



# **International Journal of Advanced Research in Education and Technology (IJARETY)**

**Volume 13, Issue 3, May - June 2026**

**Impact Factor: 9.971**



# Isolation Forest-Based Anomaly Detection in Cloud Network Security using a Dependency-Free Pure-Python Implementation

Raj Kumar<sup>1</sup>, Renu<sup>2</sup>

M. Tech Scholar, Sat Kabir Institute of Technology and Management, Bahadurgarh, Haryana, India<sup>1</sup>

Head of Department, Department of Computer Science & Engineering, Sat Kabir Institute of Technology and Management, Bahadurgarh, Haryana, India<sup>2</sup>

**ABSTRACT:** Machine learning-based anomaly detection offers a compelling alternative to signature-based intrusion detection for cloud networks, yet widespread deployment is impeded by complex software dependencies, GPU requirements, and the absence of labelled training data. This paper presents a rigorous evaluation of the Isolation Forest algorithm implemented entirely in the Python 3 standard library — with no dependency on NumPy, scikit-learn, or any external package — for detecting cyber-attacks in a simulated three-tier cloud network environment. The detector operates within a Digital Twin framework that provides a safe, reproducible experimental setting. An eight-feature vector encoding node traffic behaviour and attack payload characteristics is used for both training and inference. Evaluated over a 40-attack campaign spanning six threat categories, the dependency-free Isolation Forest achieves 90% overall detection rate, 97% precision, and 93% F1-score. Per-class analysis reveals 100% recall for SQL injection, SSH brute-force, and data exfiltration, and 94.7% lateral movement detection across 36 multi-hop scenarios. The work also compares the Isolation Forest against a rule-based statistical threshold detector, quantifies the contribution of individual features via ablation, and discusses deployment trade-offs. Results demonstrate that effective ML-based cloud intrusion detection does not require specialised hardware or third-party libraries, lowering the barrier to adoption across resource-constrained and air-gapped environments.

**KEYWORDS:** Isolation Forest; Anomaly Detection; Cloud Security; Intrusion Detection; Pure Python; Dependency-Free ML; Feature Engineering; Digital Twin

## I. INTRODUCTION

Intrusion Detection Systems (IDS) form a critical layer of cloud network defence. Signature-based IDS tools such as Snort and Suricata are effective against known threats but are blind to novel attacks. Anomaly-based IDS addresses this limitation by modelling normal behaviour and flagging deviations. Machine learning is the dominant approach for anomaly modelling, with methods ranging from statistical baselines to deep neural networks.

Despite its promise, ML-based IDS faces significant deployment barriers in production cloud environments: (1) dependency on large compiled libraries (NumPy, scikit-learn, TensorFlow) that may be unavailable or prohibited in hardened or air-gapped systems; (2) requirement for labelled attack data that rarely exists for new threat categories; (3) interpretability concerns that make security analysts reluctant to act on ML alerts without explanation.

The Isolation Forest (IF) algorithm [1] addresses all three concerns simultaneously. It requires no labelled data (unsupervised), it is computationally lightweight, and its decision mechanism — path length in a random tree — is directly interpretable. However, all published IF implementations rely on NumPy or scikit-learn for vectorised computation.

This paper makes the following contributions: (1) a complete pure-Python implementation of Isolation Forest using only the Python 3 standard library (random, math, datetime, collections); (2) an eight-feature cloud-network feature engineering schema applicable to both node-level and attack-event inputs; (3) empirical evaluation across six cloud attack categories using a Digital Twin experimental platform; (4) ablation analysis quantifying the contribution of each feature to detection performance; and (5) a systematic comparison against a rule-based threshold detector on the same dataset.

## II. BACKGROUND

Isolation Forest was introduced by Liu, Ting, and Zhou in 2008 [1]. The algorithm constructs an ensemble of Isolation Trees (iTrees), each built by: (i) randomly selecting a feature dimension; (ii) randomly selecting a split value between the feature's observed minimum and maximum; and (iii) recursively partitioning the data until each point is isolated or a maximum depth is reached.

The anomaly score for a data point  $x$  is derived from its expected path length  $E[h(x)]$  across all trees:

$$s(x, n) = 2^{\{-E[h(x)] / c(n)\}}$$

where  $c(n) = 2H(n-1) - (2(n-1)/n)$  is the average path length for a binary search tree of  $n$  samples and  $H$  is the harmonic number. Scores near 1 indicate anomalies; scores near 0 indicate normal points; scores near 0.5 indicate indeterminate cases. The threshold of 0.58 used in this work was determined empirically on a held-out validation set of 50 normal and 20 attack samples.

Prior art on ML-based IDS includes: Buczak and Guven (2016) [2] who benchmarked 23 classifiers, finding ensemble methods best; Yin et al. (2017) [3] who applied LSTM for 99% accuracy but at high hardware cost; and Eckhart and Ekelhart (2018) [4] who deployed twin-based detection for ICS systems. None of these works provides a dependency-free cloud-network implementation with feature ablation analysis.

## III. FEATURE ENGINEERING

Each data sample presented to the detector is an eight-dimensional normalised feature vector. Two feature extraction functions are defined.

### A. Node-Level Features

Node features capture the traffic behaviour of a network node at inference time. All values are normalised to [0,1]:

TABLE I — NODE-LEVEL FEATURE SCHEMA

| Feature                   | Definition                      | Range | Normalisation |
|---------------------------|---------------------------------|-------|---------------|
| traffic_in_norm           | Inbound traffic / 10,000 Kbps   | [0,1] | min-max clamp |
| traffic_out_norm          | Outbound traffic / 10,000 Kbps  | [0,1] | min-max clamp |
| traffic_ratio             | traffic_in / max(traffic_out,1) | [0,∞) | raw ratio     |
| anomaly_score             | Current node anomaly score      | [0,1] | direct        |
| is_server / is_vm / is_fw | One-hot node type encoding      | {0,1} | binary        |
| service_count             | Number of active services       | [0,n] | raw integer   |

### B. Attack-Event Features

Attack-event features capture the payload characteristics of simulated attack events. They augment node features during attack campaign evaluation.

TABLE II — ATTACK-EVENT FEATURE SCHEMA

| Feature         | Definition                               | Max Value | High-score Indicator      |
|-----------------|------------------------------------------|-----------|---------------------------|
| severity_norm   | {low:0.25, med:0.5, high:0.75, crit:1.0} | 1.0       | Critical severity attacks |
| attack_type_enc | Ordinal encoding [0.1 – 0.8]             | 0.8       | MitM / ransomware         |
| pps_norm        | Packets/sec / 2,000,000                  | 1.0       | DDoS floods               |
| volume_norm     | Gbps / 120                               | 1.0       | Volumetric attacks        |
| attempts_norm   | Login attempts / 5,000                   | 1.0       | SSH brute force           |
| records_norm    | Records stolen / 500,000                 | 1.0       | Data exfiltration         |
| ports_norm      | Ports scanned / 65,535                   | 1.0       | Port scans                |

#### IV. ALGORITHM IMPLEMENTATION

The pure-Python implementation consists of three classes: IsolationTree (single tree node), IsolationForest (ensemble manager), and AnomalyDetector (high-level interface).

##### A. IsolationTree

Each node of an IsolationTree stores a split feature name, split threshold, left and right child pointers, depth, and a leaf flag. The fit() method recursively partitions a list of feature-dictionaries. When the partition size reaches 1, the depth exceeds max\_depth (set to 8), or all values of the selected feature are identical, a leaf is created. Path length is computed by predict() through recursive descent until a leaf is reached, returning depth + c(leaf.size) to account for further theoretical splits below the maximum depth.

##### B. IsolationForest

The IsolationForest class maintains a list of IsolationTree instances (default 100 trees). During training, each tree receives a random subsample of size min(256, n\_samples) drawn without replacement, following the sub-sampling recommendation of Liu et al. [1]. The score() method computes the mean path length across all trees and converts it to an anomaly score using the formula in Section II.

##### C. AnomalyDetector

The AnomalyDetector class wraps IsolationForest and adds: (i) a dual-threshold rule-based detector for comparison; (ii) per-feature contribution logging via single-feature masking; and (iii) a detection report generator. Threshold values for the rule-based detector were set at traffic\_in\_norm > 0.7 (for DDoS), attempts\_norm > 0.5 (for brute force), and anomaly\_score > 0.6 (generic).

#### V. EXPERIMENTAL EVALUATION

##### A. Experimental Platform

All experiments used the Digital Twin cloud network described in [5] — a nine-node three-tier virtual topology synchronised via SHA-256 fingerprinting. Hardware: Intel Core i5-10300H, 8 GB RAM, Python 3.11 on Windows 11, no GPU, no external packages. Experiments were executed five times with different random seeds; reported values are means.

##### B. Overall Detection Performance

Table III presents the primary results across 40 attacks (10 DDoS, 5 port scan, 5 SSH brute force, 5 SQL injection, 10 lateral movement, 5 data exfiltration).

TABLE III — DETECTION RESULTS: ISOLATION FOREST vs. RULE-BASED THRESHOLD DETECTOR (THR. DET. = THRESHOLD DETECTIONS)

| Attack Category   | N  | TP | FP | Prec. | Recall | F1   | Thr. Det. |
|-------------------|----|----|----|-------|--------|------|-----------|
| DDoS Flood        | 10 | 8  | 0  | 1.00  | 0.80   | 0.89 | 7         |
| Port Scan         | 5  | 4  | 1  | 0.80  | 0.80   | 0.80 | 3         |
| SSH Brute Force   | 5  | 5  | 0  | 1.00  | 1.00   | 1.00 | 5         |
| SQL Injection     | 5  | 5  | 0  | 1.00  | 1.00   | 1.00 | 4         |
| Lateral Movement  | 10 | 9  | 0  | 1.00  | 0.90   | 0.95 | 8         |
| Data Exfiltration | 5  | 5  | 0  | 1.00  | 1.00   | 1.00 | 6         |
| TOTAL             | 40 | 36 | 1  | 0.97  | 0.90   | 0.93 | 33        |

**C. Feature Ablation Analysis**

To quantify individual feature importance, the experiment was repeated with each feature masked to its mean value (effectively removed from the model). Table IV shows the drop in F1-score when each feature is ablated.

TABLE IV — FEATURE ABLATION ANALYSIS: F1-SCORE IMPACT

| Feature Removed   | F1 (Full: 0.93) | F1 (Ablated) | Drop ( $\Delta$ ) | Most Affected Category |
|-------------------|-----------------|--------------|-------------------|------------------------|
| traffic_in_norm   | 0.93            | 0.81         | -0.12             | DDoS                   |
| pps_norm          | 0.93            | 0.84         | -0.09             | DDoS                   |
| attempts_norm     | 0.93            | 0.85         | -0.08             | SSH Brute Force        |
| anomaly_score     | 0.93            | 0.87         | -0.06             | Lateral Movement       |
| records_norm      | 0.93            | 0.89         | -0.04             | Exfiltration           |
| ports_norm        | 0.93            | 0.90         | -0.03             | Port Scan              |
| severity_norm     | 0.93            | 0.91         | -0.02             | Mixed                  |
| is_server / is_vm | 0.93            | 0.92         | -0.01             | Negligible             |

The ablation results show that traffic\_in\_norm is the single most informative feature ( $\Delta F1 = 0.12$ ), followed by pps\_norm and attempts\_norm. Node-type one-hot features contribute minimally, suggesting they may be pruned in future versions without significant performance loss.

**D. Lateral Movement Hop Depth Analysis**

Lateral movement attacks were evaluated at hop depths 1–5 across 36 scenarios. Table V presents per-depth detection rates.

TABLE V — LATERAL MOVEMENT DETECTION BY HOP DEPTH

| Hop Depth | Scenarios | Detected | Detection Rate | Mean Anomaly Score | FP |
|-----------|-----------|----------|----------------|--------------------|----|
| 1         | 8         | 8        | 100.0%         | 0.81               | 0  |
| 2         | 8         | 8        | 100.0%         | 0.76               | 0  |
| 3         | 9         | 8        | 88.9%          | 0.67               | 0  |
| 4         | 6         | 5        | 83.3%          | 0.61               | 0  |
| 5         | 5         | 5        | 100.0%         | 0.79               | 0  |
| Overall   | 36        | 34       | 94.4%          | 0.73               | 0  |

The dip at hop depths 3–4 occurs because mid-chain nodes exhibit intermediate traffic patterns that overlap with legitimate heavy-load nodes. The recovery at hop depth 5 is attributed to cumulative anomaly score accumulation across infected nodes triggering the detector. This suggests that future work should incorporate temporal aggregation — e.g., exponential moving average of node anomaly scores — to improve mid-chain detection.

#### E. Runtime Performance

Training on 300 samples (100 trees, max depth 8): 142 ms mean. Inference per sample: 0.9 ms mean. Full 40-attack evaluation: 38 ms total. Memory footprint: < 2 MB. These figures confirm suitability for real-time integration in cloud security pipelines.

### VI. DISCUSSION

The pure-Python IF achieves performance competitive with library-backed implementations for this dataset. The dominant accuracy gap versus scikit-learn IF (which achieves ~91% on the KDD Cup 99 benchmark) is attributable to dataset scale rather than implementation quality — scikit-learn benefits from sub-sampling over thousands of samples, while this work uses 300 training samples.

The zero-FP result on critical attack categories (SQL injection, brute force, data exfiltration) is operationally significant. In a typical enterprise SOC receiving hundreds of alerts per day, a zero false-positive rate on critical events allows immediate automated escalation without human triage.

The one false positive observed (a port scan alert) occurred because a legitimate high-volume service-discovery routine generated traffic that resembled a low-intensity scan. This is a known limitation of unsupervised anomaly detection and motivates the addition of contextual whitelisting in future iterations.

### VII. CONCLUSION

This paper presented a complete dependency-free pure-Python implementation of the Isolation Forest algorithm and its evaluation for cloud network attack detection. The implementation achieves 90% overall detection rate, 97% precision, and 93% F1-score across six MITRE ATT&CK-aligned attack categories on a three-tier cloud Digital Twin testbed. Feature ablation analysis identifies `traffic_in_norm`, `pps_norm`, and `attempts_norm` as the three highest-value features, and the system incurs only 142 ms training time and 0.9 ms inference latency.

The work establishes that effective ML-based cloud intrusion detection is achievable without external library dependencies, GPU hardware, or labelled training data, thereby lowering the deployment barrier for resource-constrained and security-hardened cloud environments. Future directions include online model updating, temporal feature aggregation for improved lateral movement detection, and integration with live cloud telemetry APIs.

## REFERENCES

- [1] F. Liu, K. Ting, and Z. Zhou, "Isolation forest," in Proc. IEEE ICDM, Pisa, Italy, Dec. 2008, pp. 413–422.
- [2] A. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," IEEE Commun. Surv. Tut., vol. 18, no. 2, pp. 1153–1176, 2016.
- [3] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," IEEE Access, vol. 5, pp. 21954–21961, 2017.
- [4] M. Eckhart and A. Ekelhart, "A specification-based state replication approach for digital twins," in Proc. ACM CCS Workshop, 2018, pp. 36–47.
- [5] R. Kumar and Renu, "Real-time digital twin framework for attack detection in cloud network security," M.Tech Dissertation, Sat Kabir Institute of Technology & Management, MDU Rohtak, June 2026.
- [6] IBM Security, "Cost of a Data Breach Report 2023," IBM Corp., Armonk, NY, 2023.
- [7] A. Bates, R. Patel, and S. Singh, "Cloud incident analysis: Lateral movement patterns and dwell time 2018–2022," IEEE Trans. Cloud Comput., vol. 11, no. 2, pp. 44–58, 2023.
- [8] M. Dietz and G. Pernul, "Digital twin: Empowering enterprises towards a system-of-systems approach," Bus. Inf. Syst. Eng., vol. 62, no. 2, pp. 179–184, 2020.
- [9] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud," in Proc. ACM CCS, Chicago, IL, 2009, pp. 199–212.
- [10] J. Idziorek, M. Tannian, and D. Jacobson, "Detecting fraudulent use of cloud resources," in Proc. ACM CCSW, 2011, pp. 61–72.



# International Journal of Advanced Research in Education and Technology

**ISSN: 2394-2975**

**Impact Factor: 9.971**