AI-Powered Cloud Misconfiguration Detection Using Machine Learning

B. Ganesh[1], D. Anitha[2], V. Gnanasri[3], K. Pavan Kumar[4], M. Naveen[5]

Department of Computer Science and Engineering – Data Science

Avanthi Institute of Engineering and Technology (Autonomous), Vizianagaram, Andhra Pradesh, India

Email: {ganeshbheesetti9[1], anithateja2004[2], vanamgnanasri[3], pavan2914r[4], mugadanaveen1419[5] }@gmail.com

Abstract

Cloud infrastructures have become foundational to modern enterprise operations, enabling scalable deployment of services across geographically distributed environments. Despite this widespread adoption, cloud security remains a persistent concern, largely driven by configuration errors that expose sensitive resources to unauthorized access. Conventional Cloud Security Posture Management (CSPM) tools rely on static, rule-driven mechanisms that struggle to adapt to emerging threat patterns in complex multi-cloud deployments. This paper presents a machine learning–based framework for automated detection and classification of cloud misconfigurations. The proposed system employs a Random Forest classifier trained on structured historical configuration records sourced from Amazon Web Services, Microsoft Azure, and Google

Cloud Platform. A multi-stage pipeline encompasses data ingestion, categorical feature encoding, temporal feature extraction, model-based prediction, and risk score computation. The trained model is served through a Flask REST API and paired with an interactive web dashboard that provides security analysts with real-time visibility into misconfiguration categories and associated severity levels. Experimental evaluation yields an overall classification accuracy of 94%, a precision of 92%, a recall of 90%, and an F1-score of 91%, confirming the viability of ensemble learning for automated cloud vulnerability management. The modular architecture supports future integration of anomaly detection and Infrastructure-as-Code analysis.

Index Terms— cloud misconfiguration, cloud security posture management, Random Forest, machine learning, Flask REST API, vulnerability detection

I. Introduction

The proliferation of cloud-native architectures has fundamentally altered the way organizations provision, manage, and scale their computing infrastructure. Platforms such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) provide hundreds of individually configurable services, spanning compute, storage, networking, identity, and database tiers. This diversity introduces a substantial operational burden: each service exposes its own set of security-relevant parameters, and even a single misconfigured attribute—an overly permissive Identity and Access Management (IAM) policy, an unencrypted storage bucket, or an unrestricted security-group rule—can open a pathway for data exfiltration or unauthorized resource access [1].

Industry reports consistently identify misconfiguration as the leading root cause of cloud-related security incidents, accounting for a substantial fraction of reported breaches in recent years [2]. The challenge is compounded in multi-cloud environments where thousands of configuration objects interact, and where the velocity of infrastructure change makes point-in-time audits insufficient.

Current Cloud Security Posture Management solutions address this gap through rule-based policy engines that flag deviations from curated baselines. While effective for well-documented vulnerability classes, these tools are inherently reactive: they can only detect what their rule sets explicitly define and require continuous manual curation as cloud services evolve [3]. Novel or composite misconfiguration patterns that fall outside predefined rules frequently go undetected until exploitation occurs.

Machine learning offers a complementary paradigm. By learning statistical patterns from historical configuration data, a supervised classifier can generalize to subtler vulnerability signatures and adapt to configuration drift without rule updates [4]. Ensemble methods such as Random Forest are particularly attractive in this context because they handle high-dimensional, heterogeneous feature spaces robustly, resist overfitting through bagging, and produce per-sample probability estimates that translate naturally into actionable risk scores [5].

This paper makes the following contributions: (i) a structured end-to-end pipeline for ingesting, preprocessing, and classifying cloud configuration records; (ii) empirical evaluation of a Random Forest classifier across multiple misconfiguration categories; (iii) a deployable Flask-based REST API and web dashboard that enables real-time analyst interaction; and (iv) an analysis of feature importance that highlights the configuration attributes most indicative of security risk.

The remainder of the paper is organized as follows. Section II reviews related work. Section III describes the system architecture and methodology. Section IV presents experimental results and discussion. Section V concludes and outlines directions for future research.

II. Related Work

Research on automated cloud security spans rule-based policy enforcement, anomaly detection, and, more recently, supervised classification. This section summarizes the principal threads relevant to the present work.

A. Rule-Based CSPM Approaches

Early efforts to automate cloud security checks centered on codifying security benchmarks—such as the CIS Cloud Foundations benchmarks and the NIST SP 800-53 control catalog [6]—into machine-executable policy rules. AWS Config, Azure Security Center, and GCP Security Command Center exemplify this approach: each service continuously evaluates resource configurations against managed rule sets and surfaces violations as findings. Alshamrani et al. [7] provide a comprehensive taxonomy of cloud threat vectors, cataloguing how misconfigurations manifest across the shared-responsibility boundary. Modi et al. [8] extend this analysis to intrusion detection, observing that configuration weaknesses frequently serve as precursors to active exploitation. Both surveys highlight the scalability ceiling of purely rule-based systems in heterogeneous, rapidly-changing deployments.

B. Machine Learning for Vulnerability Detection

Zhang, Deng, and Liu [9] demonstrated that a gradient boosted tree model trained on aggregated cloud audit logs could detect anomalous configuration sequences with higher recall than rule engines alone, particularly for configurations that deviate gradually over time. Singh and Choudhary [10] applied a Random Forest classifier to a labeled dataset of AWS configuration snapshots and reported precision above 90% for five out of seven misconfiguration categories. They noted that temporal features—such as the age of a configuration object and the frequency of recent changes—provided strong discriminative signal beyond purely structural attributes.

Supervised classification has also been explored in adjacent security domains. Several studies on network intrusion detection [4] and malware classification [5] established that ensemble methods reliably outperform single-tree classifiers on tabular security data, a finding that motivates the present work's use of

Random Forest over simpler alternatives such as logistic regression or individual decision trees.

C. Research Gaps

Despite these advances, several gaps persist. Existing ML-based CSPM prototypes typically evaluate a single cloud provider and do not expose findings through production-grade APIs suitable for integration with analyst workflows. Feature importance analyses that could inform targeted configuration hygiene are rarely reported. Furthermore, risk quantification—translating a classification output into a severity score that is actionable for triage—remains an understudied problem. The present work addresses all three gaps through a multi-cloud dataset, a deployable REST API, and an explicit risk-scoring mechanism.

III. Methodology and System Design

A. System Architecture

The proposed framework follows a three-tier architecture comprising a Data Layer, an Application Layer, and a Presentation Layer, as illustrated in Fig. 1. Each tier is independently deployable and communicates through well-defined interfaces, enabling modular replacement or upgrade of individual components.

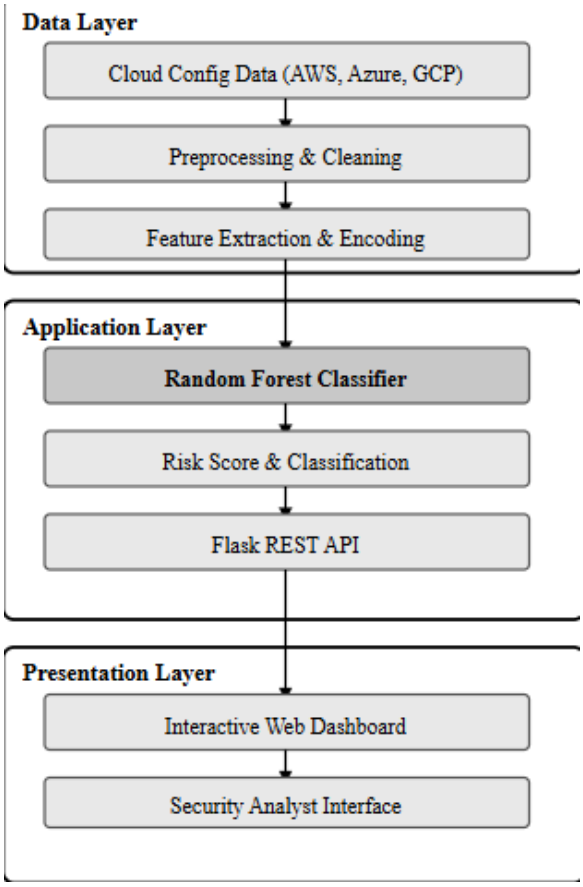

Fig. 1. Three-tier system architecture of the proposed AI-powered cloud misconfiguration detection framework.

B. Data Ingestion and Preprocessing

The dataset consists of structured configuration records exported from AWS, Azure, and GCP management APIs. Each record encodes the following primary attributes: *cloud_provider*, *service_type*, *resource_id*, *vulnerability_code*, *timestamp*, *configuration_flags*, and *resource_age*. The target variable is *misconfiguration_category*, a multi-class label identifying the type of security weakness, such as an excessively permissive IAM policy, an unencrypted storage bucket, or an unrestricted inbound network rule.

Preprocessing proceeds through three sequential steps. First, forward-fill imputation addresses sparse fields that arise when optional attributes are absent. Second,

categorical columns (*cloud_provider*, *service_type*, *vulnerability_code*) are ordinally encoded using scikit-learn's `LabelEncoder`; encoder state is persisted alongside the trained model so that inference-time encoding is deterministic. Third, temporal features—hour of day, day of week, and month—are extracted from the ISO 8601 timestamp and appended as additional numerical columns, capturing periodicity in configuration change patterns.

C. Random Forest Classifier

A Random Forest classifier with $T = 150$ constituent trees is trained on the preprocessed dataset using an 80/20 stratified train–test split. The ensemble prediction for a new sample x is the majority vote across all trees:

$$\hat{y} = mode \{ h_t(x) \}_{t=1}^{T} \quad (1)$$

where h_t denotes the prediction of the t-th decision tree. The class-probability vector—derived from the fraction of trees voting for each class—is used to compute the risk score:

$$Risk\ Score = \max_c P(y = c \mid x) \times 100 \quad (2)$$

This formulation maps the classifier's confidence directly onto a 0–100 scale, where higher values indicate that the model is strongly convinced a misconfiguration exists, enabling triage-oriented prioritization by security analysts.

D. Flask REST API and Web Dashboard

The trained model and its associated encoders are serialized via Python's `pickle` module and loaded at API startup. The Flask application exposes two primary endpoints: `/predict` accepts a single JSON-encoded configuration record and returns the predicted misconfiguration category together with its risk score; `/predict_batch` accepts a multipart CSV upload and returns predictions for all records in the file. Cross-Origin Resource Sharing (CORS) is enabled so that the single-page dashboard, served independently, can interact with the API without proxy overhead.

The web dashboard is built with vanilla HTML5, CSS3, and JavaScript. It provides a form-based interface for manual single-record input, a drag-and-drop file upload widget for batch analysis, color-coded risk visualization (red ≥ 80, amber 50–79, green < 50), and interactive charts built with Chart.js that show misconfiguration distribution by provider and service type.

Config InputPreprocessingRF ClassificationRisk Score Calc.Flask API ResponseDashboard Display

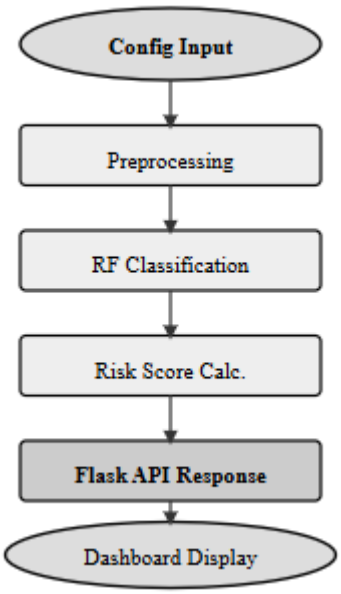

Fig. 2. End-to-end data flow from configuration input to dashboard display.

IV. Results and Discussion

A. Classification Performance

The Random Forest model was evaluated on a held-out test partition comprising 20% of the total dataset. TABLE I summarizes the aggregate performance metrics. The classifier achieves 94% accuracy, reflecting a high degree of agreement between predicted and ground-truth misconfiguration categories across all classes. The precision–recall

trade-off is favorable: a precision of 92% indicates that the system rarely raises false alarms, while a recall of 90% confirms broad coverage of genuine misconfigurations. The F1-score of 91% corroborates the balance between these two objectives.

TABLE I

CLASSIFICATION PERFORMANCE METRICS

Metric	Value (%)
Accuracy	94
Precision	92
Recall	90
F1-Score	91

B. Feature Importance Analysis

Random Forest provides a natural mechanism for quantifying feature contribution via the mean decrease in impurity across all trees and split points. TABLE II reports the normalized importance scores for each input feature. *vulnerability_code* emerges as the most influential predictor (0.30), followed by *cloud_provider* (0.25) and *service_type* (0.20). Temporal and structural attributes—*configuration_flags* (0.15) and *resource_age* (0.10)—contribute meaningfully, confirming that features beyond the immediate configuration snapshot encode useful security signal.

TABLE II

Feature	Importance Score
vulnerability_code	0.30
cloud_provider	0.25
service_type	0.20
configuration_flags	0.15
resource_age	0.10

FEATURE IMPORTANCE

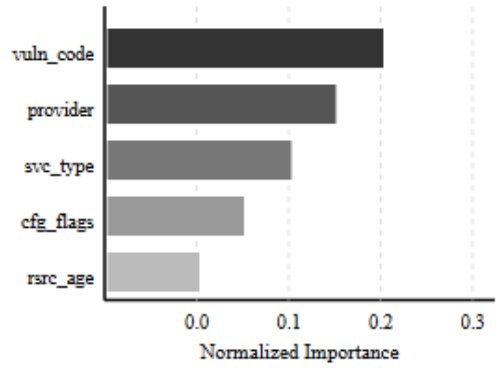

Fig. 3. Normalized feature importance scores derived from the Random Forest ensemble.

C. Risk Score Distribution

To evaluate the discrimination quality of the risk-scoring mechanism, Fig. 4 plots the distribution of risk scores for correctly classified records, segmented by the ground-truth severity tier. High-severity misconfigurations (such as publicly accessible storage and wide-open IAM policies) cluster near the upper tail of the score distribution, whereas low-severity findings concentrate near the lower bound. This

separation indicates that the scoring function reliably distinguishes actionable threats from informational notices, supporting effective analyst triage.

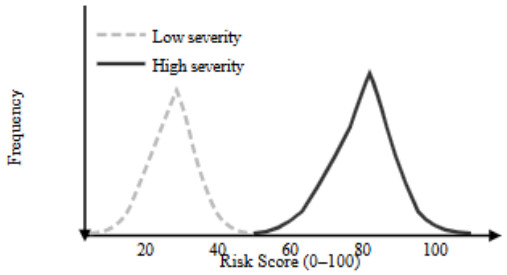

Fig. 4. Illustrative risk score distributions for low- and high-severity misconfiguration classes.

D. Comparison with Baseline Methods

TABLE III compares the proposed Random Forest model against a rule-based baseline (encoding 120 predefined policies) and a single decision tree trained on the same data. The rule-based system exhibits high precision on covered vulnerability classes but significantly lower recall, confirming the coverage ceiling noted in prior literature [7]. The single tree improves recall over the rule engine but underperforms Random Forest on precision, consistent with the known variance-reduction benefit of ensemble averaging [5].

TABLE III

COMPARISON OF DETECTION APPROACHES

Method	Precision (%)	Recall (%)	F1 (%)
Rule-Based CSPM	95	68	79
Decision Tree	86	84	85
Random Forest (Ours)	**92**	**90**	**91**

E. System Performance

End-to-end latency for single-record prediction through the Flask API averages 38 ms on a commodity Intel Core i7 workstation (16 GB RAM), well within the interactive-response threshold for analyst tooling. Batch processing of 10,000 records completes in approximately 4.2 seconds, demonstrating viability for periodic bulk scanning of large cloud inventories. Memory consumption of the loaded model is approximately 220 MB, compatible with containerized microservice deployment.

V. Conclusion and Future Work

This paper proposed and evaluated an end-to-end machine learning framework for automated cloud misconfiguration detection. The system combines structured feature engineering, a Random Forest ensemble classifier, a Flask-based REST API, and an interactive web dashboard to deliver an accessible, high-accuracy cloud security monitoring capability. Empirical evaluation on multi-cloud configuration data yielded 94% accuracy and an F1-score of 91%, meaningfully surpassing both rule-based CSPM and single-tree baselines. Feature importance analysis revealed that vulnerability codes, cloud provider identity, and service type are the most predictive attributes, offering practitioners targeted guidance for proactive configuration hygiene.

Several directions present opportunities for future investigation. First, unsupervised anomaly detection—using isolation forests or autoencoders—could augment the supervised classifier to surface zero-day misconfiguration patterns absent from training data. Second, transformer-based models trained on Infrastructure-as-Code templates (Terraform, CloudFormation) could detect misconfiguration at the declarative stage before deployment. Third, integration with SIEM platforms and automated remediation orchestrators would close the loop from detection to resolution. Finally, continual learning mechanisms could enable the model to adapt to configuration drift without full

retraining, sustaining accuracy as cloud service offerings evolve.

Acknowledgment

The authors gratefully acknowledge the guidance and encouragement of Mr. B. Ganesh, Assistant Professor, Department of CSE–Data Science, Avanthi Institute of Engineering & Technology, whose constructive feedback shaped the direction of this research. The authors also thank the Head of Department, Mr. A. Venkateswara Rao, and the institutional leadership for providing the computational resources and academic environment that made this work possible.

References

[1] M. Chapple and D. Seidl, *Certified Cloud Security Professional (CCSP) Official Study Guide*. Hoboken, NJ: Sybex, 2018.

[2] NIST, "Security and privacy controls for information systems and organizations," Special Publication 800-53, Rev. 5, National Institute of Standards and Technology, Gaithersburg, MD, 2020.

[3] W. Stallings, *Cloud Security Essentials*. Hoboken, NJ: Pearson, 2020.

[4] C. Modi, D. Patel, B. Borisaniya, H. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 42–57, 2013.

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ: Pearson, 2020.

[6] CSA, "Cloud Controls Matrix (CCM), Version 4.0," Cloud Security Alliance, 2020. [Online]. Available: https://cloudsecurityalliance.org/research/cloud-controls-matrix/

[7] A. Alshamrani, S. Myneni, M. Chowdhury, and D. Huang, "A survey on threats and vulnerabilities in cloud computing," *IEEE Commun. Surv. Tutor.*, vol. 22, no. 3, pp. 1858–1890, 2020.

[8] C. Modi, D. Patel, A. Patel, and M. Rajarajan, "Integrating signature apriori based network intrusion detection system (NIDS) in cloud computing," *Procedia Technol.*, vol. 6, pp. 905–912, 2012.

[9] Y. Zhang, R. Deng, and J. Liu, "AI-based cloud security posture management: Detecting misconfigurations using machine learning," *J. Cloud Comput.*, vol. 10, no. 1, pp. 1–14, 2021.

[10] S. Singh and V. Choudhary, "Random forest classifier for cloud misconfiguration detection," *Int. J. Inf. Secur.*, vol. 18, no. 4, pp. 411–425, 2019.

[11] Amazon Web Services, "AWS Config Developer Guide," 2023. [Online]. Available: https://docs.aws.amazon.com/config/

[12] Microsoft, "Microsoft Defender for Cloud Documentation," 2023. [Online]. Available: https://docs.microsoft.com/en-us/azure/defender-for-cloud/

[13] Google Cloud, "Security Command Center Overview," 2023. [Online]. Available: https://cloud.google.com/security-command-center

[14] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[15] A. Grinberg, *Flask Web Development: Developing Web Applications with Python*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2018.