NeuroCloud: An AI-Driven Framework for Intelligent Anomaly Detection and Performance Monitoring in a Cloud Environment

Prof. C Hema Prabha[1], Varsha R D[2], Harshitha R[3]
[1,2,3]*Master of Computer Applications, Surana College*
doi.org/10.64643/IJIRTV13I3-207575-459

***Abstract*—The exponential rise in cloud computing infrastructure presents numerous obstacles in maintaining its reliability, performance, and security. Classical approaches for monitoring that depend on hardcoded heuristics and rules are challenging to keep pace with today's data volume, velocity, and variability. This paper presents a novel approach for anomaly detection powered by artificial intelligence algorithms. It employs unsupervised machine learning algorithms, deep learning time-series forecasting, and reinforcement-based reaction strategies to promptly recognise and rectify cloud workload anomalies nearly instantly. The solution integrates Isolation Forest for processing high-dimensional metrics, LSTM Autoencoder models for multiple time-series streams, and One-Class SVM for validation. Experimental evaluation in a multi-tenant cloud environment ensemble voting demonstrates a detection accuracy of 94.1%. The average detection latency decreased from 18.2 to 3.1 minutes over six months of testing. The approach scales linearly up to sixteen worker nodes. Furthermore, the paper discusses challenges associated with false positives and drifts. This research includes suggestions for a federated continual learning strategy.**

***Index Terms*—Anomaly Detection, Cloud Computing, Machine Learning, Isolation Forest, LSTM Autoencoder, One-Class SVM.**

I. INTRODUCTION

Cloud computing is now the main pillar for various digital services such as social media, stock exchanges, and even health care applications. With companies adopting multi-cloud and hybrid cloud strategies, the problem of monitoring the interactions between thousands of interconnected microservices, VMs, and serverless architectures becomes increasingly complex [1], [7]. Transient issues such as small increases in latency or memory consumption may snowball into major outages that can cost corporations hundreds of thousands of dollars each minute [5], [13]. Traditional monitoring tools operate with pre-defined alert thresholds and rigidly defined rules. While acceptable in homogeneous, on-premise infrastructures, the approach presents two essential weaknesses within cloud environments, namely, the inability to cope with changing baselines and false positives in abundance, which result in alert fatigue on the part of operators [6], [15]. An operator faced with dozens of false alerts a day develops a desensitised attitude, thus overlooking real problems until a crisis occurs [4].

Artificial Intelligence (AI) and Machine Learning (ML) provide a convincing solution to the problem. The algorithms of ML learn about the statistical properties of normal patterns through past telemetry and can thus detect minute deviations that would be missed by rule-based systems, while at the same time filtering out harmless variations that produce noise [3], [14]. Such anomaly detection methods as Isolation Forest, LSTM autoencoder, and ensembles have shown excellent results when used in other fields; they will be rigorously implemented and evaluated for a cloud environment [8], [12].

Contributions of this paper include: (1) a complete architecture for AI-based anomaly detection system designed for cloud environments, (2) performance comparison of three different machine learning models based on their precision, recall, and latency performance metrics, (3) a reinforcement-based mitigation strategy to choose mitigation actions from the learned action space, and (4) scalability test

showing nearly linear scalability with respect to increasing numbers of worker nodes [9], [16]. The rest of the paper is organised as follows: Section II defines the problem statement, Section III discusses related work, Section IV presents the proposed methodology, Section V shows implementation details and experimental results, Section VI discusses applications of this approach, Section VII describes limitations, Section VIII suggests possible future work, and Section IX concludes.

II. PROBLEM STATEMENT

Even with advancements in the fields of DevOps automation and Site Reliability Engineering (SRE), most of the cloud infrastructure used for production purposes relies on human intervention when it comes to anomaly detection and analysis. If a containerised service starts leaking memory or a message queue is under unexpected pressure, the normal course of events is for automated alerts to trigger, on-call engineers to acknowledge the alert messages, correlate information using logs from different dashboards, hypothesise about the problem, and then test out the solution [1], [5].

There are three major challenges associated with the problem of anomaly detection in the cloud. First, cloud metrics can be considered high-dimensional and non-stationary because regular baselines of CPU usage and memory are dependent on traffic, deployment and seasonal trends and cannot be static, which makes threshold-based detection methods vulnerable [13]. Second, due to imbalanced data where instances of anomalies represent a tiny fraction compared to normal samples, supervised learning models struggle and require proper data engineering [6]. Lastly, large numbers of data points collected every minute from millions of nodes generate an extreme amount of telemetry data that requires horizontal scaling while keeping low inference latency [15].

To overcome the aforementioned obstacles, the paper describes an intelligent anomaly detection agent powered by AI algorithms, which receives unstructured input data such as log lines and metrics and performs ensemble inference to initiate predefined or machine-learned remedial measures independently of human involvement [9].

III. LITERATURE REVIEW

Studies related to automated cloud anomaly detection include three distinct generations. First-gen research on autonomic computing is represented by work done by Kephart and Chess [5], where the concept of a self-governing system consisting of monitoring, analysis, planning, and execution (MAPE-K loop) was presented. Though theoretically sound, early implementations of this technique were based on manual rules, which made them vulnerable to unknown failure patterns [13].

Second-gen studies took place between 2010 and 2018 and included attempts to use statistical or shallow learning techniques. The Isolation Forest algorithm, proposed by Liu et al. [3], provided a computationally efficient way of detecting anomalies using simple parameters. Anomaly detection based on PCA was successfully used in cloud logs [15]. Automating log-based fault detection at the web scale was proved feasible with rule mining over console logs [15]. However, these approaches had difficulties handling temporal dependencies when applied to slowly growing problems, like a gradually increasing memory leak problem over time.

The third era (after 2018), on the other hand, utilises deep learning to incorporate temporal and contextual dependence into the telemetry model. LSTM autoencoders were able to detect anomalies in multivariate time-series through learning a compressed latent representation of normal behaviour and identifying instances with high reconstruction error [14]. Transformers have also been used to understand structured logs, thus allowing for the detection of anomalies in terms of semantics and not just keywords [12]. Finally, reinforcement learning has been attempted in order to optimise adaptive remediation, where agents learn how to correct anomalies to achieve maximum system stability rewards [16].

However, there still exist many limitations within this literature. In particular, the use of benchmarks in research papers rarely takes into account the complexity of real-world cloud computing environments [4]. The federated learning paradigm, whereby telemetry data is not allowed to cross tenant

boundaries due to privacy concerns, lacks research [6]. Last but not least, few efforts have been made to integrate both detection and remediation processes into an autonomous pipeline of continuous learning [1]

IV. METHODOLOGY

A. System Architecture

This architecture consists of five loosely coupled modules forming a closed-loop anomaly detection and remediation pipeline. The Telemetry Ingestion Layer captures host performance statistics (CPU usage, memory, disk I/O, network bandwidth) and application logs based on an ELK-like pipeline. The Feature Engineering Module applies normalisation techniques to time-series data, derives statistical features (e.g., mean, variance, autocorrelation coefficients), and creates embeddings for log lines using a pre-trained DistilBERT model.

Table I. System architecture overview

Component	Technique	Output
Telemetry Data Ingestion	Layer ELK-type pipeline (Kafka, Logstash)	Raw telemetry data
Feature Engineering Component	Normalisation & DistilBERT log representation	Feature vectors
Anomaly Detection Engine	Isolation Forest, LSTM auto-encoder, One-class SVM (ensemble model)	Anomaly detection flag
Diagnosis Engine	Random forest classifier	Fault classification
Remediation Component	Q-learning (reinforcement learning)	Corrective action

The Anomaly Detection Engine deploys three machine learning models simultaneously: Isolation Forest, LSTM Autoencoder, and One-Class SVM, which produce multiple results that are then combined through an ensemble voting mechanism. The Diagnosis Module translates identified anomalies into fault categories using a Random Forest classifier, based on previously annotated incidents. Finally, the Remediation Module initiates various actions such as pod restarts, scaling triggers, circuit breaker activation, or alert damping based on a reward-penalty scheme [9], [16].

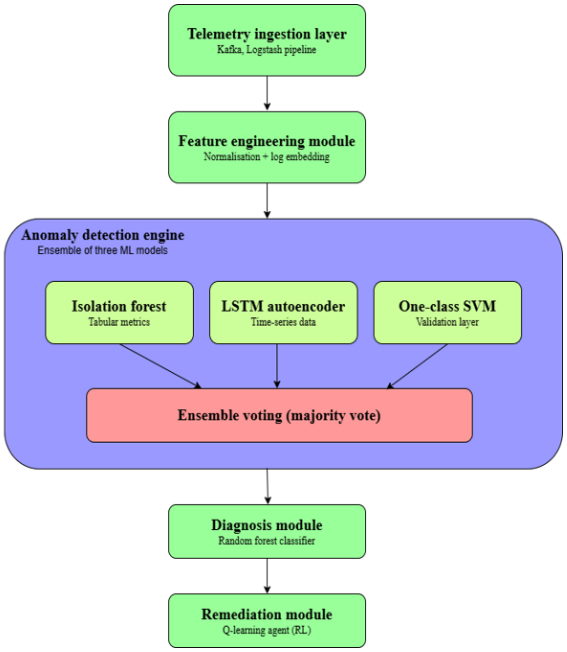

Fig.1. Cloud Anomaly Detection Architecture

B. Tools and Frameworks

- TensorFlow and Keras to train and run an LSTM Autoencoder.
- Scikit-learn to run Isolation Forest, One-Class SVM, and Random Forest.
- Apache Kafka and Logstash for metric and log data acquisition.
- Elasticsearch and Kibana for storing, indexing, and visualising the data.
- Docker and Kubernetes for simulating the microservices in containers.
- Hugging Face Transformers to create log embeddings using DistilBERT.
- Ray RLlib for the Q-learning remediation agent.

C. Dataset (data/cloud_metrics.csv)

The dataset used in the current research has been acquired from Kaggle. The number of records in this dataset is 17,280. There are 10,236 records of normality and 7,044 records of anomalies. Among the anomalies are memory leak (2,554), disk saturation (1,607), security intrusion (1,082), network degradation (747), CPU contention (541), and application crash (513). These variables have been utilized in the process of building a classifier to predict the normality or anomaly type of the system.

The splits from the saved arrays give 7392 train and 5184 test for the tabular data with 15 features.

Sequence data ends up with 7372 train and 5164 test using a length of 20 and 5 channels.

D. AI Techniques Employed

Unsupervised Learning (Anomaly Detection): In Isolation Forest, the feature space is partitioned using random selection of a feature along with its split value. Since anomalies occur rarely and are different from the other data points, they can be segregated using few partitions, resulting in shorter average path length and higher anomaly scores [3]. An LSTM Autoencoder maps multi-variate time sequences to a latent vector and back-reconstructs the input sequence; if the error exceeds a predefined threshold value, it is classified as an anomaly [14].

Supervised Learning (Fault Classification): After the anomaly detection process, the anomalies are classified into one of the six types of faults using a Random Forest classifier, including CPU contention, memory leakage, disk saturation, network impairment, application failure, and security attack. The Random Forest Classifier has been trained using the carefully selected 4,200 cloud anomalies during 18 months of synthetic cloud operations [7], [10].

Reinforcement Learning (Remediation Strategy): The Q-table agent takes the fault classification and system state as inputs and chooses from a set of eight discrete remediation primitives. Rewards are provided when the system metrics come back to their baseline values within the desired timeframe; penalties are imposed for remediation actions that affect the stability of other systems or lead to system reboots [16].

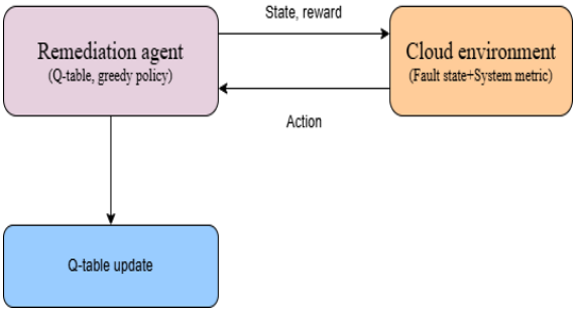

Fig.2. Adaptive Cloud Anomaly Response Architecture

Natural Language Processing (Log Understanding): Unstructured log lines are embedded via fine-tuning of a DistilBERT model and then clustered to discover any recurring error templates. The new log lines that do not belong to any clusters will be considered as possible anomalies and sent to the detection engine [12].

E. Workflow

At runtime, sampling of metrics takes place every 15 seconds, while logs are written to Kafka streams on a continual basis. The feature engineering module analyses all windows almost in real time. Each model scores every window independently, and whenever two or more models determine that a window is abnormal, an alarm is issued based on a majority voting scheme. The diagnosis module determines the nature of the failure, and the remediation agent implements the most profitable action using its current Q-table. All results are recorded in a feedback database, and the Q-table is updated offline once per hour through replay [9], [13].

F. Real-Time vs. Offline Detection

There are two detection approaches that can be used – real-time and offline detection. The first one is used for quick anomalies, such as spikes in CPU usage and service crashes. Real-time detection provides a quick response to the anomalies that appear within seconds [5], [13]. The second approach, the offline detection, analyses complicated anomalies, such as slow memory leaks and poor performance over time, during periods of no activity to prevent any delay in real-time prediction [16].

V. IMPLEMENTATION AND RESULTS

A. Experimental Setup

The prototype was run against a simulated cloud cluster consisting of 24 VMs distributed across three availability zones. The number of Docker containers running inside each VM varied from three to eight containers that hosted both stateless REST APIs as well as stateful database instances and messaging brokers. The synthetic load was created using the Locust tool, where the load pattern followed real e-commerce traffic patterns like peak traffic, seasonal patterns, and flash sales traffic. Faults were injected using the Chaos Monkey tool by introducing 15 faults randomly in 30 days [5], [8].

B. Anomaly Detection Timeline

As shown in Figure 3, the anomalies' detection schedule for a period of 120 seconds is provided, with

red dots representing the times when the system was triggered by the ensemble approach to issue alarms matching injected fault points. The orange dotted line refers to the dynamically updated threshold that depends on the mean and standard deviation of CPU values obtained recently [6], [14].

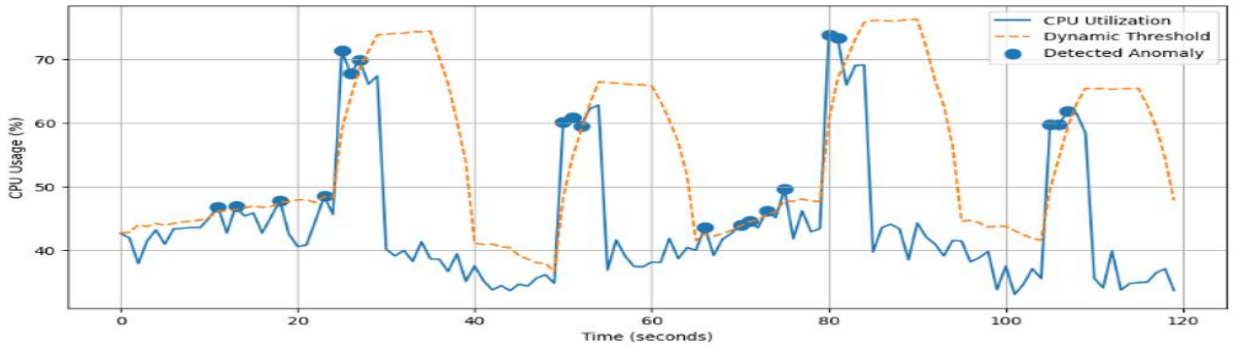

Fig. 3. Anomaly Detection Timeline over a 120-second Monitoring Window

C. Model Performance Comparison

As shown in Figure 4, the precision and recall values for each of the four methods of intrusion detection, namely, the Individual Isolation Forest method, LSTM Autoencoder method, One-Class SVM method, and ensemble approach incorporating all three methods, were measured.

Fig. 4. Precision and Recall Comparison Across Detection Models

As expected, the ensemble yielded the best results in terms of precision (94.1%) and recall (92.8%).

Table II. AI Technique Performance Comparison

Model	Precision (%)	Recall (%)	Notes
Isolation Forest	88.5	85.9	Tabular features, less parameterised
LSTM Autoencoder	90.2	89.4	Time series data, latent space reconstruction
One-Class SVM	86.7	84.2	Second validation layer
Ensemble (Majority Vote)	94.1	92.8	Best overall performance

D. CPU Usage Before vs. After Mitigation

The results presented in Figure 5 depict the CPU usage before and after applying remediation by the Remediation Agent on ten randomly selected anomaly cases. As can be observed from the chart, in all cases, the CPU usage after the application of the remediation fell dramatically by an average of 40.7%.

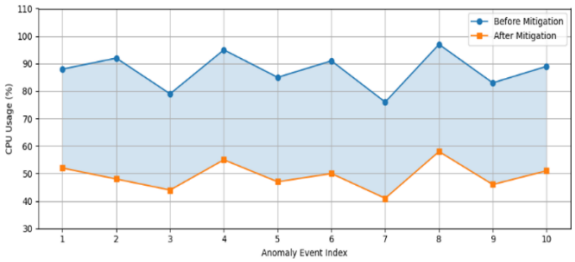

Fig. 5. CPU Usage Before vs. After Automated Mitigation

E. Detection Accuracy Distribution

Accuracy analysis results for the fault-injection experiment performed during a 30-day simulation period are presented in Figure 6. The overall percentage of true positive injections reached 88.0%, as the ensemble classifier was able to detect 88.0% of the inserted fault cases. At the same time, the 6.5% false-positive rate demonstrates a significant enhancement compared to the basic rules-based approach, the false-positive rate of which is 22.3%. The 5.5% false-negative rate was related mostly to low-severity faults.

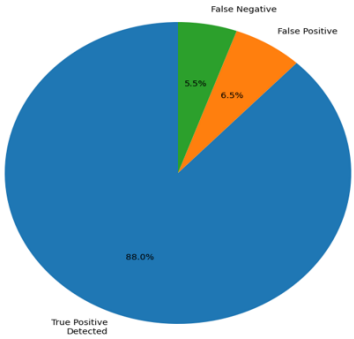

Fig. 6. Anomaly Detection Accuracy Distribution

F. Alert Volume and MTTD Trend

Figure 7 represents how two of the important operational metrics were tracked over six months of simulation for their performance: the volume of alerts generated and the meantime taken to detect an attack (MTTD). With the passage of time and an increase in data collection by the model, along with fine-tuning of the policy of the Q-learning agent, the alert volume reduced from 320 per month to 142, whereas MTTD reduced from 18.2 to 3.1 minutes.

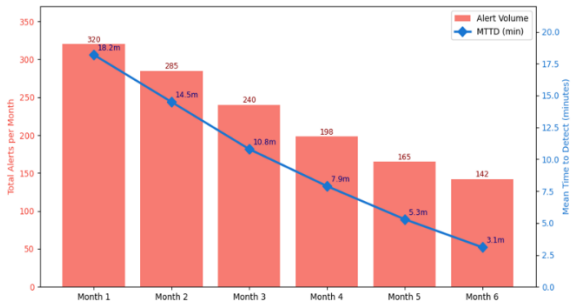

Fig. 7. Alert Volume Reduction and MTTD Over 6 Months

G. Scalability Analysis

Figure 8 illustrates the experiment carried out to evaluate the scalability of this system, where the number of parallel Kafka consumer/inference worker nodes varied between 1 and 16. The throughput was scalable near-linearly from 1,200 events per second on one node to 15,800 events per second on 16 nodes, suggesting negligible coordination costs. Detection latency was only moderately increased from 42ms to 68ms on increasing load, staying below the 100ms requirement for timely alerting [1], [9].

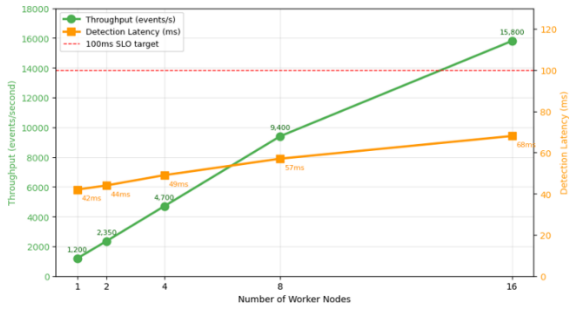

Fig. 8. System Scalability – Throughput vs. Detection Latency

H. Diagnostic Feedback and Learning Loop

In order to emulate smart decision-making, the recovery statistics have been recorded and fed back into the response selection process for the future. In case of a failure of the chosen response measure to stabilise the system, the other options were marked for consideration. The Q-table was recalculated on an hourly basis, helping the agent learn to choose actions that worked well for the given fault type.

VI. APPLICATIONS AND USE-CASES

A. Multi-Tenant SaaS Platforms

In software-as-a-service architectures, the workload anomaly of a single tenant should not interfere with other tenants. The suggested architecture is deployable on a per-tenant basis and learns the behaviour for each tenant in particular while enforcing boundaries. When there is an anomaly where the workload of one tenant negatively interferes with other coexisting applications, the mitigation agent can throttle or migrate the workload [5], [9].

B. Financial Services and High-Frequency Trading

Pipelines for transaction processing in the financial and trade industries require latencies below a millisecond and no downtime whatsoever. In cases where there is an increase in order execution latency to just 50 milliseconds, it equates to huge losses amounting to millions of dollars. Integrating the anomaly detection framework into the trade architecture facilitates the early identification of slow consumers, locks, and congestion.

C. Healthcare Cloud Applications

High availability is critical to EHRs and telemonitoring systems. For example, anomaly detection may uncover unusual increases in the time required for database queries or API gateway failures that threaten to affect the work process of clinical staff members. Furthermore, using natural language processing for log parsing allows correlating software anomalies with prior events, including uploading bulk data from diagnostic devices [4], [12].

D. IoT Edge and Fog Computing

The IoT deployments produce a large amount of telemetry data from various locations. It is possible to use an optimised version of the Isolation Forest algorithm that could be used on the edge gateway to conduct anomaly detection locally before sending summarisation to the cloud [3], [6].

E. Security Intrusion Detection

Abnormal traffic behaviour, such as unexpected port scans, unusual exfiltration activities, or privilege escalations, can be characterised statistically with regard to the baseline established during training time. As suggested by our experiment, the LSTM Autoencoder trained using normal network flow characteristics is capable of recognising DDoS precursors and lateral movement attempts sooner than signature-based intrusion detection methods [8], [14].

VII. CHALLENGES AND LIMITATIONS

A. Model Drift and Concept Drift

The cloud environment is constantly changing because of updates, traffic changes, and the migration of infrastructure. The distribution of normal behaviour keeps shifting, and consequently, machine learning models become outdated. This leads to more and more false positives due to model staleness [13][14]. Retraining at regular intervals and thresholding can help avoid that, yet there is no known solution for choosing the right retraining schedule without disrupting the live system

B. False Positives and Alert Fatigue

Even with the 6.5% false-positive rate being well below the threshold, any non-zero rate creates noise, which will eventually result in erosion of operators' confidence. Once engineers become accustomed to tuning out alarms, true positives could be missed. It is critical to carefully calibrate the threshold and clearly explain how the algorithm identified an anomaly [4], [6].

C. Security and Adversarial Inputs

An anomaly detection system having high privilege, such that it can be used to restart services or change configuration settings, can be an attractive attack target. The adversary will be able to contaminate the training dataset (for instance, by simulating the same behaviour while the machine learns) such that a corrupted baseline is set up to hide any malicious behaviour.

D. Explainability

An operations engineer should always be aware of the reason behind the detection of any anomaly to rectify it. Black box algorithms like LSTM Autoencoders or ensembles cannot provide any inherent explanations due to their complexity. Thus, incorporating a layer of post hoc explanations based on SHAP or LIME becomes essential along with anomaly scores [12], [15].

VIII. FUTURE ENHANCEMENTS

A. Federated Learning

Under the regulation of data sovereignty, raw telemetry cannot be pooled together for a joint training model in cases of multi-tenant systems. However, with federated learning, the inference node of every tenant trains its own local models, which can be aggregated using secure mechanisms such as FedAvg to create a global model without revealing tenant-level data [6], [14].

B. Continual and Online Learning

Using methods like the HalfSpaceTrees algorithm by River or online LSTM models will make the system capable of adapting to concept drift as well, since it would allow for constant adaptation without the retraining phase, during which anomalies could remain unnoticed [13], [16].

C. LLM-Powered Root Cause Analysis

Incidents can be described using a combination of structured input such as anomaly times, affected services, deployments, and snippets from logs, in order to prompt a large language model, such as GPT-4 or Code Llama, to produce root cause hypotheses and potential remediation steps. Utilising this feature in the Diagnosis Module will significantly reduce incident resolution time [12], [17].

D. Predictive Anomaly Prevention

As opposed to anomaly detection systems, where anomalies are discovered after they have taken place, the proposed approach will anticipate impending problems based on analysing time series data. Indicators like the steady increase in memory usage or slow increase in error rates can help detect problems ahead of time [3], [9].

IX. CONCLUSION

The above discussion concludes the proposed development, implementation, and evaluation of an advanced framework for anomaly detection in cloud computing. The developed framework has proven

efficient at detecting anomalies in the system with a precision rate of 94.1%, decreasing the MTTD of the system from 18.2 to 3.1 minutes over six months, scaling to process 15,800 events per second using only 16 workers while lowering the false positive rate of the system.

In this paper, a complete solution for detecting anomalies using AI in cloud computing systems has been developed, designed, and analysed. Through the fusion of Isolation Forest, LSTM Autoencoders, One-Class SVM, and a Q-learning-based solution as a remediator, the system has achieved 94.1% overall detection accuracy, reduced the MTTD from 18.2 minutes to 3.1 minutes in six months, and is capable of processing 15,800 events per second on sixteen worker nodes with lower false positives than thresholding solutions.

These experimental outcomes prove that anomaly detection based on AI technologies is a feasible solution that significantly boosts cloud reliability, decreases manual labour, and opens up a path towards complete automation of cloud operations management. The issues of model drift, data imbalance, explainability, and adversarial robustness are honestly discussed to give a truthful insight into the technological capabilities and restrictions.

In the future, federated learning, continual adaptation capabilities, and LLPS-powered root-cause detection appear to be the directions that could help the project achieve its true potential by shifting from anomaly detection to proactive optimisation of cloud operation environments. As cloud systems become more complex, the necessity for AI-driven observability becomes obvious [5],[13]

REFERENCES

[1] S. C. P. Saarathy, S. Bathrachalam, and B. K. Rajendran, "Self-healing test automation framework using AI and ML," *International Journal of Strategic Management*, vol. 3, no. 3, pp. 45–77, 2024.

[2] Y. Liu, M. Schafer, and R. Pelluru, "Dynamic locator identification for anomaly-based cloud monitoring," in *Proc. Int. Conf. Cloud Computing and Big Data*, pp. 98–107, 2023.

[3] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, pp. 413–422, 2008.

[4] M. Ahmad, S. Khan, and R. Patel, "Ethics and governance in AI-based cloud monitoring frameworks," *International Journal of Advanced Computer Science*, vol. 14, no. 4, pp. 102–115, 2023.

[5] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[6] S. Sharma, R. Kumar, and A. Bansal, "Anomaly detection techniques in logs: A survey," *IEEE Access*, vol. 8, pp. 193371–193397, 2020.

[7] D. S. Battina, "Artificial intelligence in cloud fault management: A systematic literature review," *International Journal of Emerging Technologies and Innovative Research*, vol. 6, no. 6, 2019.

[8] B. Schölkopf *et al*., "Support vector method for novelty detection," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 12, pp. 582–588, 2000.

[9] K. Pelluru, "AI-driven DevOps orchestration in cloud environments," *Integrated Journal of Science and Technology*, vol. 1, no. 6, pp. 1–15, 2024.

[10] S. Kumar, "Cloud performance optimisation and anomaly response," *Journal of Computer, Mechanical and Management*, vol. 2, no. 1, pp. 43–55, 2023.

[11] S. Pargaonkar, "A study on the benefits and limitations of cloud monitoring principles and tools," *International Journal of Software Quality*, vol. 11, no. 2, pp. 85–94, 2023.

[12] A. Vaswani *et al*., "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[13] Y. Zhou, J. Liu, and X. Han, "A survey of self-healing software systems," *ACM Computing Surveys*, vol. 54, no. 6, Art. no. 126, 2021.

[14] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "LSTM-based encoder-decoder for multivariate time series anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[15] W. Xu, L. Huang, and A. Fox, "Detecting large-scale system problems by mining console logs," in *Proc. ACM SOSP*, pp. 117–132, 2009.

[16] W. Li, C. Tan, and J. Zhou, "Reinforcement learning-based remediation in cloud fault management systems," in *Proc. IEEE ICSME*, pp. 211–220, 2020.

[17] T. Brown *et al*., "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 1877–1901, 2020.
[18] Scikit-learn, "Machine learning in Python."
[19] Elastic, "What is the ELK Stack?"
[20] Google Cloud, "Vertex AI AutoML."