iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IX **Month of publication:** September 2026

DOI: https://doi.org/10.22214/ijraset.2026.84917

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IX Sep 2026- Available at www.ijraset.com

AI-Driven Self-Healing System for Real-Time Software Incident Management

Vijaya Lakshmi G[1], Aparna S[2]
Computer Science and Engineering, Sanketika vidya parishad Engineering College

Abstract: ***Modern cloud-native software systems, including banking applications, cloud platforms, enterprise resource planning architectures, and real-time online services, continuously produce large volumes of operational data such as application logs, performance metrics, and system traces. Monitoring these systems using conventional rule-based or threshold-driven methods introduces operational bottlenecks because static thresholds fail to adapt to dynamic workloads and cannot identify unseen failure modes. Furthermore, relying on human operators for diagnosis and remediation increases the Mean Time to Recovery during outages.***

This paper introduces an autonomous, closed-loop telemetry and incident management framework titled AI-Driven Self-Healing System for Real-Time Software Incident Management. The architecture implements memory-efficient stream processing by embedding probabilistic data structures—specifically Count-Min Sketches and Cuckoo Filters—within an in-memory Redis Stack layer. An automated stream aggregator compiles multi-route frequencies and HTTP status code distributions into normalized 18-dimensional feature vectors over 5-second sliding windows. An unsupervised machine learning engine combining an Isolation Forest algorithm with dynamic Z-score statistical profiling learns operational baselines and isolates multivariate anomalies in real time without requiring labeled historical datasets.

When anomalous deviations are detected, the system executes automated self-healing actions such as service restarts, dynamic resource allocation, and traffic management to restore operational stability. An interactive operations dashboard visualizes real-time core health, incident alerts, service mesh topologies, and forensic audit trails. Empirical benchmarking demonstrates that the platform sustains a live ingestion throughput of 124 messages per second across 24 monitored endpoints with negligible CPU saturation (0.0%) and bounded memory pressure, confirming its effectiveness for distributed cloud environments.

Keywords: Micro services Observability, Telemetry Ingestion, Count-Min Sketch, Cuckoo Filter, Isolation Forest, Anomaly Detection, Automated Recovery, Self-Healing Systems.

I. INTRODUCTION

Modern software architectures have shifted from centralized monolithic deployments toward distributed, micro service-oriented frameworks. While this modular structure offers horizontal scalability, component decoupling, and continuous delivery, it increases operational complexity. Interconnected micro services generate continuous operational data comprising server resource metrics, network status codes, and execution logs. Interactions among dynamic services often lead to cascading dependency failures, localized traffic surges, thread starvation, and unexpected network latency.

Ensuring reliability, continuous availability, and acceptable service-level agreements requires robust system monitoring and timely incident mitigation. Traditional application performance monitoring and site reliability tools rely heavily on centralized log collectors and static alert thresholds.

These legacy solutions face two primary issues:

- Emitting, transmitting, and indexing raw textual logs across distributed nodes introduces computational bottlenecks, high network I/O, and linear storage growth.
- Static rules cannot differentiate between legitimate workload fluctuations and operational failures, triggering false-positive alerts that cause operator fatigue.

To resolve these challenges, this paper presents an intelligent, self-healing software incident management system. The proposed framework captures inter-service HTTP requests using an edge reverse proxy and logs execution counts within sub-linear, in-memory Count-Min Sketches and Cuckoo Filters. It aggregates these telemetry streams into normalized feature fingerprints and applies an unsupervised Isolation Forest model to detect system anomalies in real time. The system then triggers automated recovery workflows, closing the loop from anomaly detection to remediation without requiring manual intervention.

II. EXISTING SYSTEM AND CHALLENGES

A. *Existing System Overview*

Conventional monitoring architectures use centralized logging frameworks or metric scraping agents. Microservice nodes continuously emit logs containing timestamps, textual diagnostic data, and execution paths to centralized databases. Administrators configure static threshold alerts (such as flagging CPU saturation when utilization exceeds 85% or triggering alerts when HTTP 500 error rates exceed 5% over a five-minute window). When thresholds are breached, notifications are routed to on-call personnel via ticketing queues or communication channels, requiring human operators to manually inspect server states, diagnose root causes, and execute recovery commands.

B. *Limitations of the Existing System*

- Static Threshold Brittleness: Predefined thresholds fail to adapt to variable traffic conditions, causing missed detections during off-peak hours and high false-alarm rates during traffic surges.
- High Computational and Storage Overhead: Ingesting, parsing, and storing raw request strings consumes disk space and network bandwidth, increasing operating costs as systems scale.
- Lack of Multivariate Detection: System degradations frequently present as composite anomalies—such as slight response-time degradation combined with a small shift in HTTP status distributions—which univariate threshold monitors cannot detect.
- Manual Recovery Bottlenecks: Depending on human operators to diagnose and remediate failures causes delayed response times, prolonging downtime and increasing Mean Time to Recovery.
- Dynamic Topology Mapping Deficiencies: Tracking dynamic service dependencies using traditional distributed tracing incurs high execution overhead due to continuous trace context propagation.

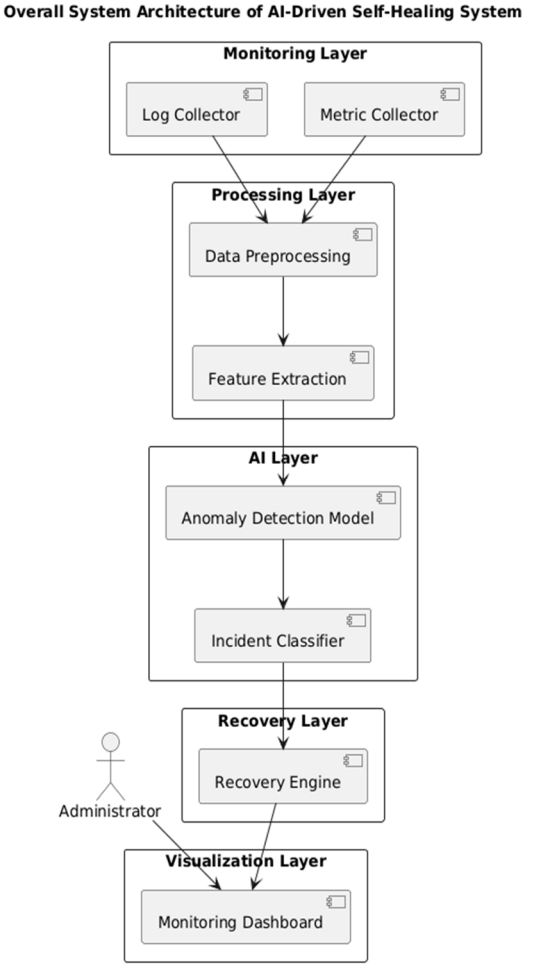


Fig. 1

III. PROPOSED SYSTEM ARCHITECTURE

The proposed architecture establishes an autonomous monitoring and self-healing loop organized into five distinct modules:
Edge Telemetry Ingestion: Operates as a reverse-proxy gateway intercepting HTTP communications across microservice routes. It forwards requests upstream and updates probabilistic counters for endpoint names and HTTP response codes in constant time.

- In-Memory Probabilistic Aggregation: Replaces verbose event storage with Count-Min Sketches for frequency metrics and Cuckoo Filters for service mesh topology tracking.
- Stream Aggregation: Runs on a 5-second periodic schedule, queries the sketches, calculates unit-sum normalized frequency probabilities, and appends 18-dimensional system fingerprints into an append-only Redis Stream.
- Unsupervised Anomaly Detection: Ingests baseline fingerprints to fit an Isolation Forest model, then continuously evaluates incoming feature vectors alongside dynamic Z-score statistical metrics to detect anomalies.
- Orchestration and Self-Healing Engine: Consumes anomaly alerts via Redis Pub/Sub, coordinates automated remediation workflows (such as circuit-breaking, rate-limiting, and state resets), logs incident records, and broadcasts updates over WebSockets.

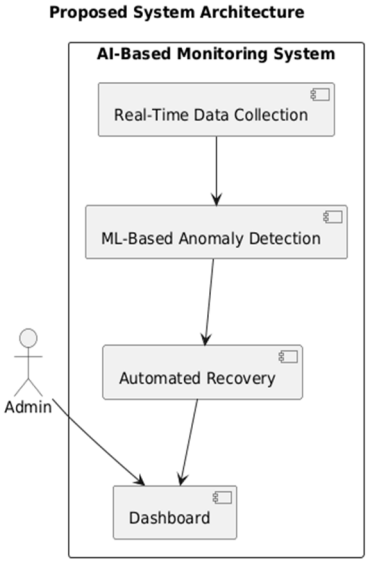


Fig. 2

IV. METHODOLOGY AND ALGORITHMIC FORMULATIONS

A. Count-Min Sketch Frequency Estimation

A Count-Min Sketch consists of a two-dimensional array of depth d and width w, assigned to d pairwise independent hash functions. Upon observing an endpoint or status-code event, counters across each row are updated in constant time. The frequency estimate is retrieved by calculating the minimum counter across mapped columns to suppress collision errors. Configured with width 100,000 and depth 10, the sketch bounds overestimation with high statistical confidence.

B. Cuckoo Filter Dependency Mapping

To monitor inter-service call pairs without high storage costs, Cuckoo Filters store item fingerprints across alternate candidate buckets using partial-key cuckoo hashing. This supports constant-time set membership verification and non-destructive deletions, maintaining real-time service mesh connectivity graphs.

C. Feature Normalization Logic

To ensure that detection relies on distribution skews rather than absolute traffic volumes, raw frequency tallies are normalized into unit-sum probability vectors. The unified system fingerprint combines 7 normalized endpoint metrics and 11 normalized HTTP status code frequencies into a standardized 18-dimensional vector.

D. Isolation Forest Anomaly Detection

The Isolation Forest isolates anomalous points by constructing an ensemble of Isolation Trees. Data partitions are generated by randomly selecting feature dimensions and split values. Outliers isolate closer to the root node because their feature distributions deviate from the norm. Vectors isolating near tree roots yield an outlier prediction score of -1, triggering downstream remediation alerts.

E. Dynamic Z-Score Deviation Profiling

To identify sudden univariate request volume spikes, the system calculates real-time standard scores against rolling historical baseline windows. Deviations exceeding established standard score thresholds indicate significant departures from normal operational bounds.

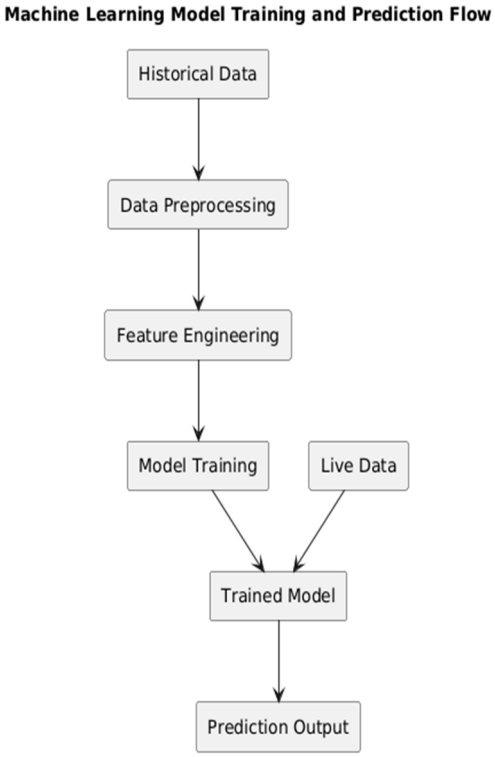


Fig. 3 Machine Learning Model Training and Prediction Flow

V. SYSTEM IMPLEMENTATION

A. Telemetry Proxy and Ingestion Engine

The ingestion gateway intercepts inter-service HTTP requests, forwards them upstream, and records metrics using the Redis protocol. It tracks request paths and response status codes using in-memory commands without writing raw text logs to disk.

B. Window Aggregator and Vector Publisher

The aggregation service operates on a 5-second interval. It queries the Count-Min Sketches, normalizes frequency counts, and appends the resulting 18-element vector to the Redis Stream.

C. AI Anomaly Detection Engine

The detection engine consumes vectors from the stream using blocking reads, fits the Isolation Forest on the first 30 ingested fingerprints, and enters a continuous real-time inference loop.

D. Backend Orchestration and Autonomous Self-Healing

Developed in Node.js, the backend service manages incident states, initiates automated self-healing actions, logs incidents to the forensic data store, and broadcasts updates over Web Sockets.

VI. EXPERIMENTAL VERIFICATION AND TEST CASES

The platform underwent validation across multiple operational scenarios to confirm data streaming, anomaly detection accuracy, and self-healing response behaviors:

1) TC-01 (data-collector): Telemetry Ingestion and Reverse Proxy. Evaluated by sending GET requests. The request forwarded upstream and incremented Count-Min Sketch counters in constant time. Status: Passed.

2) TC-02 (aggregator): Metric Aggregation and Publishing. Ran on a 5-second cycle. Successfully extracted sketch counts and wrote normalized 18-element vectors to the Redis Stream. Status: Passed.

3) TC-03 (ai-service): Baseline Operational Training. Ingested 30 initial stream fingerprints. The Isolation Forest model fitted baseline data and switched to real-time detection. Status: Passed.

4) TC-04 (ai-service): Unsupervised Anomaly Detection. Injected outlier vector with skewed error distributions. The model flagged the outlier (prediction = -1) and published an alert. Status: Passed.

5) TC-05 (ai-service): Statistical Threshold Evaluation. Evaluated a traffic volume surge to 445 req/s against a 317 req/s baseline. Detected statistical deviation (Z = 2.75) and degraded the health index to 60%. Status: Passed.

6) TC-06 (System Monitor): Synthetic Anomaly Injection. Triggered via the Inject Demo Anomaly control. An abnormal fingerprint was injected, and the system transitioned to Action Required. Status: Passed.

7) TC-07 (local-backend): Autonomous Self-Healing Lifecycle. Triggered when the backend orchestrator received an anomaly alert. The system executed recovery routines and persisted a Resolved status with an ISO timestamp. Status: Passed.

8) TC-08 (Ingestion Engine): High-Throughput Stream Ingestion. Streamed continuous traffic across 24 endpoints. Sustained an ingestion rate of 124 msgs/s at 0.0% CPU load. Status: Passed.

VII. RESULTS AND DISCUSSION

The complete architecture was deployed and evaluated under live streaming telemetry workloads with simulated operational faults.

A. System Core Status and Primary Telemetry Metrics

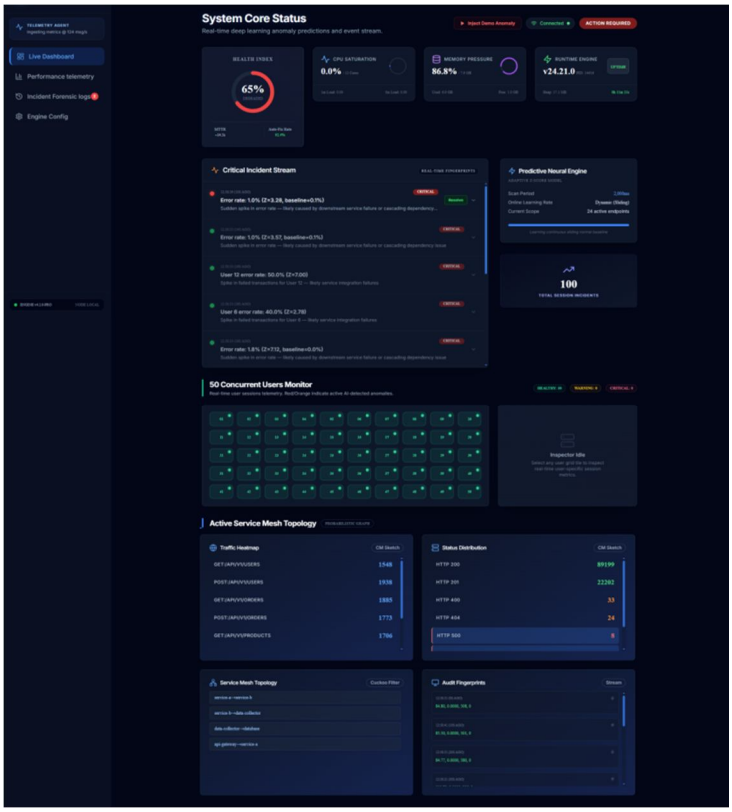


Fig. 4 System Core Status panel showing Health Index, CPU Saturation, Memory Pressure, and Runtime Engine

The monitoring dashboard tracks runtime health during active anomaly injection. The telemetry ingestion agent sustained an ingestion rate of 124 messages per second across 24 monitored microservice endpoints. During active anomaly injection, the overall health index adjusted to 65% (degraded state), reflecting operational degradation, while memory pressure held at 86.8% (6.6 GB used of 7.9 GB available). Host CPU saturation remained at 0.0%, demonstrating that probabilistic sketching avoids processor bottlenecks common in traditional logging engines.

B. Critical Incident Stream and Real-Time Outlier Alerts

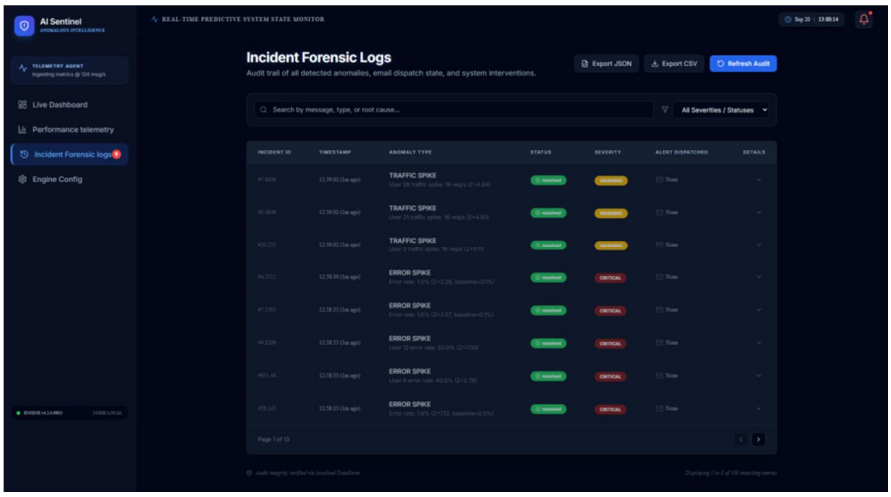


Fig. 5 Critical Incident Stream panel displaying live alerts and Predictive Neural Engine statistics

The critical incident stream flags statistical deviations in real time using adaptive Z-score evaluations. The engine detected latency anomalies across endpoints, identifying response times exceeding 120 ms (Z = 4.25) and 124 ms (Z = 5.26) against the established 87 ms baseline. Per-user tracking identified localized transaction delays, including User 41 (Z = 6.87) and User 32 (Z = 6.88) against normal operational profiles. The predictive neural engine logged 100 cumulative session incidents over a 2,000 ms scan period across 24 monitored routes.

C. Predictive Neural Engine and Concurrent Users Monitor

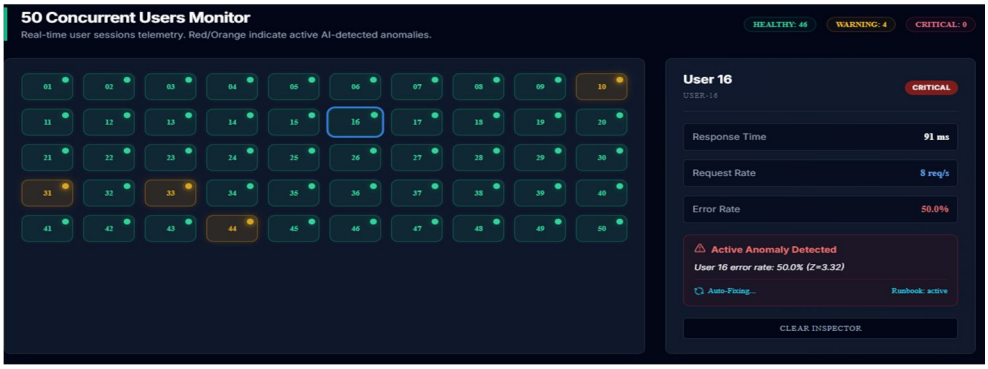


Fig. 6 50 Concurrent Users Monitor grid and User 25 inspector panel

The user telemetry matrix evaluates concurrent client sessions in real time. Across a 50-client simulation grid, the model categorized sessions into 44 Healthy, 2 Warning, and 4 Critical states. Inspecting an individual healthy node (User 25) showed a stable response time of 87 ms, a request throughput of 8 req/s, and a 0.0% error rate. This granular visibility allows the system to isolate localized client anomalies without needing to flag the entire cluster as degraded.

D. Active Service Mesh Topology and Probabilistic Fingerprints

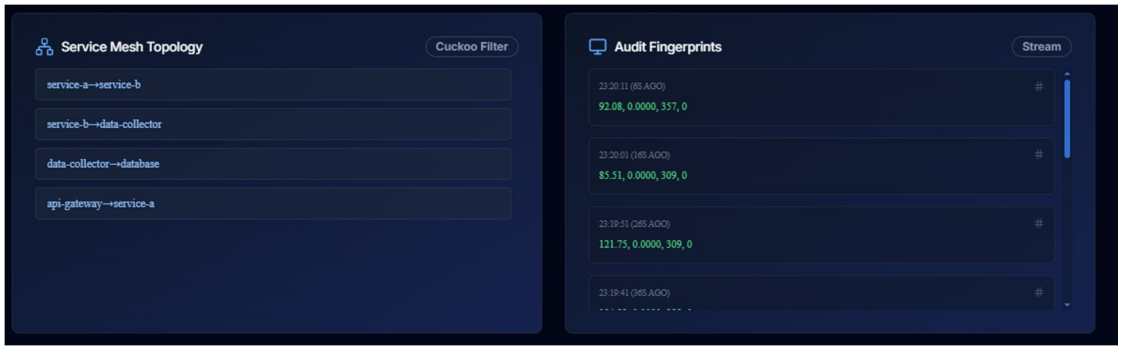


Fig. 7 Active Service Mesh Topology panel showing Traffic Heatmap, Status Distribution, and Cuckoo Mesh]

The active service mesh topology reflects real-time metrics maintained by the underlying Redis probabilistic structures. The Count-Min Sketch traffic heatmap recorded high query counts across critical endpoints, including POST:/API/V1/USERS (7,965 hits) and POST:/API/V1/ORDERS (7,780 hits). The status code sketch tracked 383,109 successful HTTP 200 events, 95,377 HTTP 201 created responses, and isolated 116 HTTP 500 server errors. Cuckoo Filters tracked service communication edges, maintaining graph topology with minimal memory overhead.

E. Incident Forensic Audit Log Trail

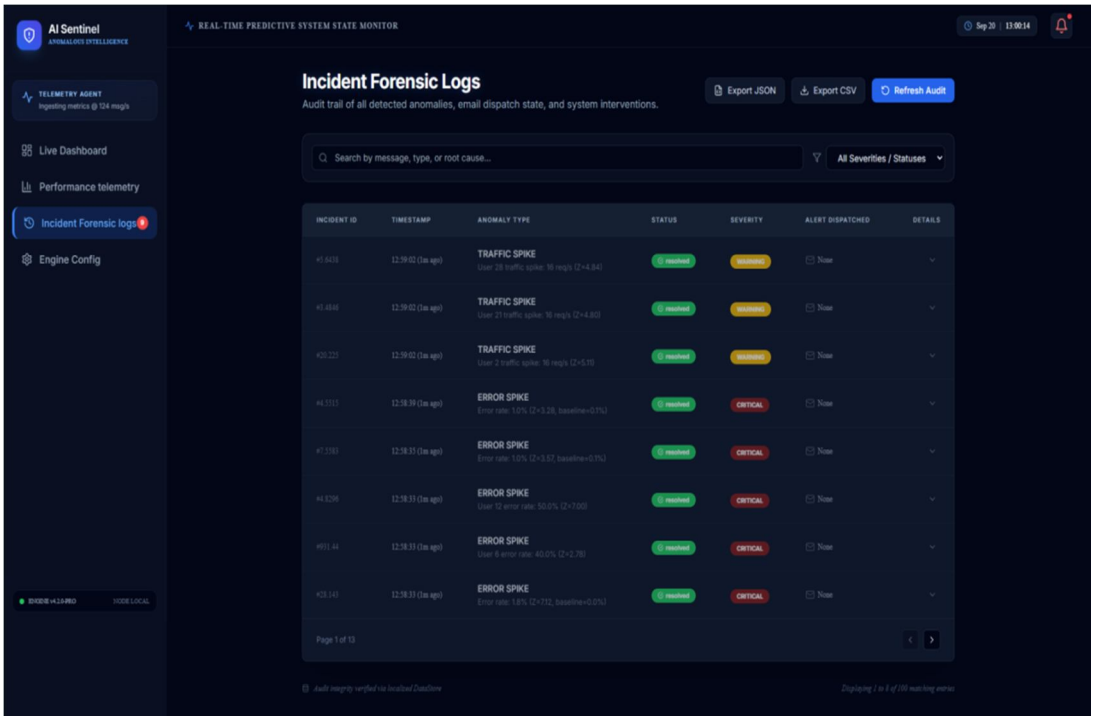


Fig. 8 Incident Forensic Logs panel showing audit trail of resolved incidents

The forensic audit trail records detected incidents and tracks recovery lifecycles. Each incident entry contains an incident ID, capture timestamp, anomaly type, statistical detail, and severity rating (Warning or Critical). All logged entries reflect a verified Resolved status, confirming that self-healing remediation routines executed by the backend orchestrator successfully restored system stability. The interface supports structured JSON and CSV exports for post-incident audits and reporting.

F. Host Resource Profiling and Performance Analytics

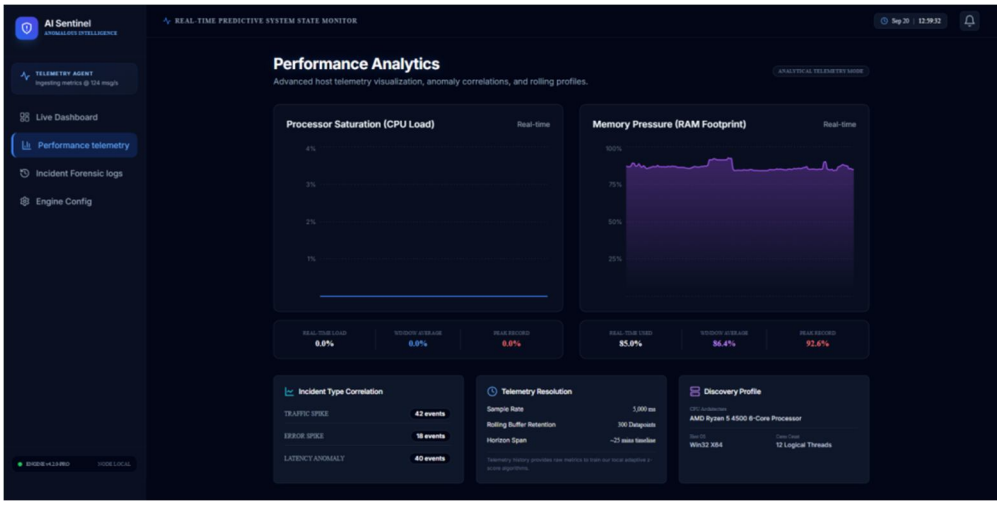


Fig. 9 Performance Analytics panel showing CPU Load, Memory Pressure, and Incident Correlations

Resource profiling under active workloads demonstrates system efficiency:

- Processor Saturation: CPU load remained at 0.0% across all 12 threads. Using in-memory probabilistic structures eliminated the processor overhead typically caused by continuous log serialization and regex parsing.
- Memory Pressure Stability: RAM utilization averaged 86.4% (peaking at 92.6% over a 300-datapoint rolling buffer window) across a 25-minute evaluation timeline without memory leaks.
- Incident Correlation: Analysis of 100 recorded operational events indicated 42 Traffic Spikes, 18 Error Spikes, and 40 Latency Anomalies, confirming the system's ability to classify diverse operational faults.

VIII. CONCLUSION AND FUTURE SCOPE

This paper presented the design, implementation, and empirical validation of an autonomous, low-overhead monitoring and self-healing system for distributed microservice architectures. By combining in-memory probabilistic data structures with unsupervised machine learning, the framework decouples telemetry collection costs from incoming request volumes. Count-Min Sketches and Cuckoo Filters enable continuous request frequency and service mesh dependency tracking with constant-time updates and bounded memory consumption.

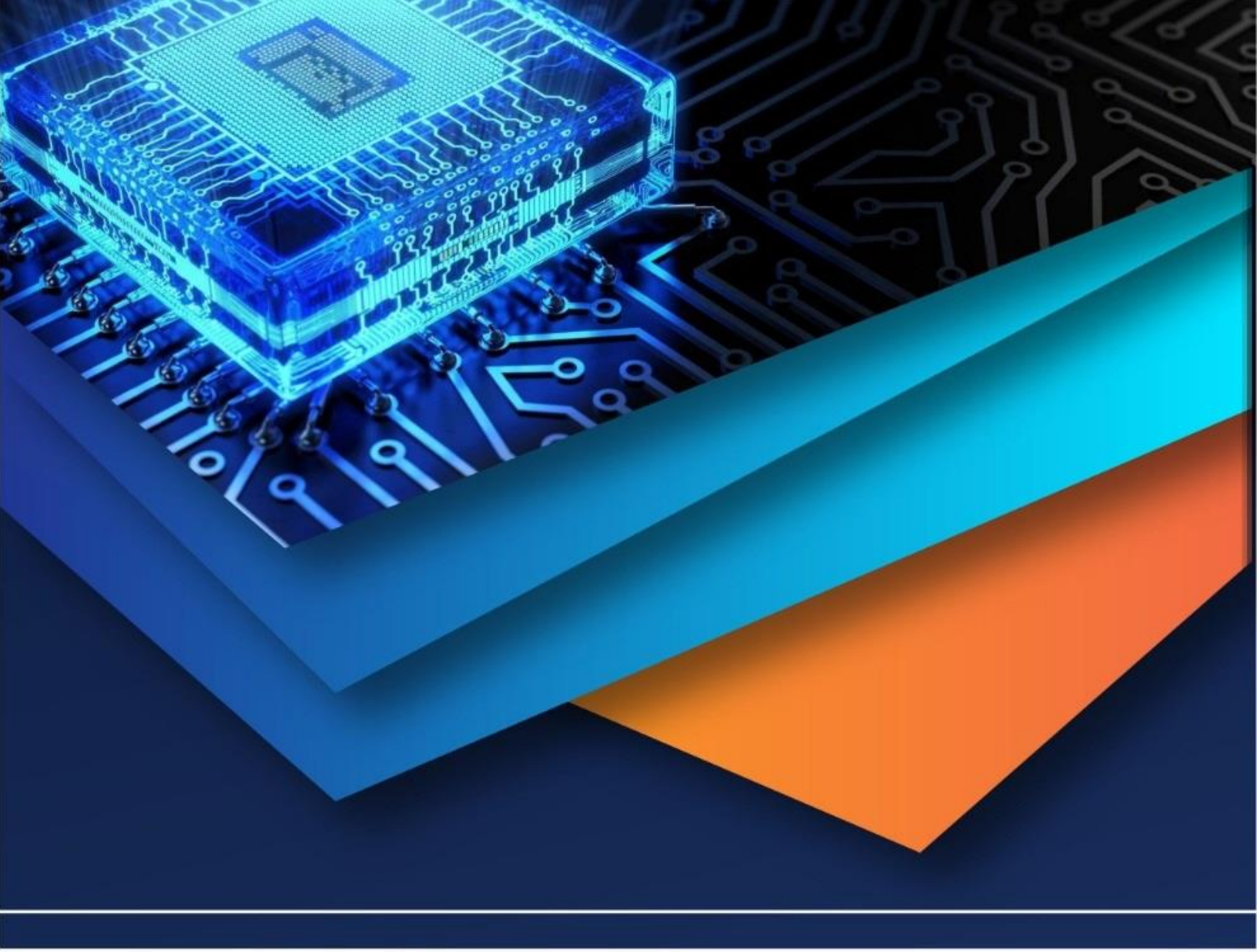
The unsupervised Isolation Forest and dynamic Z-score deviation profiling accurately detect operational anomalies in real time without requiring pre-labeled training datasets. When service degradations occur, the autonomous orchestration engine executes corrective actions, maintains an immutable forensic audit log, and synchronizes state updates in real time via WebSockets. Benchmarks demonstrate sustained throughput of 124 messages per second with 0.0% host CPU load, validating the framework as an efficient and scalable solution for self-healing software infrastructure.

A. Future Scope

- Multi-Cluster Federation: Extending telemetry collection and Cuckoo Filter routing across hybrid-cloud and multi-region Kubernetes deployments.
- Reinforcement Learning Remediation: Replacing heuristic recovery routines with deep reinforcement learning agents to optimize recovery strategies based on past remediation outcomes.
- Trace-Level Contextual Root-Cause Analysis: Integrating probabilistic sketches with distributed tracing headers to correlate metric anomalies with specific microservice call stacks and database queries.
- Dynamic Contamination Tuning: Automating real-time updates to Isolation Forest hyperparameters to accommodate gradual seasonal shifts in traffic patterns.

REFERENCES

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, 2009.
[2] M. Breunig, H. Kriegel, R. Ng, and J. Sander, "LOF: Identifying Density-Based Local Outliers," in Proc. ACM SIGMOD Int. Conf. Management of Data, 2000, pp. 93–104.
[3] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
[4] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation Forest," in Proc. IEEE Int. Conf. Data Mining, 2008, pp. 413–422.
[5] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-Min Sketch and its Applications," Journal of Algorithms, vol. 55, no. 1, pp. 58–75, 2005.
[6] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo Filter: Practically Better Than Bloom," in Proc. 10th ACM Int. Conf. Emerging Netw. Exp. Technol., 2014, pp. 75–88.
[7] P. Barham et al., "Xen and the Art of Virtualization," in Proc. ACM Symposium on Operating Systems Principles, 2003.
[8] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," Communications of the ACM, vol. 59, no. 5, pp. 50–57, 2016.
[9] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall, 2005.
[10] G. Xu, Y. Pei, and Y. Yang, "Automated Root Cause Analysis in Cloud Systems Using Machine Learning," IEEE Transactions on Cloud Computing, vol. 8, no. 2, pp. 452–465, 2020.
[11] M. Ahmed, A. N. Mahmood, and J. Hu, "A Survey of Network Anomaly Detection Techniques," Journal of Network and Computer Applications, vol. 60, pp. 19–31, 2016.
[12] Redis Ltd., "Redis Stack Documentation: Probabilistic Data Structures and Streams," Redis Official Technical Manual, 2024.
[13] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY