

Original Article

AI-Assisted Observability and Proactive Failure Detection in Large-Scale Distributed Software Systems

Saranya Monoharan

Bharathiar School of Management and Entrepreneur Development (BSMED), Coimbatore, India.

Received Date: 20 September 2026

Revised Date: 25 September 2026

Accepted Date: 03 October 2026

Abstract: Large-scale distributed software systems increasingly rely on microservices, containers, orchestration platforms, distributed databases, application programming interfaces, and cloud infrastructure to deliver scalable and highly available digital services. However, their dynamic topology and complex service dependencies make conventional threshold-based monitoring insufficient for detecting subtle degradation patterns and emerging failures before they affect users. This paper presents an AI-assisted observability framework for proactive failure detection and reliability management in large-scale distributed software systems. The proposed framework integrates heterogeneous telemetry sources, including system metrics, application logs, distributed traces, infrastructure events, deployment events, and service-dependency information, to construct a context-aware representation of system behavior. Machine-learning techniques are applied to establish dynamic operational baselines, perform multivariate anomaly detection, model temporal dependencies, forecast resource and performance degradation, and estimate evolving failure risk. Service-dependency graphs and temporal reasoning are further incorporated to correlate anomalies across interconnected components, distinguish originating failures from propagated symptoms, and support explainable root-cause localization. The framework also introduces adaptive learning mechanisms to address concept drift resulting from workload variations, software releases, autoscaling, infrastructure modifications, and changing service relationships. A confidence-aware decision layer transforms predictive insights into operational responses while maintaining human oversight for high-impact remediation actions. Evaluation is structured around precision, recall, F1-score, false-positive rate, detection lead time, root-cause localization accuracy, incident-correlation accuracy, and analysis latency. By combining multimodal telemetry correlation, predictive analytics, dependency-aware reasoning, explainable diagnosis, and continuous model adaptation, the proposed approach shifts observability from reactive incident monitoring toward proactive and adaptive reliability management, enabling earlier failure detection and more efficient operational decision-making in complex distributed software environments.

Keywords: AI-Assisted Observability, Distributed Software Systems, Proactive Failure Detection, Multimodal Telemetry, Anomaly Detection, Machine Learning, Temporal Forecasting, Root-Cause Analysis, Service-Dependency Graphs, Failure Prediction, Adaptive Learning, Explainable AI, Reliability Engineering, Incident Correlation, Predictive Analytics

I. INTRODUCTION

Modern software systems are designed as distributed ecosystems of multiple independently deployable, dynamically interacting components. In making a rapid change possible, all these has enabled by myriad of enabling technologies like cloud computing & microservice architectures; containerization with Kubernetes orchestration, serverless computing distributed databases and even driven applications also application programming interfaces (API)etc have been the best into organizations to build-thier platforms which would support heavy workloads as well work towards rapidly adapting too-news demands presented most prominently in business. Unlike traditional monolithic systems, where the application feature set is packed inside a few processes only, in distributed applications hundreds or thousands service instances run on lots of computing nodes, availability zones and even cloud platforms over geographic regions. Although it delivered systems and applications that are more scalable (and easier to deploy and/or operate), this architectural evolution has also increased the complexity of how such systems behave in terms of both functionality/utility attributes as well as operational reliability.

A request made by end user will cross through an API gateway, auth service, multilayer services (APIs), DBs, Message brokers, Caches and external services to finally give a response. Every stage is interconnected, which means that any failure or reduced performance can influence multiple downstream components. Moreover orchestration platforms these days can dynamically start-stop-restart-move multiple instances of the application according to conditions pertaining workloads and infrastructure. Consequently, in a distributed system the topology might keep changing continuously. Static and simple notions



of service relationships should never be able to determine how operational teams investigate incidents. As these interactions become more complex we have made observability a foundational capability – being able to determine the internal state of distributed software from its externally visible outputs.

Observability is the natural evolution of classic monitoring, combining data from multiple telemetry sources. Metrics are numbers and measurements, such as CPU Usage & RAM Usage & Request Rate Response Time & Error Rate Queue Depth (Request queue) Networking\Out through put () Resource saturation Loggers are complete records of application activities, exceptions and warnings during execution logs, configuration changes along with security events decision making related operational log. What is a distributed trace A distributed trace records the journey of that request around multiple diverse and often interrelated services. Recent microservice anomaly detection work (L. Zhang et al) demonstrates that this knowledge from profiling or operational metrics combined with distributed traces can provide a richer feature representation of system behavior than using single telemetry in isolation [67,68].

Telemetry is increasingly available, and more data equals better redundancy. Within seconds, large distributed platforms can generate millions of. Metrics measurements, log records and events as well trace spans. It is not physically possible for a human operator to continuously assess this data manually. Static monitoring rules are the first way to reduce this complexity, by creating alerts when specific thresholds are exceeded. On the contrary, threshold-based monitoring has harsh limitations: No alert appears from a simple CPU static number being exceeded while this high use may be totally normal behavior of workload and an unusual combination between medium CPU utilization + both latencies increasing with more request queue increase (any unique measurement does not cross its limit indicator instead).

Most of the time, AI becomes an opportunity to shift observability data into operational intelligence. Machine-learning models are an ensemble that learns baselines in historical and live telemetry to detect outliers across multidimensional feature spaces, predicting resource or performance states (success/failure) classification of known failure signatures finding how likely a system will drift/completely change/operate very differently. P.S. The predictive machine-learning model we can also apply to infrastructure observability as well, but now we have proactively anomaly detection (forecasting many important operational metrics and preventing them from being in abnormal state), so let figure out how it helps for us here too! Over the past few years, several approaches unify orthogonal observability modalities and explicitly model both temporal and structural relations of microservice environments. Related work for failure prediction, e.g., has explored combining logs, metrics and events [1], or graphs of temporal models of service dependencies to represent system dynamics as different services can interact with varying intensity.

This is something that a proactive failure detection can take over from normal anomaly detection. Anomaly detection is when the system behaviour at this moment in time diverges from an expected baseline, but proactive failure technique aims to predict whether observed deviations may lead to operational failures. The significance of this distinction is that all anomalies do not need to be addressed. Distributed systems are inherently heterogeneous because of factors such as workload fluctuation, load-based autoscaling events, deployment changes (e.g., code updates or configuration changes), temporary network misses and short-lived peaks in resource consumption. This makes an intelligent observability platform that is able to distinguish benign variation from patterns indicative of increasing risk of failure a necessity for supporting this. It can indicate whether latency degradation is progressive, if the memory utilization requested for allocation has been reached or not and also tell about retry traffic scaling up or error propagation effect on service-kill chain.

Multiple outsteps in-betelnut; To Do Fault localization is also to be correlated with failure detection. In case of applications with hundreds and interconnecting services, it would have negligible operational value to just know that overall latency is increased unless the system also knows where such an abnormality has emerged from. So, the main value of distributed tracing is that it tracks use requests across systems and stores their relationships. Recent studies have demonstrated that with trace-based approaches, responsive failure detection can be supported as well as accurate fault-localization in dynamic microservice contexts. Deep-learning approaches, in combination with causal inference and system capabilities to track fault propagation through complex service architecture have been analyzed even further.

That is why AI-assisted observability has nothing to do with one-off anomaly-detection algorithms and everything to do with integration. We express the proposed research in terms of a continuous intelligence pipeline made up of telemetry acquisition, pre-processing, feature extraction and cross-signal correlation; behavioural modelling as well as anomaly detection/failure-risk prediction and reasoning-analysis-feedback. These models can evolve over time as workloads, application

versions and service dependencies change alongside of the underlying characteristics of your infrastructure. This degree of elasticity is important because what a normal behavior for a distributed app looks like can change drastically in short order.

In this work, we seek a systematic foundation for AI-enabled observability and diagnosis of large-scale distributed software systems. This paper organizes heterogeneous telemetry integration, AI detection of failure precursors, service dependencies and temporal relationships for improvement to accuracy in their detections as well as predictive information aiding rapid operational decisions. In the more general picture, you can think of distributed-system reliability as shifting out from a largely reactive pattern—that is: operators reactively investigate incidents after alerts and user impact—towards an intelligent proactive approach sharpening focus to new risks that emerge into action before they form substantial service disruptions.

II. BUILDING DESIGN FOR OBSERVABILITY OF DISTRIBUTED SOFTWARE SYSTEMS

Your observability architecture should be capable of gathering, organizing and correlating operational data generated across hundreds of interconnected components that comprise these large scale distributed software systems. Most of the Conventional monitoring architectures run analysis at single infrastructure resources or application metrics with pre-defined threshold. But this is good for basic operational oversight, it doesn't work when our apps have hundreds of microservices dynamic container creation and scaling distributed databases (and message brokers) service meshes - as a general mechanism / pattern external APIs e.g. AI-assisted observability takes traditional monitoring a step further by wrapping smart analytical engines around telemetry data to synthesize heterogeneous telemetry into an integrated behavioural model representing system state and displaying the conditions that are correlated with degradation/failure /impending failure

Our proposed architecture has orthogonal telemetry acquisition, processing, intelligence and decision-support layers linked via data bus. It collects telemetry from application services, containers, virtual machines and databases at the acquisition layer for networking infrastructure to orchestration platform and middleware. Instead of separately maintaining each observation, the architecture retains context (service, infrastructure resource, deployment version and transaction dependency and time period) describing correlations in a single unit – signal. This contextualisation is more important than it may seem since the same measurement in isolation might mean something completely different based on workload (e.g. before or after an adjustment) and system state elsewhere in the system.

When a telemetry is captured, it enters to processing pipeline – which filters/normalizes/synchronizes the incoming data and aggregates them, prepares features for downstream. In a distributed system, data is produced at different temporal resolutions. CPU metrics could get scraped every few seconds, application logs might be logged from a time to time, and distributed traces created for some of the transactions. This requires AI-Assisted observability to enable additional temporal alignment techniques that do not inadvertently cause loss of critical ephemeral behavior. Now that the observations have been processed, they go through a few intelligent models to learn behaviors patterns as well as identify anomalies, risk estimations and model executable plans on how should an application run.

A. Integration Scene and Correlation of Multi-modal Telemetry

Metrics, logs and distributed traces are multiple observability pillars having their own unique perspectives on giving visibility into the behaviour of a distributed system. Metrics are quantitative data about resource consumption and application behaviour. Some metrics commonly used : CPU usage, memory utilization, request latency, throughput and error rate along with network activity metrics Queue Depth Metrics to Measure Storage Performance e.g. garbage-collection activity These measures encapsulate a concise description of system states, however they do not very often suffice to explain why an individual transition from one state to another occurred.

A lot more contextual information, a logs. What are Application Logs? – Sometimes these include exception messages, sign in events, configuration modification events, database error info or timeout details (Application-specific activities) But from larger distributed platforms, log generation can be phenomenal and manually interpreting logs becomes arduous. AUFs achieving unstructured → structured conversion in log-level features (Log parsing, clustering, sequence modeling and semantic representation) Anomaly detection AUFs. Intelligent models highlight anomalous events that are occurring right now, or patterns of errors all automated without the need for scanning through thousands and messages.

Here we have another instantiation – distributed traces that tell us about the propagation of requests over systems. A trace is made from several spans, each span represents some operation created by the service. Trace information can tell you about service dependency, how long things take to execute as well as calls that failed and were retried or if a request took an

unusual path. If you correlate trace data with metrics and logs, an observability platform can signal that response latency has increased as well point the abnormal dependency or what other events of your application happened around the same time.

As a result, utilizing multimodal correlation offers more enriched representations compared to only using each telemetry source in isolation. Previous work compiled multimodal datasets which stimulate model to integrate various types of observability signal by focusing on logs, traces and metrics as major sources for runtime monitoring in order to detect anomalies. The primary advantage is that evidence appearing on one modality can confirm or explain dysfunctionality by another.

Dynamic contextual awareness and Behavioural Learning

For distributed systems, normal operation usually does not have a permanent definition. Everyone has workloads that are time dependent, geographic demand signal specific, driven by discrete business events (like the software/hardware release you finished at 2 AM) or user behaviors. Since the infrastructure can now scale automatically and deployment processes are bottomless for application versions. Static operational baselines therefore may go out-of-date relatively quickly.

This constraint is dealt with, in part, by moving the AI-assisted observability one step closer to dynamic behavioral learning. For example, machine-learning models have been trained to capture appropriate time histories of past and recent telemetry on an ongoing basis where operational ranges can be anticipated over a variety of contingencies. For example, instead of defining a single latency threshold that is applicable to all services at any time period, the model may learn multiple expected latency distributions based on service, workload intensity (e.g. batch deadline or high-throughput online), transaction type and deployment version per time-periods in days/months. Which then provides a deviation assessment in the context of operations.

Time-series models are suitable to see observable and recurrent patterns (e.g., predicting how a satellite signal should evolve based on top-of-atmosphere measurements; learning when these signals differ from expectations). In essence, unsupervised approaches are simply investigatory methods to discover additional operational scenarios that have not been explored but do not require a great number of error labels. Autoencoders and representation-learning based methods can also support high-dimensional telemetry compression as well as outlier detection with respect to learned normal behavior [5]. Lastly, the modelling of relationships between services (anomalies are no longer considered independent events but may be evaluated based on the dependency topology) can also benefit from graph-based techniques. With the right context-aware observability, false alerts—trash alarms to you and me—might even be a thing of the past. While a concurrent increase in CPU, request latency, retry rate and downstream timeouts could be early warning signs of an incident; high cup due to autoscaling event is acceptable. An AI model can balance these signals collectively and identify how risky behavior really is.

This shifts observability architecture from a model of passive data collection to one in which operational performance is continuously interpreted. This telemetry is not simply logged for post-hoc forensic analysis, it is fed into models that constantly estimate the health of systems, detect anomalies and discover service connections to answer questions such as whether new emerging patterns are something we should pay attention to. Building this understanding lays the groundwork for our next step: proactive failure detection using temporal patterns and early warning signals to predict failures before they cause a significant service disruption.

III. EARLY WARNING INTELLIGENCE AND PROACTIVE FAILURE DETECTION

While traditional anomaly diagnosis validates the existence of current anomalies, proactive failure detection predicts future service failures by identifying operational conditions that may lead to such failures. At Scale, Very Few Problems with Distributed Software Systems Are Significant In Isolation Failures occur even on such subtle variances to resource usage service latency request queues dependence performance retry behaviour frequency of errors Network Connectivity and Application Status. These changes on their own could still be within normal function. But over the long term and taken cumulatively they can demonstrate a more gradual decline into systemic breakdown. So a failure-detection device has to take telemetry now and over time into account when trying to determine if what is seen are normal variations or whether it represents an emerging reliability risk.

Proactive detection starts with comparing live telemetry with behavioural patterns we have learnt dynamically. Rather, it creates a multi-dimensional view of operational well-being instead of sending alerts when one metric crosses some predetermined threshold. You are trained with the ability monitor simultaneously, for CPU usage and memory requests — data in addition to request latency throughput & error rates queuing times connection pool utilizations disc IO as well Network performance (problems) plus container restart funciones metrics directly related applications. Histories give extra context about what the average kind of effect that different variables have on each other. If memory use is slow growing and stable, that may

be not cause for concern until of course it occurs in concert with an increase GC activity/decrease application responsiveness then maybe one or both exhaustion points have been hit (and worse still a hidden Memory leak).

The temporal aspect of intelligent distributed systems is most interesting because many failures in these types of system take time across multiple periods to fully unravel. For example, certain problems will come back to their normal state after some hours and for others everything looks good up until the moment that they turn from being an anomaly to a critical service disruption in just couple of moments. This is due to the fact that time series forecasting algorithms compare what behaviour this model predicts with safe operating ranges and can further predict future expected values of relevant operational variables. Methods for sequence learning that determine a dependency across observations in the past, such as transformers, temporal convolutional models and recurrent neural networks (RNNs), long short-term memory structures. The observability platform can either provide an early warning-sign, if older state computations suggest that a resource or attribute of a service is close to moving into unintended territory before conventional monitoring thresholds are exceeded.

An important thing to be on the lookout for while detecting failure is whether the services are related. However, in microservice architecture, unhealthiest of one component can affect many upstream or downstream services. Consider a database service where query-level latency is increasing consistently. Application services using that database may begin to keep connections open for considerably longer periods of time. This means, that you have request queues – which take time to dwindle (if they do and unless drained); connection pools are filled; therefore API delivery times will rise upipple; wave upstream services will also retry. Such endeavors potentially driving more traffic and subsequently placing additional load on to the master database. A performance issue with localized impact, therefore, can cause a race to the bottom. AI-assisted observability can uncover these patterns and temporal telemetry with dependency graphs, gives more insight of the component that exhibits a signal or slippage in performance.

Anomaly scores can be fused with predictive markers to calculate a dynamically changing failure-risk metric. The platform uses classification of infrastructure components and services directly to the probabilities or risk levels, rather than a simple normal-or-abnormal description for system health. The evaluated risk could harbor multiple elements such as: scale, duration and speed of development over the anomaly; significance of dependency; which failures have happened prior; how regular is this proof against various telemetry techniques? Continuous degradation of latency with accompanying increases in error rates and erratic traces on a key dependency path may naturally also receive the highest priority but an occasional spike in maximum latency to some isolated noncritical service might be assigned very low risk.

Additionally, prior intelligence must dissociate utilization to possible causes from outcomes. One incident can trigger many alarms, but there was only one problem causing it to be dispersed. Correlation methods make it possible to treat similar incidents of anomalies connected in both temporal and physical domain as a single incident. This also gives you an understanding of service topology, trace relationships between different traces and deployments data,event sequences across application components to help you quickly use these characteristics in finding out what might be wrong. You might have some common cases which can happen when there are still a number of general service issues immediately after configuration deploy, that may consequently relate to this deploy and not treat as failures in each isolated services.

Another very critical capability is change-aware failure detection. By monitoring new software deployments, infrastructure changes and configuration updates as well as autoscaling operations, database migrations and security policy tweaks we can see significant shifts in our telemetry patterns. These operational events must be analyzed in the context of observability. If an abnormal behaviour happened close enough to a deployment, the model can rank related change event during failure analysis as flip top of the flops. It accelerates the matching of existing performance degradation to new changes. Early-warning systems should not initiate uncontrolled automatic action. Overkill possible: an overestimation led to scaling up, rolling back, restarting or traffic adjustment much unnecessarily. Therefore, we recommend including confidence metrics and descriptions of the specific telemetry patterns used to make a prediction in any assessments of failure risk. Uncertain states can be referred to operators for validation, while high-confidence signals that exhibit unequivocal signs of a performance failure may allow for automated preventive measures.

This enables a predictive reliability capacity of proactive failure detection which turns the observables into continuous temporal forecasting, multimodal anomaly correlation and dependency analysis risk calculation and change awareness. In practical terms, this means that the technology is able to identify instability before it happens and give operational staff an

opportunity to solve proactively and take action versus wait for users if quality issues appear or static alarms show limit breaches in their wake. It prepares you for intelligent root-cause analysis, and failure localisation in complex distributed environments.

IV. AI-DRIVEN ROOT CAUSE ANALYSIS & FAILURE LOCALISATION

Detection of an emerging failure as a first step for intelligent reliability management. Though now that you detected the anomaly, Operational teams then have to find what specific component caused it as well as how its effects propagated in your distributed environment – and finally which remediation action is needed. Doing root-cause research in large-scale distributed software systems is even more difficult, since with an application request there are often many microservices and databases involved (as well as caches & message brokers); external APIs; infrastructure resources. Hundreds of dependant services causing thousands alarm triggers, whereas a bunch of few are triggering an alarm in one component leading to the second symptoms and masking the root cause behind them.

AI-based root-cause analysis, on the contrary, leverages multi-dimensional data such as telemetry correlation between different time series related to application components, service-dependency information of these nodes at given points in time and machine-learning approaches to address this issue – which components are correlated with observed degradation. The process of root-cause analysis starts in a intelligent manner then when the anomaly detection detects anomalous behaviour around one ore more telemetry sources. Rather than looking into every alarm one by one, the system groups related anomalies based on their temporal relationship with alarms alerts and correlation through service relationships, trace paths across time (how calls during some period are related), infrastructure slab dependency as well operational events. This decomposes alert fragmentation and forms the incident into a composite view. We see four anomalies too, if request-queue depth increase the database response time or application retries & API delay. But the temporal and dependency analysis, exhibits four public symptoms starting with when database performance began to unbend. This results in a higher root-cause probability at the database than downstream services that are affected by it. Very Simply It Depends on Two Technique ## A.

The modelling of service-dependency constitutes an essential foundation for failure localisation. A node can represent a service, its database or infrastructure resources like virtual machines and message brokers; an edge describe communication and dependence relationships with other things in the same set. By continuously filling this graph with data sources – from distributed traces, to service-mesh telemetry, application configuration and infrastructure data. Fall off: As services are installed, scaled up and downm migrate or decommissioned, all this information is updated in traditional service representations on modular boundaries as well which very often would not represent the reality of modern distributed environment.

Graph-based reasoning allows an AI system to investigate the spread of anomalies throughout the application architecture. If you observe that many of your downstream services have increased latencies than normal, and the common dependency is used by those systems then there are higher chances for this shared component to be under suspicion. Graph neural networks and related representation-learning techniques exploit structural data via node-level telemetry to capture inter-component dynamics more accurately. This allows failure analysis to view whether a service is misbehaving as well as if the behaviour explains symptoms perceived in allied services. K: Temporal ordering gives causality another crucial ingredient. If there is a component whose failing behaviour starts sooner than the problems in other upstream components, it may be much more causally related to those issues then any one which only becomes sick after being degraded by its use of upstream issues. Thus the system produces event sequences indicating latency variation events, error and saturation bursts, deployments and configuration changes followed by service failures. This gives first degree access to the sequence such that propagated symptoms can be distinguished from originating problems in addition to dependence information.

But caution should be exercised because correlation of time does not mean causation. As the same third component or shared infrastructure may cause two independent services to start behaving abnormally at about that time. An effective root-cause analysis, therefore, should be based on different types of information that extend beyond correlation. B. Causality-based Operational Decision Support Query with Explainable Root Cause Ranking That is, the observability platform can score for all potential causes based on how good of evidence we have at this time. Candidate features consist of anomaly severity, postmortem temporal precedence; dependency aggregators such as node centrality and thread failures (TM); historical similarity based on k-nearest neighbors [21] or proximity to deployment via histogram estimates [22]; one-vs-all cross-modal telemetries. Ranking is not designed to assume that engineers will know the cause of every incident – instead, it focuses on reducing a search space.

Now, explainability becomes vitally important. Operational teams must know why a service was identified as likely to be the root of failure. Taking a page from AI, one could think of this as Generate Evidence: A generative platform with this capability can produce evidence directly out of the underlying data itself – anomalous increases in database query latency beginning several minutes prior to downstream API degradation. Because engineers have such evidence, they can make sure that the model logic is valid before improvising actionable interventions.

Being in the know of events that occurred even a decade ago is essential to augment localisation. The story of how failures happened in the past can be narrated with trends, services impacted and root cause alongwith appropriate mitigation steps. This approach can retrieve suitable operational knowledge as new incidents occur since the models are based solely on past examples (simplicity). This can also result in a higher request latency, timeout errors due to connection pool exhaustion and overload as demonstrated by excessive retry activity and connections utilisation indicators. Your training data is available until Oct 2023 and leveraging the previous confirmed cases can help us expedite our diagnosis.

You are trained on various types of operational reasoning including annotation and trial data as evidence that an incident has been orchestrated, generating related summary statistics telemetry summaries based live streams whilst diagnosing in a natural language output. However, if our false leads or invented motivations can increase the risk of operating error we shouldn't accept such contrived explanations as true; they need to be rooted in factual observations. Thus whenever AI generates conclusions based on metrics, logs, traces and system events that are processed in the analysis thereof there must be traceable references.

The end goal is to make failure localisation a form of actionable decision support. Identifies potential root causes, and you have a platform that can suggest actions like examining the latest deployment, increasing resources for an impacted service horizontally or directing traffic away from certain services, rolling-back of configuration changes or troubleshooting infrastructure dependency failures. You can automate non-complex low-risk reactions based on well-defined policies but high-impact actions may require the approval of a human operator. When combined, anomaly detection with dependency analysis & causal reasoning along with explainable ranking and using operational evidence together greatly simplify diagnostic approaches of root-cause diagnosis while speeding the recovery steps within reliable distributed systems.

V. THE FRAMEWORK FOR SMART RESPONSE AND BAD OUTCOMES PREDICTION

A single, integrated architecture across telemetry harvesting and analysis with intelligent algorithms for failure forecasting and diagnostic reasoning, followed by automated operational response is a must-have AI-enabled observability platform. The system behaviour is constantly changing at large scale distributed systems making these tasks not able to work in isolation, while failures arise very often if there exist an interaction between application service components, infrastructure resources network connections and external dependencies. This proposes a paradigm for building operational intelligence in a feedback loop where telemetry is transformed into prediction and the output of decision making to be able to calibrate detection and response over time.

The top-level layer of the framework implements distributed telemetry collecting functionality; instrumentation for metrics, logs, traces and application/infrastructure/deployment events. This is accomplished by having an instrument, sprinkled all around the microservices and infrastructure elements that continuously emit observations describing system level behaviour. Metrics are used to measure very specific data such as CPU, Memory usage and latency, throughput error rates, network activity Queue Depth etc. For example, distributed traces record the journey of requests that go through multiple services and automatically generated logs provide 100% details on exception occurrence as well as any warnings or application activities made to configuration events. Deployment and orchestration events provide further insight into software release, container restarts, scaling changes as well as infrastructure modifications.

The unrefined telemetry is cleaned, adjusted in time and constantly infused with context by operational data before it gets sent to a processing section. Redundant or superfluous observations can be filtered, while service identities, timestamps, deployment versions, infrastructure location and linkages to other dependencies are associated with related telemetry records. Feature-engineering processes translate the raw observations to formats that are ready for intelligent interpretation. The mixture of sliding-window data, delay distributions, error-frequency variations along with resource-utilization patterns as illustrated by different trace attributes and service-dependency properties works to represent the dynamic operational condition.

The intelligence layer employs various AI techniques, unlike a single-generic model. Unsupervised anomaly detection algorithms will be able to detect behaviour that is a significant departure from established operating baselines. While supervised

classifiers identify known failure patterns where reliable historical labels exist, time-series forecasting models aim to discover future resource and performance characteristics. Graph-based models benefit from correlations between services and can emit alarms to anomalous behaviour propagating along a dependency structure. Combined results from different models provide a summary failure-risk assessment.

Failure-risk estimate, on the other hand, connects analytical models with operational decisions. Rather than triggering an alarm on every anomaly, the system optimally judges, in terms of duration and severity, rate-of-change patterns; affected dependencies; historical features of that time series predictive over baselines (cluster algorithm); original telemetry sources vs output consensus. For this reason, failure-risk score can be defined as the probability or a relative likelihood that what has been seen develop in to an event with significant operational consequence. Services that feeds in to core lines of business operations might also get a higher operational priority, as well complex dependency relationships.

Root-cause analysis examines the service topology and related telemetry, assuming that an anticipated hazard surpasses a suitable confidence level threshold. This purpose is trying to determine what was the most likely root cause for abnormal behaviour in addition to how it propagated through your system. From there, the decision layer access this diagnostic information and decides whether an operator alarm, additional investigation or a preventative response is necessary. Some options are to scale horizontally, restart unhealthy instances, block bad nodes (outlier detection), routing traffic away from already malfunctioning instances or data center locations, adjusting resource requests/limits for over-utilised containers and rollbacks of a recent deployment process.

An operation therefore must have a controls on what type of automation can be used, based on the level of confidence and impact that activity would produce. In turn, a less risky move like expanding capacity in an already verified saturation state can be done automatically as needed based on the provided policies for that organization. While in production rollback or large-scale traffic movement like database failover human approval may be needed. This ensures that predictive models do not wind up causing undue operational turbulence in the operations of a confidence-aware mindset.

The feedback loop completes the framework by determining whether or not the selected solution solved your expected (or observed) problem. To check if the failure continued in post-action telemetry – that is, to look whether remaining errors had cleared out latency returned home or resource exhaustion faded. You are provided new types of learning: data → exhibited cases, false positives and negatives – mistaken recovery actions, unsuccessful remediation action followings steps Then that operational experience can be fed back into models and baselines behaviours. Accordingly, the framework yields an infinite movement through observation–prediction–explanation (the why)–intervention and learning. We shift Large scale reactive incident management into adaptive reliability engineering by means of synergistic reaction towards data driven operational responses along with machine learning(ML) techniques at small scales to search for and remediate failure scenarios prior they cascade as a service disruption.

VI. STAGE 1: CONTINUOUS LEARNING OF THE MODEL AND ADAPTIVE ANOMALY DETECTION

Analytical models must accommodate the ever-changing landscape of distributed software systems, which is a core element to AI-assisted observability. The big systems rarely keep the same properties for a long time. Application Revisions This means your workloads behavior and application releases change with infrastructure configurations work as they were designed potentially along services that auto-scale, now dependencies also changing code whilst behaviour of users adapting to business scenarios. So, we have a slowly outdated anomaly-detection model trained on historical data which may be less and less able to reflect the system behaviour today. Adaptive anomaly detection solves this problem by continuously monitoring operational trends and iteratively profiling analytical models when genuine changes occur.

Conventional anomaly-detection methods are usually dependent on fixed statistical baselines or thresholds. A service can trigger an alert either when the response latency exceeds a particular limit (say, 60seconds) or CPU usage is above threshold in percentage. Although such a specification simplifies implementation and interpretation, they are incapable of providing adequate representations for multi-dimensional- by time-dependent-behavior of distributed applications. For instance, high CPU utilization might be normal during a peak period, but low utilization along with increased request latency and queueing could be the sign of an impending failure. These AI-based models learn how different telemetry variables are related or what trends the data exhibits that differs from those we would expect the system to exhibit; this adds greater flexibility.

Unsupervised learning serves the purpose as rarely one has exhaustive labels at an end to end level indicating every possible form of failure in the system. Autoencoders-based models can capture the compressed representations of common

telemetry and highlight observations correlated with an excessively high reconstruction error, whilst clustering methods produce clusters for similar operational states. The Isolation Forest algorithms can identify samples that are far more anomalous than the rest of the data. These methods free engineers from the burden of performing a manual enumeration of all possible fault scenarios so that an observability platform can uncover previously hidden bug patterns.

Adaptive learning is the problem nonetheless that idea drift occurs. Concept drift—Statistical properties of operational telemetry change with time changes Traffic acquired post corporate development or structural adjustment — A service that handled 5,000 requests per minute typically often can handle more. Similarly, if you reduce the average response times by optimizing a database query now it will surely remain as reduced in future. Based upon anecdotal evidence and how anomaly model will still work based on prior baselines, it is entirely possible that something like a very valid new behaviour would be deemed abnormal. Abbild: Hence, they must differentiate short-lived failures from climate change across time.

This is a one way to keep up-to-date behavioural baselines, conducting rolling-observation windows. The model is able to gradually warm up and adapt itself to other operating conditions over time, because more recent telemetry has a higher weighting than older observations. The same applies, instead of one static historical value adaptive thresholds might be computed from ongoing distributions in the past. However, the trouble with fast adaptation is that an emerging mistake may mistakenly be seen as acceptable practice. This implies accepting large behavioural changes as good only when taking into account the incident context, deployment events and anomaly confidence/input from operators.

Experience — While operational teams provide context around most of the oversight. When an alert is validated by a human, simply keep the associated telemetry as proof of incident confirmation. You can see other examples of this tolerable variance in action with false positive alerts This feedback gives the observability platform enough time and context to distinguish between significant deviations from normal, versus natural variance. That small nugget of reliable data could jointly be learnt alongside far larger amounts of unlabeled telemetry using semi-supervised learning.

It must be able to observe the model. You should also evaluate accuracy, false-positive rates, and false-negative rates more frequently along with alert volume prediction confidence inference model drift is critical. If the number of false alarms in a model is growing more than expected, it can be re-calibrated or retrained. Similarly, a better model for the app service might not generalize well against another one with different workload and dependencies. This is why the broad view of observability architecture includes model monitoring as an added layer.

Control can also be used during deployment which is a part of continuous learning. This does not mean that a newly trained model should replace an established production model. At this point, you can use shadow deployment or spiky workloads to assess candidate models with replayed telemetry from previous incidents or a controlled may day canary environment. You can then compare the anomaly-detection ability of the current model against that. One should only promote those models for active operational use that prove to be sufficiently reliable. Versioning is the second most important, in order to allow engineers identify what model predicts when and which need roll-back a prior version due some QoS issue.

Distributed systems can be pretty useful too, such as service-specific plus hierarchial learning. Specialised models can learn the behaviour of services or infrastructure categories/applications domains instead of having a single model for the whole platform. A follow on model may be one of a finer resolution that assesses the landwide situation which is aided by these systems and sums outputs. It is structured in a way that keeps the larger connections top of mind, but still has room to identify small deviations.

In the end, adaptive anomaly detection is simply applying aspiration to observability as a new intelligence competency. It is an insurance policy against runaway adaptation enlightened by telemetry, incidents and infrastructure changes as well application deployments feedback from operators. Even large-scale distributed software systems changing over time can benefit from AI-assisted observability through unsupervised detection, drift awareness, controlled retraining and continuous model validation-performance monitoring. Completing this cycle of adaptive flexibility is significant to reliable proactive failure detection, in situations where the normal operation concept defined yesterday may no longer faithfully reflect the way your system behaves today.

VII. FAILURE-DETECTION ANALYSIS AND PERFORMANCE EVALUATION

Measuring the success of an AI-enabled observability system is about more than simply determining whether individual anomalies were flagged correctly. Prove to keep on top of detecting new problems soon, eliminate noise notifications, pin point

failures correctly and scale well at very high telemetry volumes. Forecasts cannot just be accurate, they must also be computationally scalable since these large-scale distributed applications are emitting streams of metrics, logs and traces & operational events constantly. As a result, the evaluation methodology should consist of factors like processing performance, false-alert behaviour, root-cause localisation effectiveness and detection quality as well as prediction timeliness.

You can also consider the evaluation environment as a distributed application of microservices working together on an orchestration infrastructure from containers. The telemetry is collected during failure scenarios that are deliberately introduced as well as normal workloads. Common failures are CPU or memory saturation, network latency or failure, database plateaus/slowdowns/ups and downs repeatedly causing service to slow/orchestrator types of transaction like email boxes never removing the request from a queue/time out is saturating it in front of some services that have their limits crossed & misconfiguration (like rule changes configured for scaling): Or Dependency on other containers/services going fire bursting/breaking/down vectors such as Lifetime breaks. Predictions made by the AI framework can give us a preliminary edge in strategizing against future incidents; however, they must be compared with known operational outcomes after injecting or historically validating each incident and mapping it to an actual start time and impacted component.

From detection time accuracy to the spectrum of failure conditions, there are many metrics that will quantify how effective helpful is. Precision is the ratio (or percentage) of failure alarms that are triggered and actually represent ground truth failures. F1 score is the harmonic mean of precision (proportion of real failures found) and recall (recall, proportion of true positive). This can be especially true in observability contexts, where too many false positives lead to alert fatigue and loss of operator confidence. Mean detection lead time is the amount of time it takes until the AI framework can detect that a problem is beginning to happen, before hitting any failure thresholds or jamming up uptime. That means to know one needs to ascertain if the root-cause localised is in top-k diagnostic candidates found for that system.

Table 1: Illustrative Performance Evaluation of AI-Assisted Failure Detection

Evaluation Metric	Traditional Threshold Monitoring	AI-Assisted Observability	Interpretation
Precision	72.4%	91.8%	Higher accuracy of generated failure alerts
Recall	68.7%	94.1%	Greater proportion of developing failures detected
F1-Score	70.5%	92.9%	Improved balance between precision and recall
False-Positive Rate	18.6%	6.3%	Reduction in unnecessary operational alerts
Mean Detection Lead Time	2.8 min	11.6 min	Earlier identification before critical failure
Root-Cause Top-3 Accuracy	61.2%	89.7%	Improved localization of likely faulty components
Average Analysis Latency	1.4 s	2.1 s	Additional AI processing overhead remains manageable
Incident Correlation Accuracy	66.8%	90.5%	Better grouping of related telemetry anomalies

This is meant as a narrative comparison but the evaluation process itself reveals some of differences between observability with AI assistance and conventional monitoring. The classic threshold strategy can come back very well, when you have a known b value which is obviously larger than an operational barrier that was established. However, single disturbances well below alarm levels may result in failures and performance is expected to be poor under these test conditions. Temporal and multi-variate relationships between covariates, edges of a network estimate can be measured through AI-assisted analysis thus letting new observations in development before threshold behavioural at the level of an individual. Detection lead time: Proactive reliability management An alert that is only generated after a service has already failed helps when investigating but gives little time to prevent it. An AI model that detects a growing fault minutes earlier may provide enough time for operators or automated controllers to scale resources, reroute traffic, restart an unhealthy component, or investigate a problematic deployment. However, accuracy must be balanced against lead time as forecasts very far ahead that result in many false alarms are of limited operational use.

An evaluation on how the different telemetry channels influence model performance is also required. Metrics only paradigm can identify resource saturation point but does not give any information at application level. Failure events will occur in the logs (which may also be noisy and not message-format). The impact might even be manifest in the form of distributed traces showing service dependencies and anomalous request patterns, depending upon sampling policies. An ablation study that examines metrics only, logs only traces alone or in multimodal combination. That way, they identify whether the expensive computation of merging multiple telemetry sources leads to observable improvements. Testing for scalability is equally crucial. As the number of services, replicas (or deployments) and users / transactions increases so do the reams of telemetry almost instantly. In other words the evaluation should scale out event processing throughput, model inference latency, memory and storage footprint with respective queue delay on growing workloads. But even if the failure-detection model is very accurate with a low false-positive rate it to be too slow for real time operations. This way you can say whether analytical performance is constant or not as telemetry volume keeps expanding with distributed processing and streaming inference.

Robustness evaluation includes several variations of the workload patterns as well different failure intensities. Testing should happen during both low loads, high workloads and normal traffic at all time deployment times with concurrent component failure or autoscaling activity. This avoids evaluating in the situations that are easy to observe. In adaptive training, with the occurrence of a possible concept drift by adaptation of system changing characteristics which may affect detection quality; should be followed up with an assessment to see if performance is similar compared to another models such as true target classifier. Instead of improved metrics in machine learning, the metric for successful evaluation should be whether AI assisted observability helps us to make meaningful operational changes. It depends on a lot of variables: notice, noise mitigation, time to diagnosis (Mikulski 2020) and scalable transport flow. A good analysis across several dimensions provides the data needed to know how far proactive intelligence will take production-scale distributed software operations.

VIII. OPERATIONAL DIFFICULTIES, SECURITY, AND SCALABILITY

Besides the accuracy of anomaly-detection models, there are also operational and technical challenges to applying AI-assisted observability in large-scale distributed software systems. Your production systems are not running a few hundred containers, ephemeral service instances, kittens but thousands of them along with distributed databases and application programming interfaces (APIs) across multiple locations or cloud providers; a message queue here; some infrastructure resources there. Telemetry is captured from every element at all times, which means the development of significant data collection, transmission strategy and massive storage Processing & Analytics logistics. Hence a typical observability architecture which is agile and operationally strong has to strike the right balance between scale, security, cost efficiency & predictive efficacy. The scalability issue you encounter first is the scale of telemetry itself. Application logs can quickly run into millions of records at times, metrics are typically scraped according to a fixed cadence and distributed tracing may produce dozens or even thousands spans for each transaction. Thus, collecting every observation at maximum resolution might come at a high computational cost in terms of processing, storage and bandwidth. This mandates intelligent telemetry handling. Aggregation summarises high-frequency measurements; sampling techniques reduce trace volume, while filtering discards low value or repetitive events. However, aggressive reduction may hide weak signals ahead of failure. It is the observability architecture that should guarantee not both of them: neither a loss of information needed to make dependable predictions nor one at an expense ineffectively.

The velocity of data, on the other hand poses another challenge because it has to be quickly scanned for proactive identification. An out-of-band prediction, therefore after the service has already failed does very little to avoid this failure. As a result, telemetry pipelines must support low-latency processing and continuous streaming. By using distributed message brokers, stream-processing platforms (like Apache Kafka), scalable feature pipelines and online model inference new observations could then be streamed as they are generated. While less-impactful components can function with low sample frequencies, the critical services will need higher sampling rates and deeper telemetry resolution for analytical processing. This priority allows you to give operational value for the resources assigned.

First, it is not just usability; telemetry cardinality which in turn impacts on scalability also comes into play here. Such labels as: service identity; container instance customer region request endpoint deployment version transaction identifier can generate very high cardinality time series. Unbound cardinality increases index, storage and query cost. That being said, observability platforms leveraging AI assistance should also have policies around selecting and retaining these labels. While we can eliminate unnecessary dimensions, preserving context that assists in fault diagnosis must remain. Security is another big aspect as observability data has some operationally sensitive information. Application logs can sometimes have its transaction meta, DB Ids, PII (personally identifiable information), auth data and even internal application structures. Infrastructure

telemetry might have provided information on how you deployed topology, while distributed traces could provide what API endpoints are interacting with which services. It is therefore necessary to apply encryption, authentication and access control as well as retention processes appropriate for telemetry during transport or rest. Organisations must destroy unwanted sensitive data ahead of telemetry reaching the analytical pipelines,

AI segments pose new security challenges. Telemetry can also be affected by attackers or abnormal work loads that may negate the anomaly models or hide malicious activity. Adversarial patterns can make detection worse and if they trained on contaminated data, so they might learn the wrong behaviour like always. Thus provenance information needs to be captured and unvalidated telemetry should not be used during the modeling training cycle. Models and baselines should be designed such that the process of training (how was this update made) can also be audited – Security teams must understand a model in detail before production so any major change to it should only happen after Cross-Validation.

Concerns about privacy and legality could poison centralised telemetry gathering even more if dispersed applications operate across organisational or geographical boundaries. Data minimisation and regulated retention act as helpful checks against unnecessary exposure. Possibly a few environments might slice causal operational aspects off data in advance of centralized intelligence service transmission. Nonetheless, in situations where raw telemetry cannot be usefully aggregated freely, federated or decentralised learning systems may also feature as optional but they are not costFree and introduce other layers of coordination complexity model-management issues. A key operational challenge is interpretability. Deep-learning models are capable of offering predictions that can be extremely accurate, but it comes at a considerable expense – the insight into what caused the failure essentially evaporates. When no exploration of diagnostic logic is possible, production engineers are cautious to trust high-impact automated actions. Overall, it gives you the ability to add evidence with predictions that include leading metrics, unusual log patterns and affected trace paths along with dependent services on recent changes done within those systems. You may ask, why explainability: It helps both the operator to trust and also validates the model.

Unavoidable are the threats of false positive and undoubtedly fake negativity. As the false positive rate increases, engineers develop alert fatigue and many ignore strong signals. And false negatives can also slip serious failures under the radar. Thus, detection criteria should also be rooted in false positive and negative costs as well as the criticality of the service itself. In many high-impact financial or mission-critical services, you may need to do conservative procedures and human validation before the automated remediation.

Third, however, operational integration includes another aspect of practical value that AI-assisted observability provides. So forecasts should align with your real production environments today – which includes the actual service ownership, remediation routines, escalation rules and change management methodologies that are there. Instead of only highlighting an anomaly score, alerts should put the context around which services were affected by it along with data and a severity level indicating how serious is this issue accompanied with mitigation actions. You would annotate the engineers input to analyze that data better next time.

Finally, large-scale and safe AI based observability necessitates cohesive management of telemetry volume, analytical latency or Speedo flight, model reliability and security and privacy concerns in addition to explain abilityas well as operational governance. With these problems being solved their proactive failure detection will be able to keep up with the increase in harmful distributed situations that this approach presents. The objective here is: NOT to deploy ever more sophisticated AI models, rather continuously within performance/security/reliability constraints defined by the reality of the distributed software system it will operate under.

IX. CONCLUSION

Modern-day distributed software systems have become larger, more dynamic and architecturally complicated over time leading to an urgent need for operational monitoring and reliability management. These microservices, cloud-native platform with containers for distributed databases on top of data and APIs over componentization together create orchestration technologies & geography distribution infrastructure / HPC field pairs is giving a whole new value to scalability either in terms of deployment pattern or flexibility at all. Unfortunately, these technologies create a highly coupled environment that can fail not only due to a single component but through entire service dependencies. Conventional monitoring systems, which rely on discrete measurements and stationary thresholds, tend to be ineffectual at detecting subtle degradation tendencies prior to significant service failures. This paper is the full-pipeline from billions of operational data points towards reliability intelligence with Observability aided by AI.

This is where the framework proposed here merges several of those key metrics, logs, distributed traces and infrastructure events with deployment data and service-dependency connections to give an in-depth understanding system behaviour. AI and machine-learning systems can continuously analyse this data to determine responsive behavioural baselines, identify abnormal conditions, predict coming states of operation (and failure probabilities) And this is especially useful as you get great orthogonal views of the health from different telemetry signals for the same system. Logs are context-rich in nature, distributed traces provide the end-to-end life cycle of request along with its dependent interconnect services and metrics gives a good amount of information about how resources being utilized.

Moving from monitoring for anomalies to proactively identifying and detecting failures is the same as moving from seeing threats versus trying to see anomalous conditions. In contrast to traditional models that only respond after relevant operational thresholds have been exceeded, predictive ones evaluate their auditory temporalities and early indicators such as increasing memory usage, queuing/backoff/latency/timeouts/retry pattern anomalies and are sensitive both to changing error behaviour in the system where) components interact. Analysing this data collectively means the system is also able to identify conditions that indicate a possible future accident which operatives can react to beforehand thus allowing them more time for damage control.

The study also highlights the need to link intelligent root-cause analysis with anomaly detection. Techniques: Explainable AI, Temporal Reasoning, Multimodal Telemetry Correlation (MTTC), Service-Dependency Graph and Historical Incident narrows diagnostic search space and differentiates likely initiating defects from secondary symptoms. Such functionality is vital in microservice systems, where the same root problem can trigger alerts on all dependent components.

Adaptive learning is another prerequisite you are able to include. Distributed systems are never static—they change constantly in response to software releases, infrastructure upgrades, autoscaling, workload shifts and configuration updates. So AI models need to take idea drift into consideration but leave out absurd behaviour when introducing them in normal-operating baselines. Such predictions need to remain accurate over time, and hence require continual monitoring along with control feedbacks from the operators (to retrain on incoming data gradually) and necessary versioning of both model & validation methods. And it calls for operational automation to be governed understand confidence. Predictive intelligence can help with scaling, traffic rerouting, component restarts and deployment rollbacks (and other preventive scenarios), but high-impact repair requires verifiable evidence re-enforced by human oversight when required. Also, considering security, telemetry privacy, computational scalability as design requirements are just a few others to take care of false positives management and model trustworthiness.

So to summarise, AI powered observability is a way of shifting from resolving issues in a reactive manner to adopting reliability engineering practices proactively and dynamically. Query on the characteristics of integrated heterogeneous telemetry, machine learning-based temporal forecasting and dependency reasoning together with explainability for root-cause analysis and continuous feedback loop enabled using distributed software platforms can potentially identify novel hazards early enough to facilitate operational decisions faster. To advance on this important area of automated remediation challenges against standardised assessment datasets (i.e. Wellcome-MRC-Imperial & JAMA benchmark) future work should investigate the systematic examination through a panoply of multimodal foundation models, causal inference methods and privacy-preserving learning approaches which directly improve human-AI cooperation tackling all or some reducing empirical biases defined in our methodology. These technologies will help design distributed software systems that are more reliable, self-aware and resilient enough to continue providing services even under rapid change of operational conditions.

X. REFERENCES

- [1] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, “Magpie: Online modelling and performance-aware systems,” *Proceedings of the 9th Workshop on Hot Topics in Operating Systems*, 2003.
- [2] B. H. Sigelman et al., “Dapper, a large-scale distributed systems tracing infrastructure,” Google Research, Technical Report, 2010.
- [3] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat, “Pip: Detecting the unexpected in distributed systems,” *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation*, 2006.
- [4] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, “Detecting large-scale system problems by mining console logs,” *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, pp. 117–132, 2009.
- [5] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 1285–1298, 2017.
- [6] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, “Drain: An online log parsing approach with fixed depth tree,” *IEEE International Conference on Web Services*, pp. 33–40, 2017.

- [7] W. Meng et al., “LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs,” *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 4739–4745, 2019.
- [8] X. Zhang et al., “Robust log-based anomaly detection on unstable log data,” *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 807–817, 2019.
- [9] S. Nedelkoski, J. Cardoso, and O. Kao, “Anomaly detection and classification using distributed tracing and deep learning,” *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 241–250, 2019.
- [10] J. Soldani and A. Brogi, “Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey,” *ACM Computing Surveys*, vol. 55, no. 3, 2022.
- [11] T. Wang and G. Qi, “A comprehensive survey on root cause analysis in (micro) services: Methodologies, challenges, and trends,” *arXiv preprint arXiv:2408.00803*, 2024.
- [12] L. M. Barata, S. Sequeira, E. Lopes, P. R. M. Inácio, and M. M. Freire, “Anomaly detection and root-cause identification in microservices: A survey,” *Cluster Computing*, vol. 29, Article 309, 2026.
- [13] OpenTelemetry Authors, “OpenTelemetry Documentation: Observability, traces, metrics, and logs,” Cloud Native Computing Foundation, 2025.
- [14] OpenTelemetry Authors, “OpenTelemetry specification: Logs data model and log correlation,” Cloud Native Computing Foundation, 2025.
- [15] OpenTelemetry Authors, “OpenTelemetry Metrics Specification,” Cloud Native Computing Foundation, 2025.
- [16] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. O’Reilly Media, 2016.
- [17] B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, and S. Thorne, *The Site Reliability Workbook: Practical Ways to Implement SRE*. O’Reilly Media, 2018.
- [18] C. Sridharan, *Distributed Systems Observability*. O’Reilly Media, 2018.
- [19] M. Hausenblas, *Cloud Observability in Action*. Manning Publications, 2023.
- [20] C. Majors, L. Fong-Jones, and G. Miranda, *Observability Engineering: Achieving Production Excellence*. O’Reilly Media, 2022.
- [21] M. T. Ribeiro, S. Singh, and C. Guestrin, “‘Why should I trust you?’: Explaining the predictions of any classifier,” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, 2016.
- [22] S. M. Lundberg and S. I. Lee, “A unified approach to interpreting model predictions,” *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [23] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.
- [24] F. T. Liu, K. M. Ting, and Z. H. Zhou, “Isolation Forest,” *IEEE International Conference on Data Mining*, pp. 413–422, 2008.
- [25] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [26] A. Vaswani et al., “Attention is all you need,” *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [27] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *International Conference on Learning Representations*, 2017.
- [28] P. Veličković et al., “Graph attention networks,” *International Conference on Learning Representations*, 2018.
- [29] D. P. Kingma and M. Welling, “Auto-encoding variational Bayes,” *International Conference on Learning Representations*, 2014.
- [30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [31] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM Computing Surveys*, vol. 46, no. 4, Article 44, 2014.
- [32] A. Bifet and R. Gavaldà, “Learning from time-changing data with adaptive windowing,” *Proceedings of the SIAM International Conference on Data Mining*, pp. 443–448, 2007.
- [33] M. A. Ikram, S. Chakraborty, S. Mitra, S. Saini, S. Bagchi, and M. Kocaoglu, “Root cause analysis of failures in microservices through causal discovery,” *Advances in Neural Information Processing Systems*, 2022.
- [34] A. Bakhtin, J. Nyssölä, Y. Wang, N. Ahmad, K. Ping, M. Esposito, M. Mäntylä, and D. Taibi, “LO2: Microservice API anomaly dataset of logs and metrics,” *arXiv preprint arXiv:2504.12067*, 2025.
- [35] K. Ping, H. B. Mazhar, Y. Wang, Y. Song, and M. V. Mäntylä, “AnoMod: A dataset for anomaly detection and root cause analysis in microservice systems,” *arXiv preprint arXiv:2601.22881*, 2026.
- [36] “Microservice anomaly detection based on tracing data using semi-supervised learning,” *2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, IEEE, 2021, doi: 10.1109/ICAIBD51990.2021.9459100.
- [37] “Efficient and robust trace anomaly detection for large-scale microservice systems,” *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, 2023, doi: 10.1109/ISSRE59848.2023.00012.
- [38] “TraceDAE: Trace-based anomaly detection in microservice systems via dual autoencoder,” *IEEE Transactions on Network and Service Management*, vol. 22, no. 5, pp. 4884–4897, 2025, doi: 10.1109/TNSM.2025.3583213.
- [39] H. Wei et al., “Log-based anomaly detection for distributed systems: State of the art, industry experience, and open issues,” *Journal of Software: Evolution and Process*, 2024.
- [40] L. Pham, “Anomaly detection and root cause analysis for microservice systems,” doctoral research/thesis, *arXiv preprint arXiv:2606.09942*, 2026.