Data Science and Computational Intelligence
https://dscij.org

Streaming Anomaly Scores and Outage Detection in Cloud Operations Logs

Kin-Lok Wan*

Division of AI, School of Data Science, Lingnan University, Hong Kong, Hong Kong SAR, China

Corresponding author: kin.l.wan@ln.edu.hk

Abstract

The exponential growth of cloud computing infrastructure has necessitated the development of robust, real time monitoring systems to ensure high availability and reliability. Cloud operations logs provide a granular, continuous stream of data reflecting the internal state of distributed systems. However, the sheer volume, velocity, and unstructured nature of these logs make manual inspection and batch processing techniques highly inadequate for timely outage detection. This paper presents a comprehensive simulation study focusing on streaming anomaly scores and their application in predicting and detecting systemic outages within cloud operations. By employing a simulated environment that mimics complex, multi tier cloud architectures, we evaluate a streaming methodology that continuously ingests operational logs, parses them in real time, and assigns dynamic anomaly scores based on sequential and quantitative deviations from baseline behaviors. The research rigorously investigates the relationship between fluctuating anomaly scores and the imminent onset of service degradation or catastrophic failures. Through extensive simulation runs injecting various fault types, including network partitions, resource exhaustion, and configuration errors, we analyze the efficacy of dynamic thresholding mechanisms for triggering outage alerts. The findings demonstrate that continuous anomaly scoring significantly reduces detection latency compared to traditional window based batch analysis, while maintaining an acceptable false positive rate. Furthermore, the study highlights the importance of context aware scoring adjustments to mitigate the impact of routine maintenance operations, which often manifest as anomalous but benign log patterns.

Keywords: Cloud Computing; Streaming Data; Anomaly Detection; Outage Prediction; Streaming Anomaly Scores

1. Introduction

1.1 Background on Cloud Operations Logs

The modern digital economy relies heavily on cloud computing infrastructure, which acts as the foundational backbone for enterprise applications, data storage, and global communications. As organizations increasingly migrate their workloads to distributed cloud environments, the complexity of managing and maintaining these systems scales exponentially. Cloud infrastructure is composed of thousands of interconnected microservices, virtual machines, containers, and networking components, all of which are in a constant state of flux. To monitor the health and performance of such intricate ecosystems, system administrators and automated orchestration tools depend on cloud operations logs. These logs are generated continuously by every component within the architecture, recording discrete events ranging from routine user authentication and routine automated tasks to critical hardware failures and network timeouts. The data contained within these logs is invaluable; it provides a sequential, historical record of the internal state transitions of the system.

Historically, log analysis was a reactive process. System administrators would archive log files and query them retrospectively only after an outage had been reported by end users. This forensic approach, while useful for post mortem analysis, is entirely insufficient for the proactive management required in modern, high availability cloud environments. The volume of data generated by a typical enterprise cloud deployment can easily reach several terabytes per day, characterized by high velocity and significant variety in formatting and structure. Consequently, the paradigm has shifted toward automated log analysis, wherein machine learning algorithms and statistical models are employed to parse and analyze logs as they are generated. The objective is to identify anomalous patterns that deviate from expected normal behavior, thereby providing early warning signs of potential system degradation before it escalates into a full scale outage. However, transitioning from theoretical models of automated analysis to practical, real time implementation presents substantial computational and algorithmic challenges [1].

1.2 The Need for Streaming Anomaly Detection

The transition from batch processing to stream processing represents a critical evolution in cloud monitoring methodologies. Batch processing involves accumulating data over a specified time window or volume threshold before

executing analytical workloads. While effective for generating daily reports or training machine learning models offline, batch processing inherently introduces latency. In the context of cloud outages, a delay of even a few minutes can result in severe financial repercussions, data loss, and damage to organizational reputation. Outages in distributed systems rarely occur instantaneously without warning; they are typically preceded by a cascade of minor errors, performance bottlenecks, and resource constraints that manifest in the operational logs as subtle anomalies. To capture these fleeting indicators, log analysis must be performed continuously on the data stream, evaluating each log entry or small micro batch of entries in near real time [2].

Streaming anomaly detection addresses this requirement by continuously updating its internal state and evaluating incoming log sequences against normal behavioral profiles. Instead of producing a binary classification of normal versus anomalous for a massive block of historical data, streaming systems generate a continuous, dynamic anomaly score. This score fluctuates based on the severity and frequency of deviations observed in the log stream. When the anomaly score surpasses a predefined or dynamically calculated threshold, the system triggers an alert, indicating a high probability of an impending outage. This proactive approach allows automated orchestration systems to initiate remediation protocols, such as scaling resources, diverting traffic, or restarting failing microservices, potentially averting the outage entirely. Despite its advantages, streaming anomaly detection introduces complex challenges regarding computational overhead, memory management, and the handling of concept drift, where the definition of normal behavior evolves over time due to software updates or changes in user behavior [3].

1.3 Research Objectives and Contributions

The primary objective of this research is to investigate the efficacy of streaming anomaly scores for outage detection through a controlled, highly detailed simulation study. Simulating cloud environments offers significant advantages over utilizing real world production data for initial methodological evaluation. Production environments are inherently noisy, and actual outages are relatively rare events, making it difficult to obtain statistically significant datasets of failure modes. By employing a simulation, we can meticulously model complex cloud topologies and deterministically inject a wide variety of fault conditions, observing precisely how the streaming anomaly detection mechanisms respond.

This paper makes several distinct contributions to the field of cloud operations management. First, we design and implement a sophisticated log simulation engine capable of generating realistic, high velocity log streams that accurately reflect the interdependencies and cascading failure mechanisms inherent in multi tier cloud architectures. Second, we propose and evaluate a novel streaming anomaly scoring mechanism that combines sequential pattern analysis with quantitative metric extraction, designed specifically to operate within strict memory and processing constraints. Third, we conduct an exhaustive comparative analysis of various thresholding strategies for outage detection, examining the trade offs between detection latency, true positive rates, and false alarm frequencies. By systematically altering the parameters of the simulation, including log volume, fault intensity, and system noise, we provide empirical insights into the robustness and scalability of streaming anomaly detection in highly dynamic environments.

2. Literature Review

2.1 Traditional Log Analysis Techniques

The evolution of log analysis in computing systems spans several decades, originating from simple rule based systems to the complex algorithmic approaches employed today. In early monolithic architectures, system logs were typically plain text files scrutinized manually using command line text processing utilities. The primary method of detecting issues relied on regular expression matching, where administrators would define specific keywords such as error, critical, or failed. When a log entry matched these predefined rules, an alert was generated. While this rule based approach was highly interpretable and required minimal computational resources, it suffered from severe limitations [4]. It was entirely reactive and relied heavily on the domain expertise of the administrators to anticipate all possible error manifestations. Furthermore, rule based systems generated an overwhelming number of false positives during routine maintenance or benign configuration changes, leading to alert fatigue among operational staff.

As systems grew more complex, the volume of logs outpaced the capacity of manual rule creation and maintenance. This necessitated the development of more sophisticated statistical methods. Researchers began applying frequency analysis and time series forecasting to operational logs, attempting to establish baselines of normal log generation rates. Significant deviations from these baselines, such as sudden spikes in log volume or the unexpected absence of routine log entries, were flagged as anomalies. While statistical methods improved upon the rigidity of rule based systems, they struggled with the unstructured nature of log data. Logs consist of free text messages that vary significantly between different software components and versions. To effectively apply statistical and subsequent machine learning techniques, researchers recognized the critical need for automated log parsing, a process that transforms unstructured log messages into structured formats, typically by extracting static templates and dynamic parameters [5]. The development of efficient log parsers marked a pivotal turning point, enabling more advanced analytics.

2.2 Streaming Data Processing in Cloud Environments

The advent of cloud computing and big data technologies fundamentally altered the landscape of data processing. Traditional relational database management systems and batch processing frameworks like Hadoop were ill equipped to handle the continuous, unbounded streams of data generated by global cloud infrastructures. This led to the emergence of distributed stream processing engines. These frameworks introduced new paradigms for handling data in motion, emphasizing low latency processing, fault tolerance, and scalable state management [6]. In the context of log analysis, these streaming platforms allowed for the continuous ingestion and processing of log events as they occurred, shifting the analytical focus from historical review to real time operational intelligence.

The application of machine learning algorithms within these streaming frameworks presented novel challenges. Traditional supervised machine learning models require labeled training data and assume that the underlying data distribution remains stationary over time. However, cloud operations logs are rarely labeled with accurate fault categorizations, and the environment is highly dynamic, characterized by frequent software deployments, workload fluctuations, and infrastructure modifications. This phenomenon, known as concept drift, rapidly degrades the accuracy of static predictive models. Consequently, recent literature has focused heavily on unsupervised and semi supervised online learning techniques [7]. These algorithms are designed to continuously update their internal parameters based on the incoming data stream, adapting to new normal behaviors while simultaneously detecting anomalous deviations. Techniques such as streaming clustering, online isolation forests, and dynamic principal component analysis have been widely explored for their applicability to real time log analysis [8].

2.3 Outage Prediction and Detection Methodologies

The ultimate goal of anomaly detection in cloud logs is the accurate and timely prediction or detection of system outages. The literature broadly categorizes outages into two types: hard failures, such as server crashes or network partitions, which manifest abruptly, and soft failures, such as memory leaks or database connection pool exhaustion, which develop gradually over time. Hard failures often produce immediate, highly visible anomalies in the log stream, making them relatively straightforward to detect, albeit difficult to predict with significant lead time. Conversely, soft failures offer a window of opportunity for prediction but require highly sensitive anomaly detection mechanisms capable of identifying subtle, cumulative deviations from normal behavior.

Many proposed methodologies in the literature rely on analyzing the sequence of log events. By modeling the normal execution paths of distributed applications as sequences of log templates, researchers have utilized techniques like finite state automata, hidden Markov models, and, more recently, deep learning architectures like recurrent neural networks and transformers to identify anomalous sequences [9]. An unexpected log template or a missing expected template within a specific execution context often indicates a software defect or a misconfiguration leading to service degradation. Other approaches focus on the quantitative aspects of the logs, analyzing the distributions of numerical parameters extracted during the parsing phase, such as response times, payload sizes, or memory utilization metrics. By tracking the time series of these dynamic parameters, researchers have applied advanced forecasting techniques to predict impending resource exhaustion [10]. Despite the wealth of theoretical models, empirical studies rigorously evaluating the real time application of streaming anomaly scores for outage detection within highly dynamic, simulated environments remain limited, providing the motivation for the current research endeavor.

3. Methodology and Simulation Design

3.1 Data Generation and Characteristics

To systematically evaluate the streaming anomaly scoring mechanism, we constructed a sophisticated discrete event simulation environment designed to mimic a large scale, multi tier cloud application. The simulated architecture consists of load balancers, web servers, application servers, and database clusters, all generating continuous streams of operational logs. The simulation engine is parameterized to control various aspects of the environment, including the request arrival rate, which follows a Poisson distribution with configurable daily and weekly seasonality patterns, representing typical user behavior. The behavior of each simulated component is governed by finite state machines that dictate the sequence of log messages generated during both normal operations and failure scenarios.

The generated logs are designed to closely resemble real world unstructured log data. Each log entry comprises a timestamp, a severity level, the source component identifier, and a free text message. The free text message is constructed by combining static string templates with dynamic variables, such as user identifiers, transaction amounts, and processing latencies. During normal operation, the simulation generates logs that reflect standard operational noise, including routine connection establishment, data retrieval processes, and automated background maintenance tasks. This baseline data generation is critical for training the anomaly detection algorithms to recognize and filter out benign activities. Table 1 details the primary configuration parameters utilized to establish the baseline simulation environment.

Table 1: Configuration Parameters for the Log Simulation Environment

Parameter Category	Specific Setting	Value Range / Description
Infrastructure Topology	Number of Web Nodes	50 to 200 dynamic instances

Parameter Category	Specific Setting	Value Range / Description
Infrastructure Topology	Database Shards	10 active, 2 standby nodes
Workload Characteristics	Request Rate Distribution	Poisson with diurnal variance
Workload Characteristics	Background Task Frequency	Scheduled every 15 minutes
Log Generation	Template Vocabulary Size	Approximately 500 distinct templates
Log Generation	Parameter Variability	High variance in latency metrics

The core of the simulation's value lies in its fault injection capabilities. We programmed the engine to introduce specific, deterministic fault conditions that replicate common cloud infrastructure issues [11]. These include CPU saturation on application servers, network latency spikes between the web and database tiers, and misconfigured routing tables causing request blackholing. When a fault is injected, the state machines of the affected components transition into error states, altering the pattern and content of the generated logs. For instance, a database connection timeout will generate specific error templates, while a memory leak will manifest as a gradual increase in garbage collection log frequency and duration. The timing, duration, and intensity of these injected faults are precisely recorded by the simulation engine, providing an absolute ground truth for evaluating the accuracy and latency of the anomaly detection system.

3.2 Streaming Anomaly Scoring Mechanism

The anomaly detection methodology employed in this study is designed for continuous, streaming execution. The architecture pipeline consists of three primary stages: log parsing, feature extraction, and anomaly scoring. As log entries stream from the simulation engine, they are immediately ingested by the parsing module. Given the unstructured nature of the raw logs, the parser utilizes an online template extraction algorithm. This algorithm dynamically constructs and maintains a dictionary of log templates by identifying the static textual components of the incoming messages and masking the dynamic variables. This step is crucial as it reduces the high dimensionality of raw text into a structured, categorical format that is amenable to mathematical analysis.

Following parsing, the system extracts features over tumbling time windows. Rather than evaluating individual log lines in isolation, the system analyzes the collective behavior of the system within small, discrete intervals. The feature extraction module calculates two distinct types of features. First, it computes the frequency distribution of log templates within the window, creating a vector that represents the volumetric behavior of the system. Second, it models the sequential transition probabilities between templates, capturing the execution flow of the underlying applications. The anomaly scoring module evaluates these extracted features against a continuously updating model of normal behavior. The system utilizes an online variant of the Isolation Forest algorithm, which is highly efficient for streaming anomaly detection. The algorithm assigns an anomaly score to each time window, reflecting the degree to which the current feature vectors deviate from the established baseline. A higher score indicates a greater structural or volumetric deviation from expected patterns.

3.3 Outage Detection Thresholding System

The generation of an anomaly score is only the first component of the detection pipeline; the subsequent challenge is determining when a fluctuating score indicates an actual, actionable outage rather than transient system noise. In many traditional systems, a static threshold is defined, and any score exceeding this value triggers an alert. However, our simulation study evaluates more sophisticated, dynamic thresholding mechanisms. Given that the baseline activity of a cloud environment fluctuates due to diurnal cycles and scheduled batch jobs, a static threshold inevitably leads to a high false positive rate during peak usage and a high false negative rate during off peak hours.

To address this, we implemented an adaptive thresholding system based on the Extreme Value Theory. The system maintains a sliding window of historical anomaly scores and continuously calculates the moving average and standard deviation. The threshold is dynamically adjusted based on these statistical properties, ensuring that it remains sensitive to local context. Furthermore, to mitigate the impact of momentary spikes in anomaly scores, which are common in distributed systems due to transient network congestion or temporary resource locks, we implemented a cumulative scoring mechanism. An alert is not triggered by a single anomalous window; instead, the system requires sustained anomalous behavior over consecutive windows. The cumulative score decays over time if normal behavior resumes, effectively filtering out isolated anomalies while rapidly escalating the alert status when systemic degradation persists. This thresholding logic is critical for translating raw mathematical anomaly scores into practical, operational intelligence.

4. Experimental Setup and Execution

4.1 Simulation Environment Configuration

The empirical evaluation of the proposed streaming anomaly detection framework was conducted using a dedicated, high performance computing cluster to ensure that the simulation ran without resource contention, which could artificially introduce latency or logging artifacts. The simulation engine was configured to run continuously for an equivalent of thirty simulated days. The first seven days of the simulation were designated as the burn in period. During this phase, no faults were injected. The purpose of the burn in period was to allow the online log parsing algorithms to stabilize their template

dictionaries and to enable the streaming anomaly detection models to build a robust baseline of normal diurnal and weekly operational patterns. The data generated during this period served exclusively for unsupervised model initialization [12].

Following the burn in period, the simulation entered the active testing phase, spanning twenty three simulated days. During this phase, an automated fault orchestration script deterministically injected a total of one hundred distinct outage scenarios. These scenarios were categorized into three primary fault classes: resource exhaustion, network partition, and application logic errors. Resource exhaustion faults simulated memory leaks and CPU overutilization, typically resulting in a gradual degradation of service. Network partition faults simulated sudden loss of connectivity between microservices, leading to immediate transaction failures. Application logic errors simulated the deployment of buggy code, which produced specific error stack traces in the logs. The fault injection script randomized the time of day and the specific nodes targeted by the faults to ensure a comprehensive evaluation across varying background workload conditions.

4.2 Evaluation Metrics

To rigorously assess the performance of the streaming anomaly scoring and outage detection system, we established a set of standardized evaluation metrics. The primary objective was to balance the timeliness of detection with the accuracy of the alerts. We defined a True Positive as an instance where the anomaly detection system generated an alert that correlated with a deterministically injected fault within a predefined acceptable time window. A False Positive was recorded when the system generated an alert during normal operations without any underlying fault. A False Negative occurred when a fault was injected and progressed to a full simulated outage without the system generating a corresponding alert.

Using these definitions, we calculated the standard metrics of Precision, Recall, and the F1 Score. Precision measures the proportion of generated alerts that were actually associated with real outages, providing an indication of the system's susceptibility to false alarms. Recall measures the proportion of actual injected faults that were successfully detected by the system. The F1 Score provides a harmonic mean of Precision and Recall, offering a single, balanced metric of overall detection accuracy. Beyond these standard classification metrics, the critical factor of detection latency was rigorously analyzed. Detection latency was defined as the time elapsed between the initial injection of the fault by the simulation engine and the moment the dynamic thresholding system triggered an alert. Minimizing this latency is paramount in cloud operations, as it directly translates to the lead time available for automated remediation systems to act before a service disruption affects end users.

5. Results and Discussion

5.1 Analysis of Anomaly Score Distribution

The continuous execution of the simulation generated an extensive dataset of time series anomaly scores. Analysis of the normal operational periods revealed that the baseline anomaly scores were not static but exhibited distinct cyclical patterns corresponding to the simulated diurnal workloads. During simulated daytime hours, when request rates and background processing tasks were highest, the baseline anomaly score maintained a higher average with increased variance. This observation validates the necessity of the dynamic thresholding mechanism implemented in the methodology section, as a static threshold would have inevitably generated false positives during peak load periods.

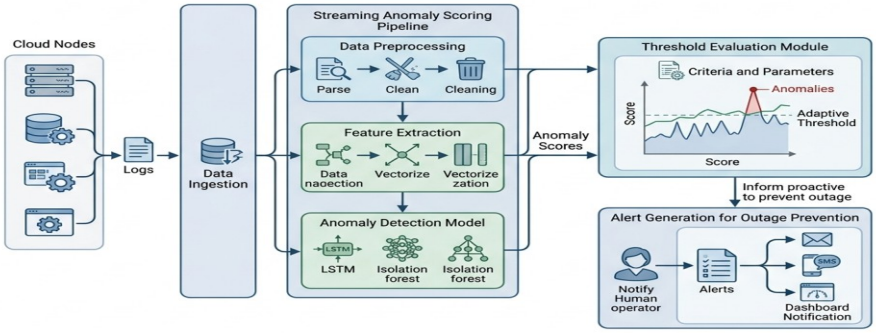

Figure 1: Comprehensive System Architecture for Streaming Log Anomaly Detection and Outage Prediction

When faults were injected, the response of the anomaly scoring mechanism varied significantly depending on the nature of the fault. Hard failures, such as network partitions, resulted in an immediate and drastic spike in the anomaly score. The log streams during these events were characterized by a sudden cessation of normal transaction logs and a concurrent explosion of timeout and connection refused error templates. The streaming model detected this volumetric and structural deviation almost instantaneously. Conversely, soft failures, such as simulated memory leaks, produced a much more nuanced response in the anomaly scores. Initially, the score would increase marginally as the garbage collection logs became slightly more frequent. As the memory pressure intensified, the score exhibited an accelerating upward trend. This gradual escalation highlights the capability of the streaming sequential analysis to detect subtle, cumulative deviations that would be entirely missed by simple rule based systems or infrequent batch processing analysis [13].

5.2 Outage Detection Efficacy and Latency

The comprehensive evaluation of the outage detection mechanisms demonstrated the clear superiority of dynamic thresholding combined with cumulative scoring over traditional static approaches. Table 2 presents a comparative analysis of different detection strategies evaluated during the twenty three day testing phase.

Table 2: Performance Evaluation of Outage Detection Mechanisms

Detection Mechanism	Precision	Recall	Average Latency
Static Thresholding	0.62	0.88	145 seconds
Dynamic Thresholding (Single Window)	0.74	0.91	120 seconds
Dynamic Thresholding (Cumulative Score)	0.94	0.89	165 seconds

The static thresholding approach yielded the lowest precision, generating a significant number of false positives during periods of high simulated load or during complex, but benign, background maintenance tasks. While it achieved a respectable recall, the operational cost of investigating numerous false alarms renders this approach suboptimal for modern cloud environments. Moving to a dynamic thresholding approach based on statistical moving averages improved precision considerably, as the system adapted to the varying baseline activity. However, relying on a single time window exceeding the dynamic threshold still left the system vulnerable to transient spikes in logging activity caused by brief network jitter [14].

The implementation of the cumulative scoring mechanism provided the most robust performance. By requiring sustained anomalous behavior before triggering an alert, the precision increased dramatically to 0.94, meaning that nearly every alert generated was indicative of a genuine injected fault. This high precision is critical for automated remediation systems, which must trust the alert signals to execute potentially disruptive recovery actions. There was a slight trade off in average detection latency, which increased to 165 seconds compared to the single window dynamic approach. This increase represents the time required for the cumulative score to breach the critical threshold. However, an analysis of the specific fault types revealed that this latency penalty primarily affected the detection of soft failures, where a delay of two to three minutes is generally acceptable and preferable to generating false alarms. For critical hard failures, the magnitude of the anomaly score was typically sufficient to breach the cumulative threshold rapidly, minimizing the latency impact [15].

Furthermore, the simulation results highlighted the importance of feature selection in the anomaly detection pipeline. The combination of volumetric features (template frequencies) and structural features (sequential transitions) proved synergistic. Volumetric features were highly effective at detecting resource exhaustion issues, where the types of logs remained normal but their frequency increased. Structural features were indispensable for detecting application logic errors and configuration faults, which altered the expected sequence of operations without necessarily changing the overall volume of generated logs. The streaming architecture successfully maintained low computational overhead throughout the simulation, parsing and analyzing thousands of log lines per second without introducing processing backlogs, thereby validating its suitability for deployment in high velocity production environments.

6. Conclusion and Future Directions

6.1 Summary of Key Findings

This comprehensive simulation study rigorously evaluated the application of streaming anomaly scores for the real time detection and prediction of outages in cloud operational logs. By constructing a highly detailed, discrete event simulation engine, we successfully modeled the complex interdependencies and logging behaviors of modern distributed architectures. The findings conclusively demonstrate that continuous, streaming log analysis provides a critical operational advantage over traditional batch processing methodologies by significantly reducing the latency of anomaly detection. The dynamic parsing and feature extraction mechanisms proved capable of handling unstructured log data efficiently, transforming raw text streams into continuous, quantifiable metrics of system health.

Crucially, the experimental results emphasize that generating an accurate anomaly score is only a partial solution; the mechanism used to interpret that score is equally vital. The comparative analysis of thresholding strategies revealed that dynamic, context aware thresholds combined with cumulative scoring logic drastically reduce false positive rates while maintaining high recall for actual outage events. This high precision is an absolute prerequisite for integrating anomaly detection systems into automated orchestration and self healing infrastructure pipelines. The simulation confirmed that while hard failures are detected almost instantaneously due to massive disruptions in log patterns, soft failures require the sensitivity of sequential and volumetric analysis to identify gradual degradation before it culminates in catastrophic service loss.

6.2 Limitations and Future Work

While the simulation provided a highly controlled environment for evaluating the core methodologies, it inherently abstracts some of the chaotic variability present in genuine production environments. The simulated fault classes, although comprehensive, cannot encompass every possible failure mode of bespoke enterprise applications. Additionally, the log templates generated by the simulation, while realistic, lack the complete semantic complexity and occasional irregularities found in human authored logging statements across disparate open source components. Therefore, deploying these algorithms directly into a live environment will require a period of calibration and tuning to account for unanticipated operational noise.

Future research directions will focus on bridging the gap between simulation and real world deployment. A primary area of investigation involves the integration of natural language processing techniques, specifically large language models, into the streaming parsing pipeline. While computationally expensive, utilizing these models to extract deeper semantic context from unstructured logs could further enhance the accuracy of anomaly detection algorithms. Additionally, exploring federated learning approaches for distributed anomaly detection across multiple edge computing locations represents a promising avenue. By allowing localized anomaly models to share insights without centralizing massive log volumes, organizations can achieve global operational intelligence while mitigating network bandwidth constraints and data privacy concerns. Ultimately, the continuous advancement of streaming analytics will remain central to ensuring the resilience and reliability of future cloud infrastructure [16].

References

1. Hu, H.; Liu, Y.; Qian, D. I/O Feature-based File Prefetching for Multi-Applications. In Proceedings of the 2010 Ninth International Conference on Grid and Cloud Computing, Nanjing, China, 1–5 November 2010; pp. 213–217.

2. Chen, Y.; Byna, S.; Sun, X.H.; Thakur, R.; Gropp, W. Hiding I/O latency with pre-execution prefetching for parallel applications. In Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, Austin, TX, USA, 15–21 November 2008; pp. 1–10.

3. Gomes, H.M.; Bifet, A.; Read, J.; Barddal, J.P.; Enembreck, F.; Pfharinger, B.; Holmes, G.; Abdessalem, T. Adaptive random forests for evolving data stream classification. Mach. Learn. 2017, 106, 1469–1495.

4. Ghemawat, S.; Gobioff, H.; Leung, S.T. The Google file system. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 19–22 October 2003; pp. 29–43.

5. Fuleky, P. Macroeconomic Forecasting in the Era of Big Data: Theory and Practice; Springer: New York, NY, USA, 2020; Volume 52.

6. Lucas Filho, E.R.; Efstathiou, A.; Yang, L.; Fu, K.; Shen, J.; Herodotou, H. DITIS: An End-to-End System-Level Simulator and Optimizer for Distributed Tiered Storage. SN Comput. Sci. 2025, 6, 746.

7. Hong, T.; Langevin, J.; Sun, K. Building Simulation: Ten Challenges. Build. Simul. 2018, 11, 871–898.

8. Choi, H.; Varian, H. Predicting Initial Claims for Unemployment Benefits. Google Inc. 2009, 1, 1–5.

9. Hornik, K. Approximation capabilities of multilayer feedforward networks. Neural Netw. 1991, 4, 251–257.

10. Tian, X.; Ahrens, B.; Rossdeutscher, L.; Alonso, L.; Parente, L. Soil science-informed neural networks for soil organic carbon density modelling under scarce bulk density data. EGUsphere 2026.

11. Zhou, L.; Zhang, C.; Liu, F.; Qiu, Z.; He, Y. Application of deep learning in food: A review. Compr. Rev. Food Sci. Food Saf. 2019, 18, 1793–1811.

12. Gui, J.; Wang, Y.; Shuai, W. Improving reading performance by file prefetching mechanism in distributed cache systems. Concurr. Comput. Pract. Exp. 2024, 36, e8215.

13. Wang, H.; Sun, Q.; Niu, X.; Liu, K.; Zhang, J.; Hao, Z.; Xu, D. Soil Organic Carbon Prediction Using an Efficient Channel Attention-Enhanced CNN-LSTM Model with LUCAS Spectral Library. Eur. J. Soil Sci. 2025, 76, e70202.

14. Liu, X.F.; Shahriar, M.R.; Al Sunny, S.M.N.; Leu, M.C.; Hu, L. Cyber-Physical Manufacturing Cloud: Architecture, Virtualization, Communication, and Testbed. J. Manuf. Syst. 2017, 43, 352–364.

15. Gill, B.S.; Modha, D.S. SARC: Sequential Prefetching in Adaptive Replacement Cache. In Proceedings of the 2005 USENIX Annual Technical Conference (USENIX ATC 05), Anaheim, CA, USA, 10–15 April 2005; pp. 293–308.

16. Choi, J.; Lee, B. Accelerating Materials Language Processing with Large Language Models. Commun. Mater. 2024, 5, 13.