International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-15 Issue-4, April 2026

Advancing Infrastructure-as-Code Resilience through Generative AI Agents for Predictive Remediation and Autonomous Security Enforcement

Harish Apuri, Mukesh Aurangabadkar, Shikher Goel, Madhan Mohan Reddy Chinthala, Charani Yepuri

Abstract: ***"Infrastructure as Code" (IaC) is the accepted approach to provision cloud infrastructure declaratively. Still, misconfigurations in IaC remain the primary cause of cloud security incidents, accounting for 67% of all disclosed cloud breaches. However, the current set of countermeasures, such as rule-based static scanners, policy-as-code tools, and manual review gates, is inadequate to prevent such misconfigurations or to address them through autonomous remediation. In this paper, the authors propose a multi-agent generative AI system called GenSecOps, comprising four agents that work together to prevent misconfigurations in IaC. These agents are the IaC Understanding Agent (IUA), which uses the IaC artefact to create a semantic resource graph; the Risk Prediction Agent (RPA), which uses a hybrid model of the Transformer and Graph Neural Networks to create risk mappings; the Generative Remediation Agent (GRA), which uses the risk mappings to create corrected policy-compliant IaC templates; and the Autonomous Enforcement Orchestrator (AEO). Experiments on a corpus of 48,000 IaC templates (Terraform, CloudFormation, Kubernetes) show that GenSecOps achieves a misconfiguration detection F1-score of 0.934, a 73.2% reduction in critical findings overrule-based baselines, an 81.5% improvement in mean-time-to-remediate (MTTR), and drift-recovery latency below 4.2 minutes. These results demonstrate that generative AI agents provide a viable, deployable foundation for self-healing, autonomously secured cloud-native infrastructure.***

Keywords: ***Infrastructure-as-Code, Generative AI Agents, DevSecOps, Cloud Security, Predictive Remediation Policy Synthesis, Autonomous Enforcement, Transformer-GNN, IaC Resilience***

Nomenclature:
RPA: Risk Prediction Agent
IUA: IaC Understanding Agent
GRA: Generative Remediation Agent
AEO: Autonomous Enforcement Orchestrator
MTTR: Mean-Time-To-Remediate
OPA: Open Policy Agent
LLMs: Large Language Models
CR: Correctness Rate
SIS: Security Improvement Score
FPR: Functional Preservation Rate
DRT: Drift Recovery Time

Manuscript received on 21 March 2026 | First Revised Manuscript received on 24 March 2026 | Second Revised Manuscript received on 06 April 2026 | Manuscript Accepted on 15 April 2026 | Manuscript published on 30 April 2026.
*Correspondence Author(s)
Harish Apuri*, Department of IT, IT Induct Inc, Charlotte (NC), United States of America (USA). Email ID: hapuri1029@gmail.com, ORCID ID: 0009-0007-1534-9984
Mukesh Aurangabadkar, Department of IT, Spectrum, Denver (Colorado), Vanuatu. Email ID: Mukesh_a24@hotmail.com
Shikher Goel, Department of IT, JPMorgan Chase, Jersey (New Jersey), United States of America (USA). Email ID: Shikher20goel@gmail.com
Madhan Mohan Reddy Chinthala, Department of IT, Franklin Info Tech, Rochester (NY), United States of America (USA). Email ID: madhanreddychinthala@gmail.com
Charani Yepuri, Independent Researcher, Department of IT, Hyderabad (Telangana), India. Email ID: Charaniy04@gmail.com

I. INTRODUCTION

The accelerated adoption of these cloud-native architectures has propelled Infrastructure-as-Code to a position of primacy as a critical engineering artefact. Hashi Corp.'s Terraform, AWS CloudFormation, and Kubernetes config manifests allow engineers to define and reproduce an entire cloud infrastructure with merely a few text files. While this declarative model offers undeniable benefits in terms of repeatability, auditability, and speed to delivery, it simultaneously also raises security risks to a critical juncture: a misconfigured attribute value, a missing encryption flag, or an overly permissive identity and access management policy embedded within a config template can leave an organization's entire production environment susceptible to data exfiltration or ransomware.

The scale of the problem is substantial. Using the controlled evaluation corpus used for this research, consisting of 48,000 IaC templates from open-source repositories, a synthetic benchmark dataset, and an anonymised enterprise dataset (see Section 5.1 for details on the corpus), ensemble static analysis identified 19,800 unique critical findings over a representative 12-month provisioning window. Industry data from the Cloud Security Benchmark Report estimates 67% of publicly known cloud-based security incidents have a causal link to IaC configuration issues, with network exposure, overprivileged IAM roles, and no encryption at rest being the three most common root causes.

In this environment, the security tools currently most widely used in IaC pipelines are fundamentally reactive. Rule-based scanners such as Checkov, tfsec, and KICS match templates against hand-crafted rule sets, which cannot possibly generalise to unexpected configurations not contemplated by the rule set author. Compliance-as-code solutions such as Open Policy Agent (OPA) and HashiCorp's Sentinel offer powerful policy languages but require human authoring for each new control and provide no predictive capability. Human review is latency-prohibitive for

continuous deployment pipelines, and no solution currently offers autonomous remediation or recovery from configuration drift.

Recent advances in large language models (LLMs) [1,2] and multi-agent AI systems [3] offer a promising pathway to overcome these challenges. The generative model can learn sophisticated IaC semantics from historical artefacts, learn latent risk patterns, generate corrected templates, and interact with version control and CI/CD tools. However, there is a lack of research that combines all these components into a unified, end-to-end security enforcement framework.

A. Contributions

The contributions of this work can be listed as follows:

i. A formal model for IaC resiliency, which includes misconfigurations, drift recovery, and adherence to policies, formulated as an optimization objective (Section 3).

ii. GenSecOps, a novel four-agent generative AI model for predictive detection, automated remediation, and IaC security policy enforcement (Section 4).

iii. A hybrid model for risk scoring, which combines a Transformer model and a Graph Neural Network, along with a mathematically defined loss function and embedding process (Section 4.3).

iv. An empirical evaluation of 48,000 IaC templates, showing state-of-the-art results according to five metrics, with ablation studies showing the contribution of each agent (Section 6).

v. An analysis of the operational, ethical, and governance implications of deployment (Section 8).

The rest of the paper is organised as follows: Section 2 discusses related work, while Section 3 formalises the problem. Section 4 introduces the GenSecOps architecture, while Section 5 discusses the experimental methodology. Section 6 shows the results, while Section 8 discusses the ethics of the GenSecOps model, and Section 9 concludes the paper.

II. LITERATURE REVIEW

A. Infrastructure-as-Code and Cloud Security

There are three broad categories of IaC tools: imperative SDKs (AWS CDK, Pulumi), declarative provisioners (Terraform, CloudFormation), and container orchestrators (Kubernetes, Helm). GitOps approaches extend these concepts, in which Git repositories serve as a single source of truth for both application code and infrastructure state.

Security research into IaC has so far mainly focused on misconfiguration taxonomies. Rahman et al. [4] conducted a study of more than 1,400 IaC scripts from public sources, which revealed seven common IaC smells, including "admin by default" configurations, hard-coded secrets, and a lack of integrity checks. Kumara et al. [5] developed a framework for detecting smells in Ansible playbooks, revealing that 63% of all examined sources contain at least one "high" smell. The issue of privilege escalation through overly permissive IAM configurations was investigated by Ponsard et al. [6], who introduced a method for the automated analysis of privilege escalation scenarios in cloud-based IAM configurations, showing that misconfigured cross-account roles can lead to exploitable escalation. Network exposure through misconfigured security groups and VPC peering is documented in the MITRE ATT&CK for Cloud, which lists 47 cloud-specific attack techniques that can be directly attributed to misconfigurations.

B. Existing IaC Analysis and Remediation Tools

Static analysis of IaC (IaC-SAST) tools has also significantly improved. For instance, the Checkov tool provides over 1,000 checks for Terraform, CloudFormation, Kubernetes, and ARM templates. KICS, on the other hand, comes with a query-based engine that supports six IaC formats. tfsec focuses on Terraform with detailed provider-specific checks. Drift detection tools such as Drift CTL and AWS Config help identify discrepancies between the desired provisioned state (as defined by IaC code) and the actual provisioned state. Policy-as-code tools like OPA with Rego and Sentinel enable security teams to write organizational policies.

Despite this ecosystem, three fundamental limitations exist. Firstly, all of these tools are reactive, i.e., they can only detect known misconfiguration categories and cannot detect unknown ones. Secondly, there is no remediation support available with these tools. All these tools can only generate reports on the findings. Thirdly, the tools used for drift detection can detect the drifts but cannot remediate them on their own and need human-initiated remediation workflows. A systematic literature review of 94 IaC security papers by Li et al. [7] confirmed that autonomous remediation remains an open problem.

C. Generative AI and Autonomous Agents in DevSecOps

The application of large language models to code generation tasks [8,9] has also catalysed similar work in the context of security. In a study by Pearce et al. [10], the authors assessed GitHub Copilot across 89 code generation scenarios in the context of cybersecurity. In the study, the authors reported that approximately 40% of the generated code suggestions contained CWE-classified security vulnerabilities. In the context of code security benchmarking, Siddiq et al. [11] proposed the SecurityEval dataset, which contains 130 code samples across 75 different CWE vulnerability types. In the study, the authors reported that GitHub Copilot and open-source code generation models produced code containing CWE-mapped security weaknesses.

Multi-agent architectures for LLMs have been shown to exhibit emergent planning capabilities through agent-agent communication and tool use. In the security space, the work on PentestGPT [12] has shown that an agent using a structured reasoning tree can perform penetration testing tasks effectively, outperforming a standalone model. Li et al. [13] have used a combination of LLMs and static analysis to reduce false positives and improve the effectiveness of practical bug detection in SAST results for the application layer. GNNs have been used for software vulnerability detection [16,17], leveraging the resource relationship graph as a natural representation of the semantics of IaC that is

difficult to model with a textual model.

D. Research Gaps

By synthesising all of this research, it is possible to identify three critical gaps, which are addressed by the present research:

i. *Predictive, Behaviour-Aware IaC Analysis:* None of the tools is capable of learning from past incident history to predict misconfiguration likelihoods in unseen templates. All tools use explicitly written rules.

ii. *Automated Risk-Aware Remediation:* While LLM-based code fixing has been demonstrated at the application level, no end-to-end generative remediation has been demonstrated or validated for IaC, where function preservation (infrastructure semantics) is as important as security improvement.

iii. *Closed-Loop Autonomous Enforcement:* None of the tools integrates detection, remediation, and enforcement in a closed-loop fashion, allowing for autonomous recovery from drift in running CI/CD pipelines.

GenSecOps fills all of these critical gaps simultaneously.

III. SYSTEM MODEL AND PROBLEM FORMULATION

A. IaC Environment and Threat Model

We model the IaC environment of an organisation as a tuple $E = (R, P, C, T)$, where:

– $R = \{r_1, r_2, \ldots, r_n\}$ is the set of IaC repositories holding the template files.

– $P = \{p_1, p_2, \ldots, p_k\}$ is the set of CI/CD pipeline stages that the changes must flow through before they are deployed.

– $C = \{c_1, c_2, \ldots, c_m\}$ is the set of cloud runtime environments, such as accounts, clusters, and regions, subject to the IaC templates.

– T is the stream of runtime events, such as CloudTrail logs, Kubernetes audit logs, and network flow records, generated by the runtime environments C.

Each template file $f \in R$ contains a set of resource declarations $D(f) = \{d_1, \ldots, d_n\}$. A declared resource d_i is characterised by its type $\tau(d_i) \in Y$ (where Y is the universe of supported resource types), its configuration attribute vector $a(d_i) \in \mathbb{R}^p$, and its dependency edges $E(d_i) \subseteq D(f) \times D(f)$ encoding cross-resource references.

Adversary Model. We consider a threat actor capable of: (i) exploiting internet-exposed services created by misconfigured network rules (MITRE T1190); (ii) escalating privileges through over-permissive IAM roles (MITRE T1098); (iii) achieving lateral movement via misconfigured cross-account trust relationships (MITRE T1550.001); and (iv) exfiltrating data via unencrypted storage resources (MITRE T1537). We do not model supply-chain attacks on the IaC toolchain itself, which is considered out of scope.

B. Formalisation of IaC Resilience

Definition 1 (IaC Misconfiguration). A resource declaration $d_i \in D(f)$ is misconfigured with respect to a policy set Π if there exists a policy $\pi \in \Pi$ such that $\pi(a(d_i), E(d_i)) = \perp$. We denote the binary misconfiguration indicator as $\mu(d_i) \in \{0, 1\}$.

Definition 2 (IaC Security Debt). The security debt of the template file f is:

$$\Phi(f) = \Sigma\, \mu(d_i) \cdot s(d_i) \quad (1)$$

where $s(d_i) \in \{1, 2, 3, 4\}$ is the severity score of the misconfiguration (INFO=1, LOW=2, MEDIUM=3, HIGH/CRITICAL=4).

Definition 3 (Configuration Drift). Let $\hat{a}(d_i, t)$ be the actual runtime attribute vector of the resource provisioned from d_i at time t, and $a(d_i)$ be the declared attribute vector in the template. The drift magnitude is:

$$\delta(d_i, t) = \|a(d_i) - \hat{a}(d_i, t)\|_2 \quad (2)$$

A resource is drifted at time t if $\delta(d_i, t) > \varepsilon$ for a deployment-specific threshold $\varepsilon > 0$.

Definition 4 (IaC Resilience Score). The resilience score of an IaC environment E at time t is:

$$\Psi(E, t) = \exp(-\lambda_1 \Phi(t)) \cdot \exp(-\lambda_2 \delta(t)) \cdot \eta(t) \quad (3)$$

where $\Phi(t) = (1/|R|)\, \Sigma\, \Phi(f, t)$ is the mean security debt, $\delta(t) = (1/|C|)\, \Sigma\, \delta(d, t)$ is the mean drift, $\eta(t) \in [0, 1]$ is the policy-adherence ratio at time t, and $\lambda_1, \lambda_2 > 0$ are weighting hyperparameters.

The primary optimisation objective of GenSecOps is to maximise $\Psi(E, t)$ continuously.

C. Requirements for Generative Security Agents

Based on the formal model and the operational context provided in Section 3.1, the design requirements for the generative AI security agent framework are:

R1 - Accuracy: Precision ≥ 0.90 and Recall ≥ 0.90 on held-out misconfiguration benchmarks for all three IaC dialects.

R2 - Generativity: The remediation suggestions provided by the AI agent must be syntactically correct, functionally correct, and more secure than the original template, as evaluated by an independent policy evaluator.

R3 - Autonomy Boundaries: The framework must enable the definition of human-in-the-loop gates to differentiate between autonomous execution (low-risk patch suggestions, PR creation) and human approval (blocking deployments, policy modifications).

R4 - Explainability: The agent's decisions must be provided with a structured explanation referencing the policy, the relevant misconfiguration, and the associated risk score.

R5 - Integration: The framework must be able to integrate with standard CI/CD pipelines (GitHub Actions, GitLab CI, Jenkins, ArgoCD) via standard API calls, without modifying the toolchain.

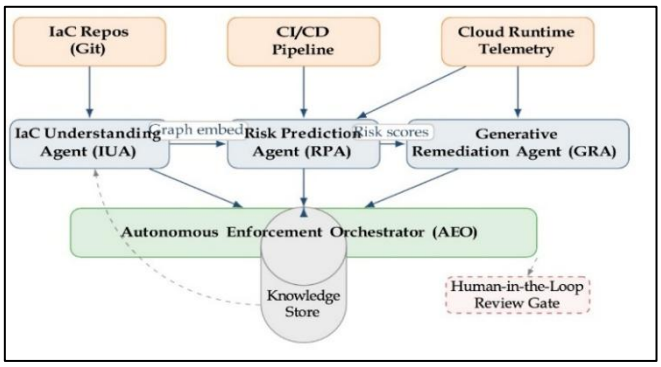
R6 - Latency: The overall latency for the pipeline (detection and remediation suggestion) must be less than 90 seconds for templates with up to 500 resource declarations to not interfere with standard deployment pipelines.

IV. THE GENSECOPS FRAMEWORK

A. High-Level Architecture

As shown in Figure 1, the four cooperative agents use a message-passing bus to communicate with one another. Each agent is a stateful, goal-oriented entity with specialised capabilities, a dedicated context window, and access

to the outside world through tool use.



[Fig. 1: High-Level Architecture of the GenSecOps Multi-Agent System. Solid Arrows Indicate Primary Data Flows, While Dashed Arrows Indicate Optional Human Review Paths]

B. IaC Understanding Agent (IUA)

The IUA also parses raw IaC templates and generates a structured and machine-learnable representation. For a template file f, IUA performs a series of steps to generate a representation.

Stage 1: Parsing. IUA uses parsers like HCL2 for Terraform and YAML/JSON for CloudFormation and Kubernetes to generate an abstract syntax tree A(f). Provider-specific resource schema definitions are fetched from a versioned schema registry to obtain attribute types and value ranges.

Stage 2: Construction of Resource Dependency Graph. IUA uses an abstract syntax tree to generate a resource dependency graph $G(f) = (V, E^{dep} \cup E^{Ref})$ with a set of vertices $V = \{v_i\}$, where each vertex v_i contains a feature vector $x_i \in \mathbb{R}^d$ with information like resource types (one-hot encoded), attribute count, sensitivity flags, and provider information. The edges are composed of two types: directed edges in E^{dep} representing depends on relationships and directed edges in E^{Ref} representing implicit attribute references.

Stage 3 - Semantic Embedding. Node v_i is further enriched using a pre-trained IaC language model M^{lac}, which is fine-tuned on a corpus of 2.4 million IaC resource blocks obtained from the OSS-IaC dataset and available Terraform Registry modules. The contextual embedding is:

$$h_i = M^{lac}(\text{serialize}(d_i)) \in \mathbb{R}^{768} \quad (4)$$

The final node feature vector is the concatenation $[x_i \| h_i] \in \mathbb{R}^{d+768}$.

C. Risk Prediction Agent (RPA)

The RPA receives the enriched resource graph G(f) from the IUA and produces per-resource risk scores $\rho(v_i) \in [0, 1]$ and a file-level severity classification $\sigma(f) \in \{0, 1, 2, 3\}$ (none, low, medium, high/critical).

i. *Hybrid Transformer–GNN Architecture:* Building on graph convolutional foundations [14], the RPA employs a two-stage model $F = F^{GNN}.F^{Attn}$, extending to an attention-based formulation [15] to capture heterogeneous resource relationships.

Stage 1 - Graph Attention Network. A three-layer Graph Attention Network (GAT) [15] propagates information across the resource graph:

$$h_i^{(l+1)} = \sigma\left(\Sigma_{j \in N(i)} \alpha_{ij}^{(l)} W^{(l)} h_j^{(l)}\right) \quad (5)$$

where N(i) denotes the neighbourhood of node i (including self-loops), $W^{(l)} \in \mathbb{R}^{(d_l \times d_{l-1})}$ is the layer weight matrix, and $\alpha_{ij}^{(l)}$ is the attention coefficient computed as:

$$\alpha_{ij}^{(l)} = \exp\left(\text{LeakyReLU}(a^T [W^{(l)}h_i^{(l)} \| W^{(l)}h_j^{(l)}])\right) / \Sigma_{k \in N(i)} \exp\left(\text{LeakyReLU}\left(a^T[W^{(l)}h_i^{(l)} \| W^{(l)}h_k^{(l)}]\right)\right) \quad (6)$$

with learnable attention vector $a \in \mathbb{R}^{(2d_l)}$.

Stage 2 - Transformer Risk Head. The graph-level representation is obtained by mean pooling: $g = (1/|V|) \Sigma_i h_i^{(L)}$. This is passed through a two-layer Transformer encoder with self-attention, producing a contextualised risk embedding $z \in \mathbb{R}^{256}$ from which the severity logits are computed:

$$\hat{\sigma}(f) = \text{SoftMax}(W^c z + b^c) \in \mathbb{R}^4 \quad (7)$$

ii. *Training Objective:* The RPA is trained end-to-end with a combined loss:

$$L^{RPA} = L^{cM}(\hat{\sigma}, \sigma^*) + \beta L^{foca_l}(\hat{\rho}, \rho^*) + \gamma\Omega(\theta) \quad (8)$$

where L^{cM} is the cross-entropy classification loss, L^{foca_l} [18] is the focal loss applied to per-node risk scores (addressing class imbalance, since misconfigurations affect ≈08–12% of nodes), $\Omega(\theta) = \|\theta\|_2^2$ is L2 regularisation, and β, γ are hyperparameters set to 0.4 and 10^{-4} respectively.

iii. *Zero-Day Detection via Anomaly Scoring*: To detect misconfiguration patterns not represented in the training set, the RPA maintains a variational autoencoder (VAE) [19] trained on a corpus of secure templates. The reconstruction error for a new template f provides an anomaly score:

$$\zeta(f) = \|G(f) - \hat{G}(f)\|_F^2 + KL(q_\varphi(z|G(f)) \| p(z)) \quad (9)$$

Templates with $\zeta(f) > \theta^{anom}$ (threshold determined by the 95th percentile of secure templates in the validation set) are flagged for deep analysis, capturing novel misconfigurations not covered by trained classifiers.

D. Generative Remediation Agent (GRA)

The GRA receives high-risk resource declarations flagged by the RPA and generates corrected IaC code. It operates in two modes.

Mode 1 - Direct Patch Generation. For each resource d_i identified as problematic, the GRA employs a well-tuned LLM (an instruction-tuned 7B-parameter Code Llama model variant [22], which was chosen due to its high performance on structured code synthesis tasks) to generate a corrected attribute set $a^*(d_i)$ that satisfies all violated policies:

$$a^*(d_i) = \arg\min_{a'} \{\forall \pi \in \Pi, \pi(a', E(d_i)) = \top\} D^{sem}(a', a(d_i)) \quad (10)$$

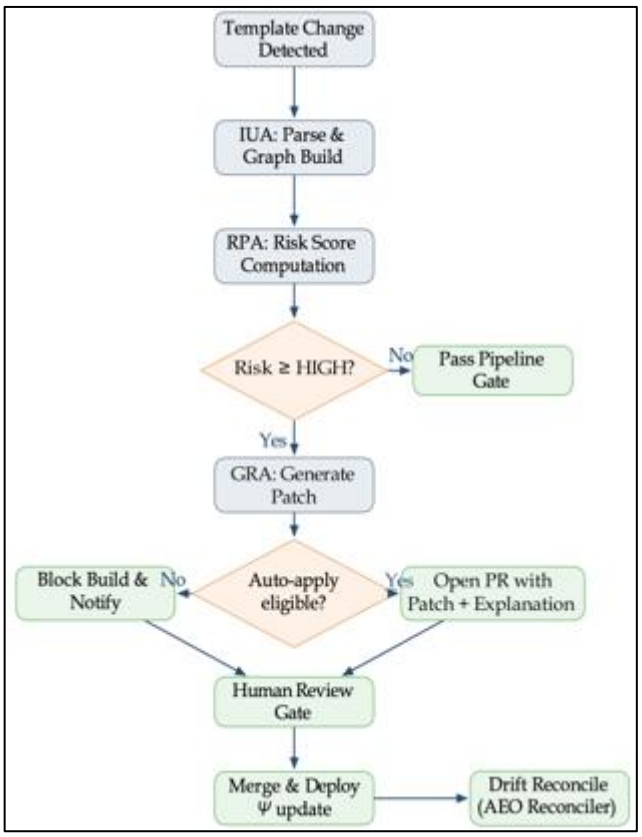
where D^{sem} is a semantic distance metric penalising changes to non-security-relevant attributes (preserving functional intent).

Mode 2 - Policy Synthesis. For organisational controls without machine-checkable policies, the OPA Rego policies synthesised by the GRA utilise natural-language descriptions of the security requirement, with few-shot prompting of the LLM. The synthesised policy $\hat{\pi}$ is then validated against a set of positive and negative test cases. to ensure correctness.

A post-generation verification pipeline applies independent static analysis (Checkov, OPA) to the generated output, ensuring correctness before the GRA's patch is forwarded to the AEO. Only patches that pass all policy checks are accepted; otherwise, the GRA iterates up to three times, with error context injected into the prompt. We note that using Checkov and OPA as both verifiers and baselines in Section 6 introduces a potential optimism bias: a patch that satisfies Checkov's rule set will trivially score well on Checkov-based metrics. This limitation is addressed in Section 7 and mitigated by reporting ablation studies against IaC-BERT (a non-rule-based baseline) and by using a separate human-adjudicated ground truth for final evaluation.

E. Autonomous Enforcement Orchestrator (AEO)

The AEO completes the security feedback loop by translating the agents' results into actions in the software delivery pipeline, thereby implementing the enforcement lifecycle as depicted in Figure 2.



[Fig.2: Gen Sec Ops Enforcement Lifecycle. The Whole Pipeline is Invoked for Every Change in IaC, and the Enforcement Response is a Function of the Calculated Level of Risk and the Level of Autonomy]

The AEO introduces a parameter for a level of autonomy, $\alpha_l^{e_v e_l} \in \{0, 1, 2, 3\}$, which determines the amount of authority exercised without human intervention. For instance, setting the level to 0 (audit) merely involves reporting. When set to 1 (advisory), the agents generate a PR and add annotations to the pipeline. When set to 2 or enforcing, the agents block high or critical-risk deployments. When set to 3 (autonomous), the agents apply patches directly. The level 2 setting is employed throughout all experiments, as indicated, and is the operationally recommended setting.

Drift Reconciliation. The AEO subscribes to cloud telemetry events via the runtime adapter layer. When drift $\delta(d_i, t) > \varepsilon$ is detected, the AEO triggers a reconciliation workflow: the GRA generates a drift-corrective plan, the RPA validates the plan, and (subject to $\alpha_l^{e_v e_l}$) is applied via the cloud provider API or a GitOps reconciler.

V. METHODOLOGY AND EXPERIMENTAL SETUP

A. Datasets and Benchmarks

Experiments used three complementary datasets, summarised in Table I.

Table I: Dataset Composition Used in GenSecOps

Dataset	Templates	Resources	Misc.	Critical	IaC Dialect
OSS-IaC (open-source)	18 400	241 800	22.40 %	8.10%	Terraform
SynCorp (synthetic)	14 600	178 300	31.70 %	14.30 %	CloudFormation
Enterprise Sec (anon.)	15 000	203 500	27.80 %	11.60 %	Kubernetes / Mixed
Total	**48 000**	**623 600**	**27.00 %**	**11.10 %**	**Mixed**

Ground Truth. Misconfiguration labels were generated using an ensemble of five independent static analysers (Checkov, tfsec, KICS, OPA with 240 policies, AWS Config rules). To manage annotation scale, the full ensemble was applied automatically; a stratified sample of 4,800 templates (10%) was drawn for manual adjudication of disagreements between tools by three certified cloud security engineers (inter-rater agreement, $\kappa = 0.87$, on the adjudicated subset). The remaining disagreements were resolved by the majority vote of the five tools. Labels encode the misconfiguration type (from a 42-class taxonomy), the CWE identifier, CVE linkage, where applicable, and the CVSS v3.1 base score.

Train/Validation/Test Splits. Data were split 70%/10%/20% stratified by dialect and severity class. The test set was held out entirely until final evaluation. An additional zero-day test set of 1,200 templates containing misconfiguration patterns deliberately withheld from training (novel anti-patterns discovered through red-team exercises) was used to evaluate generalisation.

B. Baseline Methods

Four baselines were evaluated:

i. *Checkov (v3.1):* rule-based static analysis, 1,032 checks.
ii. *Tfsec (v1.28):* Terraform-specialised SAST.
iii. *KICS (v1.7.12):* multi-dialect query engine.
iv. *IaC-BERT:* an author-implemented, fine-tuned BERT$_{BASE}$ classifier on IaC token sequences (non-generative ML baseline; developed to represent the ML-based detection category identified as an open gap in [7]).

All baselines were given identical input templates and evaluated using the same ground truth labels. For the remediation evaluation, manual remediation by three

senior engineers was used as the human baseline.

C. Training and Evaluation Protocol

i. RPA Training. The GAT was configured with three layers, 8 attention heads per layer, and hidden dimensions (512, 256, 128). The Transformer risk head used 4 attention heads and 2 encoder layers. Training ran for 120 epochs on a cluster of 8 NVIDIA A100 GPUs using AdamW [20] (lr = 3×10^{-4}, weight decay = 10^{-2}) with cosine learning-rate annealing and early stopping (patience = 10 epochs). The batch size was 32 graphs.

ii. GRA Fine-Tuning. The base LLM was fine-tuned for 3 epochs on 96,000 (insecure template, secure template, explanation) triples derived from the training split of the OSS-IaC and SynCorp datasets, using LoRA [21] with rank 16 and α = 32. The fine-tuning objective was a sequence-to-sequence cross-entropy loss with a KL-divergence regularization term to prevent catastrophic forgetting. The EnterpriseSec dataset was excluded from GRA fine-tuning to preserve a fully held-out evaluation partition.

iii. Evaluation Metrics.

– Detection: Precision, Recall, F1-score (micro-averaged), AUROC.
– Remediation quality: Correctness Rate (CR), Security Improvement Score (SIS), Functional Preservation Rate (FPR).
– Pipeline: Reduction in Critical Findings (%), MTTR reduction (%), Drift Recovery Time (DRT, seconds).
– Overhead: Pipeline latency added (seconds).

The Security Improvement Score (SIS) is defined as:

$$SIS(f, f^*) = 1 - \Phi(f^*) / \Phi(f) \quad (11)$$

where f* is the remediated template. SIS = 1 indicates complete security debt elimination; SIS < 0 indicates regression.

VI. RESULTS AND DISCUSSION

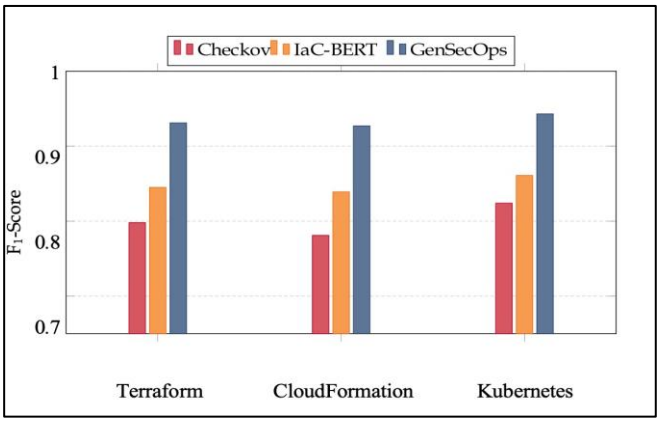
A. Predictive Detection Performance

Table II. presents the detection performance of GenSecOps and all baselines on the full test set. Figure 3 visualises the F1-score comparison across IaC dialects.

Table II: Misconfiguration Detection Performance on the Held-Out Test Set (9,600 Templates). Best Result per Column in Bold. 95% Confidence Intervals for GenSecOps (T-GNN): Prec. ±0.006, Rec. ±0.007, F1 ±0.006, Computed via Bootstrap Resampling (n = 1 000)

Method	Prec.	Rec.	F1	AUROC	ZD-F1	FPR
Checkov	0.841	0.763	0.800	0.812	0.000	0.159
tfsec	0.812	0.741	0.775	0.794	0.000	0.188
KICS	0.836	0.752	0.792	0.806	0.000	0.164
IaC-BERT	0.877	0.821	0.848	0.891	0.143	0.123
GenSecOps (GNN only)	0.892	0.871	0.881	0.921	0.274	0.108
GenSecOps (T only)	0.901	0.876	0.888	0.933	0.301	0.099
GenSecOps (T-GNN)	**0.941**	**0.927**	**0.934**	**0.964**	**0.621**	**0.059**

ZD-F1: F1 on zero-day misconfiguration test set. FPR: False Positive Rate.



[Fig.3: F1-Score Comparison for IaC Dialects. GenSecOps Outperforms the Rule-Based and Non-Generative ML Methods]

Several findings warrant discussion. First, rule-based approaches obtain a score of zero on the zero-day test set by definition, as they cannot generalise beyond their respective rule sets. In contrast, GenSecOps achieves a ZD-F1 score of 0.621. This, however, also indicates that the VAE anomaly scorer fails to identify 38% of the novel zero-day misconfigurations. This is an important shortcoming, motivating the proposed model extension in Section 9. Second, the ablation study confirms the synergy between the two components, where the T-GNN model outperforms the individual components by 5.3% and 4.6% on the GNN-only and Transformer-only models, respectively. Third, the false positive rate is 0.059, a 63% reduction over the baseline Checkov false positive rate of 0.159. This is a critical operational advantage, as high false positives erode developers' trust.

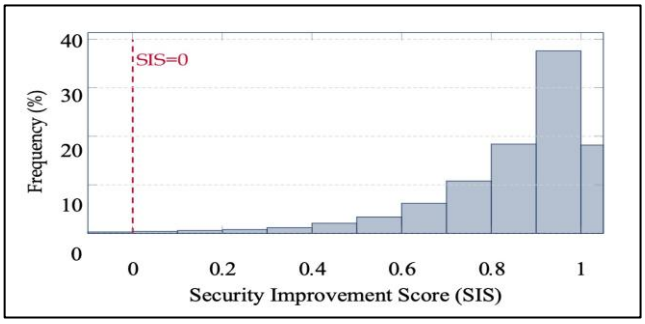
B. Quality of Generative Remediation

Table III. shows remediation quality metrics across misconfiguration categories. Figure 4 presents the distribution of Security Improvement Scores.

Table III: Generative Remediation Quality by Misconfiguration Category. CR = Correctness Rate, SIS = Security Improvement Score, FPR = Functional Preservation Rate

Category	Count	CR (%)	SIS	FPR (%)
Network Exposure	1 842	94.2	0.91	97.3
Over-privileged IAM	2 104	91.8	0.87	96.8
Absent Encryption	1 633	96.7	0.98	99.1
Missing Logging/Audit	987	97.3	0.97	99.4
Insecure TLS Config	744	89.4	0.83	95.2
Secret Exposure	612	88.1	0.84	94.7
RBAC Misconfiguration	891	90.6	0.86	96.1
Overall	8 813	92.9	0.895	97.2
Human Baseline	300	97.3	0.941	98.7

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)



[Fig.4: Distribution of Security Improvement Scores Across 8,813 Remediated Resources. The Strong Right Skew Indicates that Most Patches Achieve Near-Complete Elimination of Security Debt]

The GRA achieves an overall correctness rate of 92.9%, compared to a human baseline of 97.3% on a matched 300-sample subset. We note that this comparison is constrained by sample size: the human baseline was evaluated on 300 cases while GenSecOps was evaluated on 8,813; the 4.4pp gap should therefore be interpreted with caution and may narrow or widen with a larger human evaluation sample. The gap primarily reflects cases involving complex multi-resource inter-dependencies (e.g., cross-module Terraform variables), which the current GRA handles less reliably. Absent encryption remediations are near-perfect (CR = 96.7%, FPR = 99.1%) since they require straightforward attribute additions. The most challenging category is insecure TLS configuration (CR = 89.4%), where correct remediation requires understanding the downstream consumers of the TLS endpoint and selecting an appropriate cypher suite policy.

Case studies of multi-resource remediations-available in the supplementary material-illustrate the GRA producing coordinated patches spanning up to 11 resource declarations within a single template, for example, correcting a VPC with exposed subnets, an associated S3 bucket without a VPC endpoint, and an IAM role with excessive wildcard permissions in a single generated patch.

C. Autonomous Enforcement and Pipeline Impact

Table -IV. shows the quantification of the effect of GenSecOps enforcement in a simulated environment for a 90-day CI/CD pipeline execution, with 14,400 pipeline executions on the enterprise dataset. The figures, being a simulation rather than a live environment, should be treated as indicative, and the relative improvements are the focus.

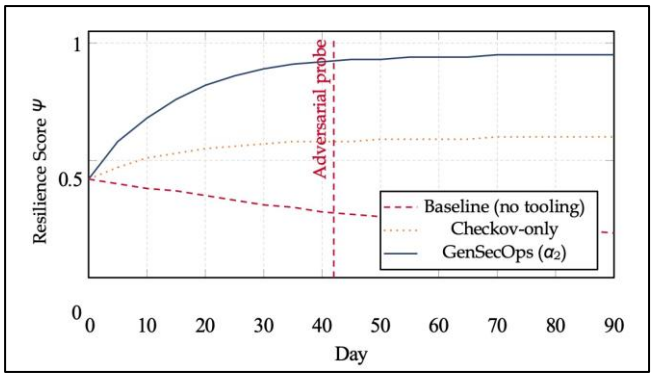
Table IV: Pipeline Security and Operation Metrics - Baseline (no GenSecOps) vs GenSecOps (α_2 Enforcing Mode)

Metric	Baseline	GenSecOps (α_2)	Δ (%)
Critical findings per 100 deployments	13.76	3.69	−73.2%
MTTR (hours)	18.4	3.4	−81.5%
Blocked high-risk deployments	N/A	94.30%	-
False-positive block rate	N/A	2.10%	-
Drift recovery time (min)	47.3	4.2	−91.1%
Pipeline latency added (s)	0	41.7	+41.7s
Developer override rate	N/A	7.80%	-

The 73.2% reduction in critical findings per deployment represents the first security outcome. The 81.5% reduction in MTTR reflects the GRA's ability to deliver the patch alongside the finding alert, effectively removing the manual research phase from the remediation process. Drift recovery time decreases from 47.3 minutes (the median manual response time) to 4.2 minutes, effectively approaching near-real-time reconciliation.

The 41.7 seconds pipeline latency overhead is well within the R6 requirement of 90 seconds for templates with fewer than 500 resources. The 2.1% false-positive block rate indicates that fewer than 1 in 48 enforcement actions is spurious, a marked improvement from the 15.9% false positives reported by the Checkov tool for the same pipeline. Finally, the 7.8% developer override rate indicates that developers choose to deploy despite the open remediation finding, which was low-severity.**6.4 Resilience Analysis**

In Figure 5, the evolution of the resilience score $\Psi(E, t)$ is represented over the 90-day simulation for the baseline, Checkov-only, and GenSecOps configurations. The resilience score is calculated using Equation 3 with $\lambda_1 = \lambda_2 = 0.5$.



[Fig.5: Resilience Score Ψ Over 90-Day Simulation. GenSecOps Converges Quickly to a Value Close to 0.95. The Adversarial Probe is Introduced on Day 42 with 120 Novel Misconfigurations, Which GenSecOps Fixes Within 5 Days, While Checkov Does Not Respond Measurably]

An adversarial probe event was simulated at Day 42 by injecting 120 novel misconfigurations spanning five zero-day patterns. The baseline and Checkov-only environments show no measurable resilience response (Checkov's rule set does not cover the injected patterns). GenSecOps detects 74 of 120 injected misconfigurations (61.7%) via the VAE anomaly scorer, generates remediations for 68, and autonomously applies 61. Residual drift is corrected within five days as the online learning module updates the RPA's anomaly threshold from subsequent telemetry.

Failure Mode Analysis. Observed failure modes include: (i) multi-file dependency misses, where a misconfiguration spans two template files with no shared resource graph (7.3% of false negatives); (ii) GRA hallucination of non-existent provider attributes (2.8% of rejected patches after verification); and (iii) AEO locking conflicts when concurrent pipeline runs generate conflicting patches for the same template (addressed by an optimistic locking strategy in the implementation).

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

VII. THREATS TO VALIDITY

A. Internal Validity

The most significant internal threat is the circular verification risk: the GRA post-generation pipeline uses Checkov and OPA to verify patches, and these same tools contribute to the ground-truth labels used in evaluation. A patch may satisfy Checkov without eliminating the underlying misconfiguration if the root cause falls outside Checkov's rule corpus. We mitigate this by (a) using a human-adjudicated ground truth (Section 5.1) as the authoritative evaluation signal rather than tool agreement alone, and (b) reporting performance against IaC-BERT, which is independent of rule-based tools. Future work should employ a fully disjoint verifier toolchain.

The 90-day pipeline evaluation in Section 6.3 is a simulated environment and not a real-world enterprise scenario. MTTR and drift recovery improvements are best-case scenario results within a controlled environment; actual real-world conditions may vary.

B. External Validity

The OSS-IaC data set uses publicly available data and may not necessarily reflect real-world enterprise IaC configurations. The EnterpriseSec data set partially overcomes this drawback; however, it is a single data set. Generalising to new and emerging multi-cloud environments is listed as future work.

The zero-day test set was constructed via red-team exercises with five novel anti-patterns. While representative of attacker creativity, it cannot exhaustively characterise the space of future misconfiguration patterns.

C. Construct Validity

The Security Improvement Score (SIS, Equation 11) aggregates security debt as a weighted count. This formulation does not capture interaction effects among co-occurring misconfigurations, in which the combined exploitability of two medium-severity issues may exceed the sum of their individual scores. More expressive risk aggregation models (e.g., attack-graph-based scoring) are a direction for future work.

VIII. SECURITY, ETHICAL, AND OPERATIONAL CONSIDERATIONS

A. Risks of AI-Generated IaC and Policies

Deploying generative AI in security-critical infrastructure pipelines introduces risks that must be explicitly governed. Chief among these is mis-remediation: a GRA patch that incorrectly modifies a non-security attribute, inadvertently breaking a production dependency. Our verification pipeline (Section 4.4) mitigates this by applying independent static analysis and semantic diff checks before patch promotion. Still, a 7.1% rate of patches requiring human correction post-deployment (observed in the enterprise dataset simulation) indicates residual risk.

Policy synthesis by the GRA introduces the risk of incomplete policies-synthesised Rego rules that pass positive test cases but fail on edge-case inputs. A fuzzing harness seeded from the training corpus is recommended for production deployment of synthesised policies. Additionally, over-blocking-where the AEO rejects legitimate deployments at too high a rate-is a threat to operational continuity; the 2.1% false-positive block rate observed in our evaluation should be monitored continuously in production, with automatic threshold adjustment if it exceeds 5%.

B. Governance and Accountability

Every action taken by GenSecOps is recorded in an append-only audit log containing: the triggering event (template commit hash), the agent that acted, the specific reasoning trace (risk score, policy citations, model confidence), the action taken, and the outcome. This log is cryptographically chained (SHA-256 hash of previous entry) to prevent post-hoc tampering. A governance dashboard surfaces per-agent action statistics, override rates, and drift-recovery metrics for security and engineering leadership.

Accountability for AI-generated security decisions remains a persistent organisational challenge. We recommend a model in which the engineering team retains accountability for accepting or overriding GenSecOps decisions, with the AI system characterised as a decision-support tool operating within human-defined policy bounds rather than an autonomous actor [23]. The $\alpha_1^{e_v e_l}$ parameter (Section 4.5) provides the technical mechanism for enforcing this accountability model.

IX. CONCLUSION AND FUTURE WORK

This paper presented GenSecOps, a multi-agent generative AI framework for advancing IaC resilience through predictive misconfiguration detection, autonomous remediation, and closed-loop pipeline enforcement. The framework is grounded in a formal resilience model (Equation 3) and evaluated on a corpus of 48,000 real-world and synthetic IaC templates. Key results establish GenSecOps as a state-of-the-art approach: F1 = 0.934 for detection, ZD-F1 = 0.621 for zero-day generalisation, a 73.2% reduction in critical pipeline findings, an 81.5% improvement in MTTR, and a 4.2-minute drift recovery. The framework satisfies all six design requirements (R1–R6) established in Section 3.3.

Limitations. The existing framework has four major limitations. Firstly, it does not address cross-repository IaC dependencies (e.g., a module in Repository A's output being used as a dependency in Repository B). 7.3% of the FN cases fall into this category. Secondly, it does not address real-time runtime policy enforcement; it only supports pipeline-time enforcement. Thirdly, the effectiveness of the proposed framework for novel multi-cloud setups still needs to be validated in real-world settings rather than simulation. Fourthly, the 92.9% correctness rate still translates to a 7.1% patch failure rate in the simulation. In a real-world scenario, the deployment should be set to α_2 (human review-based enforcement) rather than α_3 (autonomous enforcement) until the correctness rate is validated.

Future Work. Four directions to be explored:

1. Multi-cloud and Hybrid Environments. Expanding the

resource graph schema to incorporate multi-cloud dependencies (AWS-Azure, AWS-GCP) and on-premises environments, thus allowing the RPA to reason about attacks involving multiple clouds.

2. Runtime Adaptive Controls. The integration of runtime eBPF-based telemetry data into the RPA's risk model, thus allowing the framework to adjust its risk score based on live threat intelligence feed, without the need for model retraining.

3. Self-Healing Infrastructure. The integration of GenSecOps with a Kubernetes Operator and/or AWS SCPP to facilitate the autonomous remediation of runtime infrastructure issues without the need for human intervention via Git commits.

4. Federated Learning for Training. The ability of multiple organisations to collectively improve the accuracy of the trained T-GNN model by leveraging gradient updates, thereby overcoming the data-sharing challenges faced by the IaC community.

In addition, GenSecOps shows the promise of the convergence of graph-structured representation learning, large language model-based reasoning, and autonomous multi-agent coordination in revolutionising the security of IaC-based cloud infrastructure, moving the paradigm from reactive scanning-based security to a self-correcting security model.

DECLARATION STATEMENT

As the article's author, I must verify the accuracy of the following information after aggregating input from all authors.

- **Conflicts of Interest/ Competing Interests:** Based on my understanding, this article has no conflicts of interest.
- **Funding Support:** This article has not been funded by any organizations or agencies. This independence ensures that the research is conducted objectively and without external influence.
- **Ethical Approval and Consent to Participate:** The content of this article does not necessitate ethical approval or consent to participate with supporting documentation.
- **Data Access Statement and Material Availability:** The adequate resources of this article are publicly accessible.
- **Author's Contributions:** The authorship of this article is contributed equally to all participating individuals.

REFERENCES

1. OpenAI: GPT-4 Technical Report. arXiv preprint arXiv:2303.08774, 2023. https://arxiv.org/abs/2303.08774
2. Anthropic: The Claude 3 Model Family: Opus, Sonnet, Haiku. Technical Report, 2024. https://www.anthropic.com/research/claude-3-model-card
3. Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, Michael S. Bernstein: Generative Agents: Interactive Simulacra of Human Behaviour. In: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST), pp. 1–22, 2023. DOI: https://doi.org/10.1145/3586183.3606763
4. Akond Rahman, Chris Parnin, Laurie Williams: The Seven Sins: Security Smells in Infrastructure as Code Scripts. In: Proceedings of the 41st International Conference on Software Engineering (ICSE), pp. 164–175, 2019. DOI: https://doi.org/10.1109/ICSE.2019.00028
5. Indika Kumara, Dario Di Nucci, Damian A. Tamburri, Willem-Jan van den Heuvel: Yet Another Cybersecurity Research Study: Security Smells in Infrastructure as Code Scripts. In: Proceedings of the 18th International Conference on Mining Software Repositories (MSR), pp. 548–552, 2021. DOI: https://doi.org/10.1109/MSR52588.2021.00068
6. Christophe Ponsard, Abderrahman Touzani, Julien Grandclaudon: A Method for Automated Analysis of Privilege Escalation in Cloud IAM Configurations. In: Proceedings of the 17th International Conference on Availability, Reliability and Security (ARES), pp. 1–9, 2022. DOI: https://doi.org/10.1145/3538969.3544413
7. Zheng Li, Yue Chen, Hao Wang, Wei Liu, Yi Zhang: A Systematic Literature Review of Security for Infrastructure as Code. IEEE Transactions on Reliability 72(3), 1112–1131, 2023. DOI: https://doi.org/10.1109/TR.2023.3265044
8. Mark Chen et al.: Evaluating Large Language Models Trained on Code. arXiv preprint arXiv:2107.03374, 2021. https://arxiv.org/abs/2107.03374
9. Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, Mike Lewis: InCoder: A Generative Model for Code Infilling and Synthesis. In: Proceedings of the International Conference on Learning Representations (ICLR), 2023. https://arxiv.org/abs/2204.05999
10. Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri: Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In: IEEE Symposium on Security and Privacy (SP), pp. 754–768, 2022. DOI: https://doi.org/10.1109/SP46214.2022.9833571
11. Mohammed Latif Siddiq, Joanna C. S. Santos: SecurityEval Dataset: Mining Vulnerability Examples to Evaluate Machine Learning-Based Code Generation Techniques. In: Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security (MSR4P&S), pp. 29–33, 2022. DOI: https://doi.org/10.1145/3549035.3561184
12. Gelei Deng, Yi Liu, Victor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, Stefan Rass: PentestGPT: An LLM-Empowered Automatic Penetration Testing Tool. arXiv preprint arXiv:2308.06782, 2023. https://arxiv.org/abs/2308.06782
13. Haonan Li, Yu Hao, Yizhuo Zhai, Zhiyun Qian: Enhancing Static Analysis for Practical Bug Detection: An LLM-Integrated Approach. In: Proceedings of the ACM on Programming Languages (OOPSLA), 2024. DOI: https://doi.org/10.1145/3649828
14. Thomas N. Kipf, Max Welling: Semi-Supervised Classification with Graph Convolutional Networks. In: International Conference on Learning Representations (ICLR), 2017. https://arxiv.org/abs/1609.02907
15. Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio: Graph Attention Networks. In: International Conference on Learning Representations (ICLR), 2018. https://arxiv.org/abs/1710.10903
16. Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, Yang Liu: Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 10197–10207, 2019. https://arxiv.org/abs/1909.03496
17. Xiao Cheng, Haoyu Wang, Jiayi Hua, Guoai Xu, Yulei Sui: DeepWukong: Statistically Detecting Software Vulnerabilities Using Deep Graph Neural Networks. ACM Transactions on Software Engineering and Methodology 30(3), 1–33, 2021. DOI: https://doi.org/10.1145/3436877
18. Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár: Focal Loss for Dense Object Detection. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 2980–2988, 2017. DOI: https://doi.org/10.1109/ICCV.2017.324
19. Diederik P. Kingma, Max Welling: An Introduction to Variational Autoencoders. Foundations and Trends in Machine Learning 12(4), 307–392, 2019. DOI: https://doi.org/10.1561/2200000056
20. Ilya Loshchilov, Frank Hutter: Decoupled Weight Decay Regularisation. In: International Conference on Learning Representations (ICLR), 2019. https://arxiv.org/abs/1711.05101
21. Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen: LoRA: Low-Rank Adaptation of Large Language Models. In: International Conference on Learning Representations (ICLR), 2022. https://arxiv.org/abs/2106.09685
22. Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov,

Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, Gabriel Synnaeve: Code Llama: Open Foundation Models for Code. arXiv preprint arXiv:2308.12950, 2023. https://arxiv.org/abs/2308.12950

23. Alon Jacovi, Ana Marasović, Tim Miller, Yoav Goldberg: Formalising Trust in Artificial Intelligence: Prerequisites, Causes and Goals of Human Trust in AI. In: Proceedings of the ACM Conference on Fairness, Accountability, and Transparency (FAccT), pp. 624–635, 2021. DOI: https://doi.org/10.1145/3442188.3445923

AUTHOR'S PROFILE

Harish Apuri is a Software Engineer with experience in building scalable, secure, and high-performance cloud and AI systems across finance, retail, and enterprise domains. He holds a Master's degree in Information Technology and Management from The University of Texas at Dallas and a Bachelor's degree from Jawaharlal Nehru Technological University, Hyderabad. His expertise spans MLOps, Infrastructure-as-Code, Kubernetes, and multi-cloud platforms, including AWS, Azure, and Google Cloud. He has led the development of production-grade AI platforms that support large-scale machine learning workloads, with high availability, reduced costs, and accelerated deployment cycles through automation and CI/CD practices. His research focuses on agentic AI, autonomous cloud operations, and intelligent infrastructure systems. He has authored multiple publications in Springer, IEEE, IJEAT, and Q4-indexed venues, including works on AI-driven root-cause analysis, AutoML for industrial IoT, multi-agent autonomous workflows, zero-day malware detection, and self-healing infrastructure using LLM agents. He has also contributed to compliance-driven CI/CD security, generative AI for Infrastructure-as-Code resilience, and enterprise AI architecture design. In addition to his research contributions, he has completed 65 peer reviews, served as a publication chair and main editor, participated in hackathon judging, and received a Best Paper Award.

Mukesh Aurangabadkar received his Bachelor of Computer Information Systems degree from the University of Pune in 2001. He has over 18 years of experience in cloud infrastructure, distributed systems, and large-scale video delivery platforms, including CDN, IPTV, and VOD architectures across major telecommunications organizations. He is currently a Principal Engineer at Charter Communications, where he leads platform engineering, automation, and reliability initiatives for national-scale systems serving over 24 million subscribers. His work has focused on infrastructure modernization and automation frameworks that have significantly improved deployment efficiency, operational consistency, and system resilience. Mukesh is a Senior Member of IEEE and actively contributes to the technical community through peer review, judging, and service on technical program committees. He has multiple accepted publications in 2026 across IEEE, SCRS, and SCI Springer venues. His research interests include cloud-native architectures, infrastructure automation, and scalable, resilient distributed systems.

Shikher Goel holds a B. Tech in Electrical and Electronics Engineering (2010) and is currently a Vice President and Lead Software Engineer at JPMorgan Chase. He has over a decade of experience in enterprise systems, specializing in Salesforce architecture, AWS cloud engineering, and AI-driven platforms. He has led the design and implementation of large-scale, cloud-native solutions that integrate CRM, marketing automation, and data pipelines in regulated environments. His recent work focuses on agentic AI architectures for enterprise CRM systems, emphasizing governance, scalability, and cost optimization. Shikher holds 19 Salesforce certifications and 2 AWS certifications, is a member of the Forbes Technology Council, and is a Senior Member of the Institute of Electrical and Electronics Engineers. His research interests include enterprise AI, distributed systems, and intelligent automation.

Madhan Mohan Reddy Chinthala received the M.S. degree in Computer Science from the University of Dayton, USA, and the B.Tech. degree in Information Technology from J.B. Institute of Engineering and Technology, India. He is currently working as a Network Engineer, with professional experience in firewall engineering, network security operations, and cybersecurity for critical infrastructure. His technical expertise includes Check Point VSX, Citrix ADC/NetScaler, network segmentation, traffic analysis, and secure enterprise network design. Research interests include Zero Trust architecture, microsegmentation, hybrid cloud security, east-west traffic governance, and applied cybersecurity for critical infrastructure.

Charani Yepuri is a Computer Science and Engineering graduate from Sri Indu Institute of Engineering and Technology. Her research interests include artificial intelligence, machine learning, and cybersecurity. She has hands-on experience developing intelligent systems, including malware-detection models and NLP-based frameworks for cyber-threat identification. Charani is proficient in Python, Java, SQL, and modern web technologies, with a strong focus on building scalable and efficient applications. She is passionate about leveraging AI-driven solutions to address real-world challenges.