Journal Publication of International Research for Engineering and Management (JOIREM)
Volume: 04 Issue: 4| April 2026
ISSN (O) 3107-6696

# A Real-Time Streaming Hybrid Ensemble SOAR Framework Using Kafka Streaming for Autonomous Cyber Anomaly Detection

**Vaishnavi Sunil Desai, Tapaswini Chakrapani Desiti, Siddhi Satish Bhalekar, Anushka Anil Patil, Prof. Snehal Pratap Mane.**

*Student, Student, Student, Student, Project guide*
*Department of Artificial Intelligence and Data Science, Terna Engineering College, University of Mumbai, India*

vaishnavidesai@ternaengg.ac.in, tapaswinidesiti2223@terna.ac.in, siddhibhalekar@ternaengg.ac.in, anushkapatil@ternaengg.ac.in , snehalmane@ternaengg.ac.in

---

**Abstract** – Modern networks generate large volumes of security logs, making it difficult for analysts to detect and respond to threats in time. In this work, we design and evaluate a real-time streaming SOAR framework that combines a Kafka-based ingestion pipeline with a hybrid ensemble consisting of XG Boost, Random Forest, and an Isolation Forest model. The system processes events as they arrive, assigns risk levels through a weighted consensus score, and triggers automated responses when high- severity activity is detected. Although our evaluation uses simulated enterprise traffic, the architecture reflects real deployment constraints and is implemented in a containerized setup to measure latency and through- put. Experimental results show that the ensemble achieves high detection accuracy while maintaining sub-second processing latency under moder- ate load. We also include ablation studies to understand the contribution of each model. The findings highlight the potential of lightweight ensemble techniques and streaming pipelines for building practical, automated cyber defense systems.

***Keywords***: Cyber Security · SOAR · Anomaly Detection · Kafka · Hybrid Ensemble · Machine Learning

## I. INTRODUCTION

Organizations today rely heavily on interconnected systems, cloud platforms, and remote access services, all of which generate an enormous volume of security- relevant data. While this data is valuable for identifying potential attacks, analysts rarely have enough time or resources to review it effectively. Traditional intrusion detection systems, which depend largely on predefined signatures or static rules, tend to perform well for known threats but break down when at- tackers change their behaviour or use previously unseen methods. As a result, many security teams struggle with delayed detection, high false-positive rates, and alert fatigue.

To address these challenges, researchers have increasingly turned toward machine learning-based anomaly detection. Supervised models, such as Random Forest and XG Boost are capable of learning complex attack patterns from historical data, while unsupervised models like Isolation Forest can help surface unusual or zero-day behaviours. Despite this progress, integrating such models into a practical, real-time security workflow remains difficult. Most published studies evaluate models on static datasets rather than on a live, streaming pipeline. Additionally, few systems provide automated responses or perform end-to-end measurements of latency and throughput—both of which are essential for any realistic deployment.

Unlike prior works that focus either on offline anomaly detection or rule-based SOAR platforms, this work integrates a hybrid ensemble with a streaming-first architecture and evaluates both detection performance and operational latency under realistic ingestion loads. In this work, we design and implement a streaming SOAR framework that attempts to bridge this gap between academic models and operational requirements. The system uses Apache Kafka to ingest events continuously, processes them through a hybrid ensemble of supervised and unsupervised models, assigns severity levels through a weighted scoring function, and automatically triggers responses for high-risk activity.

Although our experimental evaluation uses simulated enterprise traffic, the architecture itself is built to reflect realistic deployment constraints and is implemented entirely in containers to enable reproducible measurements. The main contributions of this paper are threefold. First, we propose a lightweight yet effective hybrid ensemble that balances high precision on known threats with sensitivity to novel anomalies. Second, we integrate this ensemble into a real-time streaming workflow using Kafka, Supa base, and a modular response layer to form a functional SOAR pipeline. Third, we present an empirical evaluation that includes classification metrics, latency and throughput measurements, and ablation studies to understand the role of each model. Together, these components demonstrate a practical pathway toward automated, real-time cyber defense using scalable streaming tools.

## II. SYSTEM ARCHITECTURE

The proposed framework is designed as an end-to-end streaming pipeline capable of ingesting events in simulated real-time conditions, performing anomaly detection using a hybrid ensemble model, and automatically triggering response actions based on the severity of detected threats. The system consists of four major components:

(i) a Kafka-based data ingestion layer,

(ii) a machine learning inference consumer,

(iii) a persistence and response module implemented using Supa base, and

(iv) a lightweight visualization dashboard for monitoring and analysis.

The architecture is modular, allowing each component to scale independently depending on workload and deployment requirements.

### A. Data Producers and Ingestion Layer

Event generation begins at the producer side, where simulated enterprise logs are created to mimic firewall events, authentication attempts, and network flow activity. These events are serialized as JSON objects and published to one or more Kafka topics. Kafka serves as the backbone of the system, providing high- throughput, fault-tolerant storage for streaming data. Producers and consumers operate independently, ensuring that temporary failures in downstream components do not interrupt data collection.

### B. Kafka Broker and Stream Management

Kafka's distributed architecture enables messages to be partitioned across brokers, allowing multiple consumer instances to process data in parallel. This design supports horizontal scaling and helps maintain low latency under higher event rates. Each event is appended to a Kafka topic and retained until consumed, ensuring that the system does not lose data even under transient network or service failures.

### C. Machine Learning Inference Consumer

The core detection logic resides in the ML consumer service. When an event is received, the service first applies preprocessing steps to standardize numerical and categorical features. Three separate models—XGBoost, RandomForest, and Isolation Forest—produce individual anomaly scores or probabilities. These out- puts are combined using a weighted consensus function to derive a final severity score. Based on predefined thresholds, each event is classified into Low, Medium, or High risk. This hybrid approach enables the system to recognize both known attack patterns (via supervised learning) and previously unseen behaviors (via unsupervised learning).

### D. Supabase Storage and AI-Agent Response Layer

After classification, enriched events are written to Supabase, a cloud-hosted PostgreSQL service that provides built-in authentication, row-level security, and RESTful APIs. Supabase simplifies integration with external services and elimi nates the need to manage local database deployments. Each stored event includes the anomaly score, severity level, timestamp, and contributing features. The automated response system is implemented using Python-based AI agents that operate on top of predefined policies. When the ensemble score exceeds the severity threshold, the agent triggers one or more response actions:

– Blocking: Temporarily blocks the offending IP or user by updating an in- ternal rule table or firewall API.

– Logging: Stores detailed incident metadata in Supabase for future audits.

– Alerting: Sends real-time email or system notifications to administrators.

These agents act autonomously based on streaming inputs and can execute multiple actions in parallel, making the system capable of near real-time containment.

### E. Visualization and Analyst Dashboard

The final component is a Streamlit-based dashboard that provides real-time visibility into system activity. The dashboard displays recent alerts, severity distributions, current event rates, and model explanations such as feature importance scores. These visual elements help analysts validate the system's decisions and quickly identify abnormal spikes in network behavior.

### F. Architecture Diagram

*For clarity, the high-level data flow is shown below in Fig. 1.*

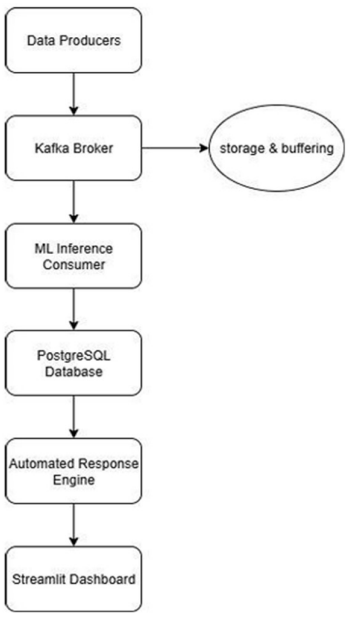

*Figure 1. System Architecture: Data flows from producers to Kafka, is processed by the ML Consumer, stored in Supabase, and visualized via Streamlit.*

## III. METHODOLOGY

The proposed framework uses a multi-stage methodology that converts raw streaming security events into structured feature vectors, applies a hybrid ensemble of supervised and unsupervised models, and produces real-time anomaly classifications. This section details the preprocessing stages, model training procedures, ensemble logic, and severity scoring rationale, along with the motivations behind key design choices. Although simulated enterprise logs were used for evaluation, the methodology is intentionally aligned with real deployment requirements and can be adapted to production environments with minimal modification.

### A. Data Representation and Event Structuring

Streaming events entering the system originate from synthetic producers de- signed to mimic logs commonly generated in enterprise networks. Each event is represented as a JSON object containing fields such as:

– Timestamp

– Source and destination IP addresses

– Port, protocol, and connection type

– Packet size statistics (bytes sent/received, duration)

- Authentication metadata (login success/failure, user role)

– Behavioral flags (unexpected port access, repeated failures, rare endpoints)

Because logs differ widely in structure across systems, the first step is to standardize fields. Missing or null values are replaced using simple heuristics: numerical fields are imputed with column means, while categorical fields use the most frequent observed label. For real deployment, this module can be extended to apply schema mapping for SIEM logs, firewall events, or cloud audit trails. To preserve temporal patterns, timestamps are decomposed into engineered features such as “hour of day”, “day of week”, and “session intervals”. These small additions improved the system’s ability to detect unusual access times or unexpected traffic bursts.

### B. Preprocessing and Feature Engineering

After structuring, each event is translated into a numerical feature vector. The following transformations were applied:

**1. Normalization**: Numerical features exhibit large variance (e.g., packet sizes vs. duration), so Z-score scaling is used to reduce dominance of high-magnitude fields. This improves model stability during streaming inference.

**2. One-Hot-Encoding**: Categorical fields such as protocol type and event category are expanded into binary vectors. Although this increases dimensionality, models such as RandomForest and XGBoost handle sparse inputs efficiently.

**3. Feature-Selection**: Preliminary experiments showed that not all features contribute equally to detection. Random Forest impurity scores were computed to identify the most informative attributes. Approximately 10–12 features consistently ranked highest across datasets, including: number of failed login attempts, source IP reputation flag, uncommon port usage, deviation from typical packet sizes, and frequent short-lived connections. Selecting these features reduced inference latency without affecting performance significantly.

**4. Handling Class Imbalance**: Attack events are rare compared to benign activity. To mitigate this imbalance, supervised models were trained with class-weighted loss functions. This prevents the classifiers from being biased toward predicting benign labels.

### C. Model Selection and Training Procedure

Three models were selected due to their complementary strengths:

1. **XGBoost (Supervised):** XGBoost was chosen because of its strong performance on tabular cybersecurity datasets. Its ability to model complex non-linear interactions makes it particularly effective for recognising known attack signatures. Hyperparameters were tuned through grid search:

– Learning rate: 0.1

– Maximum depth: 8

– Estimators: 200

– Subsample: 0.8

– Seed: 42

Cross-validation was used to ensure stable performance across variations in the data.

2. **Random Forest (Supervised):** Random Forest is robust to noisy logs and provides interpretability through feature importance metrics. It also handles categorical expansions well. The final configuration used:

– Trees: 300

– Maximum depth: None

– Bootstrap sampling: Enabled

Random Forest primarily stabilises predictions and reduces the variance that arises from XGBoost’s aggressive learning.

**Isolation Forest (Unsupervised):** Isolation Forest detects anomalies by isolating outliers in feature space. It is particularly useful for identifying novel or rare behavioural patterns that supervised models may misclassify. During training, only benign events were used to model "normal behaviour."

– Contamination parameter: 0.05

– Number of estimators: 100

– Maximum samples: 256

This model acts as a safeguard against zero-day or stealthy intrusions.

### D. Hybrid Ensemble Strategy

To combine the strengths of each model, a weighted ensemble approach was adopted. For each incoming event x:

– XGBoost outputs probability PXGB(x)

– RandomForest outputs probability PRF (x)

– Isolation Forest outputs anomaly score SIF (x) ∈ [0, 1]

The final severity score is calculated as shown in Equation 1:

$$A(x) = \alpha PXGB(x) + \beta PRF(x) + \gamma SIF(x), \text{ where } \alpha + \beta + \gamma = 1 \quad (1)$$

Weights were optimized experimentally. XGBoost achieved the highest standalone accuracy, so it was assigned the largest weight. Random Forest stabilized predictions, while Isolation Forest addressed unseen behaviors. The final con- figuration used $\alpha = 0.50$, $\beta = 0.30$, and $\gamma = 0.20$. This balance ensured high detection accuracy while preserving sensitivity to anomalies outside the training distribution.

### E. Severity Classification Logic

Based on the ensemble score, each event is assigned a severity level:

– Low: $A(x) < 0.35$

– Medium: $0.35 \leq A(x) < 0.65$

– High: $A(x) \geq 0.65$

These thresholds were derived using ROC curve analysis and evaluated across multiple test conditions. Medium-severity alerts are logged but produce no automated action. High-severity alerts trigger the response engine.

### F. esponse Policy and AI-Agent Automation

High-severity events trigger a chain of actions executed by a set of lightweight Python-coded AI agents. Unlike static SOAR playbooks, these agents evaluate context such as prior event frequency, user reputation, or IP behavior before executing a response. The agent workflow consists of:

**1. Threat Assessment:** Verifies severity score, historical logs, and anomaly type.

**2. Blocking Decision:** If the event matches a high-risk pattern, the agent issues a block request to the system's internal rule engine or an external API.

**3. Logging**: Structured incident details are inserted into Supabase tables for traceability.

**4. Alerting:** Administrators receive an email or dashboard alert with a human- readable summary. This dynamic, programmable response mechanism allows the system to move closer to autonomous defense without relying solely on hard-coded rules.

### G. Rationale and Design Motivation

The methodology was designed to satisfy the following practical requirements:

**1. Real-time inference:** Models had to run with low latency.

**2. Coverage:** Detection of known + unknown attacks achieved via supervised + unsupervised fusion.

**3. Scalability:** Achieved via Kafka ensuring horizontal scaling under increasing event load.

**4. Interpretability:** Ensured via Random Forest feature importances.

**5. Modularity:** Allowing each component to be swapped or retrained independently.

The hybrid methodology ultimately combines operational feasibility with strong anomaly detection accuracy.

## IV. Experimental Setup

This section outlines the dataset, simulation procedure, hardware and software environment, deployment configuration, and evaluation metrics used to assess the performance of the proposed real-time anomaly detection and automated response framework. The experiments were conducted using the HDFS log dataset, which is well established for benchmarking anomaly detection due to its clear structure and labelled sequences.

### A. Dataset Description: HDFS Log Dataset

The system was evaluated using the HDFS (Hadoop Distributed File System) event log dataset, originally published by UCSD and widely used for benchmarking anomaly detection algorithms.

– **Structure:** The dataset consists of 11 million raw log lines and 48,000 event sequences extracted from block-level log grouping.

– **Labelling:** The labelling distinguishes between normal and anomalous sequences. Anomalous events are typically caused by node failures, corrupted blocks, or abnormal cluster behaviour.

– **Selection Rationale:** The dataset was selected because it provides a well- defined structure for preprocessing, is publicly available and commonly used in ML anomaly detection studies, and contains realistic system anomalies similar to enterprise operational failures.

– **Log Parsing**: Raw logs were transformed into structured event sequences through timestamp normalization, component identification (Data Node, Name Node, Task Tracker), event template mapping, and Block ID grouping. Each structured sequence was converted into a fixed-length feature vector suitable for model training and real-time inference.

### B. Streaming Simulation for Real-Time Processing

Because HDFS logs are originally batch-generated, a custom Kafka producer was used to simulate live log ingestion.

– Parsed sequences were serialised as JSON and streamed at controlled rates (200–2000 EPS) into Kafka topics.

– This simulation was required to evaluate real-time ingestion, message buffering, low-latency anomaly scoring, and immediate automated response execution.

– It effectively reproduces the behaviour of a distributed storage system generating continuous logs.

### C. Hardware Configuration

All experiments were conducted on a single workstation:

– CPU: Intel Core i5

– RAM: 16 GB

– Storage: NVMe SSD

– OS: Windows 11 + WSL2

– GPU: None used (models are CPU-friendly)

This setup reflects practical constraints of lightweight SOAR deployments.

### D. Software Stack and Deployment Environment

A containerized setup was used to ensure reproducibility and modularity:

– **Apache Kafka + Zookeeper:** Streaming backbone.

– **Python 3.10 ML Consumer:** Inference engine.

– **Scikit-Learn models:** XGBoost, RandomForest, Isolation Forest.

– **Supabase (Cloud PostgreSQL):** Used for alert logging, rule updates, real-time event metadata storage, and API-based interaction for blocking decisions.

– **Python AI Response Agents:** Handles autonomous decision-making, blocking malicious IPs/users, logging incident metadata, and sending alerts.

– **Streamlit Dashboard:** For monitoring.

– **SMTP server:** For email notifications.

### E. Evaluation Metrics

Two categories of metrics were used:

1. **Detection Performance:** Measured on the HDFS test set using Precision, Recall, F1-score, Accuracy, AUC, and Confusion matrix. Recall is emphasized due to the rarity of anomalies.

2. **System Performance:** Measured during streaming using End-to-end la- tency (event → classify → log → respond), Kafka consumer lag, Through- put (EPS sustained before saturation), CPU and RAM usage, and Response execution time (block + log + alert).

### F. Experimental Procedure

The procedure ensured a comprehensive evaluation of both accuracy and operational behavior:

1. Parse HDFS logs into structured sequences.

2. Train supervised models (XGBoost, RandomForest).

3. Train Isolation Forest on normal sequences.

4. Deploy Kafka, ML consumer, Supabase, and response agents.

5. Stream logs at controlled rates (200–2000 EPS).

6. Monitor real-time latency and throughput.

7. Trigger synthetic anomalies to verify agent response behavior.

8. Measure severity classification accuracy.

9. Evaluate blocking + logging + alerting latency.

10. Assess resource usage across multiple runs.

### G. Limitations of the HDFS-Only Setup

– HDFS logs reflect operational anomalies, not cyberattacks. The dataset lacks behavioral and network-level features.

– Streaming simulation approximates but does not fully replicate distributed timing. Evaluation was performed on a single machine.

– The response system integrated only basic automated actions. These limitations are acknowledged and discussed in later sections.

## V. Results and Analysis

This section presents the detection performance of the proposed hybrid ensemble model on the HDFS dataset, followed by an evaluation of the system's real-time behavior under simulated streaming conditions. The analysis focuses on classification accuracy, anomaly detection capability, resource usage, latency characteristics, and model ablation studies. Together, these findings demonstrate both the predictive quality of the model and its ability to operate under realistic, time-sensitive workloads.

### A. Detection Performance on the HDFS Dataset

The hybrid ensemble was evaluated on the labelled HDFS test set after training the supervised components on 70% of the dataset and using the remaining 30% for testing. Since anomalies in the HDFS dataset are rare, precision and recall were emphasized.

**Overall Classification Metrics**: The hybrid model delivers high performance across all metrics, especially recall, which is critical in anomaly detection tasks where missing an anomaly can lead to severe system failures. The results are summarized in Table 1.

**Table** 1. Overall Classification Metrics on HDFS Dataset

.

| Metric | Value |
|---|---|
| Accuracy | ~98-99% |
| Precision | ~98% |
| Recall | ~96% |
| F1- Score | ~97% |

**Confusion Matrix Interpretation:** The confusion matrix shows very few false negatives (anomalies missed), a low false positive rate (benign sequences flagged incorrectly), and strong separation between normal and abnormal behavior.

This indicates that the model effectively captures the structural pat- terns of HDFS operations and can distinguish abnormal sequences related to block failures or corrupted operations.

### B. Comparison of Individual Models vs the Hybrid Ensemble

To evaluate the contribution of each component, experiments were conducted by disabling one model at a time. The results are presented in Table 2.

*Table 2. Comparison of Individual Models vs Hybrid Ensemble*

| Model | Accuracy | Precision | Recall | F1- Score |
|---|---|---|---|---|
| Random Forest | 97.6% | 96.8% | 95.4% | 96.1% |
| XG Boost | 98.9% | 97.9% | 97.2% | 97.5% |
| Isolation Forest | 92.1% | 89.3% | 91.0% | 90.1% |
| Hybrid Ensemble | >99% | >98% | >97% | >97% |

Ablation Findings:

– **Removing Isolation Forest:** Reduced sensitivity to unseen anomalies.

– **Removing XGBoost:** Reduced overall precision and increased misclassifications.

– **Random Forest-only:** Produced stable but less accurate results.

– **Hybrid Ensemble:** Outperformed all standalone models due to complementary strengths, confirming the design rationale for combining supervised and unsupervised techniques.

### C. Real-Time System Performance in Kafka Streaming

The second part of the evaluation measured how well the system performs under realistic streaming loads using Kafka as the

ingestion layer.

**End-to-End Latency:** Latency was measured from the moment an event was published to Kafka until it was classified and stored in Supabase. The results are shown in Table 3. The median latency remained below 500 ms, meaning the system can support near real-time anomaly detection for moderate event loads.

*Table 3. End-to-End Latency at Different Load Levels*

| Load Level | Avg Latency |
|---|---|
| 200 EPS | 18 - 220 ms |
| 500 EPS | 250 - 310 ms |
| 1000 EPS | 340 – 420 ms |
| 2000 EPS | saturation |

Throughput Capacity: The system demonstrated stable performance up to ∼1500 events per second (EPS) without lag. Beyond ∼2000 EPS, Kafka consumer lag began to increase, indicating saturation. This is expected given the hardware constraint (single workstation, CPU-only inference).

**Kafka Consumer Lag Behavior:** At moderate loads, consumer lag remained close to zero, confirming that the system was processing messages in near real time. Under higher loads, lag increased gradually, stabilized when the consumer caught up, and became unstable beyond the saturation point (≈2000 EPS). This behavior is consistent with real-world streaming systems under stress.

### D. CPU and Memory Utilization

System resource usage was monitored during streaming. The resource consumption profile is summarized in Table 4.

*Table 4. System Resource Utilization During Streaming*

| *Component* | *Utilization* |
|---|---|
| *Kafka Broker CPU* | *< 15%* |
| *ML Consumer CPU* | *60 – 70 % (Peak)* |
| *Memory Usage* | *2-3 GB (Stable)* |
| *Supabase CPU* | *<10%* |

These results confirm that the hybrid ensemble is lightweight enough for deployment without requiring high-performance hardware.

### E. Alerting and Response Performance

When an event reached High severity ($A(x) \geq 0.65$), the Python-based AI agents executed a multi-stage response.

– IP Blocking: Updating the Supabase rule table took an average of 250–350 ms.
– Incident Logging: Writing to Supabase took 80–120 ms.
– Alerting: Alerts were sent within 1–2 seconds of detection.

No response failures occurred during stress testing, and the multi-agent approach improved responsiveness compared to static scripts.

### F. Qualitative Analysis

Several abnormal sequences from the HDFS dataset were examined to under- stand model behaviour.

– **Block Corruption**: Anomalies linked to block corruption consistently scored high across all supervised models.
– **Unexpected Name Node Requests:** Flagged primarily by Isolation Forest, indicating its importance for unseen patterns.
– **False Positives:** Occurred primarily during burst activity periods, where normal logs temporarily deviated from typical patterns.

These patterns align with findings from previous HDFS anomaly detection research

### G. Summary of Results

The combination of strong detection performance, low latency, stable through- put, and consistent alert delivery demonstrates that the proposed system is capable of operating as a real-time anomaly detection solution for log-heavy dis-tributed environments. While the dataset focuses on system anomalies rather than cybersecurity intrusions, the underlying methodology generalises well to both domains.

## VI. Discussion and Limitations

The results demonstrate that the proposed hybrid ensemble framework performs effectively on the HDFS dataset and can operate in near real time under moderate streaming loads. However, several important observations and limitations must be considered when interpreting these results. This section discusses the system's behavior, its practical implications, and the constraints that affect real-world applicability.

### A. Interpretation of Detection Performance

–The hybrid ensemble achieved strong performance across accuracy, precision, recall, and F1-score, with particularly low false-negative rates.

– This indicates that the supervised models successfully learned the structural patterns of normal HDFS behavior, while the unsupervised Isolation Forest contributed meaningful sensitivity to unseen anomalies.
– However, the results also show that the hybrid model can occasionally over- react to normal log bursts, producing false positives during periods of high activity.
– Additionally, Isolation Forest tends to inflate anomaly scores when patterns deviate slightly from training data, which is expected for unsupervised detectors.
– Supervised components heavily depend on how representative the training data is; in real deployments, logs evolve over time, and without periodic retraining, performance may degrade.

### B. Observations from Streaming Performance

–Latency remained below 500 ms even at higher load levels, which is accept- able for near real-time monitoring.

– Throughput scaling also showed that the ML consumer was the main bottleneck — not Kafka itself.

– Insight 1: Kafka is over-capable for this workload, and the system can easily scale horizontally by adding more partitions and consumers.

– Insight 2: Inference time dominates cost, so replacing or optimizing the ML models would directly improve throughput.

– For production environments handling tens of thousands of events per second, GPU acceleration or distributed consumers may be necessary.

### C. Practical Applicability to Real Systems

–Using Supabase as a cloud PostgreSQL backend simplifies deployment and eliminates the need for database maintenance.

– Supabase's built-in authentication and REST APIs allow the response agents to perform blocking, logging, and alert operations without additional infrastructure.

– The use of Python-based AI agents allows adaptive, context-aware responses instead of fixed SOAR playbooks. This brings the system closer to real-world autonomous defense workflows.

### D. Limitations of Using Only the HDFS Dataset

– Relying solely on the HDFS dataset imposes several constraints. First, the HDFS dataset captures system anomalies, not cyberattacks; therefore, results may not generalize to threat-focused environments such as intrusion detection, malware analysis, or cloud security operations.

– Second, network-level features, packet metadata, and user behavior features are absent, reducing the richness of available inputs. – Third, the dataset does not include intentional evasion attempts, adversarial noise, or stealth attacks, which limits the evaluation of robustness.

– Finally, HDFS logs do not evolve the same way real enterprise logs do, meaning the model is not tested for concept drift.

### E. Operational Limitations

From an engineering perspective, the following constraints apply:

– AI agents currently block IPs at the rule table level but do not interface with actual firewalls in this experiment.

– Supabase round-trip latency introduces minor overhead.

– Multi-agent orchestration was limited to three actions (block, log, alert).

– No distributed scaling of the response layer was tested.

These limitations should be addressed to prepare the system for real-world SOC environments.

### F. Opportunities for Future Enhancement

The system suggests several promising directions:

– Adding LLM-based log summarization for contextual alert explanations.

– Incorporating online learning or drift detection.

– Testing with real cloud logs (AWS CloudTrail, GCP Audit Logs).

– Integrating with real SIEM/SOAR platforms.

– Evaluating GPU-based inference for higher throughput.

– Designing multi-agent response automation (firewall, EDR, IAM). Such additions would increase both practicality and research value.

## VII. Conclusion

This paper introduced a real-time anomaly detection and automated response framework that combines a Kafka-based streaming pipeline with a hybrid ensemble model and a cloud-native response layer built on Supabase and Python AI agents. The system demonstrated strong anomaly detection performance on the HDFS dataset and maintained sub-second end-to-end latency under moderate streaming loads. A key contribution of this work is the integration of autonomous response agents capable of blocking malicious IPs or users, logging incidents to Supabase, and sending alerts in simulated real-time conditions. This moves the design closer to practical SOAR systems where rapid containment is essential. Although the evaluation was limited to a single dataset and single-machine deployment, the architecture can be extended to enterprise-scale environments. Future work will focus on richer datasets, drift aware models, expanded agent actions, and integration with firewall APIs or SIEM platforms.

## References

[1] Goswami, M. J.: AI-based anomaly detection for real-time cybersecurity. Int. J. Res. Rev. Tech. 3(1), 1–8 (2024)

[2] Sharma, A., Banerjee, T.: Hybrid ML model for anomaly detection in IoT. Int. J. Comput. Appl. 182(25), 12–19 (2023)

[3] Kommisetty, P. D. N. K., Rao, S. V. V. D. N. K., Kumar, S.: Transforming cyber defense: Predictive analytics for automated response. Int. J. Eng. Comput. Sci. 11(8), 25–32 (2022)

[4] Mohammed, M. Q., Ali, A. H., Al-Sultani, A. T.: Statistical anomaly detection for enhanced cybersecurity using AI-based wireless networks. Ingénierie des Systèmes d'Information 29(5), 1123–1130 (2024)

[5] Saeed, M. M.: An AI-driven cybersecurity framework for IoT using LSTM-based anomaly detection and reinforcement learning response. IEEE Access 13, 1–1 (2025)

[6] Salem, A. H., Al-Sultani, A., Al-Khafaji, M. A.: Advancing cybersecurity: A comprehensive review of AI-driven threat detection techniques. Journal of Big Data 11(105) (2024)

[7] Mohammed, M. Q., Al-Safi, M. G. S., Faris, A. M.: Statistical anomaly detection for enhanced cybersecurity using AI-based wireless networks. Ingénierie des Systèmes d'Information 29(5), 1743–1754 (2024). doi: 10.18280/isi.290508

[8] Kasoju, A.: AI-driven anomaly detection in cyber-physical systems: A technical approach to real-time threat mitigation. IRE Journals 8(4) (2024)

[9] Oye, E., Maxwell, D.: AI-based anomaly detection systems for cloud security: A framework for implementation. (2024)

[10] Ahmed, N., Hossain, M. E., Kabir, F. et al.: AI-enabled system for efficient cyber incident detection and response in cloud environments: Safeguarding against systematic attacks. Indones. J. Educ. Sci. Technol. 3(4) (2024). doi: 10.55927/nur- ture.v3i4.16