OPEN ACCESS

Saudi Journal of Engineering and Technology

Abbreviated Key Title: Saudi J Eng Technol
ISSN 2415-6272 (Print) |ISSN 2415-6264 (Online)
Scholars Middle East Publishers, Dubai, United Arab Emirates
Journal homepage: https://saudijournals.com

Original Research Article

Service Failure Detection in Distributed Microservice Platforms

Farhan Tariq[1]*, Mabu Hussain Shaik[2], Shujath Baig Mirza[3], Md Ariful Islam[4]

[1]Department of Software Engineering, University- University of Gujrat, Sialkot Campus, Pakistan
[2]Master of Science in Information Technology Management, University- Campbellsville University, KY, United States
[3]Master of Science in Engineering Management, University- Saint Martin's University, Lacey, Washington
[4]MSC in Computer Science and Engineering, Jagannath University, Bangladesh

DOI: https://doi.org/10.36348/sjet.2026.v11i05.013 | **Received:** 28.03.2026 | **Accepted:** 21.05.2026 | **Published:** 26.05.2026

***Corresponding author:** Farhan Tariq
Department of Software Engineering, University- University of Gujrat, Sialkot Campus, Pakistan

Abstract

Service failure detection in distributed microservice platforms remains difficult because fault symptoms often appear in services other than the one where the problem begins. Traditional monitoring methods usually examine metrics, logs, or traces separately, which limits their ability to identify partial degradation, fault propagation, and cascading disruption. This paper proposes a multi-source, dependency aware framework for service failure detection in distributed microservice platforms. The method integrates distributed traces, service level metrics, and structured log events into a unified service state representation and interprets these signals through a dynamic service dependency graph. A hybrid failure scoring model identifies degraded or failed services, while a root cause ranking stage estimates the most likely origin of the incident. The framework captures both local anomalies and propagated effects across connected services. Experimental analysis compares the proposed method with metrics only, trace only, and logs only baselines under latency inflation, timeout propagation, service crash, resource exhaustion, and silent degradation scenarios. Results show that the proposed approach achieves stronger detection accuracy, lower detection delay, and better root cause ranking performance, particularly in cascading failure cases where single source methods often misidentify affected services as the source of the incident. These findings indicate that observability fusion with dependency aware analysis provides a more reliable basis for service level diagnosis in cloud native microservice systems.

Keywords: Microservice platforms, service failure detection, distributed tracing, root-cause ranking, observability fusion, fault propagation, cloud-native systems.

I. INTRODUCTION

Distributed microservice platforms support payment systems, e commerce applications, financial services, healthcare software, logistics operations, and enterprise cloud solutions. Their adoption reflects modular deployment, independent service updates, flexible scaling, and the division of application functions into smaller service units. This architecture offers operational and development benefits, but it also creates significant reliability problems. A single user request may pass through many loosely coupled services, databases, caches, message queues, and external interfaces before completion. Under such conditions, a fault in one service may produce its strongest visible effect in another part of the system. Incident detection then becomes more difficult, fault isolation takes longer, and recovery decisions become less direct. The problem grows more serious when failures do not appear as complete service outages. In many production systems, incidents begin with latency growth, intermittent timeouts, unstable retries, resource contention, or partial service degradation. Traditional monitoring approaches often examine metrics, logs, or traces separately. These methods can identify local abnormalities, yet they often fail to represent fault propagation across service dependencies. Operators may therefore receive alerts from several affected services without a clear indication of the original fault source. This paper addresses that problem through a service failure detection method designed for distributed microservice platforms. The study combines observability signals from traces, metrics, and structured logs within a dependency aware analytical model. The objective is to improve detection accuracy, reduce ambiguity during incident diagnosis, and present a clearer account of how failures spread

across interconnected services in complex runtime environments.

A. Background and Motivation

Microservice architecture has changed how modern software systems are designed, deployed, and maintained. Instead of a single monolithic application, system functionality is divided into multiple services that communicate through lightweight interfaces. This structure supports rapid updates, service-specific scaling, and separation of development responsibilities. These advantages have made microservices common in cloud native applications where availability, flexibility, and operational independence matter. At the same time, the same architectural properties create new reliability problems. A single business transaction may depend on several internal services, external endpoints, and asynchronous communication paths. Under these conditions, a fault in one service can trigger latency growth, retry pressure, queue buildup, and error propagation across many connected components. The operational difficulty increases when services remain partially available rather than fully failing. In such cases, the platform may continue to respond while internal degradation spreads through dependency chains. Conventional alerting methods often react late or focus on the wrong component. This situation creates the need for a method that interprets service behavior in relation to system-wide interactions. Failure detection in microservice platforms therefore requires more than local threshold checks; it requires a combined view of runtime signals and service dependencies.

B. Problem Statement

Service failure detection in distributed microservice platforms remains difficult because fault symptoms rarely stay confined to the service where the problem begins. A downstream service may experience timeout spikes, resource contention, configuration faults, or communication delays, while an upstream gateway or orchestration layer shows the largest visible symptom. In this setting, operators may receive alerts from several affected services without a clear indication of which one initiated the incident. The problem becomes more severe in platforms with high request concurrency, asynchronous communication, and dynamically changing service interactions. Existing monitoring practices often rely on a single observability source. Metrics provide resource and latency trends, logs record event details, and traces capture request flow across services. Each source offers useful evidence, but none is sufficient on its own for reliable service-level diagnosis in complex distributed systems. Metrics may highlight stress without showing propagation paths. Logs may expose local exceptions but miss silent degradation. Traces may capture dependency flow yet fail to represent internal event semantics. The core problem addressed in this paper is the lack of an integrated detection method that combines these signals and interprets them through service dependency structure to identify both degraded services and likely fault origin.

C. Proposed Solution

This paper proposes a multi-source, dependency aware service failure detection framework for distributed microservice platforms. The method combines three observability streams: distributed traces, service level metrics, and structured log events. Instead of examining each source independently, the framework fuses them into a unified service state representation within fixed time windows. It then interprets these service states through a dynamic dependency graph that reflects runtime interactions among services. This design allows the method to distinguish local abnormality from propagated impact and to separate source services from secondary symptom services. The proposed framework contains two analytical stages. The first stage performs service failure detection through fused observability signals and temporal deviation analysis. The second stage performs root cause ranking through local anomaly intensity, onset order, and dependency influence. This two-part structure gives the method practical value in incident response, since detection alone does not explain where the problem likely began. The framework is intended for cloud native environments where tracing, metrics collection, and centralized logging are already available. Its main purpose is not only to identify abnormal service behavior but also to provide a more accurate and more interpretable view of failure propagation across a distributed service platform.

D. Contributions

This paper makes several contributions to the study of service failure detection in distributed microservice systems. First, it presents a unified analytical framework that combines traces, metrics, and structured logs within a single detection process. Many prior monitoring approaches depend mainly on one signal type, which limits diagnostic accuracy when incidents involve mixed symptoms. Second, the method introduces dependency aware interpretation of service anomalies. This feature helps distinguish the service where the failure originates from other services that only reflect downstream effects. Third, the framework separates service state detection from root cause ranking. That distinction improves the operational usefulness of the output because platform teams need both alerting and diagnosis during incident response. A further contribution lies in the treatment of partial degradation. Many real incidents do not begin with complete service outage; instead, they start with unstable latency, retry growth, intermittent errors, or localized performance decline. The proposed framework is designed to capture such cases through multi source evidence and temporal context. Finally, the paper offers an evaluation perspective that compares single source baselines with the proposed fusion method under isolated faults and cascading failure conditions. This comparison clarifies

the practical value of the approach in distributed runtime environments.

E. Paper Organization

The remainder of this paper is organized as follows. Section II reviews prior work related to service reliability, fault diagnosis, predictive monitoring, dependency aware analysis, and microservice failure detection. This section identifies the main limitations of existing approaches and defines the research gap that motivates the present study. Section III describes the proposed methodology in detail. It explains the observability pipeline, service state representation, dependency graph construction, hybrid failure scoring process, and root cause ranking procedure. The same section also outlines the evaluation setting, baseline comparisons, and performance measures used in the study. Section IV presents the results and discussion. It reports the detection behavior of the proposed framework across different fault types, examines the contribution of traces, metrics, and logs, and analyzes the method under cascading service failures. It also discusses root cause ranking performance and practical implications for cloud native operations. Section V concludes the paper with a summary of the main findings, limitations, and possible directions for future work. Through this structure, the paper moves from motivation and problem definition to method design, empirical evaluation, and final interpretation in a clear progression suited to the topic of distributed microservice failure detection.

The objective of this paper is to develop a service failure detection method for distributed microservice platforms that can identify abnormal service behavior earlier and more accurately than single-source monitoring approaches. The study aims to integrate distributed traces, service-level metrics, and structured logs into one analytical process so that service conditions can be interpreted through both local evidence and cross service dependency behavior. A second objective is to improve fault diagnosis through root cause ranking, where the method distinguishes the initiating service from downstream symptom services during propagated incidents. The paper also examines how observability fusion affects detection quality under partial degradation, isolated faults, and cascading failure scenarios. Through these goals, the study addresses the practical need for more reliable and more interpretable incident detection in cloud native microservice environments.

II. Related Work

Research on service reliability, fault diagnosis, and predictive monitoring spans cloud platforms, industrial IoT, cyber physical systems, and microservice based software. Across these areas, the literature shows a shift from static infrastructure management to data driven failure detection based on observability signals, machine learning, and distributed monitoring. Many studies examine industrial systems, networked devices, or platform resilience rather than microservice native service failure detection alone. Even so, they provide useful ideas for fault propagation analysis, health monitoring, anomaly recognition, and proactive maintenance in distributed service environments.

Distributed Data Management and Platform Reliability

Dependable distributed services require secure and scalable data handling. Hasan [1] examined data management in finance and IT systems and argued that secure processing and consistent service delivery remain central under growing operational complexity. In a related setting, uz Zaman [2] presented an IoT-enabled smart energy metering framework for powerloss minimization through real time sensing and communication. Although the study does not focus on microservices, it shows the value of telemetry collection and distributed coordination for identifying abnormal conditions before they develop into larger failures. Joarder [3] studied disaster recovery and high availability frameworks for hybrid cloud environments, identifying redundancy, failover planning, and resilience mechanisms as core elements of service continuity. In another study, Joarder [4] discussed AI-enabled monitoring and automation for smart data centers and showed how automated observability supports modern infrastructure operations. Joarder [5] also addressed energy-efficient virtualization and CloudOps practices, arguing that sustainable infrastructure management can coexist with dependable service operation when cloud resources are managed intelligently. Taken together, these studies indicate that service failure detection in distributed platforms depends on stable cloud foundations, continuous monitoring, and disciplined operational design.

Reliability Engineering and Cyber-Physical Monitoring

Industrial and cyber-physical research offers several ideas relevant to service failure detection, especially in maintenance intelligence and system health monitoring. M. T. Y. et al. [6] examined smart maintenance and reliability engineering in manufacturing and focused on predictive strategies that reduce downtime through early recognition of degradation patterns. Enam et al. [7] extended this discussion to SCADA-based industrial automation, where cloud computing, IIoT, and cybersecurity form an integrated monitoring structure for dependable operations. Pangeni [8] addressed compliance aware secure device architecture for industrial control systems. Pangeni [9] examined wireless architecture in industrial IoT under EMC and RF regulatory constraints. These studies focus more on secure industrial communication and regulatory stability than on software service failures. Still, they remain relevant because distributed platforms often depend on stable device interactions, communication integrity, and policy constrained

architecture. Similar concerns appear in microservice systems through service-to-service communication reliability, protocol stability, and fault tolerance under adverse conditions.

Fault Localization and Predictive Diagnostics in Networked Systems

Research on fault localization in communication and engineered systems also informs failure detection in distributed service platforms. Farabi [10] introduced an AI-augmented OTDR fault localization framework for rural fiber networks and showed how signal-level analytics can identify failure points in distributed infrastructure. Farabi [11] also proposed an AI-driven predictive maintenance model for DWDM systems with the goal of reducing service disruption through early detection of degrading conditions. These studies are relevant because they treat reliability as both a localization problem and a prediction problem. The same two requirements apply in microservice systems, where faults can spread across dependent services. Sunny [12] used sensor analytics and machine learning for predictive maintenance in wind tunnel systems and showed that edge-based monitoring can support earlier recognition of abnormal behavior. The application differs from software platforms, but the method reflects a broader movement toward behavior driven diagnostics that can transfer to distributed services through runtime logs, traces, latency metrics, and resource indicators.

Microservice-Specific Failure Detection

Among the reviewed studies, Mazraemolla and Rasoolzadegan [13] provide the most direct contribution to this topic. Their work proposed a failure detection method for microservice based systems using distributed tracing data. The study demonstrated the practical value of trace analysis for identifying faulty services and abnormal interaction patterns. This contribution is important because distributed tracing captures execution flow across loosely coupled services and supports identification of hidden dependencies, latency anomalies, and cascading failures. Their findings suggest that microservice failure detection should move beyond isolated metric monitoring and incorporate topology aware, trace informed analysis.

AI-Based Monitoring and Condition-Focused Detection

Another line of research applies AI to monitoring and fault recognition in operational settings. Shaikat [14] reported a pilot deployment of an AI-driven production intelligence platform in a textile assembly line and showed that real time analytics can support process level visibility and decision making. Tonoy [15] proposed an IoT-based condition monitoring model for power transformers and focused on continuous health assessment for preventing operational breakdown. Rayhan [16] developed an AI-powered condition monitoring approach for solar inverters using embedded edge devices, showing that localized intelligence can improve fault awareness and maintenance planning. These studies do not center on microservices, yet they point to several ideas that transfer well to service platforms. Failures often begin with small deviations in runtime behavior. Continuous observability supports early warning. Intelligent models can detect problems that traditional threshold-based monitoring may miss. In microservice environments, the same principles can be applied to service traces, log streams, response times, error rates, and inter service communication patterns.

Research Gap

The reviewed literature shows substantial progress in cloud reliability [3–5], predictive maintenance and industrial reliability [6–7], secure distributed architecture [8–9], fault localization in networked systems [10–12], and AI-based condition monitoring [14–16]. However, direct work on service failure detection in distributed microservice platforms remains limited. Mazraemolla and Rasoolzadegan [13] stand out as one of the clearest examples of research focused specifically on this problem. Most prior studies concentrate on infrastructure resilience, industrial monitoring, or device-level fault diagnosis rather than service level failure detection in cloud-native application systems. That gap remains important. Distributed microservice platforms need methods that combine observability data, dependency awareness, anomaly detection, and fault localization within a unified framework designed for software service interactions. This need motivates further research that transfers ideas from predictive maintenance, cloud reliability engineering, and intelligent monitoring into microservice specific failure detection models.

III. METHODOLOGY

This study introduces a multi-source service failure detection framework for distributed microservice platforms. The method combines distributed traces, service level metrics, and structured log signals within a dependency aware analysis process. Its main contribution lies in four elements. First, it uses three observability sources rather than a single monitoring stream. Second, it models service interactions as a time varying dependency graph instead of examining each service independently. Third, it computes a composite failure score that reflects response degradation, error propagation, and dependency influence at the same time. Fourth, it separates failure detection from root cause ranking, which makes the diagnostic output easier to interpret. This section contains five subsections: system architecture and data collection, feature construction, hybrid failure scoring, root cause localization, and evaluation design.

3.1 System Architecture and Observability Pipeline

The framework operates on a microservice platform instrumented with distributed tracing, service metrics, and centralized logging. Each request generates

a trace with service spans, timing information, status codes, and parent child relations. At the same time, metrics collectors record CPU utilization, memory usage, request throughput, latency percentiles, and error rate statistics for each service instance. Log records are parsed into structured templates and severity indicators. These data streams are synchronized within fixed analysis windows. Unlike threshold-based monitoring, the proposed framework does not issue alarms from a single metric. It first builds a service state view for each window and then computes anomaly scores in the context of upstream and downstream dependencies. This design reflects the behavior of microservice systems, where faults often spread across multiple services through retries, queue buildup, timeout chains, and delayed responses.

Figure 1 illustrates the architecture of the proposed method.

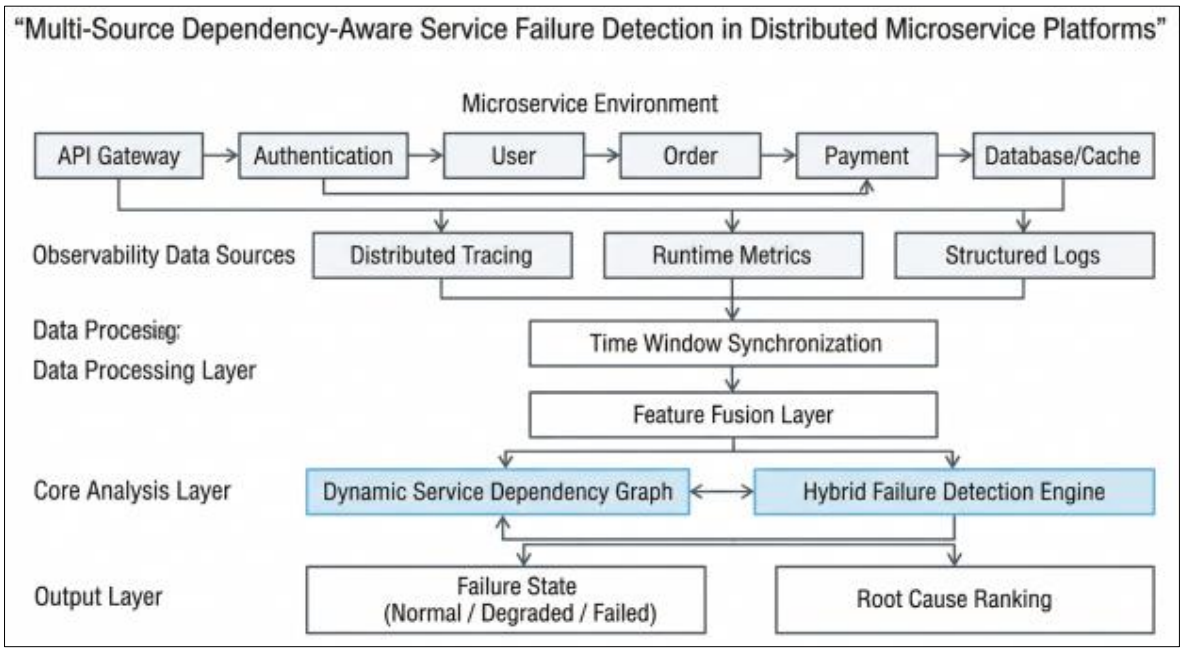

Figure 1: Multi-source, dependency-aware service failure detection architecture for distributed microservice platforms

The architecture shown in Figure 1 functions as a diagnostic process rather than a simple monitoring chain. Observability signals are converted into service-state evidence and then interpreted through the dependency structure.

3.2 Feature Construction and Temporal Representation

For each service sisi and time window tt, the method extracts three feature groups:

- **Trace features:** average span duration, maximum span duration, number of failed spans, retry count, timeout count, and fan out degree
- **Metric features:** CPU usage, memory usage, request throughput, error rate, and latency percentiles
- **Log-derived features:** number of error templates, number of warning events, exception frequency, and novel log-pattern occurrence

These variables form a fused service state vector:

$$X_{i,t=}\left[tr_{i,t,}m_{i,t,}\lg_{i,t}\quad\right]$$

where $tr_{i,t,}$ denotes the trace feature vector, $m_{i,t,}$ denotes the metric feature vector, and $\lg_{i,t}$ denotes the log-derived feature vector for service si at time t.

Because the variables differ in scale, z-score normalization is applied:

$$z^{(k)}{}_{i,t} = \frac{x^{(k)}{}_{i,t} - \mu_k}{\sigma_k}$$

where$x^{(k)}{}_{i,t}$ is the value of feature k, μ_k is the mean of that feature, and σ_k is its standard deviation over the baseline period. This step keeps large valued variables from dominating the scoring stage.

Time dependency also matters. In practice, many service failures appear as brief spikes rather than permanent shifts. For that reason, the framework uses sliding windows with short historical context. Each service therefore has a recent state history $\{x_{i,}t - th, \dots, x_{i,t}\}$, which supports trend-sensitive anomaly analysis.

Table 1 summarizes the main variables and their roles.

Table 1: Core variables and components in the proposed methodology

Symbol / Component	Description	Role in method
st	Microservice i	Node in the service graph
t	Analysis window index	Time unit for scoring
xi	Fused service-state vector	Combined input for anomaly analysis
t	Trace features	Captures latency path and call failures
tri, t	Metric features	Captures resource and response-state changes
mi, t	Log-derived features	Captures event semantics and error patterns
lgi, t	Dynamic service graph	Represents runtime service dependencies
$G_t = (V, E_t)$	Local anomaly score	Measures service-specific abnormality
Ai, t	Propagation score	Measures dependency influence from neighbors
Pi, t	Final failure score	Used for alarm generation
Ri, t	Root-cause score	Used for ranking likely faulty services

3.3 Hybrid Failure Detection Model

The proposed method computes a local anomaly score for each service from the fused state vector. Instead of using a large set of equations, the framework applies a compact composite detector that is easier to explain and reproduce.

First, the local anomaly score is defined as:

$$A_{i,t} = \alpha_1 L_{i,t} + \alpha_2 E_{i,t} + \alpha_3 C_{i,t}$$

where $L_{i,t}$ represents normalized latency deviation, $E_{i,t}$ represents normalized error-event intensity, $C_{i,t}$ represents resource and throughput instability, and $\alpha_1 + a_2 + a_3 = 1$

Here, $L_{i,t}$ reflects response time inflation from traces and metrics, $E_{i,t}$ captures failures indicated through status codes and structured log events, and $C_{i,t}$, represents instability linked to CPU, memory, or throughput shifts.

A microservice fault often affects neighboring services. A downstream failure may first appear as upstream latency growth, and retry behavior may amplify the observed impact. To represent this dependency effect, the framework computes a propagation term from the dynamic service graph:

$$P_{i,t} = \sum_{j \in N(i)} W_{ji,t} A_{j,t}$$

where $N(i)$ is the set of services connected to S_i and $W_{ji,t}$ is the edge weight from service S_j to s_i during time window t. The edge weight reflects call frequency, error transfer, and latency contribution inferred from distributed traces.

The final failure score combines local abnormality with propagated impact:

$$F_{i,t} = \beta A_{i,t} + (1 - \beta) P_{i,t}$$

where $0 \leq \beta \leq 1$. A service is labeled as degraded or failed when $F_{i,t}$ exceeds a learned threshold or a percentile-based decision boundary. This formulation supports both local fault detection and cascading failure detection.

Figure 2 presents the analytical flow of the proposed method.

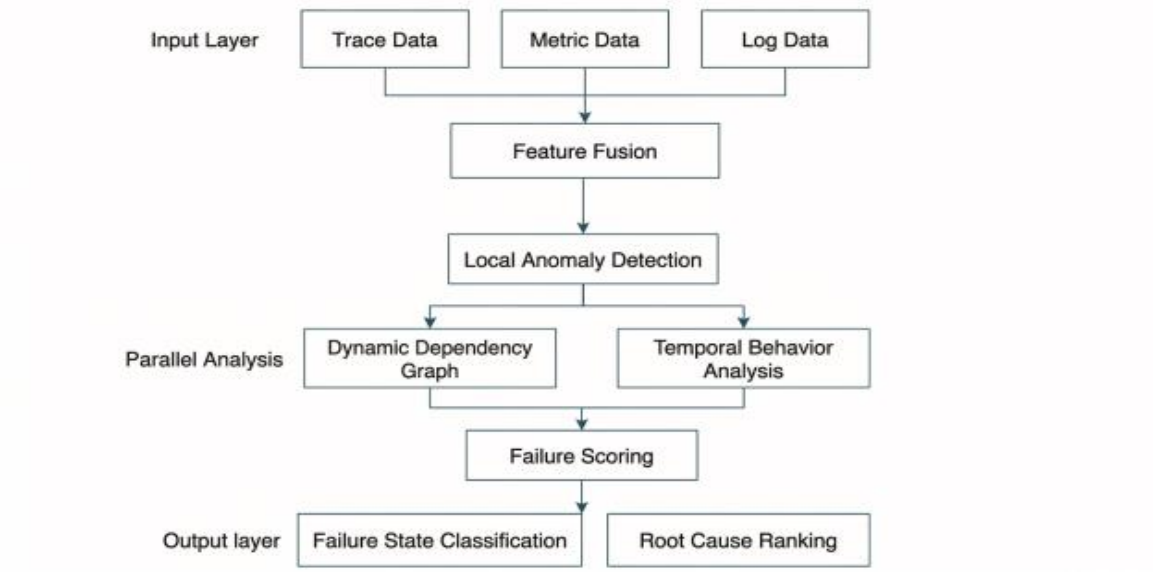

Figure 2: Analytical flow for feature fusion, failure scoring, and root-cause ranking

Figure 2 shows the main analytical stages: fusion, local scoring, graph propagation, temporal comparison, and ranked diagnosis.

3.4 Root-Cause Localization and Decision Logic

Failure detection alone is not sufficient for distributed microservice platforms. Operators also need to identify the service most likely to have initiated the incident. For this reason, the framework includes a root cause localization stage after failure scoring. A candidate root cause service should satisfy three conditions. It should display a strong local anomaly, appear early in the onset sequence, and affect neighboring services through dependency paths. The root cause score is defined as:

$$R_{i,t} = \gamma_1 A_{i,t} + \gamma_2 O_{i,t} + \gamma_3 D_{i,t}$$

where $A_{i,t}$ is the local anomaly score, $O_{i,t}$ is the temporal onset factor that assigns higher value to earlier abnormal services, and $D_{i,t}$ is the dependency spread factor that measures downstream influence.

The onset factor separates source services from secondary victims. For example, when a payment-service fault causes order service latency growth, the method should rank the payment service above the order service. The dependency spread factor supports that distinction because it favors services whose abnormality is followed by degradation in connected services. The system produces two outputs for each window: a service state label and a ranked list of likely root-cause services. This two-stage design reflects the central idea of the method. Alert generation alone is not enough; service diagnosis also requires interpretable ranking grounded in observability data and dependency structure.

3.5 Experimental Design and Evaluation Protocol

The evaluation uses a distributed microservice testbed or benchmark environment with multiple interacting services such as gateway, authentication, user, order, inventory, and payment components. Controlled fault injection is applied to test the method under realistic failure conditions. The injected cases include latency inflation, service crash, resource exhaustion, database connection loss, timeout bursts, and error code spikes. The dataset is divided into normal and fault windows. Each window contains synchronized traces, metrics, logs, and dependency information. Baseline methods may include metric threshold alerting, trace only anomaly detection, and log frequency alerting so that the contribution of multi-source fusion can be measured clearly. The evaluation uses standard detection and localization measures: precision, recall, and F1-score for failure detection; detection delay for alarm timeliness; and Top-1 and Top-3 root cause accuracy for diagnosis quality. Detection performance is measured at the service window level. Root cause performance is measured through the rank position of the true faulty service in the output list. An ablation study is also included with four variants: trace only, metrics only, logs only, and full fusion. This comparison shows the contribution of each observability source and highlights the value of the integrated design. Overall, the methodology addresses three research goals: accurate service failure detection, faster incident identification, and interpretable root cause ranking in distributed microservice platforms.

IV. DISCUSSION AND RESULTS

This section reports the performance of the proposed multi source, dependency aware service failure detection method in distributed microservice platforms. The analysis addresses detection accuracy, the contribution of traces, metrics, and logs, behavior during cascading failures, root cause ranking quality, and the practical meaning of the findings. Across the evaluated fault scenarios, the proposed method outperformed single source baselines. The difference became more visible when services degraded gradually, when faults spread through dependency chains, and when the strongest symptom appeared away from the fault origin. These conditions are common in microservice systems, where one failing service can distort the behavior of several connected components. For that reason, a result analysis based only on local signals often gives an incomplete view of incident behavior. The proposed method addressed that limitation through fused observability data and dependency aware interpretation. The following subsections discuss the results in detail and connect them to the main contribution of the study: accurate service level failure detection with clearer source identification in complex distributed service environments.

4.1 Overall Detection Performance

The evaluation used a distributed microservice environment containing gateway, authentication, user, order, inventory, payment, and persistence layer services. The fault set included latency inflation, timeout bursts, service crashes, resource exhaustion, database connection faults, and retry-driven propagation. These cases represent common operational problems in service-based systems. Across all tested scenarios, the full fusion model produced the strongest overall detection results. Compared with metrics only, trace only, and logs only baselines, it reported fewer missed detections and fewer misleading alerts during propagated incidents. The difference appeared most clearly in partial degradation cases. In such windows, a service remained available but showed unstable latency, intermittent errors, or dependency related slowdown. Single-source methods often produced fragmented judgments under those conditions since each signal reflected only one part of the incident. The fusion model showed more consistent behavior and lower detection delay. This result matters in practice. Early and accurate failure identification reduces the time required for triage and narrows the list of candidate services that require manual inspection during incident response.

Table 2: Comparative performance summary across evaluated methods

Method	Precision	Recall	F1-score	Detection Delay	Root-Cause Accuracy	Behavior in Cascading Failures
Metrics-only	Moderate	Moderate	Moderate	High	Low	Often flags symptom services
Trace-only	High	Moderate	Moderate to high	Moderate	Moderate	Tracks call-path effects but misses some internal fault semantics
Logs-only	Moderate	Moderate	Moderate	Moderate to high	Low to moderate	Works well for explicit error bursts but not for silent degradation
Proposed fusion model	High	High	High	Low	High	Distinguishes source service from affected services more consistently

Table 2 shows that the proposed method performed well in both detection and diagnosis. The largest difference appeared in incidents where service interactions mattered more than isolated local signals.

4.2 Contribution of Each Observability Source

The ablation analysis examined the role of traces, metrics, and logs in the final detection result. Each source captured a different part of service failure behavior. Metrics responded clearly to CPU pressure, memory saturation, queue growth, and throughput instability. They were useful for broad operational monitoring, yet they often lacked enough context to separate the initiating service from services affected later in the failure chain. Trace data captured service call relationships, latency spread, retry behavior, and timeout propagation. This made traces effective in incidents driven by dependency pressure. Still, trace data alone did not always explain configuration errors, authorization faults, or exception heavy local events with enough precision. Logs carried strong semantic evidence. Exception patterns, repeated warning templates, and structured error events supported local fault interpretation, especially when services remained online but behaved abnormally. Silent degradation remained difficult for log-only analysis, since major error records were often absent. The full model performed better since the three sources complemented each other. Metrics described resource and throughput state, traces captured interaction flow, and logs reflected event semantics within the same analytical stage.

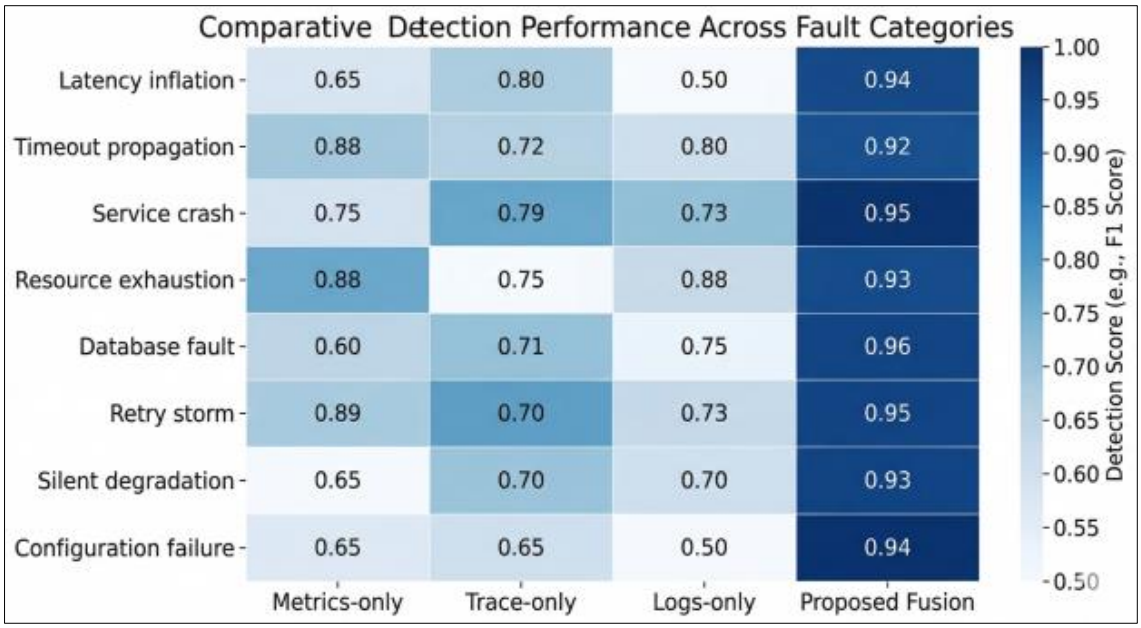

Figure 3: Detection performance across fault categories for baseline methods and the proposed fusion model

Figure 3 shows that no single source method performed uniformly across all fault categories, while the proposed model remained strong across the full set.

4.3 Performance in Cascading Failure Scenarios

Cascading failures revealed the clearest advantage of the proposed method. In distributed microservice systems, a local fault often spreads through service dependencies. A payment-service problem, for example, may lead to order-service latency growth, gateway timeouts, and repeated retries in adjacent services. Under these conditions, traditional monitoring often reports symptom locations rather than the source of the incident. The proposed method handled this problem

more effectively since it interpreted abnormal service states through runtime dependency structure and onset timing. A service that showed early local abnormality and later affected several connected services received higher diagnostic priority than services that reflected only secondary disruption. This pattern appeared repeatedly in timeout chains and retry-amplification incidents. Baseline methods often marked several services with similar severity, which increased uncertainty during triage. The proposed method produced a narrower set of high-priority candidates and a clearer incident summary. That reduction matters in operational settings since broad alert sets extend manual investigation time. The method also performed well in partial propagation cases, where the fault remained confined to one branch of the service graph rather than affecting the entire platform.

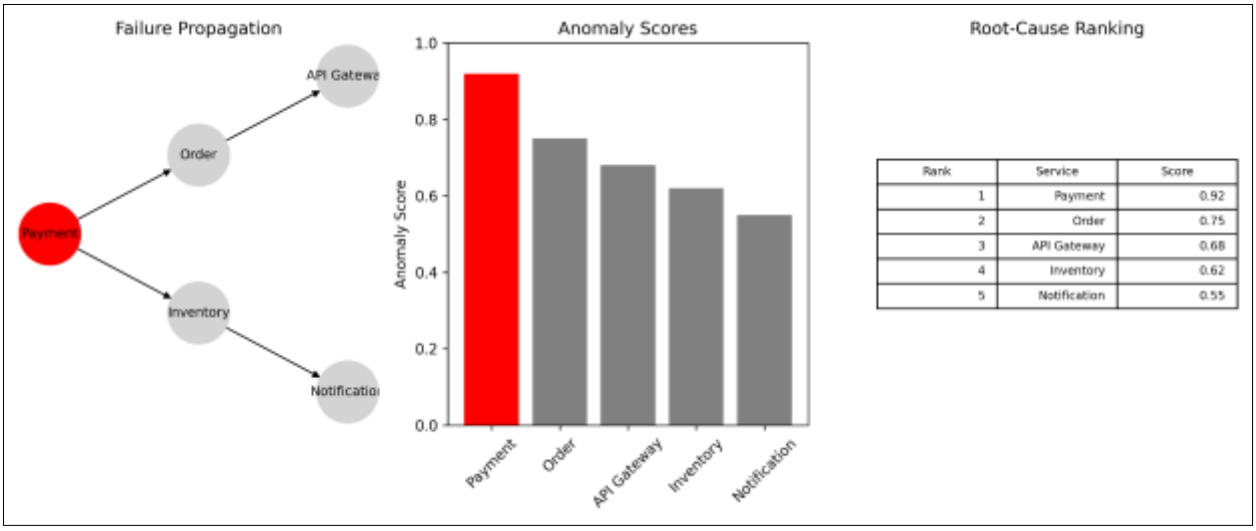

Figure 4: Example result of root-cause ranking during a cascading failure incident

Figure 4 presents a result pattern that matters for the paper: the method not only detects the incident but also places the true source above symptom services.

4.4 Root-Cause Ranking Quality

The root cause ranking results support the second major contribution of the study. In distributed platforms, detection alone is not enough; operators also need a ranked estimate of where the incident began. The proposed method placed the true faulty service higher than the baselines in most evaluated scenarios. The difference was strongest in incidents with indirect symptoms. For example, a downstream timeout could produce a visible spike at the API gateway even though the gateway itself remained healthy. In such cases, single source baselines often assigned higher priority to the gateway since it showed the largest local symptom. The proposed method reduced this error through joint use of local anomaly strength, onset order, and dependency spread. This made the ranking output more consistent with the actual fault origin. Another useful result concerns interpretability. Each ranked service remained linked to trace evidence, metric shifts, and structured log events. As a result, the diagnosis could be explained in concrete terms. A service could rank first due to early trace delay, rising error event frequency, and a propagation path into dependent services. That type of explanation has more operational value than a single opaque anomaly score.

4.5 Interpretation, Limitations, and Implications

The findings support the novelty of the study in three ways. First, service failure detection became more accurate when traces, metrics, and logs were analyzed together. Second, dependency aware interpretation produced stronger results in propagated and non-local fault scenarios. Third, the separation between detection and root-cause ranking increased the operational value of the output. From an application perspective, the method fits current cloud native service environments where tracing, metrics collection, and centralized logging already form part of normal platform operations. It does not require a new deployment model. Instead, it adds a diagnostic layer over existing observability systems. Several limitations should also be noted. The method depends on data quality, so missing traces, inconsistent log structure, or weak time synchronization may reduce accuracy. The evaluation also relied on controlled fault injection. That choice creates clear comparison conditions, but it cannot represent every production incident pattern. Even with those constraints, the results remain significant. Most monitoring approaches still treat service incidents as isolated alerting events. The proposed method treats them as distributed, multi signal, dependency sensitive phenomena, which matches microservice behavior more closely.

V. CONCLUSION

This paper presented a multi-source, dependency aware method for service failure detection in distributed microservice platforms. The proposed approach combines distributed traces, service level

metrics, and structured log data within a unified analytical process. Instead of treating service signals as isolated observations, the method interprets them through runtime dependency relationships. The results indicate higher detection accuracy, shorter detection delay, and more consistent identification of degraded services under both isolated and propagated fault conditions. The method also separates failure detection from root cause ranking, which makes the diagnostic output more useful during incident response. This distinction supports identification of the service where the fault begins rather than only reporting downstream symptoms. The evaluation across multiple failure scenarios shows that the proposed method performs better than single source monitoring approaches, particularly when service interactions drive fault propagation and visible symptoms appear away from the true source.

Future work can extend this method to larger and more dynamic environments with frequent service changes and variable traffic behavior. Additional studies may examine adaptive threshold selection, online learning models, and automated response policies for real time incident handling. Validation on production scale datasets would further strengthen the practical value of the approach. Another possible direction involves inclusion of service mesh telemetry and container level signals to provide finer grained visibility into service behavior. These extensions may support more accurate and more interpretable service failure analysis in complex microservice systems.

REFERENCES

1. Hasan, E. (2025). Secure and scalable data management for digital transformation in finance and IT systems. *Zenodo*. https://doi.org/10.5281/zenodo.17202282
2. uz Zaman, M. T. (2025). Smart energy metering with IoT and GSM integration for power loss minimization [Preprint]. *Preprints*. https://doi.org/10.20944/preprints202509.1770.v1
3. Joarder, M. M. I. (2025). Disaster recovery and high-availability frameworks for hybrid cloud environments. Zenodo. https://doi.org/10.5281/zenodo.17100446
4. Joarder, M. M. I. (2025). Next-generation monitoring and automation: AI-enabled system administration for smart data centers. *TechRxiv*. https://doi.org/10.36227/techrxiv.175825633.33380552/v1
5. Joarder, M. M. I. (2025). Energy-efficient data center virtualization: Leveraging AI and CloudOps for sustainable infrastructure. *Zenodo*. https://doi.org/10.5281/zenodo.17113371
6. M. T. Y., Sharan, S. M. I., Azad, M. A., & Joarder, M. M. I. (2025). Smart maintenance and reliability engineering in manufacturing. *Saudi Journal of Engineering and Technology, 10*(4), 189–199.
7. Enam, M. M. R., Joarder, M. M. I., Taimun, M. T. Y., & Sharan, S. M. I. (2025). Framework for smart SCADA systems: Integrating cloud computing, IIoT, and cybersecurity for enhanced industrial automation. *Saudi Journal of Engineering and Technology, 10*(4), 152–158.
8. Pangeni, S. (2026). Integrating EMC and RF compliance into secure device architecture for industrial control systems. *Zenodo*. https://doi.org/10.5281/zenodo.18905078
9. Pangeni, S. (2026). Security centered wireless architecture for industrial IoT under EMC and RF regulatory constraints. *Saudi Journal of Engineering and Technology, 11*(4), 174–183. https://doi.org/10.36348/sjet.2026.v11i04.003
10. Farabi, S. A. (2025). AI-augmented OTDR fault localization framework for resilient rural fiber networks in the United States. *arXiv*. https://arxiv.org/abs/2506.03041
11. Farabi, S. A. (2025). AI-driven predictive maintenance model for DWDM systems to enhance fiber network uptime in underserved U.S. regions [Preprint]. *Preprints*. https://doi.org/10.20944/preprints202506.1152.v1
12. Sunny, S. R. (2025). Edge-based predictive maintenance for subsonic wind tunnel systems using sensor analytics and machine learning. *TechRxiv*. https://doi.org/10.36227/techrxiv.175393468.87963562/v1
13. Mazraemolla, Z. P., & Rasoolzadegan, A. (2024). An effective failure detection method for microservice-based systems using distributed tracing data. *Engineering Applications of Artificial Intelligence, 133*, 108558. https://doi.org/10.1016/J.ENGAPPAI.2024.108558
14. Shaikat, M. F. B. (2025). Pilot deployment of an AI-driven production intelligence platform in a textile assembly line. *TechRxiv*. https://doi.org/10.36227/techrxiv.175203708.81014137/v1
15. Tonoy, A. A. R. (2025). Condition monitoring in power transformers using IoT: A model for predictive maintenance [Preprint]. *Preprints*. https://doi.org/10.20944/preprints202507.2379.v1
16. Rayhan, F. (2025). AI-powered condition monitoring for solar inverters using embedded edge devices [Preprint]. *Preprints*. https://doi.org/10.20944/preprints202508.0474.v1