

An Open-Agent Framework for Intelligent Compliance Analysis in Infrastructure as Code Using Large Language Models

Ayah Al-Jabali

`aya20248059@std.psut.edu.jo`

Princess Sumaya University for Technology

Mu'awya Al-Dala'ien

Princess Sumaya University for Technology

Research Article

Keywords: Cloud security, Cloud compliance, Cloud governance, Cloud-native security, Infrastructure as Code, Policy-as-Code, Open Policy Agent, Large Language Models, NIST SP 800-53, Terraform, Governance Risk and Compliance

Posted Date: October 19th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-10907487/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.
[Read Full License](#)

Additional Declarations: No competing interests reported.

An Open-Agent Framework for Intelligent Compliance Analysis in Infrastructure as Code Using Large Language Models

Ayah Al-Jabali^{1*} and Mu'awya Al-Dala'ien¹

^{1*}Department of Cybersecurity, Princess Sumaya University for Technology, Khalil Saket Street, Amman, 11941, Jordan.

*Corresponding author(s). E-mail(s): aya20248059@std.psut.edu.jo;
Contributing authors: m.aldalaien@psut.edu.jo;

Abstract

Cloud misconfiguration is a leading cause of data breaches in production cloud environments, and Infrastructure as Code (IaC) has amplified this risk by letting a single insecure configuration propagate across an entire cloud estate in one deployment. This paper presents an open-agent framework for intelligent compliance analysis of IaC configurations, integrating Open Policy Agent (OPA)-based Policy-as-Code enforcement with Large Language Model (LLM)-driven contextual reasoning in a single hybrid pipeline. Prior work has treated deterministic policy enforcement and LLM-based analysis independently; the proposed framework instead combines the precision of OPA-based Rego rule evaluation with the explanatory and remediation capabilities of an LLM to produce structured, audit-ready compliance evidence aligned to a formal control framework. The framework is instantiated with a custom 16-rule Rego policy library spanning three NIST SP 800-53 control families Access Control (AC), System and Communications Protection (SC), and Audit and Accountability (AU) across eight AWS resource types, and evaluated on 231 labeled Terraform configurations derived from the IaC-Eval benchmark. The OPA enforcement layer achieves 96.9% precision, 90.5% recall, and 93.6% F1 score. The LLM enrichment layer, guided by a deterministic safeguard against LLM recall errors and explicit prompt-level format constraints, produces reliable, audit-ready control mappings and remediation guidance across 16 violation types. The performance advantage over an industrial static-analysis baseline (Checkov) is confirmed statistically significant via McNemar's exact test ($\chi^2 = 13.0$, $p < 0.001$), demonstrating that the hybrid approach delivers compliance analysis outcomes that neither paradigm achieves alone.

Keywords: Cloud security, Cloud compliance, Cloud governance, Cloud-native security, Infrastructure as Code, Policy-as-Code, Open Policy Agent, Large Language Models, NIST SP 800-53, Terraform, Governance Risk and Compliance

1 Introduction

Cloud computing is now the dominant deployment model for modern digital services, a shift that has moved the burden of securing infrastructure from on-premise systems to the configuration of cloud resources. Under the shared-responsibility model adopted by every major cloud provider, correctly configuring a cloud resource is the customer’s responsibility, not the provider’s [1]. A single misconfigured storage bucket, identity policy, or audit-logging setting can therefore expose an entire production environment to unauthorized access or regulatory non-compliance. Industry estimates place the great majority of cloud security failures on the customer side, with misconfiguration as the leading cause [2], and as cloud adoption has scaled, organizations operating under regimes such as NIST SP 800-53, FedRAMP, ISO/IEC 27001, and PCI-DSS have come under growing pressure to demonstrate continuous, evidence-based compliance across rapidly changing multi-cloud estates.

Infrastructure as Code (IaC) has emerged as a foundational response to this pressure, letting organizations define, provision, and manage cloud infrastructure through machine-readable configuration files rather than manual processes or interactive consoles [3]. Terraform, built on HashiCorp Configuration Language (HCL), has become the dominant IaC tool in industry, with independent survey data placing its share of production IaC deployments above 70% [4, 5]. Because infrastructure definitions are encoded in a declarative language, every resource, permission, and network rule can be reviewed, scanned for vulnerabilities, and evaluated against compliance requirements before it ever reaches production [6]. That auditability is what makes IaC a natural entry point for automated compliance analysis.

Policy as Code (PaC) builds on this entry point by expressing security and compliance rules as machine-executable code, so that the security posture of an infrastructure definition can be enforced before any resource is deployed [7]. Open Policy Agent (OPA), a graduated project of the Cloud Native Computing Foundation, is the most widely adopted open-source engine for this purpose, evaluating structured representations of Terraform configurations against policies written in Rego, a purpose-built declarative query language [8]. In a compliance-automation setting, this lets every infrastructure change be checked against a control framework such as NIST SP 800-53 in seconds, producing consistent, auditable results. A deterministic engine such as OPA, however, only tells half the story: it can report that a rule was violated, but not why the violation matters or how to remediate it in language a non-technical auditor can use.

This is where Large Language Models (LLMs) enter. LLMs have shown strong capability in code understanding, security analysis, and natural-language generation, which makes them attractive for enhancing IaC security workflows [9], and recent studies report that LLM-based systems can surface context-specific misconfigurations

in IaC templates that static analyzers miss [10]. Used alone, however, LLMs are prone to hallucination and output inconsistency, which makes them unreliable for enforcement tasks that require deterministic, repeatable results [11]. The resulting tension between deterministic precision without explanation on one side, and rich explanation without determinism on the other is what motivates the hybrid architecture proposed in this paper.

Why NIST SP 800-53. NIST SP 800-53, ISO/IEC 27001, COBIT, and PCI-DSS are all directly relevant to cloud governance, so the choice of NIST SP 800-53 as this paper’s instantiation target deserves explicit justification: the decision is architectural, not a judgement that the other frameworks are less important. NIST publishes its control catalog in the Open Security Controls Assessment Language (OSCAL), a machine-readable JSON/XML/YAML format designed specifically so that tooling can parse and automate against individual controls [12], whereas ISO/IEC 27001 and COBIT are licensed, prose-based standards without a comparably structured, freely reusable form [13, 14]. NIST SP 800-53 is also the baseline catalog beneath FedRAMP and FISMA. Because the proposed framework expresses controls as executable Rego predicates, a catalog that is already structured and machine-oriented is the more natural fit for a first instantiation; the architecture itself is not tied to NIST and could be re-instantiated against any control framework expressible in Rego.

1.1 Problem Statement

Cloud misconfiguration is one of the leading causes of data breaches in production cloud environments, and the rapid adoption of IaC has amplified its impact: a single insecure configuration can now be replicated across hundreds of cloud resources in a single deployment. LLMs are increasingly used to author and review IaC, yet up to 75% of LLM-generated Terraform configurations have been found to contain security misconfigurations [11], making standalone LLMs unsuitable for compliance enforcement. This sharpens the problem into a precise research gap: no existing peer-reviewed system integrates OPA-based PaC enforcement with LLM-driven contextual reasoning for the compliance analysis of existing IaC configurations, aligned to a formal control framework and evaluated quantitatively against a labeled dataset. Toprani and Madiseti [10] demonstrate LLM-based IaC misconfiguration detection but do not integrate deterministic PaC enforcement, do not align their output to any formal compliance framework, and report results from only ten manually crafted templates. Mohammed [15] validates the conceptual viability of PaC for compliance automation but presents a theoretical framework with no LLM integration and no empirical evaluation. Verdet et al. [16] carry out the most comprehensive empirical study of IaC security adoption to date, yet rely entirely on existing static analysis tools, with no LLM enrichment, no compliance-framework alignment, and no audit-ready output. ARPaCCino [17] combines LLMs with OPA-based Rego enforcement, but is aimed at automated policy rule generation rather than compliance analysis of existing IaC configurations. Taken together, these systems expose a clear architectural gap: deterministic enforcement and intelligent reasoning have been pursued as independent lines of work, and no current system unifies them in a single pipeline that produces structured, audit-ready compliance evidence mapped to a formal control framework.

1.2 Research Questions and Contributions

This work is guided by three research questions. **RQ1:** Can a hybrid pipeline that couples deterministic Policy-as-Code enforcement with LLM-driven reasoning detect NIST SP 800-53 compliance violations in Terraform configurations with precision and recall competitive with, or exceeding, existing LLM-only and static-analysis-only approaches? **RQ2:** Can an LLM enrichment layer reliably generate accurate, audit-ready control mappings and remediation guidance for violations detected by a deterministic policy engine, and at what API cost? **RQ3:** Which NIST SP 800-53 control families are most frequently violated in realistic Terraform configurations, and does this distribution corroborate independent empirical findings in the literature? These questions are addressed empirically in Sections 5 and 6.

The contributions of this paper are threefold:

- **A hybrid open-agent architecture.** A pipeline design that integrates OPA-based, deterministic Policy-as-Code enforcement with LLM-driven contextual reasoning in a single workflow, so that every finding is both formally verified by a rule engine and enriched with a plain-language explanation, an official control mapping, and a suggested remediation.
- **A Rego policy library mapped to NIST SP 800-53.** A custom-authored library of sixteen Rego rules spanning three NIST SP 800-53 control families Access Control (AC), System and Communications Protection (SC), and Audit and Accountability (AU) covering misconfiguration patterns across eight AWS resource types (S3, RDS, EBS, IAM, CloudTrail, Load Balancer, Security Group, and DynamoDB).
- **A labeled evaluation dataset and empirical evaluation.** A labeled evaluation dataset of 231 Terraform configurations constructed from the IaC-Eval benchmark [18] the first application of this benchmark for security compliance evaluation rather than code generation together with an empirical evaluation reporting 96.9% precision, 90.5% recall, and 93.6% F1 score for the OPA enforcement layer, reliable audit-ready control mapping and remediation guidance from the LLM enrichment layer achieved through deterministic safeguards against LLM recall errors and explicit prompt-level format constraints rather than through unconstrained model behaviour and a statistically significant improvement over the Checkov industrial baseline (McNemar’s exact test, $\chi^2 = 13.0$, $p < 0.001$).

The term *open-agent* in the framework’s name refers to two design choices rather than to LLM-based autonomy. First, it references Open Policy Agent (OPA), whose own documentation describes it as a general-purpose policy enforcement agent, and which serves as the deterministic enforcement layer in the proposed pipeline. Second, the LLM component acts as a contextual reasoning agent over the structured violation output produced by OPA. The framework is therefore not an autonomous agent in the sense of planning-based systems such as ReAct or AutoGPT [19]; it does not employ a planning loop, dynamic tool selection, or autonomous goal-pursuit. Each component instead performs a well-defined, deterministic role, a deliberate design choice for compliance-critical workflows where predictability and repeatability are first-class requirements.

The remainder of this paper is organized as follows. Section 2 reviews related work in IaC security, Policy-as-Code, LLM-based security analysis, and IaC benchmarks. Section 3 presents the framework architecture. Section 4 describes the evaluation dataset. Section 5 details the experimental setup, and Section 6 reports the empirical results. Section 7 discusses the findings, threats to validity, and limitations, before Section 8 concludes the paper.

2 Related Work

2.1 Infrastructure as Code Security Analysis

The security implications of IaC have attracted substantial research attention since the recognition that misconfigurations in IaC scripts constitute a primary attack vector in cloud environments. Rahman et al. [20] extended the foundational study of IaC security smells through a replication study across Ansible and Chef scripts, confirming that recurring vulnerability patterns hard-coded secrets, missing encryption, overly permissive access policies persist across configuration management tools beyond the original Puppet-focused corpus [21]. Verdet et al. [16] conducted the most comprehensive empirical study of Terraform security policy adoption to date, analyzing 812 open-source GitHub projects across AWS, Azure, and Google Cloud using Checkov and tfsec. Their study found encryption at rest to be the most neglected security policy across all three cloud providers a finding directly corroborated by the present work, in which SC-28 (Protection of Information at Rest) constitutes 64.3% of all detected violations but their approach measures developer adoption rates rather than building an automated compliance analysis system, with no compliance-framework alignment, no LLM enrichment, and no quantitative detection metrics. War et al. [22] extended this line of inquiry with a large-scale census of IaC vulnerability types and responsible parties, and a subsequent study [23] extends the seven-sins taxonomy of Rahman et al. [21] to a 62-category taxonomy spanning seven IaC tools; neither offers compliance-oriented output or automated remediation. Among static analysis tools, Checkov has emerged as the de facto industrial baseline against which new IaC security tooling is compared [24], but its rules are not aligned to a formal regulatory control framework and it provides no contextual explanation or remediation beyond a static rule description; Section 6 reports an empirical comparison against Checkov on the same evaluation dataset. Beyond rule-based static analysis, Vo et al. [25] recently showed that an LLM used directly for Terraform code-smell detection can outperform static linters on smell breadth, reinforcing the observation central to the motivation of this paper that LLMs capture patterns static rules miss, although their system has no deterministic enforcement layer, no compliance-framework alignment, and no audit-oriented output.

2.2 Policy-as-Code Enforcement for Compliance Automation

Opdebeeck et al. [7] provided a comprehensive review of Policy-as-Code approaches for cloud governance, establishing OPA as the dominant open-source engine for this purpose. This picture is corroborated by Foale et al. [26], who conducted the first

large-scale study of PaC adoption in open-source software across 399 GitHub repositories and nine PaC tools, finding that security and compliance are the dominant real-world use case and that adoption centres on a small set of policy engines, OPA foremost among them; their study, however, characterises how PaC tools are used in practice rather than proposing a compliance-analysis system. Mohammed [15] presented the most directly relevant PaC work, validating OPA as a reliable mechanism for enforcing NIST SP 800-53, HIPAA, and PCI-DSS compliance controls in financial and healthcare cloud environments, but presents a conceptual framework without empirical evaluation against a labeled dataset and produces machine-readable outputs that are not interpretable by non-technical stakeholders limitations the present work addresses through an empirically evaluated system with LLM-driven enrichment. Rajapakse et al. [27] systematically reviewed challenges in DevSecOps adoption, establishing the shift-left principle central to the design of the framework presented here.

2.3 LLM-Based Approaches to IaC Security

Toprani and Madiseti [10] represent the most closely related LLM-based IaC security system prior to this work: a multi-agent, Retrieval-Augmented-Generation architecture that detects context-specific misconfigurations in AWS CloudFormation templates at an 85% detection rate, but is constrained to CloudFormation, incorporates no deterministic PaC enforcement, and was evaluated against only 10 manually crafted templates. The present work addresses these limitations by combining LLM-driven reasoning with deterministic OPA-based enforcement, aligning detection output to a formal compliance framework, and evaluating against a labeled dataset more than an order of magnitude larger. Firouzi et al. [11] empirically showed that up to 75% of LLM-generated Terraform configurations contain security misconfigurations and explicitly recommended hybrid architectures combining LLM reasoning with rule-based enforcement the recommendation this paper operationalises. This finding is reinforced by Vargas et al. [28], who benchmarked seven LLMs and small language models on Terraform generation using a dual-scanner pipeline and found that syntactic validity and security compliance are largely orthogonal properties, independently corroborating the syntax-versus-compliance gap discussed in Section 6. GenKubeSec [29] extended LLM-based security analysis to Kubernetes misconfiguration detection, but does not pursue compliance-framework alignment. ARPaCCino [17] combines LLMs with OPA-based Rego enforcement but is directed at automated policy rule *generation* rather than compliance *analysis* of existing configurations, producing no compliance-mapped, audit-ready output. Tony et al. [30] investigated prompting techniques for secure code generation, informing the prompt design adopted in the LLM enrichment layer here. Most recently, Alsaid et al. [31] introduced a five-layer oracle framework, TerraProbe, for evaluating whether LLM-generated Terraform security repairs are genuinely valid rather than merely appearing successful under a single static-analysis check; applying it to 288 repairs from three LLMs, they found that targeted-finding removal reached 83.3% for the best model, yet full-scanner cleanliness dropped to 10.4% and successful `terraform plan` generation to 39.6%. This gap between apparent and genuine remediation success is directly corroborated by a supplementary

syntax validation reported in Section 6, in which an initial set of LLM-generated remediation snippets exhibited comparably low syntactic validity before the enrichment prompt was constrained.

2.4 IaC Benchmarks and Evaluation Datasets

The IaC-Eval benchmark [18] provides 458 human-curated Terraform scenarios spanning a wide range of AWS services, originally developed for evaluating LLM-based IaC code generation. Subsequent work including MACOG [32] and TerraFormer [33] has used IaC-Eval exclusively for generation evaluation. To the best of the authors’ knowledge, the present work represents the first application of IaC-Eval as a source of Terraform configurations for security compliance evaluation, repurposing its human-curated scenarios from a generation benchmark into a labeled security assessment testbed. Beyond IaC-Eval, TerraGoat [34] is a deliberately misconfigured Terraform repository designed as a sandbox for static scanners, but provides no ground-truth labels or compliant counterparts suitable for precision/recall computation. The benchmark landscape has also begun to move beyond Terraform’s declarative model: Tarasova et al. [35] introduced SWE-InfraBench, evaluating LLMs on incremental code edits to AWS CDK repositories (an imperative IaC paradigm), finding that even strong coding models succeed on well under half of realistic infrastructure-editing tasks; this targets code-editing correctness rather than security-compliance labelling and is complementary to the dataset constructed in Section 4.

2.5 Runtime Drift and the Static-Analysis Gap

A recurring limitation of static IaC analysis, including the framework proposed here, is that it inspects intended-state configurations rather than the deployed runtime state of cloud resources. Once provisioned, infrastructure can diverge from its IaC definition through manual changes, automated remediation, or provider-side mutations a phenomenon commonly termed configuration drift. Industry tooling such as AWS Config [36] addresses this gap by continuously evaluating live resource state, and recent academic work has begun to characterise drift empirically [37] and to explore LLM-agent-based drift detection [38]. Bridging static IaC analysis with runtime drift detection is identified as a direction for future work in Section 7.

2.6 Summary of Research Gaps

Table 1 summarises the gap analysis across the studies discussed above, and Table 2 lists systems that are topically adjacent but outside the direct comparison scope of this paper (generation benchmarks, imperative-IaC benchmarks, and runtime-drift or policy-authoring systems that do not perform compliance analysis of existing configurations). No existing peer-reviewed system simultaneously addresses four dimensions central to practical IaC compliance analysis: Terraform support, formal compliance-framework alignment, LLM-powered explainability, and empirically reported precision, recall, and F1 metrics evaluated against a labeled dataset. The case for unifying these four dimensions is grounded in an explicit recommendation from Firouzi et

Table 1 Summary and Comparison of Related Works

Study	Focus	IaC	Framework		Hybrid	Audit	Metrics
[10]	Misconfig. detection	CloudFormation	None		No	No	Detection rate
[11]	LLM reliability	Terraform	None		No	No	No
[15]	Conceptual PaC compliance	Multi-cloud	NIST SP 800-53		No	No	No
[16]	Adoption study	Terraform	CIS marks	Bench-	No	No	No
[29]	K8s misconfig. analysis	Kubernetes	None		No	Partial	No
[17]	PaC rule generation	Terraform	None		Yes	No	No
[31]	LLM repair validity	Terraform	None		No	No	Layered-oracle
This Work	Compliance analysis	Terraform*	NIST 800-53*	SP	Yes	Yes	Yes

*Architecture is independent of the IaC format and compliance framework and can be extended through Rego policies.

Table 2 Related Systems Outside the Direct Comparison Scope

Study	Problem Addressed	Relation to This Work
[32]	Multi-agent Terraform code generation	Uses IaC-Eval as a generation benchmark, not for compliance evaluation
[33]	Verifier-guided IaC generation	Uses IaC-Eval as a generation benchmark, not for compliance evaluation
[35]	Imperative (AWS CDK) code-editing benchmark	Different IaC paradigm and task; targets editing correctness, not compliance
[38]	Runtime configuration drift detection	Addresses the runtime side of the static/runtime divide (Section 2.5)
[39]	Natural-language-to-Rego policy generation	Shares this work’s OPA/Rego substrate but addresses policy authoring, not compliance analysis

al. [11], whose empirical study concluded by recommending hybrid architectures combining LLM reasoning with deterministic enforcement; the framework presented here operationalises that recommendation and evaluates it against a labeled dataset, closing the loop between the recommendation and its empirical realisation. Each of the four capabilities also addresses a distinct stakeholder requirement that no individual approach satisfies alone: deterministic OPA-based enforcement provides repeatable, low-false-positive detection for security engineers, but its machine-readable output is inaccessible to non-technical auditors; LLM-driven enrichment supplies the plain-language explanation and remediation guidance developers and compliance officers depend on, but cannot be trusted as a standalone enforcement mechanism [11]; formal compliance mapping converts raw detection output into defensible audit evidence suitable for FedRAMP and FISMA; and empirical precision/recall metrics provide the quantitative basis for production-adoption decisions.

Table 3 Summary of the Five-Component Pipeline

Component	Input	Function	Output
1. Ingestion & Normalization	Raw Terraform <code>.tf</code> file (HCL)	Parse HCL into a Python dictionary, flatten the list-of-dictionaries representation into an OPA-compatible flat resource map, and select the policy packages relevant to the resource types present.	Normalized JSON object and a set of relevant Rego packages
2. Policy Engine	Normalized JSON + Rego package set	Invoke OPA against each relevant package, collect structured violation records, and deduplicate using a composite (rule, resource) key.	Deduplicated list of violation objects
3. Security Policy Library	Normalized JSON (loaded by OPA)	Express NIST SP 800-53 controls as Rego predicates that trigger only on explicit misconfiguration signals or verifiably absent security blocks.	Boolean decisions consumed by Component 2
4. LLM Enrichment Engine	Unique (rule, control) pairs	Generate a plain-language explanation, an official control mapping, and a corrected IaC snippet for each unique violation type; cache and broadcast to all matching violations.	Enriched violation records
5. Audit Report Generator	Enriched violation records	Aggregate by file, control family, and rule; render four structured sections in Markdown suitable for GRC workflows.	Audit-ready Markdown report

3 Framework Architecture

The proposed open-agent framework is designed as a five-component pipeline in which each component receives structured input from the preceding stage and produces structured output consumed by the next. As illustrated in Figure 1, the pipeline transforms raw IaC configurations into structured, audit-ready compliance evidence through five sequential stages. Table 3 summarises each component’s input, function, and output for ease of reference; the remainder of this section describes each component in detail. The framework is not an autonomous agent in the sense of LLM-based planning systems; rather, it is a hybrid enforcement pipeline in which each component performs a well-defined, deterministic role, a design choice that is intentional, as predictability and repeatability are requirements in compliance-critical workflows. The complete implementation of the framework and supporting artifacts are available as an open-source repository to facilitate reproducibility and future research [40].

3.1 Component 1: IaC Configuration Ingestion and Normalization

The framework accepts Terraform configuration files as input. Each file is parsed from its native HCL syntax into a structured Python dictionary using the `python-hcl2` library, which converts the declarative resource definitions into a nested data structure. A normalization step is subsequently applied to transform the raw `python-hcl2` output a list-of-dictionaries representation into a flat resource map compatible with OPA’s query interface, mapping the resource key to a dictionary of resource types, each containing a dictionary of named resource instances with their attribute values. The list-of-dictionaries structure that `python-hcl2` returns natively would require every

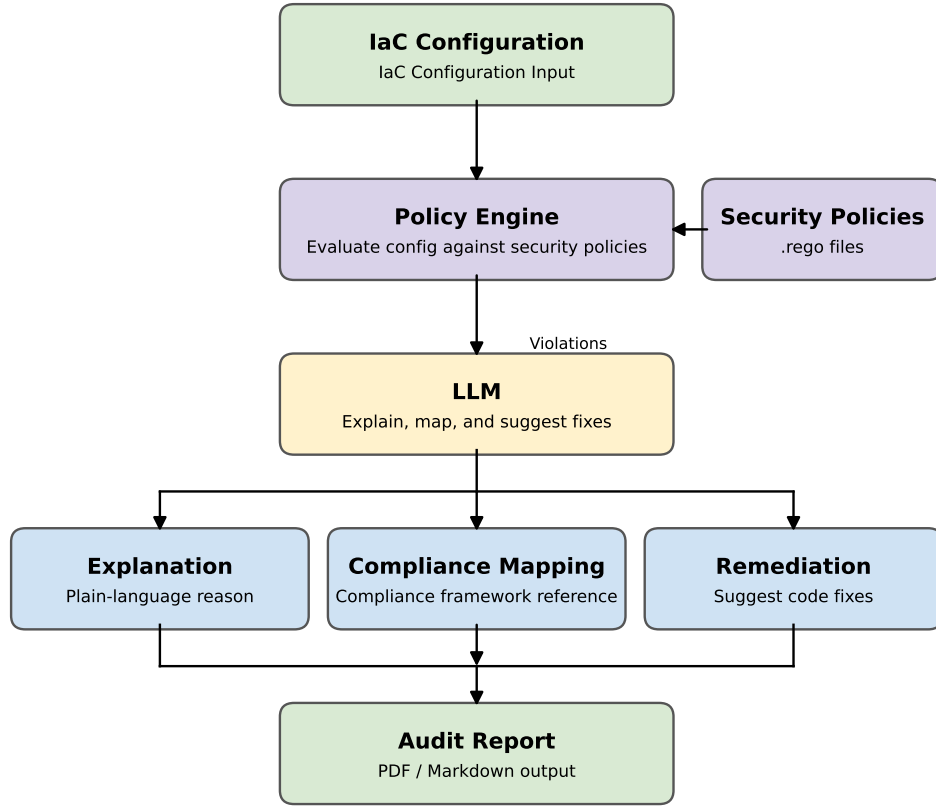


Fig. 1 System architecture of the proposed framework.

Rego rule to search across a list to find a given resource; the flat **resource** $\rightarrow$ **type** $\rightarrow$ **name** structure lets a rule address any declared resource directly by name, keeping the Rego predicates described in Section 3.3 simple, single-purpose lookups rather than list-traversal logic.

A resource-aware filtering step is applied prior to OPA evaluation. The pipeline inspects the resource types present in each normalized JSON object and constructs a set of relevant policy packages based on the resource-to-policy mapping in Table 4.

Files containing only resource types outside the scope of the implemented policy library such as DNS zones, VPC networking components, or serverless functions are excluded from evaluation. This design minimizes false positive detections by ensuring that policy rules are applied only to configurations containing the specific resource types they are designed to assess; a configuration file is evaluated against a given policy package only if it declares at least one resource type listed in that package’s row of Table 4.

Table 4 Resource-to-Policy-Package Routing

Terraform Resource Type	Routed Packages	Rules
aws_s3_bucket	terraform.sc, .au, .s3	3
aws_db_instance	terraform.sc	1
aws_ebs_volume	terraform.sc	2
aws_lb_listener	terraform.sc	2
aws_iam_role_policy	terraform.ac	1
aws_security_group	terraform.ac	2
aws_cloudtrail	terraform.au	4
aws_dynamodb_table	terraform.sc	1

3.2 Component 2: Policy Engine

The policy engine uses Open Policy Agent version 1.16.1 as the deterministic enforcement layer. OPA evaluates the normalized JSON representation of each Terraform configuration against a library of Rego policies written in Rego v1 syntax, using the `import rego.v1` directive and the `violation contains result if` construct. OPA is invoked programmatically via subprocess calls to the OPA binary, with the evaluation query targeting each relevant policy package independently; the policy engine does not blindly apply every rule to every input, but selects only the packages whose target resources appear in the configuration, which is the principal source of the framework’s high precision by construction, a rule designed for one resource type cannot generate a violation against an unrelated resource type. The structured violation output from each policy package is parsed from OPA’s JSON response format and deduplicated using a composite key of rule identifier and resource name, since two distinct rules within the same package can sometimes fire on the same resource for overlapping reasons (for example, an EBS volume that triggers both `ebs_encryption_disabled` and `ebs_missing_encryption`).

3.3 Component 3: Security Policy Library

The security policy library comprises four Rego policy files covering the three NIST SP 800-53 control families selected for this study, spanning sixteen rules across eight AWS resource types. Table 5 presents the complete policy library with the corresponding NIST control mappings, target resource types, and severity levels.

The Access Control policy file targets IAM role policies with wildcard permissions (AC-6) and security groups or S3 buckets with unrestricted or unblocked public access

Table 5 Security Policy Library NIST SP 800-53 Control Mappings

Rule Identifier	NIST	Family	Target Resource	Severity
s3_missing_encryption	SC-28	SC	aws_s3_bucket	HIGH
rds_encryption_disabled	SC-28	SC	aws_db_instance	HIGH
ebs_encryption_disabled	SC-28	SC	aws_ebs_volume	HIGH
ebs_missing_encryption	SC-28	SC	aws_ebs_volume	HIGH
dynamodb_encryption_disabled	SC-28	SC	aws_dynamodb_table	HIGH
lb_not_using_https	SC-8	SC	aws_lb_listener	MEDIUM
lb_protocol_missing	SC-8	SC	aws_lb_listener	MEDIUM
wildcard_iam_policy	AC-6	AC	aws_iam_role_policy	HIGH
unrestricted_ingress	AC-3	AC	aws_security_group	HIGH
unrestricted_ingress_all_protocols	AC-3	AC	aws_security_group	HIGH
s3_public_access_not_blocked	AC-3	AC	aws_s3_bucket	HIGH
cloudtrail_logging_disabled	AU-2	AU	aws_cloudtrail	HIGH
cloudtrail_not_global	AU-2	AU	aws_cloudtrail	MEDIUM
cloudtrail_no_log_validation	AU-9	AU	aws_cloudtrail	MEDIUM
cloudtrail_validation_field_missing	AU-9	AU	aws_cloudtrail	MEDIUM
s3_logging_disabled	AU-12	AU	aws_s3_bucket	MEDIUM

(AC-3). The System and Communications Protection policy file targets S3, RDS, EBS, and DynamoDB encryption (SC-28) and load-balancer HTTPS/protocol enforcement (SC-8). The Audit and Accountability policy file targets CloudTrail logging configuration (AU-2, AU-9) and S3 access logging (AU-12). Each rule follows one of two evaluation patterns: a positive existence check on the target resource, and a negative existence check on a sibling resource whose presence would render the configuration compliant, designed to minimize false positives by triggering only on explicit misconfiguration signals or verifiably absent security configuration blocks. All sixteen rules were triggered at least once against the evaluation dataset described in Section 4, confirming that the policy library’s full coverage is exercised by the constructed dataset rather than containing dormant, untested rules.

3.4 Component 4: LLM Enrichment Engine

The LLM enrichment layer receives the structured violation output from the OPA policy engine and generates contextual enrichment for each unique violation type. A deduplication strategy is employed prior to API invocation: rather than submitting one API call per detected violation, the enrichment layer first collects the set of unique rule–control pairs present in the evaluation results and submits exactly one enrichment request per unique pair. The enrichment outputs are cached and subsequently applied to all violations of the corresponding type, reducing the total number of API calls from 157 to 16 in the evaluation dataset an 89.8% reduction.

Each enrichment prompt provides the LLM with the violation rule identifier, control reference, severity level, and violation message. The model is instructed to return a JSON object containing exactly three fields: **explanation**, a two-to-three sentence plain-English explanation of the security risk; **nist_mapping**, a nested object containing the official control identifier, control name, and a one-sentence control description; and **remediation**, a corrected IaC code snippet (Terraform HCL) that addresses the specific violation. The prompt further constrains the remediation field to pure

HCL syntax, explicitly prohibiting markdown code fences, explanatory prose, and semicolon-delimited statements, and requiring a complete resource block rather than an isolated changed argument.

To eliminate the risk of LLM recall errors in the `nist_mapping` field a risk empirically confirmed during development, where the model produced the incorrect control name “Access Control for Remote Access Sessions” for AC-3 instead of its official name, “Access Enforcement” the control identifier, control name, and control description fields are deterministically assigned from a hardcoded lookup table populated from the official NIST SP 800-53 Rev. 5 control catalog, keyed on the control identifier already known from the Rego policy library, rather than left to LLM generation. The LLM’s generative role is thereby scoped to the two fields for which no fixed ground-truth answer exists: `explanation` and `remediation`.

The enrichment layer communicates with the Cloudflare Workers AI REST API using the `@cf/meta/llama-3.3-70b-instruct-fp8-fast` model via direct HTTP requests. A two-stage JSON extraction routine is applied as a post-processing step, since the endpoint returns the model output as a nested dictionary within the `result` field rather than a plain string. A retry mechanism with linear backoff (30, 60, and 90 seconds on successive attempts) handles rate-limit responses.

3.5 Component 5: Audit Report Generator

The audit report generator assembles all enriched violation records into a structured Markdown document organized for direct use in GRC workflows, structured into four sections: (1) an executive summary presenting aggregate statistics including total files scanned, total violations detected, and severity distribution; (2) a violations-by-NIST-control table; (3) a violations-by-rule table; and (4) a detailed findings section presenting each file with violations as a subsection, with each violation documented by its severity, resource identifier, NIST control, violation message, LLM-generated explanation, compliance mapping, and remediation snippet. This structure mirrors the way GRC reporting is typically consumed in practice: the executive summary serves leadership and auditors who require aggregate posture at a glance, the by-control and by-rule breakdowns serve compliance officers, and the detailed findings section serves security engineers and developers who remediate individual violations.

3.6 Design Principles

The framework is designed around three core principles. *First, hybrid enforcement:* the deterministic OPA layer provides the precision and repeatability guarantees essential for security-critical applications, while the LLM layer provides the contextual reasoning and natural language generation that deterministic tools cannot produce; neither layer alone is sufficient. *Second, modularity and instantiation-agnostic design:* each pipeline component is independently configurable and replaceable the IaC ingestion layer can be adapted to alternative IaC formats, the Rego policy library can be extended with additional control families or replaced with policies aligned to CIS Benchmarks, ISO/IEC 27001, or PCI-DSS, and the LLM provider can be substituted without modifying the pipeline logic. *Third, audit traceability:* every output element,

from violation detection to control mapping to remediation snippet, is explicitly linked to a specific IaC resource, a specific Rego rule, and a specific control identifier.

4 Evaluation Dataset

The dataset was designed to satisfy three criteria: (i) *realism* configurations must represent authentic, production-representative IaC infrastructure definitions; (ii) *labeled ground truth* each configuration must carry an explicit compliance label; and (iii) *scope alignment* configurations must contain resource types within the scope of the implemented policy library.

4.1 Source: IaC-Eval Benchmark

The primary dataset source is the IaC-Eval benchmark [18], a publicly available collection of 458 human-curated Terraform configuration scenarios spanning a wide range of AWS services. Three properties make it well-suited as the foundation for a labelled compliance testbed: the reference configurations are human-authored rather than synthetically generated; the benchmark spans a broad surface of AWS services rather than a single resource type; and the reference solutions are intended to be functionally correct rather than security-hardened, meaning a non-trivial proportion contain authentic, naturally-occurring misconfigurations. To the best of the authors’ knowledge, this work represents the first application of IaC-Eval as a source of Terraform configurations for security compliance evaluation; all prior work using IaC-Eval, including MACOG [32] and TerraFormer [33], has applied it exclusively as a code-generation benchmark.

4.2 Scope-Controlled Filtering

Of the 458 IaC-Eval scenarios, 249 define infrastructure resources outside the scope of the implemented policy library (DNS zones, VPC networking components, serverless functions, notification services, and similar). Evaluating these would produce trivially correct true-negative results that artificially inflate specificity without contributing meaningful compliance signal, so, consistent with the evaluation methodology of established tools such as Checkov and tfsec, these 249 configurations were excluded from quantitative metric computation. The remaining 209 configurations contain at least one in-scope AWS resource type and form the evaluation-eligible subset. Figure 2 summarises the construction of the final evaluation dataset, and Table 6 reports the distribution of in-scope resource types across the final 231-file dataset (the total exceeds 231 because individual configurations may declare multiple in-scope resource types).

4.3 Ground Truth Labeling

Ground truth labels were assigned through two distinct procedures. The 22 injected configurations (Section 4.4) were authored with known violation types and carry exact ground truth by construction. The 209 evaluation-eligible IaC-Eval scenarios were labelled through an automated heuristic procedure implemented as a lightweight

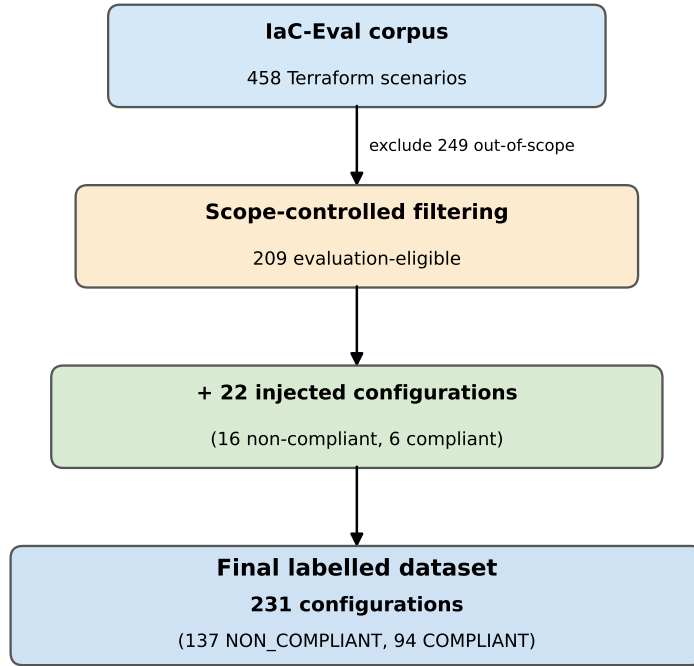


Fig. 2 Construction of the evaluation dataset.

Table 6 Distribution of In-Scope AWS Resource Types Across the 231-File Dataset

Resource Type	Configurations	% of Dataset
aws_s3_bucket	92	39.8%
aws_security_group	83	35.9%
aws_db_instance	60	26.0%
aws_dynamodb_table	24	10.4%
aws_lb_listener	23	10.0%
aws_iam_role_policy	14	6.1%
aws_cloudtrail	4	1.7%
aws_ebs_volume	3	1.3%

substring-based classifier over the raw Terraform source text, checking for: missing encryption (absence of the substring `encrypted` in resources whose type indicates an encryptable storage primitive); missing logging; wildcard permissions (literal wildcard tokens); missing public-access blocks on S3 buckets; and unrestricted security-group ingress (the CIDR block `0.0.0.0/0` specifically within an `ingress{}` sub-block, so that an open egress rule is not mistaken for an open inbound rule). A file is labelled `NON_COMPLIANT` if any pattern is matched and `COMPLIANT` otherwise. Table 7 illustrates the procedure with a matched pair of S3 bucket configurations.

The unrestricted-ingress indicator was added after cross-validating the heuristic’s labels against the OPA detection layer’s own output revealed that files with fully-open inbound security-group rules were being mislabelled `COMPLIANT`; this is treated as a genuine correction rather than a limitation, since it brings the heuristic’s coverage closer into line with what the Rego policy library evaluates. A related, structurally similar gap was identified but not corrected: the encryption-keyword check for RDS instances scans the entire file rather than the specific `aws_db_instance` resource block, meaning an unrelated resource that happens to mention “encrypted” can cause a genuinely unencrypted RDS instance to be labelled `COMPLIANT`. This heuristic procedure is deliberately lightweight and approximates, rather than perfectly captures, the structural definitions evaluated by the Rego policy engine; it cannot distinguish between encryption configured inline, through a companion sibling resource, or through a Terraform module, since all three patterns produce a `COMPLIANT` label provided the keyword appears anywhere in the file. The implications of this approximation for the validity of the reported precision and recall estimates are discussed in Section 7.

Table 7 Matched Pair of S3 Bucket Configurations Illustrating the Labelling Procedure

Configuration	Label and Reasoning
S3 bucket with a companion server-side encryption resource	<code>COMPLIANT</code> . The companion resource satisfies the SC-28 encryption requirement; no labelling indicator is triggered.
S3 bucket only, no inline or companion encryption resource	<code>NON_COMPLIANT</code> . The missing-encryption indicator triggers, mapping to SC-28 (Protection of Information at Rest).

4.4 Injected Configurations for Coverage

The IaC-Eval subset alone does not exercise every rule in the policy library, so 22 additional Terraform configurations were injected: 16 non-compliant configurations spanning all three implemented NIST control families, and 6 compliant configurations serving as negative samples, following the same approach used by Toprani and Madisetti [10]. Table 8 reports the distribution by violation type; each injected configuration is small (5–20 lines of HCL) and contains exactly one violation type. All ground-truth

labels are published in the accompanying `ground_truth.csv` file in the open-source repository [40].

Table 8 Distribution of Injected Configurations by Violation Type and NIST Control

Violation Type	NIST Control	Count
S3 missing encryption	SC-28	2
RDS encryption disabled	SC-28	1
EBS encryption disabled	SC-28	1
EBS missing encryption	SC-28	1
DynamoDB encryption disabled	SC-28	1
Load balancer not using HTTPS	SC-8	1
Wildcard IAM policy	AC-6	2
Unrestricted ingress (TCP)	AC-3	1
Unrestricted ingress (all protocols)	AC-3	1
S3 public access not blocked	AC-3	1
CloudTrail logging disabled	AU-2	1
CloudTrail not global	AU-2	1
CloudTrail no log validation	AU-9	1
S3 logging disabled	AU-12	1
Non-compliant subtotal		16
Compliant baseline (S3, RDS, IAM, Security Group, DynamoDB)	—	6
Total injected		22

4.5 Dataset Statistics

Table 9 presents the final composition of the evaluation dataset: 231 labeled Terraform configurations, of which 137 (59.3%) are `NON_COMPLIANT` and 94 (40.7%) are `COMPLIANT`. To the best of the authors’ knowledge, this is the largest labeled Terraform compliance evaluation dataset reported in the peer-reviewed literature, a 23-fold increase over the 10-template corpus used by Toprani and Madiseti [10]. Table 10 reports the distribution of ground-truth violations across the three control families; a single `NON_COMPLIANT` file may carry violations across more than one family, so the row counts sum to more than 137.

4.6 Threats to Validity

Construct validity. The dataset is constructed to evaluate compliance detection, not LLM code generation. Although the underlying IaC-Eval scenarios were originally written as code-generation tasks, the labelling procedure treats each reference configuration as a static infrastructure definition whose compliance posture is determined by the resource declarations it contains; this repurposing is methodologically sound because the reference configurations are independent artefacts whose security properties can be evaluated without reference to the natural-language prompts that originally produced them.

Internal validity. The ground-truth labelling procedure for the IaC-Eval subset is the principal source of internal validity risk. The substring-based detection of

Table 9 Evaluation Dataset Composition

Category	Count	%
Total IaC-Eval scenarios	458	100%
Excluded (out-of-scope)	249	54.4%
Evaluation-eligible (IaC-Eval)	209	45.6%
Injected non-compliant	16	–
Injected compliant	6	–
Total evaluation dataset	231	–
NON_COMPLIANT	137	59.3%
COMPLIANT	94	40.7%

Table 10 Distribution of Ground-Truth Violations Across NIST SP 800-53 Control Families

Family	Family Name	NON_COMPLIANT Files
SC	System and Communications Protection	111
AC	Access Control	91
AU	Audit and Accountability	108

missing encryption matches the keyword **encrypted** anywhere in the source file, so configurations that mention the keyword in comments or unrelated arguments can be mislabelled **COMPLIANT** even when genuinely non-compliant an effect concretely observed for **aws_db_instance** resources during development. The substring-based resource-type relevance check also keys against fragments of the Terraform identifier, so resources whose canonical name does not contain the searched fragment (most notably **aws_db_instance**, which represents RDS but does not contain the substring **rds**) are not exercised by the corresponding labelling rule. The net effect of these noise sources is asymmetric in direction: heuristic under-coverage produces files labelled **COMPLIANT** that contain genuine violations, depressing the precision estimate; heuristic over-coverage produces files labelled **NON_COMPLIANT** with no detectable violation, depressing the recall estimate. No claim is made that these effects cancel; the reported precision of 96.9% and recall of 90.5% (Section 6) should be treated as point estimates whose true confidence bounds are wider than the values alone suggest. A small-scale manual audit by a second annotator, with inter-annotator agreement reported through Cohen’s κ , is identified as a priority direction for future work.

External validity. The dataset is drawn from a single benchmark source and is restricted to AWS Terraform configurations. Generalisation to other cloud providers or IaC technologies is supported in principle by the framework’s architecture but not empirically demonstrated here. Despite these limitations, the dataset improves substantially on prior labelled evaluation corpora in the LLM-based IaC security literature

and is the first to combine a published benchmark source, an explicit labelling procedure, control-mapped ground truth, and public release of labels and configurations together, making it reusable as a benchmark independent of the specific framework proposed in this paper.

5 Experimental Setup

5.1 Implementation Environment

The framework was implemented in Python 3.13 on a Dell XPS 14 (9440) workstation running Windows 11 (64-bit), equipped with an Intel Core Ultra 7 155H processor and 32 GB of RAM. Although the LLM enrichment layer is invoked through a cloud-hosted inference endpoint and therefore imposes no local GPU requirement, the workstation specification is reported to support reproducibility of the local pipeline stages. Table 11 lists the principal software dependencies; all versions correspond to the exact configuration under which the experimental results were obtained. The complete implementation, including the Terraform parsing pipeline, Rego policy library, evaluation scripts, and labelled dataset, is publicly available [40].

Table 11 Principal Software Dependencies Used in the Implementation

Component	Version	Role
Python	3.13	Pipeline implementation language
python-hcl2	4.3.2	HCL-to-Python parsing of Terraform files
Open Policy Agent	1.16.1	Deterministic Rego policy evaluation
requests	2.32.3	HTTP client for Cloudflare Workers AI API
Checkov	3.3.6	Industrial baseline static analysis tool
Llama 3.3 70B Instruct (FP8)	—	LLM used for violation enrichment

5.2 Policy Configuration

The Rego policy library comprises four policy files covering 16 violation rules across three NIST SP 800-53 control families (Table 5). All policies use OPA’s Rego v1 syntax with the `import rego.v1` directive. Each policy file is assigned to a dedicated OPA package namespace: `terraform.ac`, `terraform.sc`, `terraform.au`, and `terraform.s3` for S3-specific rules spanning multiple control families. At evaluation time, the policy engine inspects the resource types present in each file and selects only the relevant packages using the mapping in Table 4; configurations declaring only out-of-scope resource types are skipped without invoking OPA at all. All 16 implemented rules were triggered at least once against the evaluation dataset.

5.3 LLM Configuration

The LLM enrichment layer was configured to balance output consistency, response completeness, and API cost. Llama 3.3 70B Instruct (FP8 quantization) was selected because it sits at a useful operating point on the open-weights frontier: large enough to produce well-structured technical JSON, small enough for acceptable latency through a hosted endpoint, and accessible without institutional licensing constraints. The model is invoked at a temperature of 0.1 with a maximum token limit of 1,024; across the sixteen unique violation types enriched, no response was truncated by the token limit. As described in Section 3.4, the `nist_mapping` field is deterministically assigned from an authoritative lookup table rather than generated, after an empirical observation that the model produced an incorrect control name for AC-3 when prompted to recall it from memory; this restriction was verified by direct comparison against the official NIST SP 800-53 Rev. 5 control catalog for all sixteen implemented rules. The 157 raw violations detected in the evaluation dataset were reduced to 16 API calls via caching on the composite key (rule id, NIST control).

5.4 Evaluation Metrics

Framework performance is evaluated using four standard binary classification metrics computed at the file level, since compliance workflows operate on deployable configuration units and a single non-compliant file can introduce multiple violations into a production environment simultaneously. A file is classified as a True Positive (TP) if its ground-truth label is `NON_COMPLIANT` and the OPA engine detects at least one violation; as a False Negative (FN) if its label is `NON_COMPLIANT` and no violations are detected; as a False Positive (FP) if its label is `COMPLIANT` and at least one violation is detected; and as a True Negative (TN) if its label is `COMPLIANT` and no violations are detected.

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN} \quad (1)$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \quad \text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (2)$$

Precision and recall are reported in preference to accuracy as the primary indicators, because the evaluation dataset is class-imbalanced (59.3% `NON_COMPLIANT`) and the cost profile of compliance enforcement is asymmetric: a false positive imposes a remediation-triage burden on a developer, while a false negative allows a non-compliant configuration to reach production. LLM enrichment quality is assessed through control-mapping accuracy, defined as the proportion of enriched violation types for which the reported control name correctly identifies the official NIST SP 800-53 control corresponding to the associated identifier. The statistical significance of the performance difference between the proposed framework and the Checkov baseline is additionally assessed using McNemar’s exact test for paired binary classification, applied to the per-file correctness outcomes of the two systems on the identical evaluation dataset, which evaluates whether the observed difference is distinguishable from chance using only the subset of files on which the two systems disagree.

Table 12 OPA Policy Engine Detection
Performance at File-Level Granularity (with
95% Wilson score confidence intervals)

Metric	Value (95% CI)
True Positives (TP)	124
False Negatives (FN)	13
False Positives (FP)	4
True Negatives (TN)	90
Precision	96.9% [92.2, 98.8]
Recall	90.5% [84.4, 94.4]
F1 Score	93.6%
Accuracy	92.6% [88.5, 95.4]

Confidence intervals computed using the Wilson score method ($z = 1.96$) on the underlying TP/FP/FN/TN counts.

5.4.1 Baseline Comparison Setup

Checkov, the most widely adopted open-source static analysis tool for IaC [24], was selected as the comparison baseline because it is the de facto baseline against which new IaC security tooling is compared in recent literature [16, 23], its rule catalog covers the same AWS resource types targeted by the Rego policy library, and it produces structured, machine-readable output suitable for automated metric computation. Checkov (version 3.3.6) was executed against the identical 231-file evaluation dataset with no modifications to its default rule catalog and no custom rules added, in an isolated Python environment to avoid a dependency conflict with the version of `python-hcl2` pinned for the proposed framework. A file with at least one failed Checkov check is treated as Checkov-flagged; the same TP/FN/FP/TN accounting rules described above are then applied to compute Precision, Recall, F1, and Accuracy for Checkov on the identical dataset.

6 Results and Analysis

6.1 OPA Policy Engine Detection Performance

Table 12 presents the confusion matrix and derived performance metrics, with 95% Wilson score confidence intervals, for the OPA policy enforcement layer evaluated against the 231-file labelled dataset. The framework correctly identified violations in 124 of the 137 `NON_COMPLIANT` files ($TP = 124$), missed violations in 13 `NON_COMPLIANT` files ($FN = 13$), incorrectly flagged 4 `COMPLIANT` files as containing violations ($FP = 4$), and correctly identified the remaining 90 `COMPLIANT` files as compliant ($TN = 90$).

Precision analysis. The precision of 96.9% indicates that the OPA policy engine rarely flags compliant configurations as containing violations: only 4 of 128 flagged files were `COMPLIANT` in ground truth. Inspection of the four false-positive files revealed two distinct causes. Three files involved S3 buckets whose companion server-side

encryption resources were supplied through a Terraform module or a dynamic block rather than a directly declared resource, an inherent consequence of static analysis against a flat resource map rather than a fully resolved Terraform plan. The fourth exposed a substring-matching limitation in the `wildcard_iam_policy` rule: its predicate matches any literal asterisk in the policy document, including asterisks appearing as scoped wildcard suffixes within an otherwise fully-enumerated ARN (e.g., `arn:aws:logs:*:*:*` on an otherwise specific action list), rather than only genuine least-privilege violations. This is identified as a concrete target for future refinement of the rule’s predicate (Section 7).

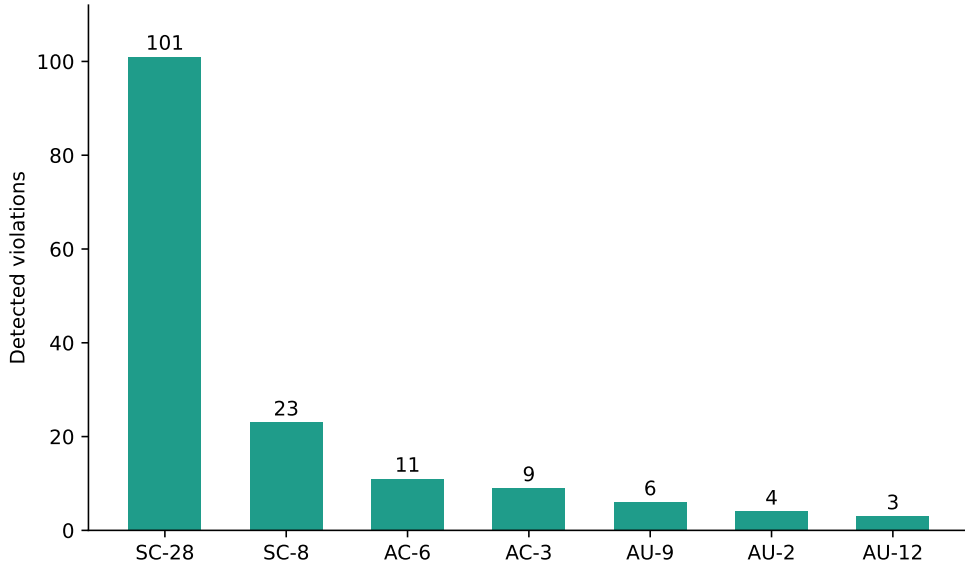
Recall analysis. The recall of 90.5% indicates the engine detected violations in 124 of 137 `NON_COMPLIANT` files, missing 13. Analysis of the false negatives confirms the dominant pattern identified during development: security controls absent by omission rather than explicitly disabled. Two missing-field companion rules were successfully added to the final policy library — `lb_protocol_missing`, which fires when a load balancer listener declares no protocol attribute at all, and `cloudtrail_validation_field_missing`, targeting CloudTrail trails that omit log-file validation each contributing additional true positives without introducing new false positives. A third candidate rule, targeting RDS instances with no `storage_encrypted` field, was implemented and evaluated but withdrawn after it introduced 55 additional false positives, traced to the ground-truth labelling heuristic’s whole-file (rather than resource-scoped) keyword matching rather than to a flaw in the missing-field detection principle itself. A fourth candidate rule, targeting CloudTrail’s `enable_logging` field, was rejected because AWS defaults CloudTrail logging to enabled when the field is absent, so a missing-field rule keyed on this attribute would invert the correct default. These three outcomes illustrate that missing-field detection is not a uniformly safe extension to a conservative, explicit-signal policy library: its safety depends on whether the underlying cloud-provider default is genuinely insecure, and whether the ground truth against which the rule is evaluated is itself scoped precisely enough.

6.2 Violation Distribution Analysis

Table 13 and Figure 3 present the distribution of the 157 detected violations across the implemented NIST SP 800-53 controls. SC-28 (Protection of Information at Rest) accounts for 101 violations (64.3% of the total), consistent with the findings of Verdet et al. [16], who identified encryption at rest as the most neglected security practice across 812 open-source Terraform projects. SC-8 (Transmission Confidentiality and Integrity) accounts for 23 violations (14.6%), predominantly from load-balancer listeners not enforcing HTTPS. AC-6 (Least Privilege) contributes 11 violations (7.0%) from IAM policies with wildcard permissions. SC-28 towers over every other control at more than four times the combined total of the remaining six, and the bars drop off sharply after SC-8; this concentration is consistent with the resource-type distribution in Table 6, since S3 buckets are the most common in-scope resource and are declared without a companion encryption resource far more often than any other resource type is left insecure. Table 14 breaks the same 157 violations down by the specific Rego rule; `s3_missing_encryption` alone accounts for 96 of the 101 SC-28 violations and 61.1% of all detections.

Table 13 Detected Violations by NIST SP 800-53 Control

NIST	Control Name	Violations	%
SC-28	Protection of Information at Rest	101	64.3%
SC-8	Transmission Confidentiality and Integrity	23	14.6%
AC-6	Least Privilege	11	7.0%
AC-3	Access Enforcement	9	5.7%
AU-9	Protection of Audit Information	6	3.8%
AU-2	Event Logging	4	2.5%
AU-12	Audit Record Generation	3	1.9%
Total		157	100%

**Fig. 3** Detected violations by NIST SP 800-53 control.

6.3 LLM Enrichment Performance

The LLM enrichment layer processed all 16 unique violation types; every enrichment request completed successfully and no response was truncated by the configured token limit (Table 15). As described in Section 3.4, the control identifier, control name, and control description are assigned deterministically rather than generated, so control-mapping accuracy reduces to a verification that the lookup table itself is correctly populated; this was confirmed by direct comparison against the official NIST SP 800-53 Rev. 5 control catalog for all sixteen rules.

Table 14 Detected Violations by Rego Rule

Rule	NIST Control	Violations
s3_missing_encryption	SC-28	96
lb_not_using_https	SC-8	21
wildcard_iam_policy	AC-6	11
unrestricted_ingress (specific port)	AC-3	7
cloudtrail_validation_field_missing	AU-9	3
cloudtrail_no_log_validation	AU-9	3
s3_logging_disabled	AU-12	3
cloudtrail_not_global	AU-2	2
cloudtrail_logging_disabled	AU-2	2
ebs_missing_encryption	SC-28	2
lb_protocol_missing	SC-8	2
dynamodb_encryption_disabled	SC-28	1
ebs_encryption_disabled	SC-28	1
rds_encryption_disabled	SC-28	1
s3_public_access_not_blocked	AC-3	1
unrestricted_ingress_all_protocols	AC-3	1
Total		157

A supplementary syntax validation of the sixteen unique LLM-generated remediation snippets was conducted using HCL parsing (`python-hcl2`), separate from the control-mapping metric. An initial validation pass, prior to strengthening the enrichment prompt’s formatting constraints, found that only 7 of 15 then-available snippets (46.7%) were syntactically valid, with failures attributed to natural-language prose in place of executable code and non-HCL syntax artefacts. Following a revision to the prompt explicitly prohibiting markdown fences, prose, and semicolons, and requiring a complete resource block, a repeat validation of all sixteen final snippets found every remaining snippet to be syntactically valid HCL, indicating that the improvement in reliability is attributable to the explicit format constraints imposed on the prompt rather than to an inherent property of the underlying model’s unconstrained output. This finding is directly corroborated by Alsaïd et al. [31], whose independent evaluation of LLM-generated Terraform repairs found that apparent remediation success under a single static-analysis check can substantially overstate genuine validity.

Table 15 LLM Enrichment Layer Performance Summary

Metric	Value
Total raw violations detected by OPA	157
Unique (rule, control) pairs	16
API calls issued	16
API call reduction via caching	89.8%
Enrichment requests completed successfully	16 / 16
Responses truncated by token limit	0
Control mappings verified against NIST catalog	16 / 16
Remediation snippets syntactically valid (final)	16 / 16

Table 16 Performance Comparison with Related Work and Industrial Baseline

System	Dataset	Precision	Recall	F1
Toprani & Madisetti [10]	10 templates	N/A	N/A	85% [†]
Checkov 3.3.6 (baseline)	231 files	59.3%	100.0%	74.5%
This Work	231 files	96.9%	90.5%	93.6%

[†]Reported as detection rate rather than F1 score; precision and recall not reported in the original work.

6.4 Comparison with Prior Work and Industrial Baseline

Table 16 compares the detection performance of the proposed framework against both the closest related system in the LLM-based IaC security literature and the Checkov industrial baseline. The proposed framework achieves an F1 score of 93.6%, surpassing the 85% detection rate reported by Toprani and Madisetti [10], on an evaluation set more than an order of magnitude larger (231 configurations versus 10 manually-crafted templates).

Checkov 3.3.6 achieves a recall of 100.0% on the identical 231-file dataset, flagging every NON_COMPLIANT configuration at least once, but its precision is 59.3%: Checkov flagged all 231 files in the dataset (TN = 0), meaning every compliant file also triggered at least one Checkov finding. Manual inspection of the most frequently firing Checkov checks (S3 lifecycle configuration, S3 event notifications, KMS encryption by default, RDS enhanced monitoring, among others) revealed that the great majority of these findings fall outside the three NIST control families (AC, SC, AU) targeted by the proposed framework, reflecting general AWS Well-Architected best practices rather than access-control, encryption, or audit-logging violations. This is a genuine scope mismatch between the two systems rather than evidence that Checkov’s within-scope detection is unreliable, and is acknowledged as a limitation of the baseline comparison in Section 7. Within this caveat, the trade-off remains informative: the proposed framework trades 9.5 percentage points of recall (90.5% versus 100%) for a 37.6 percentage-point gain in precision (96.9% versus 59.3%), yielding a 19.1 percentage-point improvement in F1. At Checkov’s precision, 40.7% of flagged files require triage only to be dismissed as non-issues within the proposed framework’s scope of concern; at the proposed framework’s precision, only 3.1% do.

To establish whether this difference is statistically distinguishable from chance, McNemar’s exact test was applied to the paired per-file correctness outcomes of the two systems across the 231-file dataset. The proposed framework classified 90 files correctly that Checkov misclassified, while Checkov classified 13 files correctly that the proposed framework misclassified; both systems agreed on the remaining 128 files (124 both correct, 4 both incorrect). McNemar’s test yielded $\chi^2 = 13.0$, $p < 0.001$, indicating the observed difference in classification accuracy is statistically significant. Table 17 and Figure 4 present the side-by-side confusion matrices: Checkov produces 23.5× more false positives than the proposed framework (94 versus 4) while gaining only 13 additional true positives (137 versus 124). The two systems occupy genuinely different operating points on the precision–recall curve: Checkov’s behaviour, particularly its

zero true-negative count on this dataset, suits an exploratory, broad-coverage scanning context, while the proposed framework’s behaviour suits a compliance-gating context scoped specifically to NIST SP 800-53 AC, SC, and AU controls, in which every flagged finding incurs downstream remediation cost.

Table 17 Side-by-Side Confusion Matrix Comparison on the 231-File Dataset

Metric	Checkov 3.3.6	This Work
True Positives (TP)	137	124
False Negatives (FN)	0	13
False Positives (FP)	94	4
True Negatives (TN)	0	90
Precision	59.3% [52.9, 65.4]	96.9% [92.2, 98.8]
Recall	100.0% [97.3, 100.0]	90.5% [84.4, 94.4]
F1 Score	74.5%	93.6%

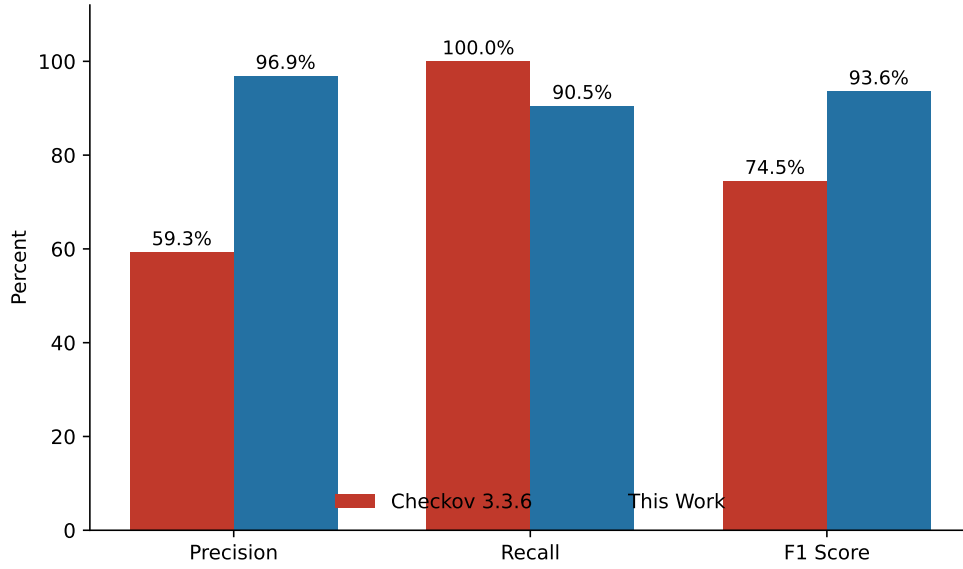


Fig. 4 Precision, recall, and F1 score of the proposed framework versus the Checkov 3.3.6 industrial baseline.

6.5 Per-Family Recall Analysis

The aggregate recall of 90.5% averages over heterogeneous detection behaviour across the three control families. The ground truth contains 111 NON_COMPLIANT files carrying at least one SC-family violation, 91 carrying at least one AC-family violation, and 108 carrying at least one AU-family violation (Table 10), indicating the three families are roughly comparably represented at the file level. The framework detected, respectively, 124 SC-family violations, 20 AC-family violations, and 13 AU-family violations. AU detection in particular remains substantially below the family’s ground-truth prevalence, consistent with the missing-field recall gap discussed above: audit-logging configurations are predominantly declared through field-presence rather than field-equality patterns, and the two AU-family missing-field rules added during development narrow but do not close this gap. Consumers of framework-generated reports should be aware that absence of detected AU findings does not equate to absence of AU non-compliance in the underlying infrastructure.

7 Discussion, Threats to Validity, and Limitations

7.1 Discussion of Key Findings

Hybrid architecture as the source of performance. The headline result of this study is that the hybrid OPA + LLM architecture achieves a file-level F1 score of 93.6% on a 231-file labelled dataset while simultaneously producing human-readable, NIST-mapped, audit-ready evidence. The 96.9% precision is the more architecturally significant of the two figures, because it confirms that the determinism of the OPA layer is preserved end-to-end: the LLM enrichment layer operates only over violations already confirmed by deterministic policy evaluation, so contextual reasoning is added on top of the precision guarantees of the rule engine rather than replacing them the architectural answer to the reliability concern raised by Firouzi et al. [11].

Precision–recall trade-off against the industrial baseline. Checkov achieves perfect recall on the evaluation dataset but at a precision of 59.3%, while the proposed framework trades 9.5 percentage points of recall for a 37.6 percentage-point gain in precision, a difference confirmed statistically significant by McNemar’s exact test. This comparison is qualified by a genuine scope mismatch (Section 6): the two systems are best understood as complementary rather than substitutes, with an organisation reasonably running Checkov as a broad best-practices screen and the proposed framework as a NIST-scoped compliance-evidence generator.

Cross-validation of the encryption-at-rest finding. The concentration of violations in SC-28 (64.3% of all detected violations) is consistent with Verdet et al. [16], who identified encryption at rest as the single most neglected security practice across 812 open-source Terraform projects. This convergence across independent studies differing in dataset, methodology, and detection approach provides cross-validation that encryption-at-rest misconfiguration is a systemic property of real-world IaC practice.

Practical viability of the LLM enrichment layer. The consistently high API completion rate and the 89.8% reduction in API call volume demonstrate that LLM-based enrichment is practically viable as a pipeline component. However, two findings

complicate any claim that the layer is reliable by default, without safeguards, and both point to the same underlying lesson: the reliability observed here is a product of deliberate constraint, not an inherent property of the model. First, the deterministic reassignment of `nist_mapping` fields was necessitated by an observed LLM recall error on AC-3, indicating that LLM-generated control names cannot be trusted without independent verification even at low sampling temperature; the control-mapping correctness reported in Section 6 therefore reflects the correctness of the hardcoded lookup table rather than the model’s own reasoning. Second, an initial validation of LLM-generated remediation snippets found only 46.7% syntactic validity before the enrichment prompt’s formatting constraints were strengthened, after which syntactic validity improved substantially a finding directly corroborated by the independent, concurrent result of Alsaïd et al. [31], and one that indicates the improvement should be credited to the explicit format constraints imposed on the prompt rather than to more capable reasoning by the model itself. Read together, these findings support a general principle for hybrid architectures of this kind: LLM output that is either factual (a control name) or structurally constrained (valid HCL) should not be accepted uncritically; it is deterministic post-processing and explicit, enforced format constraints in the prompt that make the output reliable, not an assumption that the model will behave well by default.

7.2 Threats to Validity

Following common practice in empirical software engineering, the limitations of the present work are organized below according to the type of validity threat they pose; dataset-specific threats are discussed in Section 4.6.

Construct validity. The LLM enrichment quality evaluation reported in Section 6 is limited to control-mapping accuracy and remediation syntactic validity; it does not directly measure explanation clarity or audit utility, the constructs that ultimately determine practical value to GRC stakeholders. A human-subject evaluation with GRC practitioners is identified as a direction for future work.

Internal validity. Inspection of the four OPA false positives and thirteen false negatives (Section 6) shows that both error modes trace to identifiable, documented causes (module/dynamic-block resolution limits, a substring-matching gap in the wildcard-IAM rule, and the missing-field recall gap) rather than to unexplained system behaviour, which supports the internal validity of the reported detection mechanism; the residual risk lies in the labelling heuristic discussed in Section 4.6 rather than in the detection layer itself.

External validity. The Rego policy library is scoped to three NIST SP 800-53 control families (AC, SC, AU) and eight AWS resource types, a scope determined by the resource coverage of the IaC-Eval benchmark; extending to additional families (e.g., SI, IA, CM) and resource types would increase both coverage and per-family recall. The framework also currently supports only AWS Terraform configurations; the OPA-based architecture is cloud-agnostic and the policy library could be extended to Azure and Google Cloud resource types [34], but the present evaluation is limited to AWS as a direct consequence of the IaC-Eval benchmark’s focus. The Checkov baseline

comparison is subject to the scope-mismatch limitation discussed in Section 6; a scope-filtered comparison, restricting Checkov’s findings to checks semantically aligned with AC, SC, and AU controls before computing the confusion matrix, would provide a fairer like-for-like comparison and is identified as a direction for future work.

Conclusion validity. With TP/FP/FN/TN counts of 124/4/13/90 ($n = 231$ files), the 95% Wilson confidence intervals reported in Table 12 are moderately wide, particularly for Recall (84.4–94.4%). The framework also operates as a static analysis pipeline and does not account for dynamic or runtime infrastructure states; bridging this gap would require integration with cloud provider compliance APIs [36] or recent work on state-reconciliation defects [37] and LLM-agent-based runtime drift detection [38]. Finally, the OPA detection layer is deterministic and would produce identical results on repeated runs, but the LLM enrichment layer’s explanation and remediation fields are generated at a non-zero temperature (0.1) and are therefore subject to small inter-run variability, though the deterministic reassignment of NIST control fields eliminates this variability for control-mapping accuracy specifically.

8 Conclusion

This paper presented an open-agent framework for intelligent compliance analysis of Terraform Infrastructure as Code configurations, integrating Open Policy Agent-based Policy-as-Code enforcement with LLM-powered contextual reasoning to generate structured, audit-ready compliance outputs aligned to NIST SP 800-53. The framework addresses a well-documented gap in the literature: no prior peer-reviewed system simultaneously provides Terraform support, formal compliance framework alignment, LLM-powered explainability, and empirically reported precision/recall/F1 evaluation metrics. Evaluated against 231 labeled Terraform configurations derived from the IaC-Eval benchmark the first application of this dataset for security compliance evaluation the OPA enforcement layer achieved a precision of 96.9% (95% CI: 92.2–98.8%), recall of 90.5% (95% CI: 84.4–94.4%), and F1 score of 93.6%, surpassing the 85% detection rate reported by the closest comparable system and outperforming the Checkov industrial baseline (74.5% F1) by a margin confirmed statistically significant via McNemar’s exact test ($\chi^2 = 13.0$, $p < 0.001$). The LLM enrichment layer, aided by a deterministic safeguard against LLM recall errors in control mapping and by explicit prompt-level format constraints, produced structured explanations, deterministically verified control mappings, and syntactically valid remediation snippets across all 16 evaluated violation types across all 16 evaluated violation types; a supplementary syntax validation showed that these prompt-level constraints substantially improved LLM-generated remediation code beyond its initial 46.7% syntactic-validity baseline, a finding corroborated by independent, concurrent work on LLM-generated Terraform repair validity [31]. Violation analysis confirmed that SC-28 (Protection of Information at Rest) accounts for 64.3% of all detected violations, corroborating prior empirical findings on the prevalence of encryption misconfiguration in real-world Terraform infrastructure [16].

Three architectural properties distinguish the framework from prior work: the hybrid pipeline preserves the precision guarantees of deterministic policy evaluation

while adding LLM-driven contextual reasoning on top of, rather than in place of, the rule engine; the resource-aware policy routing strategy minimises spurious detections; and the modular architecture is independent of the specific IaC technology and compliance framework against which it is instantiated. The development process also surfaced a methodological finding of independent interest: cross-validating the OPA detection layer against its own heuristic-derived ground truth identified and, in one case, corrected a genuine labelling gap, while a second, related gap was identified but found to require more sophisticated resource-scoped text parsing than a lightweight heuristic classifier can safely support illustrating that ground-truth quality and detection-rule design are not independent concerns and should be developed in tandem.

Directions for future work include extending the Rego policy library with additional NIST SP 800-53 control families and resource-scoped missing-field rules; re-instantiating the architecture against Azure and Google Cloud resource types; integrating the static-analysis pipeline with cloud provider compliance APIs or LLM-agent-based drift detection systems to bridge the gap between intended and deployed infrastructure state; conducting a formal Cohen’s κ inter-annotator agreement study to quantify ground-truth labelling reliability; performing a blind control-mapping evaluation that withholds the NIST control identifier from the LLM enrichment prompt; and conducting a qualitative user study with GRC practitioners. The labeled evaluation dataset and policy library are released as open artifacts to support reproducibility and benchmarking of future IaC compliance analysis research. The results of this study demonstrate that the hybrid OPA + LLM architecture is a practically realisable design pattern that delivers compliance analysis outcomes neither paradigm achieves alone, and that the reliability of such hybrid systems depends substantially on treating LLM output as requiring verification whether through deterministic post-processing of factual fields or explicit format constraints on generative fields rather than as trustworthy by default.

Declarations

Funding

This research received no external funding.

Competing Interests

The authors declare that they have no competing interests.

Ethics Approval and Consent to Participate

Not applicable.

Consent for Publication

Not applicable.

Data Availability

The evaluation dataset, including the ground-truth labels used in this study, is available in the project repository: <https://github.com/Ayaa859/open-agent-iac-compliance>

Materials Availability

Not applicable.

Code Availability

The source code, Rego policy library, evaluation scripts, and supporting artifacts developed for this study are publicly available at: <https://github.com/Ayaa859/open-agent-iac-compliance>

Author Contributions

Ayah Al-Jabali and Mu'awya Al-Dala'ien conceived the research idea and designed the study. Ayah Al-Jabali developed the framework, implemented the solution, conducted the experiments, and analyzed the results. Mu'awya Al-Dala'ien supervised the research, contributed to the methodological design, validated the research approach, and provided critical revisions throughout the study. Ayah Al-Jabali prepared the manuscript with contributions from Mu'awya Al-Dala'ien. All authors reviewed and approved the final manuscript.

References

- [1] IBM Security and Ponemon Institute: Cost of a data breach report 2024. Technical report, IBM Corporation, Armonk, NY, USA (2024). Accessed: 2026
- [2] CIO: The Top Cloud Security Threat Comes from Within. <https://www.cio.com/article/416343/the-top-cloud-security-threat-comes-from-within.html>. Reports Gartner's estimate that, through 2025, 99% of cloud security failures will be attributable to the customer rather than the cloud provider. Accessed: 2026 (2022)
- [3] Morris, K.: Infrastructure as Code: Dynamic Systems for the Cloud Age, 3rd edn. O'Reilly Media, Sebastopol, CA, USA (2025)
- [4] Cloud Native Computing Foundation: CNCF Annual Survey 2024. <https://www.cncf.io/reports/cncf-annual-survey-2024/>. Reports Terraform's share of production Infrastructure-as-Code deployments. Accessed: 2026 (2024)
- [5] HashiCorp: Terraform by HashiCorp (2022). <https://www.terraform.io>

- [6] Rahman, A., Shamim, S.I., Bose, D.B., Pandita, R.: Security misconfigurations in open source Kubernetes manifests: An empirical study. *ACM Trans. Softw. Eng. Methodol.* **32**(4), 1–36 (2023) <https://doi.org/10.1145/3579639>
- [7] Opdebeeck, R., Alfadel, M., Rahman, A., Kashiwa, Y., Ferreira, J.F., Kula, R.G., De Roover, C.: An empirical study of policy as code: Adoption, purpose, and maintenance. In: *Proceedings of the 23rd International Conference on Mining Software Repositories. MSR '26*. ACM, New York, NY, USA (2026). <https://doi.org/10.1145/3793302.3793355>
- [8] Open Policy Agent Project: Open Policy Agent (2024). <https://github.com/open-policy-agent/opa>
- [9] Xu, H., Wang, S., Li, N., Wang, K., Zhao, Y., Chen, K., Yu, T., Liu, Y., Wang, H.: Large language models for cyber security: A systematic literature review. *arXiv preprint arXiv:2405.04760* (2024)
- [10] Toprani, D., Madiseti, V.K.: LLM agentic workflow for automated vulnerability detection and remediation in Infrastructure-as-Code. *IEEE Access* **13**, 69175–69181 (2025) <https://doi.org/10.1109/ACCESS.2025.3560911>
- [11] Firouzi, E., Bhatt, S., Ghafari, M.: Can developers rely on LLMs for secure IaC development? In: *2026 IEEE International Conference on Software Analysis, Evolution and Reengineering — Companion (SANER-C)*, pp. 257–264. IEEE, ??? (2026). <https://doi.org/10.1109/SANER-C67878.2026.00041>
- [12] National Institute of Standards and Technology: OSCAL: The Open Security Controls Assessment Language. <https://pages.nist.gov/OSCAL/>. Accessed: 2026 (2024)
- [13] International Organization for Standardization: ISO/IEC 27001:2022 Information Security Management Systems (2022)
- [14] ISACA: COBIT 2019 framework: Governance and management objectives (2020)
- [15] Mohammed, R.: Compliance-as-code: Automating governance and security controls in financial and healthcare clouds. *Int. J. AI BigData Comput. Manag. Stud.* **6**(4), 85–94 (2025) <https://doi.org/10.63282/3050-9416.IJAIBDCMS-V6I4P109>
- [16] Verdet, A., Hamdaqa, M., Silva, L.D., Khomh, F.: Assessing the adoption of security policies by developers in Terraform across different cloud providers. *Empirical Software Engineering* **30**(3), 74 (2025) <https://doi.org/10.1007/s10664-024-10610-0>
- [17] Romeo, F., Arena, L., Blefari, F., Pironti, F.A., Lupinacci, M., Furfaro, A.: ARPaCCino: An agentic-RAG for policy as code compliance. In: *New Trends in*

Database and Information Systems. Communications in Computer and Information Science, vol. 2676. Springer, Cham, Switzerland (2026). https://doi.org/10.1007/978-3-032-05727-3_39 . ADBIS 2025

- [18] Kon, P.T.J., Liu, J., Qiu, Y., Fan, W., He, T., Lin, L., Zhang, H., Park, O.M., Elengikal, G.S., Kang, Y., Chen, A., Chowdhury, M., Lee, M., Wang, X.: IaC-Eval: A code generation benchmark for cloud infrastructure-as-code programs. In: Advances in Neural Information Processing Systems, vol. 37, pp. 134488–134506 (2024)
- [19] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: ReAct: Synergizing reasoning and acting in language models. In: Proceedings of the 11th International Conference on Learning Representations (ICLR) (2023)
- [20] Rahman, A., Rahman, M.R., Parnin, C., Williams, L.: Security smells in ansible and chef scripts: A replication study. ACM Transactions on Software Engineering and Methodology **30**(1), 1–31 (2021) <https://doi.org/10.1145/3408897>
- [21] Rahman, A., Parnin, C., Williams, L.: The seven sins: Security smells in infrastructure as code scripts, 164–175 (2019) <https://doi.org/10.1109/ICSE.2019.00033>
- [22] War, A., Diallo, A., Habib, A., Klein, J., Bissyandé, T.F.: Vulnerabilities in infrastructure as code: What, how many, and who? Empirical Software Engineering **30**(5), 120 (2025) <https://doi.org/10.1007/s10664-025-10672-8>
- [23] War, A., Diallo, A., Habib, A., Klein, J., Bissyandé, T.F.: Security Smells in Infrastructure as Code: A Taxonomy Update Beyond the Seven Sins. arXiv:2509.18761 (2025). <https://arxiv.org/abs/2509.18761>
- [24] Bridgecrew: Checkov: Static Code Analysis Tool for Infrastructure as Code. <https://github.com/bridgecrewio/checkov>. Accessed: 2026 (2024)
- [25] Vo, Q.-H., Dao, H., Fukuda, K.: Harnessing the power of LLMs for code smell detection in Terraform infrastructure as code. In: 2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 533–542 (2025). <https://doi.org/10.1109/COMPSAC65507.2025.00075>
- [26] Foale, P.L., Khomh, F., Silva, L.D., Merlo, E.: An Empirical Study of Policy-as-Code Adoption in Open-Source Software Projects. arXiv:2601.05555 (2026). <https://arxiv.org/abs/2601.05555>
- [27] Rajapakse, R.N., Zahedi, M., Babar, M.A., Shen, H.: Challenges and solutions when adopting devsecops: A systematic review. Information and Software Technology **141**, 106700 (2022) <https://doi.org/10.1016/j.infsof.2021.106700>

- [28] Vargas, F.L.S., Mansilha, R.B., Kreutz, D.: Security-First Evaluation of Text-to-Terraform: Benchmarking LLMs and SLMs for Secure IaC Generation. arXiv:2608.02672. Accepted for publication at SBSeg 2026 (2026). <https://arxiv.org/abs/2608.02672>
- [29] Malul, E., Meidan, Y., Mimran, D., Elovici, Y., Shabtai, A.: GenKubeSec: LLM-based Kubernetes misconfiguration detection, localization, reasoning, and remediation. arXiv preprint arXiv:2405.19954 (2024)
- [30] Tony, C., Díaz Ferreyra, N.E., Mutas, M., Dhif, S., Scandariato, R.: Prompting techniques for secure code generation: A systematic investigation. *ACM Transactions on Software Engineering and Methodology* **34**(8), 1–53 (2025) <https://doi.org/10.1145/3722108>
- [31] Alsaid, M., Nebolisa, C., Abbas, F.: TerraProbe: A Layered-Oracle Framework for Detecting Deceptive Fixes in LLM-Assisted Terraform. arXiv:2606.26590. Preprint; submitted to Empirical Software Engineering (2026). <https://arxiv.org/abs/2606.26590>
- [32] Khan, R.N.H., Wasif, D., Cho, J.-H., Butt, A.: Multi-Agent Code-Orchestrated Generation for Reliable Infrastructure-as-Code. arXiv:2510.03902 (2025). <https://arxiv.org/abs/2510.03902>
- [33] Jana, P., Davidson, S., Bhasker, B., Kan, A., Deoras, A., Callot, L.: TerraFormer: Automated Infrastructure-as-Code with LLMs Fine-Tuned via Policy-Guided Verifier Feedback. arXiv:2601.08734 (2026). <https://arxiv.org/abs/2601.08734>
- [34] Bridgecrew: TerraGoat: Vulnerable by Design Terraform Repository. <https://github.com/bridgecrewio/terragoat>. Accessed: 2026 (2024)
- [35] Tarasova, N., Balp-Straffon, E., Iancheruk, A., Sielskyi, Y., Kozodoi, N., Byrne, L.H., Butler, J., Jiang, D., Czelej, M., Ang, A., Shah, Y., Blanco, R., Ivanov, S.: SWE-InfraBench: Evaluating Language Models on Cloud Infrastructure Code. arXiv:2606.05249. Accepted to the NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle (2026). <https://arxiv.org/abs/2606.05249>
- [36] Amazon Web Services: AWS Config Developer Guide. <https://docs.aws.amazon.com/config/>. Accessed: 2026 (2024)
- [37] Hassan, M.M., Salvador, J., Karmaker Santu, S.K., Rahman, A.: State reconciliation defects in infrastructure as code. *Proceedings of the ACM on Software Engineering* **1**(FSE), 1–24 (2024) <https://doi.org/10.1145/3660790>
- [38] Abuzakuk, S., Crijns, L., Kermarrec, A.-M., Pires, R., Vos, M.: RIVA: Leveraging LLM agents for reliable configuration drift detection. In: *Proceedings of the Sixth European Workshop on Machine Learning and Systems*. EuroMLSys '26 (2026). <https://doi.org/10.1145/3805621.3807644>

- [39] Gupta, V., Sreenivasamurthy, D.: Prose2Policy (P2P): A Practical LLM Pipeline for Translating Natural-Language Access Policies into Executable Rego. arXiv:2603.15799. Preprint (2026). <https://arxiv.org/abs/2603.15799>
- [40] Al-Jabali, A.: Open-Agent IaC Compliance Framework. Software repository (2026). <https://github.com/Ayaa859/open-agent-iac-compliance>