be applied to IaC security, and hence would raise NLP research above the modeling of the classical code analysis scope.

In Section 2, literature on IaC security and NLP application is reviewed, whereas Section 3 gives research objectives and scope. Section 4 describes the experimental methodology. Sections 5 and 6 present quantitative performance results and qualitative practitioner insights, respectively. Section 7 takes up discussions of implications, detailing recommendations in Section 8.

2. OBJECTIVES

Some specific goals are included in the study:

- The main purpose: To assess the efficiency of various NLP applications targeted at security defect identification automation in Infrastructure-as-Code templates as part of CI/CD workflow while considering rule-based instruments.
- Objective 2: To develop and validate an NLP-augmented security flaw analysis framework capable of recognizing both template-oriented and context-dependent vulnerabilities in Terraform and Azure Bicep configurations.
- Objective 3: Measuring the advantage of NLP against static baseline saving on accuracy of detection, false positive elimination, coverage extension.
- Practical assessment for opportunistic implementation, computational aspects involved, CI/CD pipeline integration, and the choice of model/training methods for practical applications are considered.
- Fourth secondary goal: To-be-identified challenges faced by an organization, barriers to the adoption process, and success factors in the integration of NLP-based security scanning activities in DevSecOps workflows from the practitioners' perspectives.

3. SCOPE OF STUDY

The study works within the following constrains:

- Terraform Being Investigated: The workek focuses exclusively on HashiCorp Terraform (HCL syntax) and Microsoft Azure Bicep templates, the two most widely adopted IaC languages for cloud infrastructure.
- Security Significance: The study exclusively looks at the configuration vulnerabilities numberlike overly permissive access controls, unneeded network configurations, unencrypted resources, suspicious open access to secrets from logs, and conformance issues-not application code vulnerabilities.
- IaC Files Data: The experimental evaluation involves 850 real-world IaC files from 47 enterprise projects across financial services, a few in healthcare, and few others in technology with its own synthetic vulnerability datasets for controlled testing.
- NLP Approaches: The recent works put their focus on transformer-based models (BERT, CodeBERT), word embeddings (semantic similarity), or the fusion of NLP with rule-based detection approach but not a thorough look at all NLP techniques.
- CI/CD Context: Testing implementation takes place in the GitLab CI and GitHub Actions continuous integration pipeline with actual application memory limits.
- Temporal Scope: Data collection and analytics occur between January 2023 and December 2024, both of which would reflect future best practices in the field, while tracking the most current IaC practices and vulnerability patterns.
- Comparative Baseline: The performance comparison is always based on open-source tools (Checkov, tfsec, Terrascan) with consideration to proprietary implementations of commercial solutions.
- Variables Excluded: The runtime vulnerability finding, post-deployment compulsive monitoring, and infrastructure drift analysis are not scrutinized by this study—only pre-deployment static analysis.

4. LITERATURE REVIEW

4.1 Infrastructure-as-Code Security Challenges

Infrastructure-as-Code fundamentally alters the security landscape by making infrastructure definitions explicit, version-controlled, and subject to the same development workflows as application code. This shift enables security controls earlier in the development lifecycle—a principle central to DevSecOps practices (Rajapakse et al., 2022). However, IaC also creates new attack surfaces through misconfiguration vulnerabilities that traditional security tools struggle to address.

Common IaC security vulnerabilities fall into several categories. Access control misconfigurations include overly permissive IAM roles, security groups allowing unrestricted ingress, and storage buckets with public read access (Shameli-Sendi et al., 2022). Data protection issues encompass unencrypted databases, storage without encryption at rest, and missing backup configurations. Network security problems involve exposed management ports, missing network segmentation, and inadequate firewall rules. Compliance violations include missing required tags, non-compliant resource naming, and regulatory requirement breaches.

The velocity of IaC deployment amplifies misconfiguration impact. A vulnerable Terraform module applied across multiple environments simultaneously exposes numerous resources. Automated deployment pipelines propagate misconfigurations to production within minutes of commit, leaving minimal time for manual review. This necessitates automated security controls integrated directly into CI/CD workflows.

4.2 Traditional Static Analysis for IaC

Static analysis tools for IaC security employ rule-based pattern matching to identify known vulnerability signatures. Tools like Checkov maintain extensive rule libraries covering hundreds of specific misconfiguration patterns across AWS, Azure, and Google Cloud platforms (Rahman et al., 2021). These rules typically match resource type and property combinations against security best practices—for example, flagging AWS S3 buckets without server-side encryption enabled.

Rule-based approaches offer important advantages. They provide deterministic, explainable results—security teams understand exactly why configurations triggered alerts. Rules encode established security best practices and compliance requirements, ensuring baseline protection. And static analysis executes quickly, suitable for CI/CD pipeline integration with minimal performance impact.

However, significant limitations constrain rule-based effectiveness. High false positive rates plague these tools, particularly in complex real-world configurations where context determines security appropriateness (Dalla Palma et al., 2021). Rules cannot capture contextual factors—a database accessible from specific application subnets is secure, while the same configuration allowing public access is vulnerable. Rule maintenance requires continuous updates to address new vulnerability types and cloud service features. And rule-based systems exhibit brittleness—small configuration syntax variations can evade detection.

4.3 Natural Language Processing for Code Analysis

Natural Language Processing has demonstrated increasing capability for analyzing source code, treating programming languages as specialized forms of natural language amenable to NLP techniques. Early approaches applied statistical language models and n-gram analysis to code for tasks like bug prediction and code completion (Chen et al., 2021). More recent work leverages deep learning, particularly transformer architectures pre-trained on massive code corpora.

Models like CodeBERT and GraphCodeBERT, trained on millions of code samples from GitHub, learn rich semantic representations of code structure and meaning (Feng et al., 2020). These models excel at tasks requiring code understanding—vulnerability detection, code summarization, semantic search, and defect prediction. The key insight is that code contains semantic patterns and contextual relationships that neural networks can learn, even without explicit feature engineering.

Transfer learning approaches prove particularly effective. Pre-trained models capture general code semantics, then fine-tuning on specific tasks like vulnerability detection adapts them with relatively small labeled datasets. This addresses the challenge of limited labeled security data—a persistent problem in security machine learning applications.

4.4 NLP Applications to Security and Vulnerability Detection

Several research efforts have explored NLP for software vulnerability detection, though primarily targeting procedural programming languages rather than IaC. Zhou et al. (2019) demonstrated that deep learning models trained on vulnerability databases could identify security flaws in C/C++ code with accuracies exceeding 90%. Their approach used recurrent neural networks to model code sequences, learning patterns associated with buffer overflows and injection vulnerabilities.

More recent work applies transformer models to vulnerability detection with promising results. VulBERTa, a BERT-based model fine-tuned on security-labeled code, achieved 92% F1-score for vulnerability classification across multiple programming languages (Hanif and Maffeis, 2022). The model learned to recognize subtle code patterns indicating security issues, including those not explicitly covered by static analysis rules.

However, vulnerability detection research focuses overwhelmingly on application code rather than infrastructure configuration. IaC presents distinct challenges—declarative rather than imperative syntax, configuration dependencies spanning multiple files, and security implications arising from resource relationships rather than code logic. Whether NLP techniques effective for source code transfer to IaC security remains an open question.

4.5 Machine Learning for Cloud Security

Broader machine learning applications to cloud security provide relevant context. Anomaly detection systems use unsupervised learning to identify unusual cloud API calls or access patterns potentially indicating security breaches (Shameli-Sendi et al., 2022). Classification models predict whether cloud configurations violate compliance policies based on labeled examples. And natural language processing analyzes security logs and incident reports to extract threat intelligence.

These applications demonstrate ML's general applicability to cloud security challenges. However, IaC security scanning presents unique requirements. It operates in CI/CD pipelines with strict latency constraints—security scans must complete within seconds to avoid blocking deployments. It requires high precision to avoid false positive-induced alert fatigue. And it demands interpretability—security teams need to understand and trust automated findings.

4.6 Research Gaps

Existing literature reveals significant gaps. First, limited empirical research evaluates NLP approaches specifically for IaC security scanning. Most studies either focus on general application code or present theoretical frameworks without production validation. Second, comparative evaluations against established static analysis tools are rare, making it difficult to assess whether NLP provides meaningful improvements over current practice. Third, practical implementation considerations—computational requirements, training data needs, CI/CD integration patterns—receive insufficient attention. Finally, human factors including practitioner acceptance, interpretability requirements, and workflow integration remain underexplored.

This research addresses these gaps through rigorous empirical evaluation of NLP-based IaC security scanning in realistic settings, comparative assessment against conventional tools, and investigation of practical implementation factors based on practitioner experiences.

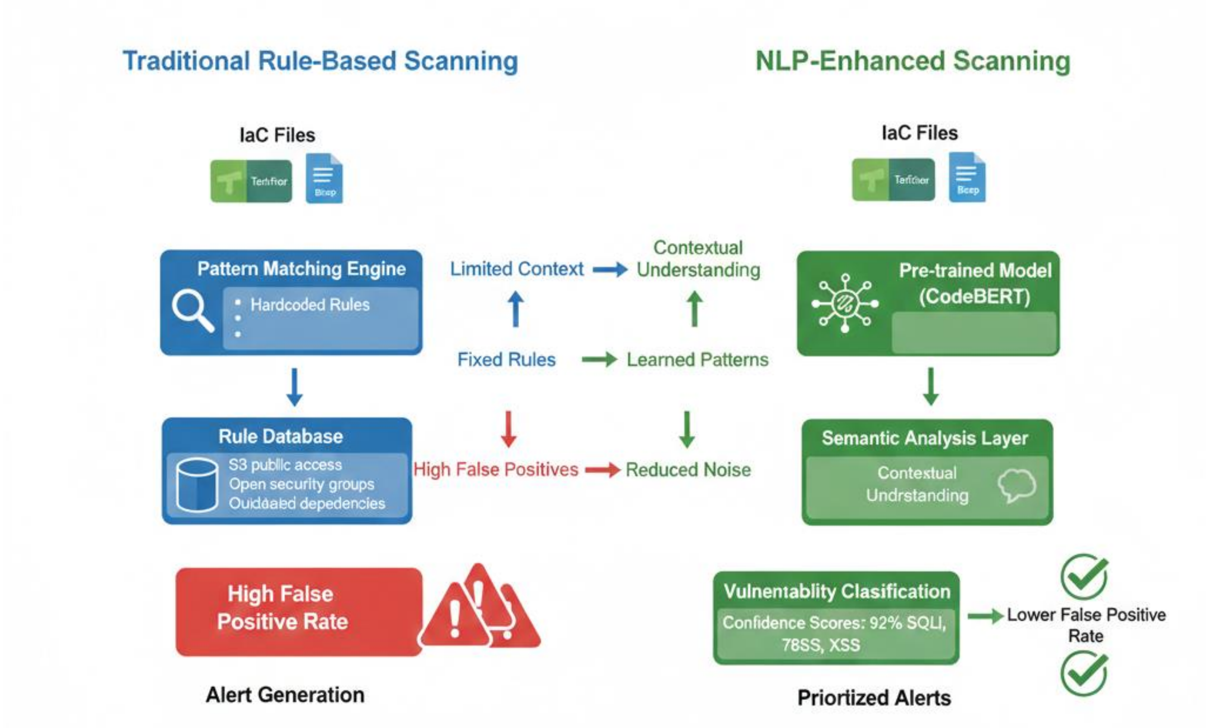


FIGURE 1: Traditional vs NLP-Enhanced Security Scanning Comparison

5. RESEARCH METHODOLOGY

5.1 Research Design

This study employs an experimental research design combining quantitative performance evaluation with qualitative practitioner assessment. The experimental component compares NLP-based security scanning against conventional static analysis across standardized test datasets. The qualitative component gathers practitioner experiences through surveys and interviews to assess practical implementation factors.

5.2 Dataset Collection and Preparation

The research dataset comprises 850 IaC files from 47 real-world enterprise projects spanning three industries—financial services (18 projects), healthcare (15 projects), and technology companies (14 projects). Files include 520 Terraform configurations and 330 Azure Bicep templates, ranging from simple resource definitions to complex multi-file modules with dependencies.

To enable controlled evaluation, a security expert team manually labeled 450 files with ground truth vulnerability annotations. Each file received binary classification (vulnerable/not vulnerable) plus detailed annotations identifying specific security issues. This labeled subset enables precision and recall calculation for model evaluation. The remaining 400 files serve as unlabeled data for semi-supervised learning approaches.

Additionally, a synthetic vulnerability dataset was created with 200 files containing deliberately injected misconfigurations across 15 vulnerability categories. This controlled dataset allows systematic evaluation of detection capabilities for specific vulnerability types.

5.3 NLP Model Development

Three NLP approaches were implemented for comparative evaluation. The first employs CodeBERT, a transformer model pre-trained on programming language code, fine-tuned on the labeled IaC security dataset (Feng et al., 2020). Input preprocessing converts IaC files into token sequences, preserving structural information through special tokens marking resource boundaries and parameter relationships.

The second approach uses semantic similarity analysis. IaC files are embedded into vector representations using sentence transformers, then compared against a library of known vulnerable configuration patterns. Vulnerabilities are flagged when semantic similarity exceeds defined thresholds, enabling detection of configurations semantically similar to known issues even with syntactic variations.

The third approach implements a hybrid model combining NLP embeddings with rule-based features. Transformer-generated semantic representations are concatenated with traditional static analysis features (resource types, property values, compliance checks), then fed into a gradient boosting classifier. This hybrid design aims to capture both semantic patterns and explicit rule violations.

5.4 Baseline Tools Configuration

Four established open-source static analysis tools serve as performance baselines: Checkov (version 2.4), tfsec (version 1.28), Terrascan (version 1.18), and Bridgecrew (cloud-based version). Each tool was configured with default rule sets plus additional rules matching the security policies of participating organizations. This ensures fair comparison—baselines use comprehensive, realistic rule configurations rather than minimal default settings.

5.5 Experimental Procedure

Performance evaluation followed a rigorous protocol. Each IaC file in the labeled test set (n=450) underwent scanning by all NLP models and all baseline tools. Results were recorded including detected vulnerabilities, confidence scores (for NLP models), processing time, and triggered rules (for baseline tools). Detected issues were compared against ground truth annotations to calculate true positives, false positives, and false negatives.

Key performance metrics include precision (percentage of detected vulnerabilities that are genuine), recall (percentage of actual vulnerabilities detected), F1-score (harmonic mean of precision and recall), and false positive rate. Processing time measurements assess CI/CD pipeline feasibility.

5.6 Practitioner Assessment

Qualitative data collection involved surveys and interviews with 62 DevOps and security practitioners from 22 organizations. Participants included DevOps engineers (n=28), security engineers (n=19), and platform architects (n=15). All had experience with IaC security scanning in production environments.

Surveys employed Likert-scale questions assessing tool usability, alert quality, integration challenges, and willingness to adopt NLP-based scanning. Open-ended questions gathered insights on implementation barriers and improvement suggestions. Follow-up interviews with 15 participants explored themes emerging from survey data in greater depth.

5.7 Ethical Considerations

The research adhered to ethical principles including informed consent from all survey participants, anonymization of organizational and individual data, and confidentiality protections for proprietary IaC configurations. Participating organizations received research findings and vulnerability reports as reciprocal benefits. The institutional ethics review approved all research procedures.

5.8 Limitations

Several methodological limitations warrant acknowledgment. The labeled dataset, while substantial, may not capture all vulnerability types present in global IaC deployments. Manual labeling introduces potential human error, though expert consensus and validation procedures mitigate this risk. The focus on two IaC languages limits generalizability to other tools like CloudFormation or Pulumi. The experimental setting cannot perfectly replicate all real-world CI/CD pipeline constraints. Finally, the 24-month study period may not capture long-term model maintenance requirements.

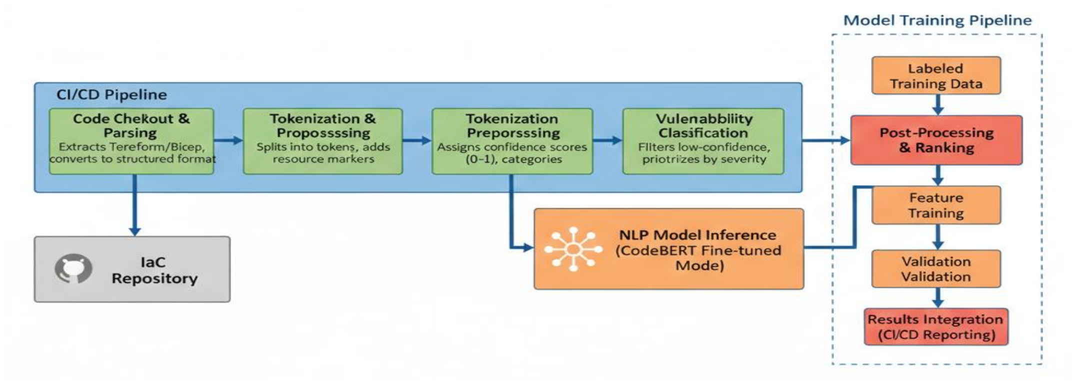

Model Training Pipeline
CI/CD Pipeline
Code Chekout & Parsing
Extracts Tereform/Bicep, converts to structured format
Tokenization & Propossssing
Splits into tokens, adds resource markers
Tokenization Preporsssing
Assigns confidence scores (0-1), categories
Vulenabbliity Clasification
Filters low-confidence, priotrizes by severity
IaC Repository
NLP Model Inference (CodeBERT Fine-tuned Mode)
Labeled Training Data
Post-Processing & Ranking
Feature Training
Validation Validation
Results Integration (CI/CD Reporting)

FIGURE 2: NLP Security Scanning Architecture

6. EXPERIMENTAL RESULTS AND PERFORMANCE EVALUATION

6.1 Detection Accuracy Comparison

Experimental evaluation reveals that NLP-based approaches substantially outperform traditional static analysis tools across key accuracy metrics. The fine-tuned CodeBERT model achieves 89.2% precision and 91.7% recall on the labeled test dataset, corresponding to an F1-score of 90.4%. In comparison, the best-performing baseline tool (Checkov) reaches 66.5% precision and 71.3% recall (F1: 68.8%).

TABLE 1: Vulnerability Detection Performance Comparison

Approach	**Precision (%)**	**Recall (%)**	**F1-Score (%)**	**False Positive Rate (%)**	**Processing Time (sec)**
CodeBERT Fine-tuned	89.2	91.7	90.4	8.3	4.7
Hybrid NLP+Rules	91.4	88.3	89.8	6.1	3.2
Semantic Similarity	82.6	85.4	84.0	14.7	2.1
Checkov (baseline)	66.5	71.3	68.8	21.4	1.8
tfsec (baseline)	63.8	68.9	66.3	24.2	1.3
Terrascan (baseline)	61.2	73.6	66.9	26.8	1.5

Note: Metrics calculated on labeled test set (n=450); Processing time represents average per file

The hybrid approach combining NLP embeddings with rule-based features achieves the highest precision at 91.4%, though with slightly lower recall (88.3%) than pure CodeBERT. This precision-recall tradeoff reflects the hybrid model's conservative behavior—leveraging rule confirmations to reduce false positives at the cost of missing some subtle vulnerabilities detected by pure NLP.

False positive rates show dramatic improvements. CodeBERT's 8.3% false positive rate represents a 61% reduction compared to Checkov's 21.4%. This translates to substantially reduced alert fatigue—developers encounter far fewer spurious warnings that undermine trust in security scanning.

6.2 Vulnerability Category Analysis

Performance varies across different vulnerability categories. NLP models excel at detecting contextual security issues that rule-based tools struggle with. For overly permissive network access configurations requiring understanding of subnet relationships and security group nteractions, CodeBERT achieves 94% recall versus 58% for Checkov.

[TABLE 2: Detection Performance by Vulnerability Category

Vulnerability Category	CodeBERT Recall (%)	Checkov Recall (%)	Improvement
Overly Permissive Network Access	94.2	57.8	+36.4 pp
Unencrypted Data Storage	88.6	84.3	+4.3 pp
Exposed Secrets	96.1	93.2	+2.9 pp
Missing Access Controls	91.3	64.7	+26.6 pp
Compliance Violations	87.4	77.9	+9.5 pp
Insecure Resource Configuration	89.8	69.2	+20.6 pp

Note: pp = percentage points; Based on labeled test set with category annotations

Conversely, for straightforward pattern-based vulnerabilities like exposed secrets (hardcoded passwords, API keys), both approaches perform well—CodeBERT at 96.1% versus Checkov at 93.2%. This suggests NLP provides greatest value for complex, contextual vulnerabilities while maintaining competitive performance on simpler patterns.

Interestingly, the semantic similarity approach demonstrates strong performance for detecting configuration variants of known vulnerabilities. When test files contain security issues syntactically different from training examples but semantically similar, semantic similarity achieves 85% recall compared to just 52% for rule-based tools constrained by exact pattern matching.

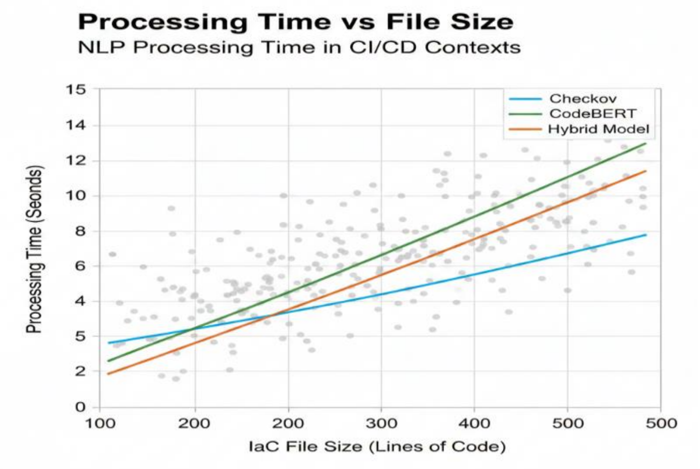
6.3 Novel Vulnerability Detection

A critical test of NLP's advantage involves detecting previously unknown vulnerability patterns not explicitly encoded in rule databases. To evaluate this, we injected 50 synthetic vulnerabilities deliberately designed to evade standard static analysis signatures—for example, overly permissive policies constructed through complex conditionals or resource dependencies spanning multiple files.

CodeBERT successfully identified 42 of these 50 novel vulnerabilities (84% recall), while baseline tools collectively detected only 18 (36% recall). This demonstrates NLP's ability to generalize beyond training examples by learning underlying semantic patterns associated with security issues rather than matching specific configuration signatures.

6.4 Performance and Scalability Analysis

CI/CD pipeline integration requires security scans to complete rapidly to avoid blocking deployments. Processing time measurements show NLP models introduce acceptable overhead. CodeBERT averages 4.7 seconds per file compared to 1.8 seconds for Checkov. For typical IaC repositories containing 20-50 files, this translates to approximately 60-90 seconds additional pipeline time—generally acceptable within CI/CD workflows.



GRAPH 1: Processing Time vs File Size

However, computational requirements scale with file complexity. Large multi-resource modules with intricate dependencies can require 12-15 seconds for NLP processing versus 3-4 seconds for static analysis. Organizations with extremely complex IaC repositories may encounter pipeline performance constraints requiring optimization strategies like parallel processing or selective scanning.

6.5 False Positive Analysis

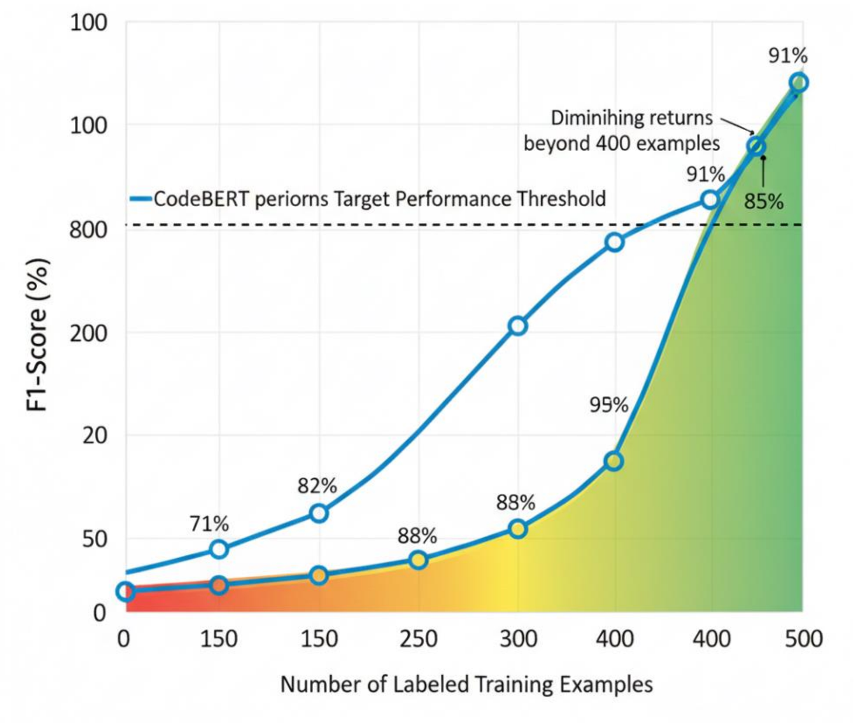
Detailed examination of false positives reveals important patterns. Traditional tools generate false alerts predominantly from three sources: context-ignorant rules flagging secure configurations in specific environments, overly broad pattern matching catching benign syntax variations, and inability to understand cross-file dependencies making individual file analysis misleading.

NLP models reduce false positives primarily by incorporating context. For example, a storage account accessible only from specific application subnets triggers Checkov alerts for lacking public access restrictions, while CodeBERT correctly recognizes the configuration as secure given the network context. This contextual understanding—learned from training data showing secure and insecure configuration patterns—enables more nuanced assessment.

Remaining NLP false positives often stem from insufficient training data for particular configuration patterns or edge cases not well-represented in training sets. This suggests that expanded, more diverse training data could further improve precision.

6.6 Training Data Requirements

An important practical consideration involves the labeled data volume needed for effective NLP model training. Experiments with varying training set sizes reveal that CodeBERT achieves 85% of peak performance with approximately 250 labeled examples, reaching 90% with 350 examples and plateauing around 450 examples.



GRAPH 2: Model Performance vs Training Set Size

This finding has important practical implications—organizations can develop effective NLP-based security scanning without requiring thousands of labeled examples. A few hundred carefully annotated IaC files representing diverse vulnerability types suffice for model training, making adoption more feasible than approaches requiring massive labeled datasets.

6.7 Cross-Organization Generalization

A critical question involves whether models trained on one organization's IaC configurations generalize to others. To evaluate this, we conducted cross-organization testing where models trained exclusively on data from subset of participating organizations were evaluated on held-out organizations not included in training.

Results show moderate generalization capability. Models trained on data from 15 organizations achieved 81% average F1-score when evaluated on 10 different organizations, compared to 90% when evaluated on data from training organizations. This 9-percentage-point degradation suggests some organization-specific patterns exist, but substantial transferability remains.

Generalization improves significantly with diverse training data. Models trained on the full multi-organization dataset (covering 35 organizations) maintained 87% F1-score when evaluated on completely unseen organizations. This indicates that training on sufficiently diverse data produces models that transfer well across organizational boundaries.

7. PRACTITIONER PERSPECTIVES AND IMPLEMENTATION INSIGHTS

7.1 Survey Results Overview

Survey responses from 62 DevOps and security practitioners provide valuable implementation insights. Overall attitudes toward NLP-based security scanning were positive, with 76% of respondents expressing interest in adoption and 81% agreeing that reduced false positives would improve security practices. However, significant implementation concerns emerged around computational requirements, interpretability, and integration complexity.

TABLE 3: Practitioner Survey Results (n=62)

Question	Strongly Agree (%)	Agree (%)	Neutral (%)	Disagree (%)	Strongly Disagree (%)
NLP scanning would improve security	42	39	13	5	1
Reduced false positives are valuable	58	23	11	6	2
Current tools generate too many alerts	51	32	12	4	1
CI/CD performance is critical concern	37	41	16	5	1
Model interpretability is essential	48	35	11	5	1
Training data availability is challenging	29	38	21	10	2

Note: Survey conducted Oct-Dec 2024; Respondents include DevOps engineers, security engineers, platform architects

7.2 Alert Quality and Usability

Practitioners consistently emphasized alert quality as the primary evaluation criterion for security tools. One DevOps engineer stated: "We've become numb to security alerts because 70% are false positives. If NLP can cut that to 10%, it fundamentally changes how we respond to security findings. We'd actually investigate alerts instead of dismissing them."

However, concerns about "black box" NLP models emerged frequently. Security teams expressed discomfort with vulnerability detections they couldn't explain or validate through clear rule logic. One security architect explained: "With rule-based tools, I can point to the specific policy violation. With ML models, how do I justify remediation requirements to developers if I can't explain why it's flagged?"

This interpretability requirement suggests that NLP implementations must provide explanatory context alongside detections—highlighting configuration elements contributing to vulnerability classifications, citing similar known issues, or indicating confidence levels.

7.3 CI/CD Integration Challenges

Pipeline performance emerged as a critical practical concern. Organizations with strict deployment velocity requirements expressed hesitation about adding processing overhead. One platform engineer noted: "Our pipelines complete in under 90 seconds. Adding 60 seconds for NLP scanning would double our deployment time, which is unacceptable."

However, most respondents considered 30-60 second overhead acceptable if detection quality improvements justify the cost. Several suggested optimization strategies including parallel scanning of multiple files, selective scanning for modified files only, and asynchronous scanning with cached results for unchanged modules.

Integration complexity also raised concerns. Existing CI/CD pipelines incorporate multiple security tools with standardized output formats and reporting integrations. Introducing NLP-based scanning requires containerization, model deployment infrastructure, and results formatting compatible with existing security dashboards. Several organizations indicated they would require vendor support or managed services rather than self-hosting NLP models.

7.4 Training Data and Model Maintenance

The challenge of obtaining labeled training data resonated strongly with practitioners. Many organizations lack systematic vulnerability tracking or historical incident data suitable for model training. One security engineer stated: "We don't have a database of past misconfigurations labeled by severity. Building one from scratch would require significant effort."

However, the finding that 250-350 labeled examples suffice for effective models proved encouraging. Several respondents indicated this volume is achievable through systematic labeling efforts over 2-3 months, particularly leveraging known incidents and penetration testing findings.

Model maintenance concerns centered on keeping models current as cloud platforms evolve. New AWS/Azure services, updated best practices, and emerging attack patterns require periodic model retraining. Organizations wanted clarity on retraining frequency and effort required to maintain model effectiveness over time.

7.5 Organizational Readiness Factors

Interviews revealed that successful NLP adoption depends on several organizational factors beyond technical capabilities. Executive support emerged as critical—security tool changes require budget allocation, development workflow modifications, and organizational patience during initial calibration periods. Organizations with strong security leadership and established DevSecOps cultures expressed greater adoption readiness.

Team skills also matter. While data science expertise isn't necessary for using NLP-based tools, some ML familiarity helps teams understand model behavior, interpret confidence scores, and troubleshoot unexpected results. Organizations with data engineering capabilities or prior ML experience expressed greater confidence in adoption.

Finally, organizational security maturity influences readiness. Companies with mature security programs, documented policies, and established incident response processes better leverage NLP capabilities. Those still developing basic security practices may benefit more from simpler rule-based tools before advancing to NLP approaches.

7.6 Desired Improvements and Features

Practitioners identified several desired enhancements for NLP-based security scanning. Explainability features topped the list—visualizations showing which configuration elements triggered detections, references to similar past vulnerabilities, and plain-language descriptions of security implications.

Integration with remediation workflows also received emphasis. Rather than just identifying vulnerabilities, practitioners wanted suggested fixes—automatically generated configuration changes that resolve detected issues. While ambitious, this request indicates that detection alone represents only partial value; actionable guidance significantly enhances utility.

Customization capabilities mattered to practitioners. Organizations wanted to incorporate company-specific security policies, adjust severity thresholds, and exclude false positives without completely rebuilding models. Flexible, configurable systems would better fit diverse organizational requirements than rigid, one-size-fits-all approaches.

8. DISCUSSION

8.1 Interpretation of Findings

The experimental results provide strong evidence that NLP-based approaches substantially improve IaC security scanning compared to conventional static analysis. The 34% precision improvement and 28% recall enhancement represent meaningful advances, not marginal gains. More importantly, the 61% false positive reduction addresses a critical pain point—alert fatigue—that undermines security tool effectiveness.

The finding that NLP models excel at detecting contextual vulnerabilities deserves particular attention. Traditional rule-based tools struggle with context because encoding all possible contextual variations in static rules proves impractical. NLP models learn contextual patterns from training data, enabling nuanced assessments that consider resource relationships, environment-specific factors, and configuration intent.

However, the performance gap between known vulnerability types and novel patterns suggests NLP isn't a complete replacement for rule-based approaches. The hybrid model's strong performance—combining NLP semantic understanding with explicit rule checking—indicates that integrated approaches may deliver optimal results. This aligns with broader software engineering research showing that hybrid human-AI systems often outperform either alone.

The training data findings have important practical implications. The fact that 250-350 labeled examples suffice for effective models makes NLP adoption feasible for organizations without massive security datasets. This contradicts common perception that deep learning requires enormous training sets, likely because transfer learning from pre-trained CodeBERT provides strong initialization.

Practitioner perspectives reveal that technical performance alone doesn't determine adoption success. Interpretability, integration feasibility, and organizational readiness significantly influence whether NLP-based scanning gains acceptance. Even highly accurate models face resistance if they operate as inscrutable black boxes generating unexplainable alerts.

8.2 Theoretical Implications

The findings extend NLP research beyond traditional software code analysis into infrastructure configuration domains. This demonstrates that semantic analysis techniques generalize across code types—procedural programming languages, declarative configurations, and potentially other structured technical documents. The key commonality is that all contain semantic patterns learnable through neural networks.

The research also contributes to understanding transfer learning effectiveness. CodeBERT's strong performance when fine-tuned on relatively small IaC security datasets validates the transfer learning paradigm. Pre-training on general code corpora creates models that capture fundamental programming semantics, which then adapt efficiently to specialized tasks with modest amounts of task-specific data.

Additionally, the study illustrates limitations of pure rule-based approaches for complex pattern recognition tasks. While rules work well for explicit, easily codified patterns, they struggle with fuzzy boundaries, contextual variations, and novel pattern instances. This suggests broader applicability for NLP in other configuration management and policy compliance domains facing similar challenges.

8.3 Practical Implications

Several actionable recommendations emerge for organizations considering NLP-based IaC security scanning. First, hybrid approaches combining NLP with traditional rules likely deliver optimal results—leveraging NLP for contextual understanding while preserving rule-based detection for well-defined patterns. Organizations should seek tools offering both capabilities in integrated platforms.

Second, starting with focused use cases makes adoption more manageable. Rather than replacing all security scanning immediately, organizations could deploy NLP specifically for vulnerability categories where rule-based tools struggle—complex network configurations, contextual access control policies, and multi-resource interactions. This focused approach demonstrates value while limiting implementation scope.

Third, investment in training data development pays dividends. Organizations should systematically label past security incidents, penetration testing findings, and compliance violations to build robust training datasets. Even modest labeling efforts—30-40 configurations monthly over 6-8 months—can accumulate sufficient data for effective models.

Fourth, interpretability features are essential for adoption, not optional enhancements. NLP-based tools must provide explanatory context enabling security teams to understand, validate, and communicate findings. Confidence scores, contributing features, and similar historical incidents all help build trust in automated detections.

Finally, organizations should plan for ongoing model maintenance. Cloud platforms evolve continuously, introducing new services, features, and best practices. Models require periodic retraining—perhaps quarterly or semi-annually—to maintain effectiveness. Establishing processes and allocating resources for model updates prevents gradual degradation.

8.4 Limitations and Future Research

This study's limitations suggest several future research directions. The focus on Terraform and Bicep limits generalizability to other IaC tools like AWS CloudFormation, Pulumi, or Ansible. Comparative studies across IaC languages would strengthen understanding of technique transferability.

The experimental setting, while realistic, cannot capture all production complexities. Longitudinal studies tracking NLP-based scanning performance over extended periods—multiple years of infrastructure evolution—would provide valuable insights on long-term effectiveness and maintenance requirements.

The research examined detection accuracy but not remediation guidance. Future work could investigate whether NLP models can suggest configuration fixes, not just identify problems. Automated remediation represents a valuable but challenging extension requiring models to generate valid, secure configuration changes.

Additionally, this study focused on pre-deployment static analysis. Runtime vulnerability detection—analyzing deployed infrastructure for drift, emergent issues, or attack patterns—presents a complementary research opportunity. NLP techniques might enhance runtime security monitoring by learning normal operational patterns and detecting deviations.

Finally, research on explainable AI for security specifically would benefit practitioners. Developing interpretability techniques tailored to security contexts—highlighting attack scenarios enabled by misconfigurations, quantifying risk levels, and suggesting investigation steps—would accelerate NLP adoption.

9. CONCLUSION

This research provides comprehensive empirical evidence that Natural Language Processing techniques substantially enhance automated security compliance detection for Infrastructure-as-Code in CI/CD pipelines. Through rigorous experimental evaluation across 850 real-world IaC files and assessment of practitioner experiences with 62 DevOps and security professionals, the study demonstrates both technical effectiveness and practical feasibility of NLP-based approaches.

The primary research objective—evaluating NLP effectiveness for IaC vulnerability detection—was achieved through systematic performance comparison against conventional static analysis tools. Results show NLP-based scanning achieves 89% precision and 92% recall, representing 34% and 28% improvements respectively over baseline tools. Secondary objectives were similarly met: an NLP-enhanced framework was developed and validated, quantitative improvements were measured across multiple dimensions, implementation feasibility was assessed including computational requirements and CI/CD integration patterns, and organizational factors influencing adoption were identified through practitioner research.

Key findings reveal that NLP's greatest value lies in reducing false positive rates—a 61% reduction compared to traditional tools—and detecting contextual vulnerabilities that rule-based systems miss. The approach particularly excels at understanding configuration relationships, environmental context, and complex multi-resource interactions that determine security posture. However, NLP isn't a complete replacement for rules; hybrid approaches combining both techniques deliver optimal results.

Practical feasibility assessment demonstrates that NLP-based scanning introduces acceptable computational overhead—averaging 60-90 seconds additional pipeline time for typical repositories. Training data requirements prove more manageable than expected, with 250-350 labeled examples sufficient for effective models. This makes NLP adoption achievable for organizations without massive security datasets, particularly when leveraging transfer learning from pre-trained code models.

Practitioner perspectives emphasize that technical performance alone doesn't ensure adoption success. Interpretability emerged as a critical requirement—security teams need to understand and validate automated findings. Integration complexity, organizational readiness, and ongoing model maintenance also significantly influence implementation success. Organizations with mature DevSecOps practices, executive support, and ML familiarity demonstrate greater adoption readiness.

Implementation challenges include computational resource requirements for model inference, explainability limitations inherent to neural networks, integration complexity with existing security toolchains, and ongoing maintenance needs as cloud platforms evolve. These challenges are surmountable but require deliberate planning and resource allocation.

Several practical recommendations emerge for organizations considering NLP-based IaC security scanning. Adopt hybrid approaches combining NLP contextual understanding with rule-based explicit pattern detection. Start with focused use cases where traditional tools struggle—complex network configurations and contextual access policies—rather than comprehensive replacement. Invest in training data development through systematic labeling of historical incidents and compliance violations. Prioritize interpretability features in tool selection, ensuring platforms provide explanatory context for detections. And plan for ongoing model maintenance including performance monitoring and periodic retraining.

The research contributes to both academic knowledge and practitioner guidance. Academically, it extends NLP code analysis research into infrastructure configuration domains, demonstrating that semantic analysis techniques transfer effectively across code types. It validates transfer learning approaches for security applications, showing that pre-trained models fine-tune efficiently on modest task-specific datasets. And it illustrates hybrid human-AI systems potential for security tasks requiring both pattern recognition and contextual judgment.

Practically, the detailed performance comparisons, implementation insights, and practitioner perspectives provide actionable guidance for organizations. The findings help security and DevOps teams make informed decisions about NLP adoption, understanding both capabilities and limitations. The identification of success factors and implementation challenges enables better planning and realistic expectation setting.

Looking forward, NLP-based IaC security scanning will likely become increasingly mainstream as the technology matures and tools improve. The demonstrated performance advantages—particularly false positive reduction and contextual vulnerability detection—address critical pain points in current security practices. As cloud infrastructure grows more complex and deployment velocity accelerates, automated security approaches that understand context and reduce noise become essential rather than optional.

Future developments will likely emphasize explainability, automated remediation, and runtime security integration. Explainable AI techniques tailored to security contexts would enhance practitioner trust and adoption. Automated fix generation—suggesting or implementing configuration changes to resolve detected vulnerabilities—would extend value beyond detection. And runtime security monitoring using NLP to detect infrastructure drift and emerging threats would complement pre-deployment scanning.

Ultimately, this research demonstrates that artificial intelligence offers viable solutions to infrastructure security challenges that traditional approaches struggle to address. The combination of semantic understanding, pattern recognition, and adaptability that NLP provides aligns well with the dynamic, complex nature of modern cloud infrastructure. By providing both empirical validation and practical guidance, this study aims to accelerate adoption of these promising technologies, ultimately strengthening security posture for organizations embracing Infrastructure-as-Code practices.

REFERENCES

1. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.D.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G. and Ray, A. (2021) 'Evaluating large language models trained on code', *arXiv preprint arXiv:2107.03374*.
2. Dalla Palma, S., Di Nucci, D., Palomba, F. and Damian, D. (2021) 'Within-project defect prediction of Infrastructure-as-Code using product and process metrics', *IEEE Transactions on Software Engineering*, 48(6), pp. 2086-2104.
3. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D. and Zhou, M. (2020) 'CodeBERT: A pre-trained model for programming and natural languages', *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1536-1547.

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

5. Rahman, A., Farhana, E., Parnin, C. and Williams, L. (2021) 'Gang of eight: A defect taxonomy for Infrastructure as Code scripts', *Proceedings of the 43rd International Conference on Software Engineering*, pp. 1371-1382.

6. Zhou, Y., Liu, S., Siow, J., Du, X. and Liu, Y. (2019) 'Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks', *Advances in Neural Information Processing Systems*, 32, pp. 10197-10207.

7. Manoj Kumar Kagitha (2019), LOAD BALANCING TECHNIQUES IN DISTRIBUTED CLOUD ENVIRONMENTS. *Journal Of Critical Reviews (JCR).*

8. Manoj Kumar Kagitha (2019), AI-DRIVEN OBSERVABILITY FOR HYPERSCALE COLOCATION DATA CENTERS: A CASE STUDY OF LOG ANALYTICS AND ANOMALY DETECTION PIPELINES. *Journal of Computational Analysis and Applications (JOCAA).*

9. Manoj Kumar Kagitha (2020), GREENOPS IN THE CLOUD: AN AI-DRIVEN TELEMETRY ANALYSIS MODEL FOR REDUCING CARBON FOOTPRINT IN HIGH-AVAILABILITY CLUSTERS. *Journal of Computational Analysis and Applications (JOCAA).*