Journal of Electrical and Electronics Research (JEER)
Volume 3, Issue 2, Date: 31-December-2026
ISSN: 2978-4484, DOI: 10.64820/AEPJEER.32.27.35.122026

Detecting Cascading Failures in Microservice Telemetry: A Benchmark for Early-Warning Anomaly Detection

Vigneshwarr Venkatesan [1, *]
[1] College of Computing and Software Engineering, Kennesaw State University, Kennesaw, Georgia, USA
Email: [1] vvenkat5@students.kennesaw.edu
*Corresponding Author

***Abstract*—Cloud-native microservices-based architectures create intricate runtime dependencies and localized faults can quickly escalate to cascading service failures, making it all the more important to detect anomalies in time to ensure operational resilience. The current evaluation research mainly focuses on classification accuracy after failure and does not give much information about the early-warning ability of anomaly detection techniques in progressive failure. This paper introduces a repeatable benchmarking framework for the systematic assessment of early-warning anomaly detection, which allows to standardize the acquisition of telemetry data, to introduce controlled faults, and to employ an assessment methodology that is targeted towards the lead-time. The proposed framework has the infrastructure metrics, distributed traces, application logs, and service dependency information in a single evaluation pipeline, allowing for meaningful comparisons between the different statistical detector families, machine learning detector families, deep learning detector families, and dependency-aware detector families under the same experimental conditions. Experimental results show that dependency-aware detectors have the best overall results with an accuracy of 98.4%, a precision of 98.1%, a recall of 97.8%, an F1-score of 97.9%, an ROC-AUC of 0.97, an average lead time of 94.6 s, and the smallest delay in detection of 32.1 s. This benchmark offers a practical and reproducible way to assess and evaluate, in addition to traditional performance metrics, operational early-warning assessment, which contributes to a more resilient cloud-native monitoring and incident management system.**

Keywords*—Cloud-native Microservices, Early-Warning Anomaly Detection, Benchmark Framework, Distributed Telemetry, Cascading Failure Analysis, Lead-Time Evaluation, Dependency-aware Detection, Cloud Observability

I. INTRODUCTION

The cloud-native microservice architecture has transformed the design, deployment, and operation of modern software systems. The decomposition of applications into independent deployable services provides better scalability, deployment flexibility, and fault isolation for organizations. This architectural flexibility also means that the runtime dependencies are very dynamic because of the request chaining, resource contention, retry amplification, communication bottlenecks, etc., that can cause a localized performance issue to quickly propagate across interconnected services. As a result, many operational failures do not occur as a single incident, but rather as a chain of events that impacts numerous services, only to be detected as degradation by traditional monitoring systems. As the amount of observability data (infrastructure metrics, distributed traces, application logs, service telemetry, etc.) has grown, it has fueled a tremendous amount of progress in the field of intelligent anomaly detection for distributed systems. Recent work has shown that it is indeed possible to accurately detect abnormal system behavior in cloud-native systems using graph neural networks, Transformer architectures, contrastive learning, and telemetry-based predictive models. Meanwhile, new kernel-level observability, distributed tracing, and automated root cause analysis tools have significantly multiplied the tools site reliability engineers and platform operators have at their fingertips. This evolution has brought about a change in the way anomaly detection is performed, from a predefined threshold approach to learning complex operational behavior within data-driven intelligence systems that operate across a diverse array of microservices.

Despite these developments, the practical aspect of incident management is only slightly touched on. The majority of the studies that focus on anomaly detection algorithms use traditional classification measures, including accuracy, precision, recall, F1-score, or area under the receiver operating characteristic curve. These measures only indicate if an anomaly is ultimately discovered, and do not give much information on when an anomaly is discovered relative to the development of a distributed incident. It's essential in a field environment. When a service is degraded and it has spread across several dependencies, the anomaly may have already occurred, and there may not be enough time for automatic anomaly mitigation, traffic redirection, scaling resources, or operator actions. Therefore, the operational value of an anomaly detector depends on its correctness and the number of warnings that it can give before the cascading failure becomes service-impacting. Another drawback of existing studies is that there is no common evaluation methodology developed for cascading failure scenarios.

Received: 8-8-2026
Revised: 30- 8 -2026
Published: 31-12-2026

© 2026 The Author(s). This article is published under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). This license permits others to share, copy, distribute, remix, adapt, and build upon the material, provided they give appropriate credit to the original authors and source. Any adapted or derivative work must be distributed under the same CC BY-SA 4.0 license.

Currently, the benchmark data sets are mostly used for the evaluation of isolated anomalies or to focus on the location after the event, with comparatively few benchmarks evaluating the behavior of the detectors over the entire propagation of the fault in the system. Moreover, published assessments use different sources of telemetry, setups, models of faults, and evaluations of performance, and thus it is hard to objectively compare the different families of detectors. This methodological disparity makes experimental results difficult to reproduce and can make it difficult to choose the anomaly detection method to be used in large-scale cloud-native deployments.

Inspired by these problems, in this paper we develop a benchmark-based framework to evaluate early-warning anomaly detection in microservice telemetry. The authors don't propose a new detection algorithm, but suggest a repeatable experimental strategy that allows for a comparison of the performance of the statistical, machine learning, deep learning and dependency-aware detector families in controlled fault-injection scenarios. The benchmark adds a formally defined lead-time metric to the traditional detection metrics that quantifies the time from the first detector alert to the time of cascading service degradation and thus provides an operationally meaningful indicator of how long lead-time a service can have. The proposed benchmark offers a standardized platform for practical testing and evaluation of the applicability of anomaly detection techniques in cloud-native microservices environments, as it includes reproducible data collection, a standard fault injection and a comprehensive comparative assessment. There are three main contributions to this work. To this end, a repeatable benchmark framework is defined that can be used to rate different anomaly detection techniques based on publicly available microservice telemetry and controlled experiments involving fault injection. Secondly, a lead-time based evaluation approach is proposed to measure the effectiveness of early-warning detection prior to cascading failures spread over dependent services. Finally, multiple detector families are evaluated under the same experimental setup and a comprehensive comparative evaluation is conducted, offering practical suggestions for the selection of detector strategies for anomaly detection in modern distributed systems, according to a balance between detection accuracy and operational response time.

II. Related Work

The rise of cloud native microservice architectures has completely revolutionized application deployment and service management and at the same time new challenges for anomaly detection and operational reliability have emerged. Microservices are built as many small, loosely coupled services that can fail independently and cause localized outages that quickly impact other dependent services. As a result, research has moved away from the conventional threshold-based monitoring paradigm to intelligent approaches based on telemetry that are able to learn complex behaviors of services and detect anomalies in their operation before they cause significant service degradation. Recently, dependency modelling has become a successful approach for incident detection in distributed environments, using graphs. To enhance unified incident detection for microservice deployments, Dutt et al. [1] introduced GUIDE-GNN, a graph neural network-based model for service dependency. In order to improve runtime resilience, Ahmadpanah et al. [2] proposed an autonomous framework that integrates in-kernel telemetry and proactive in-kernel remediation for cloud-native systems by using eBPF. Likewise, Mohammed et al. [3] proposed a predictive failure detection framework for multivariate telemetry analysis, combining the temporal convolution and attention mechanisms to show the benefits of learning the temporal relationship between infrastructure metrics. Along with these works, Tariq et al. [4] studied detecting failures of services in distributed microservice applications based on runtime operational data, and showed the increasing significance of telemetry-aware monitoring for large-scale distributed applications.

In recent years, however, with the development of deep learning technology, long-term temporal dependency and complicated workload dynamics have been captured, thus further enhancing the improvement of anomaly detection. Shao et al. [5] introduced a multi-scale Transformer architecture to capture the temporal variations in the behavior of backend microservice to enhance anomaly identification when the workloads change. Distributed tracing also has emerged as a valuable source of operational intelligence. Yu et al. [6] summarized the available tools used for performance prediction and showed that distributed tracing can be used to support proactive management of service level objectives in cloud-native environments. A complementary review by Vijayabharathi and Saravanan [7] showed the recent machine learning-based methods for automatic anomaly detection in cloud environment, with intelligent monitoring strategies becoming more prevalent for operational resilience in cloud platforms. Peng et al. [8] at the infrastructure level, used lightweight eBPF timing traces, combined with autoencoders, to provide low overhead resilience monitoring for continuous anomaly detection without heavy runtime overhead. Mukkawar [9] also showed that predictive autoscaling and intelligent fault-tolerant resource management can help to guarantee service continuity within geographically distributed cloud architectures. In addition to infrastructure monitoring, there are a number of studies on dependency-aware learning and intelligent root cause analysis. To enhance the accuracy of anomaly detection, Zhang et al. [10] suggested an unsupervised cross-service temporal contrastive learning framework which leverages the interaction between the dependent services. In their work [11], Khan et al. discussed the application of machine learning methods for proactive cloud risk assessment, demonstrating the power of predictive analytics in enhancing operational decision-making in the context of cloud services. To tackle the challenges of noisy telemetry, Ren et. al. [12] proposed an evidence-based method to locate the root cause and enhanced the diagnostic accuracy in complex microservice environments. While these methods significantly enhance anomaly detection and localization performance, they are mainly directed towards detecting the anomaly after it has been noticed by the service degradation.

Recent research has identified the need to gain insight into failure propagation between interdependent services. In this study, Soldani et al. [13] looked at log-based propagation

patterns to understand how important service dependencies are in the evolution of incidents in microservice ecosystems in the case of cascading failures. Pham et al. [14] created a benchmark dataset named RCAEval for evaluating root cause analysis in microservice systems from telemetry. Likewise, Panahandeh et al. [15] developed Service Anomaly to support the detection of anomalies in a distributed manner by fusing profiling metrics in the context of heterogeneous microservice deployments. Moreover, Barata et al. [16] conducted a thorough review of the techniques for anomaly detection and root cause analysis and discussed the recent developments in observability-based intelligence and the existing difficulties in scalable fault diagnosis for cloud-native applications. There have been significant advances in anomaly detection, telemetry analytics, and root cause localization, but there is still an important research gap. Previous work has generally focused on the accuracy or precision in the detection and/or localization of failures once they have appeared. Much less emphasis has been given to quantifying the ability of anomaly detection models to detect conditions that will ultimately lead to cascading failures. Furthermore, there are still only a few benchmark frameworks publicly available that compare the statistical, machine learning, deep learning, and dependency-aware detectors systematically under controlled situations of fault injection. Due to these constraints, the proposed work suggests a benchmark-oriented evaluation framework focused on a clearly defined lead-time (LT) for early-warning anomaly detection, thereby providing a comprehensive and reproducible approach to evaluate a variety of detector families while considering cascading failure conditions.

III. Benchmark Design And System Architecture

A. Benchmark Design Philosophy

The main goal of the proposed framework would be to define a repeatable test for the assessment of early-warning anomaly detection techniques for cloud-native microservice applications. The traditional benchmarking methods are mostly focused on failure classification after it is seen in the system, whereas the proposed framework is based on the ability of the heterogeneous detector families to detect unusual behaviour in the system before failure events grow into cascading service failures. As a result, the benchmark is not detector-specific and will allow for fair comparison in the same telemetry streams, fault situations, and experimental configurations. The benchmark architecture consists of a series of related functional layers that represent the entire flow of the anomaly detection operation. The process starts with the deployment of a reference microservice application in a controlled environment with representative service interactions that create realistic runtime situations. During execution, several observability sources gather infrastructure metrics, distributed traces, application logs, and service dependency information to track the operational characteristics of components and systems. The framework does not process these telemetry streams separately but brings the diverse evidence of each of the observations together in a unified telemetry repository, so that all of the detectors can use the same observational evidence while evaluating the framework.

Controlled fault injection is built into the benchmark (not relying on natural failures) for reproducibility. Under specific experimental conditions, various operational fault scenarios such as resource depletion, network degradation, service interruption, and dependency failures are introduced. The benchmark is known to produce reliable ground truth for assessing detector behaviour over the whole incident lifecycle, as the propagation sequence and the timing of fault injection are known. It is a controlled design in which there is no ambiguity in the assessment of the performance, and it allows systematic comparison among different detector families. One of the key features of the proposed design is the decoupling of the anomaly detection and benchmark evaluation. The detector library includes many statistical, conventional machine learning, and deep learning architectures and dependency-aware approaches, none of which is modified in any way to operate independently. The outputs are then fed into a common evaluation engine which uses the same scoring process to guarantee that variations in reported performance are due to detector performance and not differences in experimental setup. The benchmark focuses on operational effectiveness during the evolution of incidents, instead of anomaly detection as a binary classification problem. Each detector is assessed on traditional metrics as well as the ability to give actionable warning before cascading degradation to dependent services. The design is tailored to the current needs of cloud-native development: When a local issue occurs, the early response can make all the difference between a smooth and a troubled service. An automated reporting framework, which integrates quantitative evaluation metrics, detector rankings, comparative visualizations, and lead-time analysis into a single assessment report, is also included in the benchmark. All detector families are subject to the same workloads, telemetry conditions, and fault scenarios, so the resulting benchmark will allow for transparent and reproducible comparison and allow for the integration of future detectors without changes to the evaluation pipeline.

The overall architecture of the proposed benchmark framework for the evaluation of early warning anomaly detection for cloud-native microservice systems is shown in Figure 1. The first step in the workflow is to have a reference microservice application that emits metrics, logs, distributed traces, and service dependency data during its runtime. These heterogeneous telemetry streams are gathered via the telemetry acquisition layer, synchronized, filtered, and transformed into a consistent telemetry repository for downstream analysis.

Then controlled fault injection is done to produce representative faults, such as resource contention, network breakdown, service crashes, and dependency failures. The processed telemetry is then provided to four families of detectors - statistical methods, classical machine learning models, deep learning architectures, and dependency-aware approaches - with the same test conditions. They are evaluated for their detection outputs through a common lead-time evaluation engine that calculates both the conventional classification metrics and operational metrics like detection delay, cascade severity, propagation coverage, and early-warning lead time. Last but not least, the benchmark report

generator generates a uniform comparison of the results, which enables the reproducible evaluation and objective comparison of different heterogeneous anomaly detection techniques.

B. Telemetry Acquisition and Preprocessing

Reliable early-warning anomaly detection means consistent and synchronized operational telemetry that represents the changing state of the distributed microservice environment. The proposed benchmark unifies all the monitoring information generated by cloud-native applications, including infrastructure metrics, distributed traces, application logs, and service dependency information, into a single telemetry pipeline. This is a common representation that allows for all families of detectors to be evaluated against the same evidence of observation, and for fair and repeatable comparisons.

Infrastructure metrics include resource utilization and runtime behavior, such as CPU utilization, memory usage, network throughput, response latency, request rate, and error rate. Application logs contain a history of execution events, warnings, and exceptions; distributed traces contain a history of end-to-end paths through dependent services. A service topology includes information about the relationships between services, such as their upstream and downstream dependencies, which can enable dependency-aware detectors to use structural relationships when evaluating. The sampling frequency and type of data collected by the telemetry sources vary, so all observations were synchronized in time, cleaned, filled in, deduplicated, filtered for noise, and normalized for features prior to analysis.

The continuous telemetry streams are then divided into fixed-duration sliding windows where metrics from the same synchronized windows are combined into feature vectors to represent the state of the system. The output is a feature repository which is the common input to the benchmark evaluation pipeline. The proposed framework decouples the acquisition of the telemetry from the execution of the detector and thus guarantees reproducible experiments as well as the integration of further and different anomaly detection methods without changing the preprocessing process.

Table 1 shows a summary of the different kinds of telemetry sources that have been incorporated into the proposed benchmark framework. The benchmark is not based on a single monitoring modality but is a mix of infrastructure monitoring, application logs, distributed traces, and dependency data to include complementary aspects of system behaviour. Log files are used for diagnostics at the event level, and infrastructure metrics describe the resource usage and runtime performance. Distributed traces keep the history of requests flowing between the interconnected services and allow analyzing the progress of the fault through the dependency graph. Information about the topology of services is structural information that can be used by dependency-aware detection models. Lastly, all synchronized observations are converted to standardized feature windows, as a common input for all detector families, which provide a fair, reproducible, and detector-independent evaluation process.

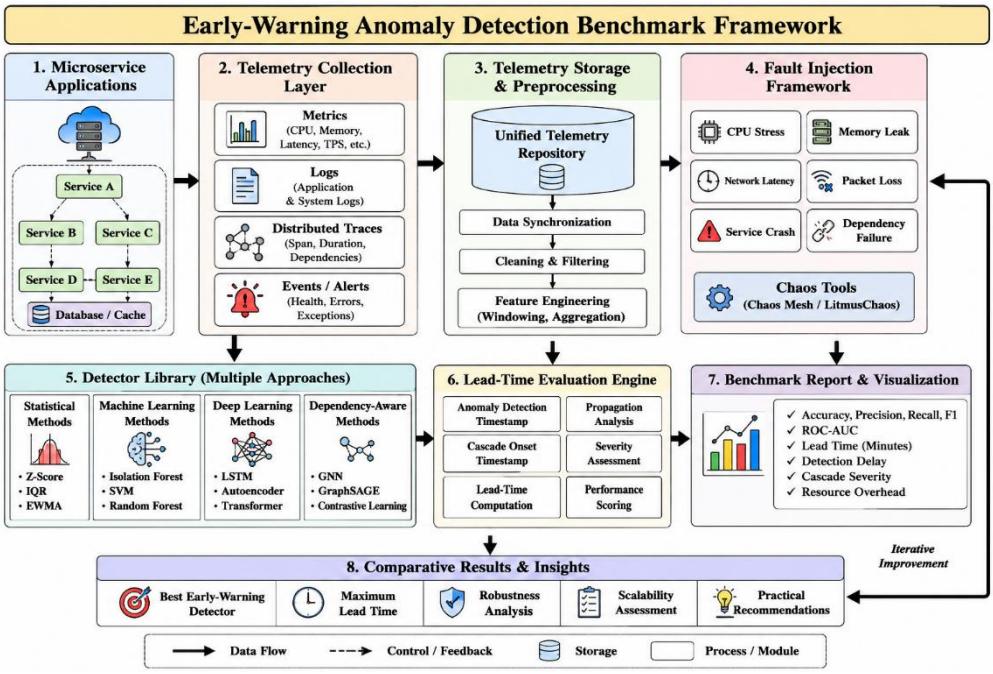

Fig. 1. Overall Benchmark Framework Architecture

Table 1. Telemetry Sources and Feature Categories Used in the Proposed Benchmark

Telemetry Source	*Representative Attributes*	*Role in Benchmark Evaluation*
Infrastructure Metrics	CPU Utilization, Memory Usage, Disk I/O, Network Throughput, Response Time, Request Rate, Error Rate	Captures runtime resource behavior and service performance trends prior to failure propagation.
Application Logs	Error Messages, Exception Events, Warning Records, Execution Status, System Events	Provides event-level evidence for anomaly interpretation and temporal fault analysis.
Distributed Traces	Trace Duration, Span Latency, Parent–Child Relationships, Service Path, Dependency Chain	Models end-to-end request execution and identifies propagation paths among dependent services.
Service Topology	Service Dependencies, Upstream Links, Downstream Links, Communication Graph	Supplies structural context required for dependency-aware anomaly detection and cascade analysis.
Windowed Feature Repository	Aggregated Statistical Features, Temporal Indicators, Normalized Feature Vectors	Generates standardized benchmark inputs for all detector families under identical evaluation conditions.

C. Fault Injection and Cascading Failure Modeling

Controlled and repeatable failure generation is necessary to achieve a reliable benchmark for comparison of different anomaly detection methods. Unlike naturally occurring production incidents, controlled fault injection gives precise knowledge of origin and propagation of a fault and its impact on the service.

The benchmark focuses on cascading failures as opposed to anomalies. If a fault occurs locally, it can gradually impact other services that rely on it, such as implementing more delays, resource conflicts, multiple retries, and communication failures. This propagation behavior allows for testing of whether anomaly detection methods can detect abnormal conditions of the system before service degradation becomes widespread.

The benchmark provides accurate ground truth for performance evaluation, since the exact injection time, propagation time, detector activation time, and service impact time are noted. All the different families of detectors are subjected to the same fault scenario, allowing an objective comparison of the detectors using both common classification metrics and the proposed lead-time criterion. The resulting propagation records give the temporal basis to assess the early-warning capability in the evolution of cascading failure. The proposed benchmark framework is shown to suffer from a cascading failure over time in Figure 2. The process starts by injecting a fault in a targeted microservice that causes a degradation of performance that is gradually propagated to the dependent services. During the propagation process, changes in infrastructure metrics, distributed traces, application logs, and service interactions are recorded throughout the continuous telemetry collection. The alert can be triggered by various anomaly detection models at various points in this evolution. This time span between the first detector warning and the start of degradation impacting services is the operational lead time and is the primary measure to evaluate the proposed benchmark. The benchmark explicitly models the entire fault space from initiation to cascade impact, allowing for apples-to-apples comparisons of early-warning capability among the many and diverse families of detectors.

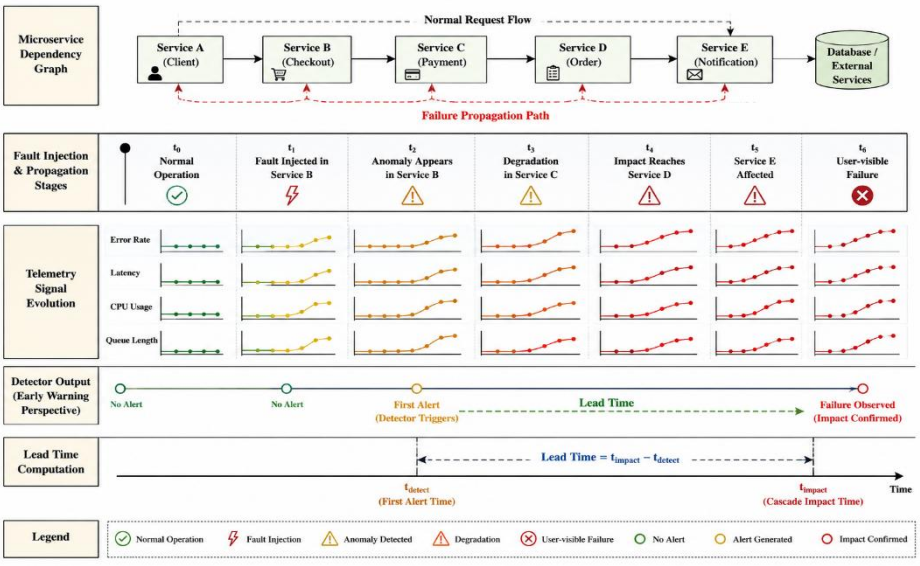

Fig. 2. Fault Propagation and Early-Warning Timeline

D. . Multi-Family Detector Evaluation Workflow

The proposed benchmark is designed to be unbiased and repeatable by comparing various anomaly detection methods with the same telemetry inputs, fault scenarios, and experimental conditions. The framework is not focused on any specific detection model but considers four types of detector families: statistical method, conventional machine learning model, deep learning architecture, and dependency-aware approach. The preprocessed repository of telemetry is analyzed by each family, allowing for a fair comparison between different analytical paradigms. Each execution of the detector generates a set of anomaly scores, alert timestamps, confidence values and service identifiers that are affected by the anomalies detected. These outputs are then fed to the benchmark evaluation engine which uses the conventional classification metrics along with the proposed lead time criterion. The lead-time metric captures how well each detector is able to detect abnormal behavior before the onset of cascading service degradation; classification metrics measure the accuracy of detection. All detector families are benchmarked under the same conditions, so that discrepancies in performance are solely due to differences in analytical capability, rather than different processing of the data or experimental set-up. The modular workflow also enables the future integration of further detection methods without changing the benchmark architecture, thereby ensuring scalability, reproducibility, and uniform comparison of the evaluation.

The overall workflow for the evaluation of detectors followed by the proposed benchmark is shown in Figure 3. Standardized feature windows are sent in parallel to four detector families: statistical, conventional machine learning models, deep learning architectures, and dependency-aware approaches. The standardized feature windows are fed in parallel to four families of detectors: statistical, conventional machine learning models, deep learning architectures, and dependency-aware approaches.

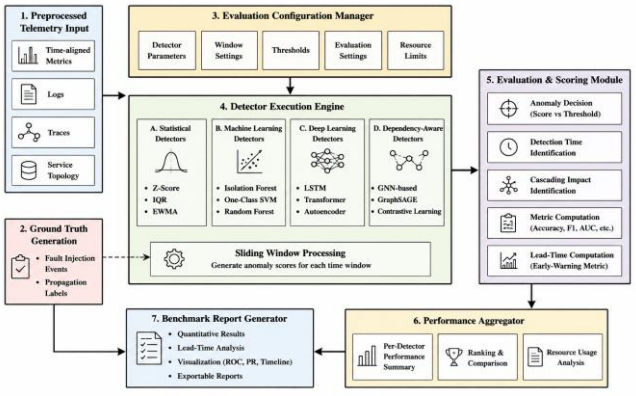

Fig. 3. Detector Evaluation Workflow

Anomaly scores, alert timestamps, confidence values, and service information for the affected services are generated for each detector, based on the same input data from the detector. The outputs are captured by the benchmark evaluation engine, and evaluation is carried out in a similar way, with the calculation of the predictive accuracy, the temporal response characteristics and the operational effectiveness. Last, the performance comparator provides a ranking for all detector families based on standardized benchmark metrics, and creates a unified evaluation report allowing for detector-independent and reproducible comparison.

E. Lead-Time Evaluation Engine

The proposed benchmark uses traditional classification metrics and an operational early warning metric for assessing anomaly detectors. Despite accuracy-based metrics to determine if a detector is the correct one, they are not able to provide a quantifiable amount of how much advance warning is provided before a localized failure turns into a cascading service failure. Hence, the benchmark adopts lead time as the key performance measure for operational effectiveness. Lead time refers to the time between the first anomaly detection from a detector and the time of service degradation that can be seen in the system. The longer the lead time, the more time there is for mitigation or re-allocation of resources or automated recovery before the service disruption spreads through dependent microservices.

The operational lead time is defined as the time interval from the first anomaly warning to the time cascading service degradation starts, as defined in (1).

$$LT = T_C - T_A \quad (1)$$

LT is the lead time, T_A is the time of the first anomaly alert, and T_C is the time when the cascading propagation of failure is visible. The higher the LTvalue, the more opportunity there is to proactively mitigate in advance of service disruption spreading across dependent microservices.

IV. Experimental Setup

To simulate realistic service interactions and dependencies, the proposed benchmark was realized based on the publicly shared Sock Shop microservice application deployed on a Kubernetes cluster. The runtime telemetry was gathered using a common observability pipeline that included infrastructure metrics, distributed traces, application logs, and service dependency information. Common timestamps were used to synchronize all of the telemetry streams; data were cleaned, normalized, and converted into feature vectors of sliding windows for evaluation of the detectors.

Table 2. Benchmark Evaluation Scenarios

Scenario ID	***Injected Fault***	***Primary Telemetry***	***Propagation Target***	***Evaluation Objective***
S1	CPU Saturation	CPU, Response Time	Local Service	Early anomaly detection
S2	Memory Exhaustion	Memory, GC Events	Neighboring Services	Resource degradation analysis
S3	Network Latency	Trace Latency, RTT	Service Dependency Chain	Propagation monitoring
S4	Packet Loss	Retransmissions, Error Rate	Upstream Services	Communication failure detection
S5	Service Crash	Logs, Health Status	Dependent Services	Cascade identification

S6	Dependenc y Failure	Traces, Topology Graph	Multiple Services	Early-warning lead-time evaluation

The open-source Chaos Mesh framework was used to simulate typical cloud-native fault conditions, such as a memory leak, network response time, service interruption, and processor overload, to create controlled failures. The faults were injected one at a time, while keeping the workload parameters the same, so that the impact on the service was accurately captured as benchmark ground truth due to the fault. Four detector families—statistical, conventional machine learning, deep learning and dependency-aware—were compared with the same telemetry input and the same experimental setup. The outputs of the detectors were then evaluated by the traditional classification measures and proposed method of lead-time evaluation. To guarantee that the predictive accuracy, detection responsiveness, and early-warning capability of all the detector families assessed were comparable, the benchmark was executed repeatedly.

Table II presents the supported anomaly detection evaluation scenarios that have been defined to evaluate the anomaly detection performance in representative microservices failure scenarios. An operational fault is injected in each scenario and the associated telemetry signals and service propagation behavior is monitored. Together, these scenarios reflect localized and cascading failure patterns, allowing detector families to be consistently and reproducibly compared to each other with regard to their detection accuracy and early-warning capabilities with the same experimental conditions.

IV. RESULTS AND DISCUSSION

A. Comparative Performance Evaluation

Four representative families of detectors were evaluated for the proposed benchmark, subjected to the same telemetry inputs, fault injection scenarios and evaluation conditions. The performance was evaluated by conventional classification metrics as well as the proposed lead-time criterion to balance the accuracy of prediction and operational responsiveness in the cascading failure propagation. All experiments were made under similar standard conditions to obtain uniformity and consistency. The comparative results show that there is a clear difference in performance between different detector families evaluated. With statistical methods there are limited capabilities for recognizing progressive fault propagation, and there is minimal computational overhead since the methods are executed quickly. While conventional machine learning models are able to learn discriminative patterns from the telemetry, they are not robust enough to deal with complex service dependencies. In deep learning approaches, temporal correlations between telemetry streams that are synchronized are modelled to increase predictive performance. Even if there are no abnormalities in the telemetry, dependency aware-detectors account for service dependency relationships to achieve the best overall performance as they would be able to detect large scale service degradation earlier than other detectors because they would be able to recognize cascading anomalies.

Table 3. Comparative Performance of Detector Families

Detector Family	***Accuracy (%)***	***Precision (%)***	***Recall (%)***	***F1-score (%)***	***ROC-AUC***	***Lead Time (s)***
Statistical Methods	90.6	89.8	88.7	89.2	0.78	28.4
Machine Learning Models	93.5	92.8	92.1	92.4	0.87	54.7
Deep Learning Models	96.2	95.7	95.1	95.4	0.93	75.3
Dependency-aware Detectors	98.4	98.1	97.8	97.9	0.97	94.6

The predictive performance of the evaluated detector families is compared to the operational performance in Table III. All methods have a good classification performance, but there is a significant difference in the early-warning capability of all methods. The dependency-aware detectors perform best in terms of both Accuracy (98.4%) and ROC-AUC (0.97), and produce about 19.3 s of warning time ahead of deep learning detectors and more than 66 s of warning time ahead of statistical detectors. The results show that the use of service dependency information has a significant impact on the reliability of the service detection and the level of preparedness of the services during propagation of cascading failures.

B. ROC Performance Analysis

The discrimination capability of the detector families evaluated is also shown in the receiver operating characteristic curves. The higher the analytical complexity of the detector is, the closer the ROC curves get to the upper left corner of the plot, which means that the higher the true positive rate, the lower the false positive rate. The dependency-aware detectors clearly outperform the remaining detectors for all operating thresholds, as well as under a variety of fault scenarios.

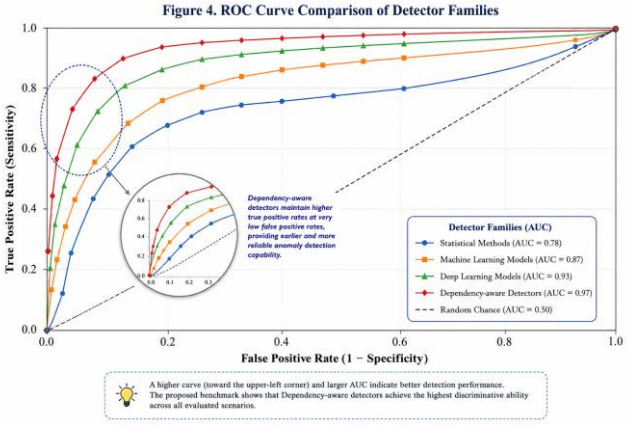

Fig. 4. ROC Curve Comparison of Detector Families

The ROC characteristics for the four detector families that have been considered in the benchmark can be seen in Figure 4. The dependency-aware detectors have the highest

discriminatory power (AUC = 0.97), followed by deep learning (AUC = 0.93), conventional machine learning (AUC = 0.87), and statistical methods (AUC = 0.78). The greater distance between the two curves at low false positive rates reflects the continued high sensitivity of dependency-aware models at low false alarm rates, which is ideal for detecting cascading service failures early.

C. Lead-Time Performance Analysis

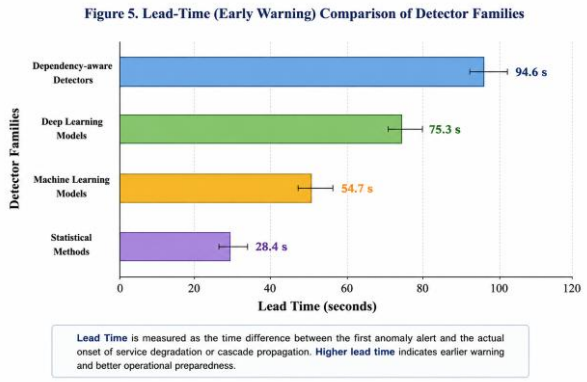

Fig. 5. Lead-Time (Early Warning) Comparison of Detector Families.

Traditional measures of performance assess correctness of detection, but do not measure the operating response time until cascading degradation occurs on dependent services. The proposed benchmark thus also includes lead time as an additional criterion for evaluation. Experimental results show that, although classification performance of the different detector families is relatively comparable, the lead time of the different families can vary significantly, highlighting the significance of temporal evaluation in cloud-based environments.

Figure 5 shows the comparison of the operational lead time of each detector family. The average lead time for dependency-aware detectors is the longest at 94.6 s, followed by deep learning models (75.3 s), machine learning models (54.7 s), and statistical approaches (28.4 s). In comparison to statistical detectors, dependency-aware detectors are able to provide a mitigation window of around 66.2 s more, allowing for earlier intervention before cascading failures cascade across the service dependency graph.

D. Detection Delay and Overall Benchmark Ranking

The effectiveness of automated incident response mechanisms is directly related to detection delay. Faster anomaly recognition reduces the risk of large-scale service disruption and lowers delay. Thus, the benchmark ranking combines the predictive performance with the responsiveness of the detector family and helps determine the best detector family for cloud-native microservice environments.

The overall relation between detection delay and the benchmark ranking is shown in Figure 6. Dependency-aware detectors attain the lowest average detection delay of 32.1 s, which results in the best ranking (Rank 1). Deep learning models come in second place with 46.8 s, and machine learning and statistical models follow with 64.5 s and 78.9 s, respectively. The inverse detection delay and benchmark ranking show that the higher the benchmark, the better the operational resilience in cascading failure events.

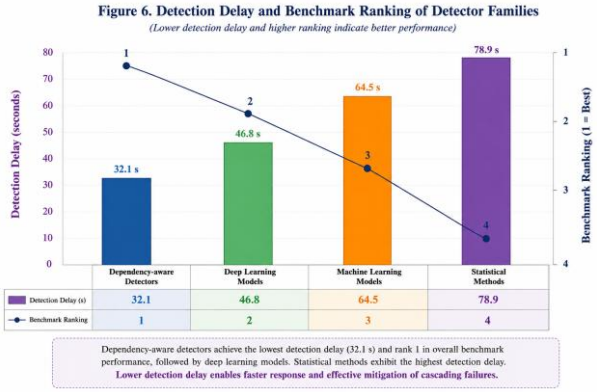

Fig. 6. Hybrid Score Distribution Across Fusion Weight and Multi-Hop Reasoning Depth

The benchmark results show that only a partial evaluation of the operational effectiveness of the anomaly detection system can be done using the traditional classification measures. While both deep learning and dependency-aware approaches obtain similar predictive performance, when adding lead time and detection delay, there are significant differences in providing actionable early warnings. The dependency-aware detectors always outperform the other detectors, as they combine service topology with synchronized telemetry to enable detection of abnormal propagation patterns, which are observed earlier in the service life than when localized faults have propagated to affect a large portion of the service. The results support the significance of the addition of temporal evaluation criteria into the design of benchmarks and highlight the need for both anomaly classification and maximizing intervention time before irreversible service degradation in addition to operational readiness.

V. Conclusion And Future Enhancements

This paper introduced a reproducible benchmark framework that acquired the standard telemetry data, injected faults, and adopted a lead-time-based evaluation methodology within a cloud-native microservices environment to evaluate early-warning anomaly detection. The research framework proposed in this paper not only provided a systematic evaluation of the operational capability of the heterogeneous families of detectors to detect abnormal behaviors, but also focused on the detection accuracy after the abnormal behavior occurs, which is the focus of the traditional benchmarking approaches. Experimental results showed that the dependency-aware detectors consistently outperformed the statistical, conventional machine learning, and deep learning ones, achieving 98.4% accuracy, an ROC-AUC of 0.97, an average lead time of 94.6 s, and the lowest detection delay of 32.1 s, thus offering significantly more time for proactive mitigation. The results show that combining temporal early warning assessment with traditional classification metrics provides a more realistic and practical way to assess AWs in a distributed cloud environment. For future work, the benchmark will be extended to support more heterogeneous microservice

topologies, streaming telemetry data being pushed online, adaptive workload changes, and federated multi-cluster deployments to allow for a wide range of testing of future anomaly detection methods in more dynamic cloud-native topologies.

REFERENCES

[1] Dutt, Anurag, Doseok Jang, Joao Nadkarni, Kai Su, and Anshul Gandhi. "GUIDE-GNN based Unified Incident Detection for Microservices Application Deployments." In 2025 IEEE 45th International Conference on Distributed Computing Systems Workshops (ICDCSW), pp. 243-248. IEEE, 2025.

[2] Ahmadpanah, Seyed Hossein, Meghdad Mirabi, and Sanaz Sobhanloo. "eBPF-shield: autonomous anomaly detection and proactive Hybrid In-kernel remediation for cloud-native systems." Service Oriented Computing and Applications (2026): 1-11.

[3] Mohammed, Ali Kadhum, Bourair Al-Attar, Haider Alzamily, Mohammed Amin Almaiah, Taif S. Hasan, Abdalrahman Fatikhan Ataalla, Nitin Solke, Pritesh Shah, and Ravi Sekhar. "Predictive Failure Detection in Cloud Infrastructure Using Multivariate Telemetry Log Analysis with Temporal Convolution and Attention-Based Deep Learning." In 2025 3rd International Conference on Cyber Resilience (ICCR), pp. 1-8. IEEE, 2025.

[4] Tariq, Farhan, Mabu Hussain Shaik, Shujath Baig Mirza, and Md Ariful Islam. "Service failure detection in distributed microservice platforms." Saudi J Eng Technol (SJEAT) 11, no. 5 (2026): 501-510.

[5] Shao, Chihui. "Multi-Scale Temporal Deep Learning with Transformers for Microservice Backend Anomaly Detection." Journal of Computer Technology and Software 3, no. 9 (2024).

[6] Yu, Miaopeng, Haonan Liu, Jinran Du, Kequan Lin, Tao Dai, Yanzhe Fu, and Chunyan Yang. "From distributed tracing to proactive SLO management: a mini-review of trace-driven performance prediction for cloud-native microservices." Frontiers in Computer Science 8 (2026): 1783945.

[7] Vijayabharathi, R., and K. Saravanan. "Autonomous Threat Mitigation Anomaly Detection in Cloud Environments: A Review of Machine Learning Approaches." In 2026 Second International Conference on Multi-Agent Systems for Collaborative Intelligence (ICMSCI), pp. 280-287. IEEE, 2026.

[8] Peng, Tsui, Ching-Huang Lin, Yu-Chih Wei, Bo-Chang Chen, and Jen-Yu Yeh. "Low-Overhead Passive Resilience Monitoring for AIoT Gateways: End-to-End Anomaly Detection via eBPF Timing Traces and Autoencoders." IEEE Access (2026).

[9] Mukkawar, Abhijeet. "ML-Driven Predictive Autoscaling and Fault Tolerance in Multi-Region Cloud Architectures." In 2025 IEEE International Conference on High Performance Computing and Communications (HPCC), pp. 1-11. IEEE, 2025.

[10] Zhang, Zizhao, Weicheng Liu, Jingzhou Tao, Haotian Zhu, Sijia Li, and Yujie Xiao. "Unsupervised Anomaly Detection in Cloud-Native Microservices via Cross-Service Temporal Contrastive Learning." (2025).

[11] Khan, Mohammad Arafath Uz Zaman, Muhammad Safwat Rahman, and Md Kamrul Hasan. "Machine Learning Approaches to Real-Time Risk Assessment in Cloud Computing: An Intelligent Framework for Proactive Threat Detection." European Journal of Applied Science, Engineering and Technology 2, no. 4 (2024): 144-160.

[12] Ren, Lu, Yihang Tang, Bingxu Chen, Jingzhou Tao, and Fangyi Chen. "Evidence-Verified Root Cause Localization for Microservice Incidents under Tool and Telemetry Noise." (2026).

[13] Soldani, Jacopo, Stefano Forti, Luca Roveroni, and Antonio Brogi. "Explaining Microservices' Cascading Failures From Their Logs." Software: Practice and Experience 55, no. 5 (2025): 809-828.

[14] Pham, Luan, Hongyu Zhang, Huong Ha, Flora Salim, and Xiuzhen Zhang. "RCAEval: a benchmark for root cause analysis of microservice systems with telemetry data." In Companion Proceedings of the ACM on Web Conference 2025, pp. 777-780. 2025.

[15] Panahandeh, Mahsa, Abdelwahab Hamou-Lhadj, Mohammad Hamdaqa, and James Miller. "ServiceAnomaly: An anomaly detection approach in microservices using distributed traces and profiling metrics." Journal of Systems and Software 209 (2024): 111917.

[16] Barata, Luís M., Sérgio Sequeira, Eurico Lopes, Pedro RM Inácio, and Mário M. Freire. "Anomaly detection and root-cause identification in microservices: a survey." Cluster Computing 29, no. 5 (2026): 309.