PRISM: Cross-Layer Observability Framework for Mobile Core Networks

Sehan Samarakoon*†, Nitinder Mohan†, Fernando Kuipers†
*VodafoneZiggo, Netherlands
†Delft University of Technology, Delft, Netherlands
sehan.samarakoon@vodafoneziggo.com, N.Mohan@tudelft.nl, F.A.Kuipers@tudelft.nl

***Abstract*—Cloud-native 5G core networks span service, orchestration, and infrastructure layers, each producing telemetry in different formats, at different rates, with different semantics. Yet monitoring tools operate per layer, leaving operators without cross-layer visibility needed to diagnose faults that cascade across architectural boundaries. We present PRISM, the first-of-its-kind cross-layer observability framework designed to plug into existing cloud-native cellular core network deployments, capturing evolving structural and temporal dependencies. PRISM collects and correlates logs, metrics, and configuration state from three layers into a unified, continuously updated representation, using an ontology-driven temporal knowledge graph (TKG) as core data structure. To support fault investigation, PRISM provides an incident-time subgraph extraction algorithm that, given incident time and rollback window, produces a compact diagnostic view of the TKG by highlighting anomalous entities and dependency paths connecting them. We evaluate PRISM on an Open5GS 5G core deployment under four fault scenarios spanning resource stress and configuration changes. Across all scenarios, PRISM produces operator-usable incident-time subgraphs that capture dominant anomalous components and their dependency structure, enabling focused troubleshooting and accelerating issue resolution time in virtualized mobile core networks.**

I. Introduction

Mobile core networks are undergoing a fundamental architectural transformation with 5G. Network functions have transitioned from monolithic, hardware-based appliances to containerized microservices deployed on cloud-native orchestration platforms [1]. This architecture spans three layers: the *service layer*, where 5G functions (e.g., AMF, SMF etc.) realize the 3GPP Service-Based Architecture; the *orchestration layer*, where runtimes and control-plane components manage placement and lifecycle; and the *infrastructure layer*, comprising the host OS, kernel services, and physical or virtual hardware (see Fig. 2). Each layer continuously produces its own telemetry (i.e. logs, metrics) at different rates, in different formats, and with different semantics [2]. With 6G, this complexity will grow with the edge-cloud continuum, native AI, and increasingly distributed deployments [3].

Despite this, monitoring tools in today's mobile networks operate largely in isolation. Operators rely on separate metric backends, log aggregators, and orchestration dashboards for each layer, but lack a unified view of cross-layer dependencies and interactions [4]. A single fault may appear as a log burst in one network function, a resource anomaly on the host, and a stale configuration binding in the orchestration layer. Diagnosing such faults requires operators to manually correlate heterogeneous data across layers and organizational teams, often under time pressure [5]. As networks grow more complex, this correlation-heavy, expertise-driven troubleshooting becomes increasingly impractical in efficiency and cost.

ISBN 978-3-903176-82-9 © 2026 IFIP

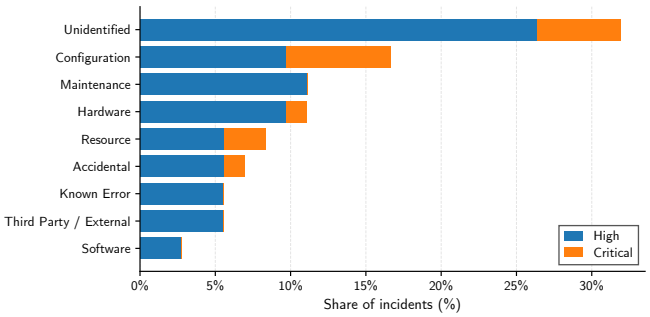


Fig. 1: Root-cause distribution of severe faults in the virtualized mobile core of a major mobile operator.

To quantify this problem, we analyze PRB records from a major European mobile operator covering high- and critical-severity incidents in its virtualized core network. Fig. 1 shows the distribution of identified root causes. The largest category is *Unidentified*: incidents with service or performance impact whose root cause could not be determined after the event, often because symptoms spanned multiple layers and teams, self-healed before detailed inspection, and lacked cross-layer dependency visibility. In such cases, post-incident analysis could confirm the impact, but not reconstruct the cross-layer fault propagation needed to isolate a single root cause. For example, one identified critical incident reported multiple voice and data alarms at the same time for a short time, while the eventual analysis linked the trigger to a power event that caused high CPU on applications, illustrating how symptoms can span infrastructure, platform, and service behavior. The second largest category is configuration errors, underscoring the need to treat configuration state and evolution as first-class diagnostic inputs, since configuration changes can trigger failures that propagate across layers over time. These findings indicate that effective diagnosis in virtualized mobile core networks requires a temporal model of cross-layer dependencies spanning multiple layers of the stack.

In this paper, we present PRISM, the first-of-its-kind cross-layer observability framework designed for cloud-native cellu-

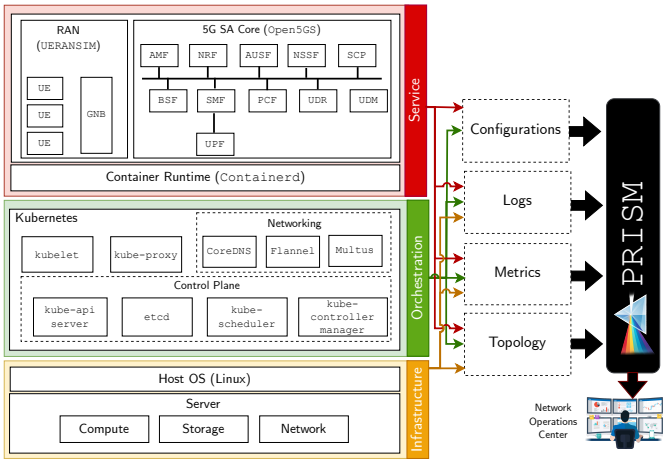


Fig. 2: Multi-modal telemetry collected by PRISM across three layers, each producing logs, metrics, and configuration state at different rates and in different formats.

lar core deployments. As shown in Fig. 2, PRISM continuously collects logs, metrics, and configuration state from the service, orchestration, and infrastructure layers of a virtualized mobile core deployment and integrates them into an ontology-driven temporal knowledge graph. A knowledge graph is a natural fit for this setting as it encodes typed, cross-layer dependencies between network components, preserves temporal evolution through time-stamped facts, and supports structured queries over both topology and operational state. Specifically, our contributions are as follows.

1) We present PRISM, a cross-layer observability framework for cloud-native 5G cores, together with an ontology that maps heterogeneous operational data from the service, orchestration, and infrastructure layers into typed entities and relations in a unified temporal knowledge graph, enabling querying and incident reconstruction (§III).
2) Building on this, we propose an incident-time subgraph extraction algorithm that, given a reported fault time, produces a compact diagnostic view of anomalous entities and their cross-layer dependency paths to support operator-assisted fault diagnosis (§III).
3) We validate PRISM on an Open5GS 5G core under four common fault scenarios spanning resource stress and configuration changes. PRISM identifies the primary fault source in all cases and produces compact subgraphs that preserve the cross-layer context needed for diagnosis (§V).

Note that PRISM focuses on observability rather than root cause analysis. However, by exposing cross-layer dependencies and temporal context at a finer granularity than existing per-layer stacks, it enables more targeted diagnosis, whether operator-driven or automated (§VI).

II. Background and Related Work

A. Observability Challenges in Mobile Core Networks

Observability in cloud-native 5G core networks can rely on different data modalities such as logs, metrics & KPIs, and network packets & traces. Among these, logs and metrics (including KPIs/counters) provide the initial observable signals for cross-layer monitoring and operational analysis in telecom networks. Logs record discrete, timestamped events; metrics provide continuous quantitative signals on different performance indicators within the network. Beyond these runtime observability signals, configuration changes are also a major source of failures and performance degradation in 5G core networks [6]. Configuration state therefore provides an essential additional view of the declared and effective runtime parameters of network components. These modalities differ in structure, semantics, and temporal granularity, and a single fault may simultaneously manifest as a log burst in one component, a metric deviation in another, as a result of a stale configuration entry in a separate component [7].

In a cloud-native 5G core, these signals originate from multiple architectural layers. The service layer comprises 5G network functions (e.g., AMF, SMF, UPF) implementing the 3GPP Service-Based Architecture [1]; the orchestration layer includes the runtime, scheduler, and control-plane components managing workload placement and lifecycle [8]; and the infrastructure layer comprises the host OS, kernel services, and physical or virtual hardware. Faults frequently propagate across these layers [3], [5]. For instance, a resource constraint at the infrastructure level may surface as scheduling delays in the orchestration layer and eventually as signaling failures at the service layer. Effective diagnosis therefore requires a shared representation that integrates heterogeneous modalities, captures cross-layer dependencies and time, and supports near-real-time updates – motivating the need for PRISM.

B. Fault Diagnosis and Observability

Cellular Networks. While multi-modal observability and fault diagnosis are well established in distributed systems [9], fewer studies address these challenges specifically in 5G core networks. Recent ML-based approaches [10]–[12] target 5G core fault localization, but remain layer-specific or rely on a single modality. Similarly, prior work on cloud-native B5G observability [13]–[15] emphasizes monitoring stacks and eBPF-based collection rather than cross-layer dependency modeling. Among dependency-aware approaches, Heeb et al. [3] propose topology-based diagnosis in IoT-extended 5G microservice architectures, while Tan et al. [16] present Zoom-inRCL for root-cause localization in NFVI layers. However, both rely on metrics and topology alone and localize faults at the network-function level. To the best of our knowledge, no prior work builds a near-real-time 5G core representation that integrates logs, metrics, and configuration state across layers.

Microservice Systems. The 5G/6G core is increasingly adopting a cloud-native, microservice-based architecture, making methods from the broader microservice domain relevant for observability and fault diagnosis in mobile networks. Among them, graph-based RCA approaches go beyond multi-modal data collection by correlating heterogeneous operational signals, such as logs, metrics, and traces, through structural dependency models [17]–[20]. Nevertheless, they typically remain centered on the service layer, using application- or

	Observability Layers			Config. States
	Service	Orch.	Infra.	
Ranjitha K. et al. [25]	✓		✓	
Rupashree R. et al. [26]	✓		✓	
Richard L. et al. [27]	✓		✓	
Wang J. et al. [28]	✓		✓	
PRISM	✓	✓	✓	✓

TABLE I: Comparison of prior work across observability layers and configuration-state support.

service-level telemetry to identify anomalies. To represent richer dependencies across heterogeneous operational data, knowledge graphs have emerged as a practical model with explicit semantics, although often still within individual layers. While these approaches provide useful foundations for our work, their applicability to holistic observability and fault diagnosis in mobile core networks is limited by static graph assumptions, insufficient support for cross-layer correlation, dependence on historical fault-pattern matching, and the lack of explicit treatment of configuration-state changes [21]–[24].

Among works on cross-layer observability and fault diagnosis, [25] uses eBPF-based monitoring to capture network- and host-level information and maintain application-to-network context mappings in the kernel. [26] studies service-layer HTTP timeouts caused by database failures by propagating unique request identifiers across layers. DeepStitch [27] enriches service-level distributed tracing with kernel-level information by learning cross-service system-call patterns using LSTM and stitching them into a global view of the application. [28] correlates service-level traces with infrastructure resource metrics for fault diagnosis using a transformer-based model that captures container interactions under topology changes.

Table I summarizes the limited prior work on cross-layer correlation across network layers. No prior work spans all three layers or incorporates configuration-state changes into diagnosis. Majority of service-layer graph-based approaches lack orchestration and infrastructure layer coverage, and multi-layer data observability platforms do not explicitly model them as continuously evolving dependencies. A further gap not captured by the table is that the data sources across these layers do not conform to a shared schema or ontology, making their integration into a near real-time unified knowledge graph a non-trivial construction challenge. PRISM is the first framework to cover three observability layers and in addition also incorporate configuration states in a continously evolving representation. It designs a cross-layer ontology that resolves heterogeneous, schema-less data from the service, orchestration, and infrastructure layers into typed entities and relations, with configuration state versioned alongside telemetry. The resulting temporal knowledge graph makes cross-layer dependencies between network components explicit and queryable, enabling operator-driven fault diagnosis without requiring historical patterns or causal models.

III. System Overview

This section describes the architecture of PRISM, whose pipeline is organized into three distinct layers (see Fig. 3). The *Data Layer* (§III-A) ingests and normalizes raw telemetry from the mobile core, persists it, and transforms the resulting records into ontology-driven graph updates. The *Knowledge Layer* (§III-B) maintains the temporal knowledge graph that serves as the central, queryable representation of network state. The *Graph Analytic Layer* (§III-C) computes per-entity anomaly features and extracts compact incident subgraphs for operator-driven diagnosis. The end-to-end processing flow follows the circled numbers in the figure. The *Data Collector* and *Data Processor* ingest and normalize raw logs, metrics, topology, and configuration state from the core network ❶. The *Data Pipeline* ❷ transports these records into the *Database* ❸ for persistent storage. The *Extract-Transform-Load (ETL) Engine* retrieves stored records ❹, maps them onto the domain ontology, and commits incremental updates to the *Knowledge Graph Store* ❺. After each update, the *Feature Builder* computes per-entity anomaly features from the latest graph state ❻. When an incident occurs ❼, the operator specifies the incident time and a rollback period ❽, and the *Incident-Time Subgraph Generator* performs anomaly scoring across the corresponding time range before extracting a compact, context-preserving subgraph for diagnosis ❾. The following subsections detail each layer and its components.

A. Data Layer

The Data Layer collects multi-modal telemetry from the mobile core infrastructure, normalizes and persists it, and transforms the resulting records into ontology-driven graph updates. It spans five components that collectively convert heterogeneous operational signals from the service, orchestration, and infrastructure layers into a form suitable for further layers.

Data Collector. The *Data Collector* acquires logs, metrics, topology, and configuration states from all three layers of the core network. To achieve cross-layer observability, PRISM ingests three complementary log streams from the service, orchestration, and infrastructure layers. Metrics complement this discrete event view with continuous, quantitative signals on resource consumption and I/O behavior at the pod, container, and node level. The collector periodically queries the monitoring backend to retrieve sliding-window aggregates, each tagged with workload identifiers for cross-layer correlation.

Beyond these common observability signals, PRISM also treats configuration and topology states as inputs. The collector periodically captures the current workload inventory, runtime placement, and configuration bindings from the cluster control plane. Snapshots are emitted only when the observed state changes, producing a compact change-event stream that records scaling, restarts, rescheduling, roll-outs, and configuration updates. Each snapshot carries a collection-time timestamp for cross-modality alignment while retaining object-level timestamps inside the payload. Across all modalities, the collector ensures continuity by resuming deterministically after failures to prevent gaps or duplicate processing. Each modality is emitted as a timestamped stream and delivered to the *Data Processor* ❶ for cross-layer correlation.

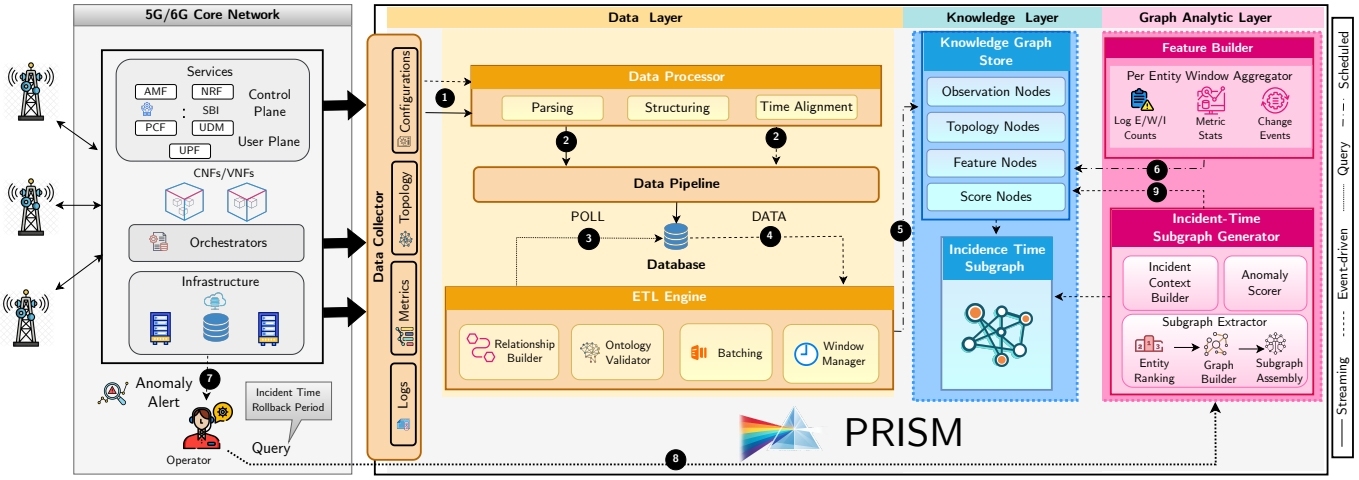


Fig. 3: Overview of the PRISM architecture, organized into Data, Knowledge, and Graph Analytic layers. Circled numbers indicate the end-to-end processing flow from telemetry collection to incident subgraph extraction.

Data Processor. The *Data Processor* transforms heterogeneous raw inputs into a consistent, time-aligned representation suitable for knowledge graph construction. Since logs, metrics, and configuration data differ substantially in structure and semantics, the component applies modality-specific processing while preserving temporal fidelity. For logs, the primary challenge is semantic inconsistency across layers. Service, orchestration, and infrastructure logs use different field names, severity taxonomies, and timestamp conventions; in some cases, even logs from the same component vary by format. For example, an AMF log may appear as a structured JSON record with fields such as `time`, `level`, and `msg`, a Kubernetes component log may use compact prefixes such as `E0912 10:41:22`, and a host log may follow syslog formatting, requiring field and timestamp normalization before cross-layer correlation. PRISM normalizes each record to a canonical set of fields and a unified time representation to preserve temporal ordering (see §IV for implementation).

Metric exporters expose heterogeneous naming and labeling conventions across hosts, pods, and containers. The processor enforces canonical component identification, layer tagging, and timestamp normalization before graph integration. Topology and configuration feeds are often noisy and partially unstructured. Non-informative payloads are filtered, and the remaining data is transformed into a compact representation that enables reliable change detection and reduces false correlations. Finally, the processor structures all modalities into a unified schema for robust identity resolution and publishes them as separate streams to the *Data Pipeline* ❷.

Data Pipeline and Database. The *Data Pipeline* transports processed streams to the *Database* in real-time. It decouples producers and consumers via persistent queuing and flow control, so transient bursts are absorbed without propagating backpressure to the upstream components. Combined time- and size-based eviction policies bound storage while keeping end-to-end latency predictable. The *Database* maintains separate indices per data modality, enabling low-latency retrieval of raw logs, metric aggregates, and configuration snapshots for graph construction. It complements the knowledge graph (§III-B) by preserving full-fidelity records, particularly for configuration data where operators often require detailed fields beyond the graph abstractions. To bound storage, the database enforces time-based retention policies.

ETL Engine. The *ETL Engine* converts normalized records into incremental updates to the temporal knowledge graph. It continuously polls the indexed store ❸ for processed records, reads them ❹, aligns them to a common time base, and applies ontology-driven mappings. It resolves each record to canonical component identities across layers and instantiates the corresponding ontology classes and relations. The record payload is then attached as typed properties on the resulting graph elements. For example, a normalized AMF error log is resolved to the corresponding network-function instance, instantiated as a log-observation node, linked to the AMF container node, and stored with typed properties such as severity, message, and timestamp and other meta data. All inserted data carry explicit temporal attributes, enabling the graph to represent both discrete events and evolving state. Updates are applied in periodic batches to provide stable graph snapshots and limit write amplification under high telemetry rates. Within each batch, observations are grouped by fixed time windows and committed together with explicit window boundaries. Configuration state is treated as a temporally versioned entity. When a change is detected, the engine closes the validity interval of the previous state and opens a new version, maintaining both the current configuration and its history for incident reconstruction. This batching and versioning strategy enables near-real-time graph maintenance while preserving the temporal structure required for downstream diagnosis ❺.

B. Knowledge Layer

The Knowledge Layer centers on a single component, the *Knowledge Graph Store*, which maintains the temporal knowledge graph that serves as the central representation of network

state in PRISM. It stores both structural topology and operational observations, and provides the substrate for downstream feature computation and subgraph extraction. The *Knowledge Graph Store* maintains a near-real-time, dependency-aware view of network state based on the PRISM ontology, updated at each batch window. It stores both the current graph state and its temporal evolution, so operators can query not only *what* is connected but also *when* a relation, observation, or configuration state was valid. Formally, we model PRISM as a TKG $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E} \times \mathcal{T}$, where each fact $(s, r, o, t) \in \mathcal{G}$ denotes that relation r holds from entity s to entity o at time t; the snapshot at time t is the directed labeled graph $G_t = (V_t, E_t)$ induced by all facts with timestamp t [29].

In PRISM, $\mathcal{E}$ includes topology entities (e.g., network function instances, pods, nodes) as well as observation-derived entities (e.g., log, metric, configuration-state). This organization separates relatively stable topology structure from fast-changing operational observations while preserving relationships. $\mathcal{R}$ captures structural dependencies (e.g., RUNS_ON, HAS_CONTAINER, HAS_POD) and observation attachments (e.g., HAS_METRIC, EMIT_LOG) between the entities.

We realize each time-scoped snapshot in the *Knowledge Graph Store* as a property graph for efficient querying and analytics. Specifically, each snapshot G_t is represented as a directed labeled graph whose nodes and edges carry typed properties (e.g., timestamp, identifiers, batch-window boundaries). This property-graph encoding preserves the formal TKG semantics while supporting efficient indexed retrieval and analytics over both structure and attributes. The store also maintains feature and score nodes attached to the topology nodes to support downstream anomaly scoring and incident-time subgraph extraction. Each batch yields a coherent time-scoped snapshot, enabling queries over dependencies and their evolution across the rollback horizon considered during incident analysis. To bound graph growth in continuous operation, PRISM applies time-based retention policies in the *Knowledge Graph Store*. Observation nodes and their associated edges are retained for a configurable period and expired thereafter. The retention period is selected to preserve sufficient history for reactive fault management and post-incident analysis.

C. Graph Analytic Layer

The Graph Analytic Layer operates on the temporal knowledge graph to support operator-driven incident analysis. It comprises two components that together produce a compact, interpretable subgraph from the broader knowledge graph upon an operator query.

Feature Builder. After each batch update to the knowledge graph, the *Feature Builder* computes per-entity features used for anomaly scoring. Features are derived from logs, metrics, and configuration states, and the framework is extensible to the broader KPIs collected in telecom core networks. They are generated for each batch window, linked to topology nodes ❻, and expired under a time-based retention policy.

Incident-Time Subgraph Generator. Upon an anomaly alert ❼, the operator issues a query specifying the incident time and rollback period ❽, and the *Incident-Time Subgraph Generator* constructs the corresponding diagnostic subgraph. First, anomaly scoring is performed across the specified rollback period. For each feature or KPI, the value at incident time is compared with a baseline computed over a stable reference window preceding the incident. A temporal gap between the baseline window and the incident reduces contamination by early fault propagation effects, so the resulting deviations better reflect incident-related change. Anomaly detection is feature-dependent, and PRISM can incorporate outputs of existing anomaly detection techniques in the network itself.

The resulting anomaly signals are attributed through features into a per-window score for each topology node, using weights that reflect the relative importance of different features or KPIs that can be assigned by operator. After scoring, score nodes are attached to topology nodes in the knowledge graph for each batch window in the rollback period. Using these scores, PRISM constructs the incident-time subgraph ❾. Topology nodes are ranked by their maximum score over the rollback period, and this ranking is used by Algorithm 1 to extract the subgraph. Algorithm 1 takes as input the ranked topology nodes $\mathcal{T} = \{(v_i, s_i)\}_{i=1}^{K}$, where s_i is the anomaly score of node v_i, and operates on the topology-only graph G_T. It initializes the subgraph with the two highest-ranked nodes and the shortest path between them, yielding the initial selected terminal set $\mathcal{T}'$ and node set S. For each subsequent candidate v_k, the algorithm connects v_k to the current subgraph S through a shortest path in G_T, so that intermediate dependency nodes are retained as diagnostic context. After adding v_k, the algorithm measures the marginal expansion $\Delta N = |S| - |S_{\text{before}}|$ and defines an efficiency score $E = s(v_k)/c$, where $c = \Delta N / \max(1, |S_{\text{before}}|)$ captures the relative growth cost. Expansion stops when either the subgraph exceeds the size bound $N_{\max}$, the added path causes an unusually large increase in node count, or the efficiency ratio drops sharply relative to previous additions. The output is therefore a compact subgraph that preserves highly ranked anomalous nodes together with the topology paths needed for operator interpretation.

Takeaway — The three-layer architecture enables PRISM to unify heterogeneous telemetry from the service, orchestration, and infrastructure layers in a temporal knowledge graph and extract focused diagnostic views on demand. Two design properties distinguish PRISM from prior observability approaches: (i) ontology-driven graph construction preserves explicit cross-layer dependencies rather than relying on post-hoc correlation, and (ii) time-bounded subgraph extraction surfaces anomalous entities with the structural context needed for fault-propagation reasoning, without requiring historical pattern databases or learned causal models.

IV. Implementation

We realize the architecture described in §III using standard cloud-native observability and data infrastructure components.

Data Collection & Processing. Service-layer logs are collected by tailing container runtime log files for

Algorithm 1: Incident-time subgraph extraction

Input: ranked terminals $\mathcal{T} = \{(v_i, s_i)\}_{i=1..K}$, hop bound H, max size $N_{\max}$, stop params $\alpha, m_{\min}$
Output: subgraph nodes S, selected terminals $\mathcal{T}'$

Build topology-only graph G_T;
if $K < 2$ **then**
  return $(\emptyset, [\,])$
$S \leftarrow \{v_1, v_2\}$, $\mathcal{T}' \leftarrow [v_1, v_2]$;
$S \leftarrow S \cup \text{nodes}(\text{SHORTESTPATH}_{G_T}(v_2, \{v_1\}, H))$;
$\Delta, Q \leftarrow [\,]$, $E_{\text{prev}} \leftarrow \perp$;
for $k \leftarrow 3$ **to** K **do**
  $v \leftarrow v_k$; **if** $v \in S$ **then**
    continue
  $N_{\text{before}} \leftarrow |S|$;
  $P \leftarrow \text{SHORTESTPATH}_{G_T}(v, S, H)$;
  $S \leftarrow S \cup \{v\} \cup \text{nodes}(P)$; append v to $\mathcal{T}'$;
  $\Delta N \leftarrow |S| - N_{\text{before}}$; append ΔN to Δ;
  $c \leftarrow \Delta N / \max(1, N_{\text{before}})$; $E \leftarrow s(v)/c$;
  if $E_{\text{prev}} \neq \perp$ **then**
    $r \leftarrow E / E_{\text{prev}}$; append r to Q;
    if $\text{LOWOUTLIER}(Q, r, \alpha, m_{\min})$ **then**
      remove last terminal; **break**
  $E_{\text{prev}} \leftarrow E$;
  if $|S| > N_{\max}$ ***or*** $\text{HIGHOUTLIER}(\Delta, \Delta N, \alpha, m_{\min})$ **then**
    remove last terminal; **break**
return $(S, \mathcal{T}')$;

the 5G network functions in the target namespaces. Orchestration-layer logs are collected from Kubernetes (`k8s`) control-plane components (`api-server`, `scheduler`, `controller-manager`, and `etcd`) and node-level components (`kubelet` and `kube-proxy`) via container logs and the node journal. Infrastructure logs are collected from the host system log stream with selective filtering. All log streams are normalized to a canonical field set and unified time representation to enable cross-layer correlation. Resource metrics at node, pod, and container granularity are exported by `Node Exporter` and `cAdvisor`, scraped by `Prometheus` using curated `PromQL` queries, and aggregated over 30 s sliding windows for CPU, memory, network, and disk I/O. Each metric pull is serialized as a `JSON` snapshot every 5 s with stable workload and infrastructure identifiers. Topology and configuration state are collected from the `k8s` control plane every 1 min using a custom script, and emitted only when a content-hash change is detected.

Data Pipeline & Storage. Processed records are normalized into a common `JSON` schema, routed by `Fluentd`, streamed through `Kafka`, and indexed in `Elasticsearch` with separate indices per data modality. `Kafka` provides persistent, decoupled transport so that transient telemetry bursts are absorbed without propagating back-pressure to upstream collectors. `Elasticsearch` enables low-latency, per-modality retrieval for downstream graph construction.

Knowledge Graph Store. The temporal knowledge graph is maintained in `Neo4j`, which natively supports property graphs and the indexed traversal queries required by the ETL engine and the subgraph extraction algorithm. The ETL engine commits graph updates in fixed 30 s batch windows. This window size was selected based on the operating-point analysis presented in §V-B, which evaluates the trade-off between processing overhead and observability latency.

Anomaly Scoring. We implement the anomaly scorer using a modified Z-score technique based on median and MAD statistics [30]. The baseline for each feature is computed over a 15-minute reference window ending 10 minutes before the incident time, so that early fault propagation does not contaminate the reference statistics. Features include log-derived event counts (error, warn, and info levels, and their ratios), resource metrics, and configuration change events. Configuration changes and version updates contribute as dedicated score terms, capturing abrupt state transitions alongside continuous metric and log-count deviations.

V. EVALUATION

We evaluate `PRISM` on a cloud-native 5G core deployment under four fault scenarios. The evaluation addresses three questions: (i) how accurately does the anomaly scorer identify and rank affected components (§V-C), (ii) how effectively does the subgraph extraction algorithm recover cross-layer diagnostic context while remaining compact (§V-D), and (iii) how does the batch window size affect the trade-off between processing overhead and observability granularity (§V-B).

Testbed. We deploy `PRISM` on a single-node `Kubernetes` cluster (v1.28.15) with `containerd` (v1.7.27) and `Flannel`+`Multus` as the CNI stack. The cluster hosts an `Open5GS` 5G core with containerized AMF, SMF, UPF, AUSF, PCF, BSF, NRF, and NSSF on Ubuntu 24.04.2 LTS, running on an Intel Xeon Silver 4509Y (8 cores / 16 threads) with 30 GiB RAM. `UERANSIM` emulates UE behavior via continuous registration attempts toward the AMF. Each fault scenario is repeated for at least 10 independent runs.

A. Fault Scenarios

The Access and Mobility Management Function (AMF) serves as the central control-plane entry point of the 5G core, handling access, mobility, and signaling from the RAN. Through Service-Based Interfaces (SBIs), the AMF interacts with multiple core functions for authentication, session management, and policy control, making it sensitive to faults originating in neighboring components. This role makes the AMF an appropriate anchor for cross-layer validation. We inject two resource-related and two configuration-related faults:

- **AMF-CPU-Stress** (`amfcpu`)**.** `Chaos Mesh` (CPU mode) stresses the AMF container for 3 min, inducing control-plane processing delays and scheduling pressure.
- **AUSF-Memory-Stress** (`ausfmem`)**.** `Chaos Mesh` (memory mode) stresses the AUSF container for 3 min, creating

	Service						Orchestration							Infra	
Fault	AMF-C	AUSF-C	NRF-C	SCP-C	SMF1-C	SMF2-C	AMF-P	AUSF-P	PCF-P	NRF-P	SCP-P	SMF1-P	SMF2-P	Kubelet	Node	Host
amfcpu	●						●								●	●
ausfmem	●	●	●				○	●		○				●	○	●
amfconf	●			●	●	●	●			○	○	○	○	●	○	●
pcfconf	●			●	●	●	○		●	○	○	○	○	●	○	○

TABLE II: Ground-truth nodes per fault scenario. ● = anomalous, ○ = context. C = Container, P = Pod.

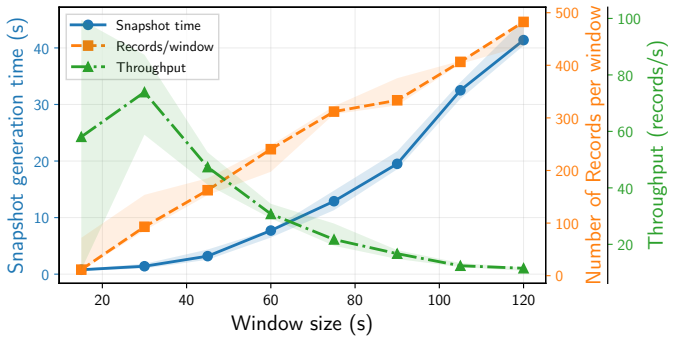


Fig. 4: ETL performance across batch window sizes (median with IQR): snapshot generation time, records per window, and throughput.

memory pressure that affects authentication responsiveness and downstream AMF interactions.

- **AMF-Configuration-Change** (amfconf)**.** Configured SCP port in the AMF is modified, disrupting SBI connectivity during service discovery and request routing.
- **PCF-Configuration-Change** (pcfconf)**.** Advertised SBI port in PCF is changed, creating endpoint-mismatch that propagate to control-plane signaling failures at the AMF.

Although the injections target specific components, their effects propagate across layers through dependency chains. Table II reports the ground-truth anomalous and the additional context nodes required for a complete diagnosis of each fault.

B. Operating-Point Analysis

The batch window size determines the temporal granularity of graph updates and affects both processing overhead and observability latency. Fig. 4 reports three metrics across window sizes ranging from 15 s to 120 s: snapshot generation time, the number of records processed per window, and the resulting throughput in records per second.

Records per window grow linearly with window size, reflecting the steady ingest rate of the data pipeline. Snapshot generation time grows super-linearly: 15–40 s windows complete in seconds, whereas 100–120 s windows require tens of seconds per batch. Throughput peaks at intermediate window sizes and declines for larger windows as the per-batch processing cost outpaces the record accumulation rate.

For windows below 40 s, each snapshot completes well within the window duration, ensuring that batch processing keeps pace with telemetry ingest. Beyond 60 s, the growing

Fault	P@K				MAP@K			
	1	3	5	10	1	3	5	10
amfcpu	1.00±.00	.67±.00	.75±.19	1.00±.00	1.00±.00	.67±.00	.65±.14	.80±.12
ausfmem	1.00±.00	1.00±.00	.80±.15	.92±.17	1.00±.00	1.00±.00	.80±.18	.83±.19
amfconf	1.00±.00	1.00±.00	.80±.15	.69±.12	1.00±.00	1.00±.00	.80±.15	.62±.09
pcfconf	1.00±.00	1.00±.25	.80±.15	.71±.14	1.00±.00	1.00±.25	.78±.28	.68±.23

TABLE III: Median Precision@K and MAP@K by fault type over 10 runs. Entries report median ± IQR.

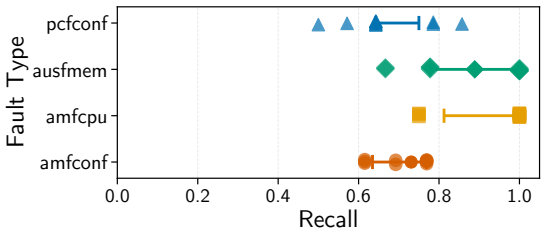


Fig. 5: Subgraph recall by fault type. Points denote runs, boxes extents marks IQR, and vertical lines the median.

snapshot generation time erodes processing headroom, and throughput drops substantially. We therefore use a 30 s batch window in all subsequent experiments, providing sub-minute observability with stable throughput.

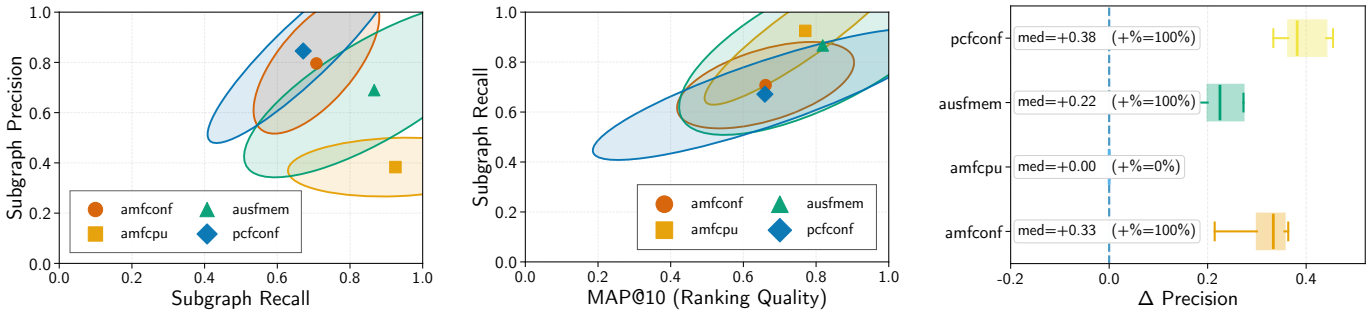
Takeaway — A 30 s batch window achieves the best trade-off between temporal granularity and processing stability, maintaining real-time graph updates without processing backlogs.

C. Anomaly Ranking Quality

We evaluate whether the anomaly scorer correctly identifies and ranks the components affected by each fault. It compares per-entity feature values at incident time against a baseline from a 15-minute reference window ending 10 minutes before the incident (§III). Features include log-derived event counts (error, warn, info, and their ratios), resource metrics, and configuration-change events, with configuration changes contributing dedicated score terms for abrupt state transitions. We assess ranking quality using Precision@K (PR@K) and Mean Average Precision@K (MAP@K) [22]. PR@K measures the fraction of true anomalous components in the top-K predictions, normalized by $\min(K, |V|)$, where V is the ground-truth anomalous set. MAP@K captures how early true positives appear in the ranking, with higher values indicating stronger concentration near the top.

Table III reports median PR@K and MAP@K across 10 runs for each fault type. All four faults achieve perfect PR@1 (median 1.00), meaning the highest-ranked component is always truly anomalous. At K=3, amfcpu drops to 0.67, while ausfmem and both configuration faults remain at 1.00. We attribute this to ground-truth size: amfcpu affects only four components (Table II), so a single false positive in the top 3 incurs a larger penalty.

At K=10, amfcpu achieves the highest PR@10 (1.00) because its small ground-truth set is fully recovered within the top 10. ausfmem follows at 0.92, while amfconf and pcfconf score 0.69 and 0.71, respectively. The corresponding MAP@10 values reflect a consistent pattern: 0.80 for



(a) Precision vs. recall. (b) Subgraph recall vs. MAP@10. (c) Δ precision with ground truth.

Fig. 6: Subgraph quality across fault scenarios. In (a) and (b), each marker denotes the centroid of runs for a fault type, and the surrounding ellipse its 95% confidence region. Tighter ellipses indicate greater consistency, while orientation reflects covariance between the axes (c) reports the precision change when the ground truth is expanded to include context nodes.

`amfcpu`, 0.83 for `ausfmem`, and 0.62 (`amfconf`) to 0.68 (`pcfconf`). The lower MAP@10 for configuration faults reflects the larger number of affected components spread across multiple layers, which distributes true positives deeper into the ranking.

Takeaway — The anomaly scorer consistently places the most critical component at rank 1 across all fault types. Ranking quality is highest for resource faults (MAP@10 $\geq$ 0.80) and moderately lower for configuration faults (MAP@10 $\approx$ 0.65), where anomaly evidence is distributed across a larger set of components.

D. Incident Subgraph Quality

We evaluate whether Algorithm 1 produces compact, operator-usable diagnostic views. Beyond a ranked list, it recovers intermediate dependency nodes needed to interpret cross-layer fault-propagation paths. We assess three properties: coverage (recall), compactness (precision), and context quality (relevance of nodes beyond the strict anomaly set). Precision is computed as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$ and recall as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$, where ground-truth nodes are those listed in Table II.

Coverage. Figure 5 shows subgraph recall for each fault type across 10 runs. Resource-related faults achieve the highest recall: `amfcpu` and `ausfmem` recover the largest fraction of ground-truth nodes, with tightly clustered runs indicating consistent behavior. Configuration faults exhibit lower and more variable recall. This difference is expected: configuration faults affect a larger set of components (8 anomalous nodes for `amfconf` versus 4 for `amfcpu`, per Table II), so subgraph recall depends more heavily on whether the upstream scoring stage ranks all affected entities near the top.

Figure 6b confirms this dependency. In this figure, each marker represents the centroid of all 10 runs for a given fault type, plotted in MAP@10 (x-axis) versus subgraph recall (y-axis) space; the surrounding ellipse captures the 95% confidence region, so a tighter ellipse indicates more consistent behavior across runs. The figure reveals a clear positive association between MAP@10 and subgraph recall: fault types whose anomalous entities are ranked highly by the scorer (rightward centroids) consistently achieve higher recall (upward centroids). In other words, the observed variation in recall is primarily driven by the quality of the anomaly ranking rather than by the subgraph extraction algorithm itself.

Compactness. Figure 6a plots subgraph precision against recall, using the same centroid-and-ellipse representation as Fig. 6b: each marker is the mean across 10 runs for one fault type, and the ellipse is the 95% confidence region. Points in the upper-right corner indicate subgraphs that are both high-coverage and highly selective. Configuration faults (`amfconf`, `pcfconf`) achieve the tightest precision-recall trade-off, with centroids in the upper-right quadrant and compact ellipses. `ausfmem` shows moderate precision with broader variance. `amfcpu` exhibits the lowest precision despite high recall. We attribute this to the small ground-truth set for `amfcpu`: only 4 entities are designated as anomalous with no context nodes (Table II), so the intermediate dependency nodes that the algorithm includes to connect anomalous entities are counted as false positives. From an operational perspective, this remains desirable: the extracted subgraph captures all relevant anomalous entities while exposing the cross-layer dependency structure needed for diagnosis.

Context quality. To assess whether nodes beyond the strict anomaly set are meaningful, Fig. 6c reports the precision change (Δ precision) when the ground truth is expanded to include context nodes. A positive Δ indicates that nodes counted as false positives under anomaly-only evaluation are actually relevant contextual entities needed to preserve diagnostic structure. `pcfconf` shows the largest median gain (+0.38), followed by `amfconf` (+0.33) and `ausfmem` (+0.22), with 100% of runs improving. `amfcpu` shows zero median change because its ground truth contains no context nodes by construction. These gains confirm that the nodes retained by `PRISM` are predominantly meaningful dependency context rather than irrelevant graph expansion.

Takeaway — The subgraph extraction algorithm produces compact diagnostic views that capture most ground-truth nodes while remaining focused. Nodes beyond the strict anomaly set are largely relevant cross-layer context: expanding the ground truth to include them improves precision by

+0.22 to +0.38 across all applicable fault types.

VI. Discussion and Conclusion

We presented PRISM, a cross-layer observability framework for cloud-native mobile core networks. PRISM continuously integrates logs, metrics, and configuration state from the service, orchestration, and infrastructure layers into an ontology-driven temporal knowledge graph, and extracts compact incident-time subgraphs that surface anomalous entities together with their cross-layer dependencies. Our evaluation on an Open5GS deployment under four fault scenarios demonstrates that PRISM identifies primary fault sources across all scenarios and produces interpretable diagnostic views.

The cross-layer representation maintained by PRISM has implications beyond the fault scenarios evaluated here. By exposing typed dependencies and temporal context across architectural layers, PRISM provides the structured input that existing root cause analysis frameworks require but lack for mobile core networks. Graph-based methods such as DiagFusion [17], causal approaches [22], and retrieval-based techniques [31] each assume access to a dependency-aware, multimodal representation of the system under diagnosis. PRISM produces exactly this representation, and its incident-time subgraphs can serve as direct input to such frameworks, enabling finer-grained, cross-layer root cause localization without requiring changes to the analysis algorithms themselves. Moreover, the ontology-driven design is not inherently tied to 5G core networks. Any orchestrated service deployment with a layered architecture, including edge computing platforms, network function virtualization, and general microservice systems [32], exhibits similar cross-layer dependencies and multimodal telemetry. Adapting PRISM to such settings requires extending the domain ontology and data collectors while preserving the core graph construction and analytics pipeline. The implementation code is publicly available [33].

Acknowledgement

This research was supported by the National Growth Fund through the Dutch 6G flagship project "Future Network Services"

References

[1] 3GPP, "System architecture for the 5G System (5GS)," 3rd Generation Partnership Project, Technical Specification TS 23.501, 2024, version 19.0.0, Release 19.

[2] C. E. Menin *et al.*, "Enabling cloud-native observability for white-box performance analysis of the 5g core," in *NOMS 2025-2025 IEEE*. IEEE, 2025, pp. 1–6.

[3] Z. Heeb *et al.*, "Iomirca: Root cause analysis in iot-extended 5g microservice environments," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, 2023, pp. 106–108.

[4] N. Kratzke, "Cloud-native observability: the many-faceted benefits of structured and unified logging—a multi-case study," *Future Internet*, vol. 14, no. 10, p. 274, 2022.

[5] A. Asghar *et al.*, "Self-healing in emerging cellular networks: Review, challenges, and research directions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 1682–1709, 2018.

[6] A.-I. Manolopoulos *et al.*, "An ai-assisted framework for lifecycle management of beyond 5g services," *IEEE Access*, vol. 12, pp. 179 449–179 463, 2024.

[7] N. Fu *et al.*, "Intelligent root cause localization in microservice systems: A survey and new perspectives," *ACM Computing Surveys*, 2025.

[8] N. Saha *et al.*, "MonArch: Network slice monitoring architecture for cloud native 5G deployments," in *2023 IEEE/IFIP NOMS*. IEEE, 2023, pp. 1–7.

[9] S. Zhang *et al.*, "Failure diagnosis in microservice systems: A comprehensive survey and analysis," *ACM Transactions on Software Engineering and Methodology*, vol. 35, no. 1, pp. 1–55, 2025.

[10] S. Enz *et al.*, "Ml driven root cause analysis (rca) in telco microservices continuum," in *2024 IEEE ICC Workshops*. IEEE, 2024, pp. 371–377.

[11] Y. Tan *et al.*, "Deep learning-based log anomaly detection for 5g core network," in *2023 IEEE/CIC ICCC*. IEEE, 2023, pp. 1–6.

[12] W. Wang *et al.*, "Distributed online anomaly detection for virtualized network slicing environment," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 11, pp. 12 235–12 249, 2022.

[13] G. Z. Bruno *et al.*, "Anomaly detection in cloud-native b5g systems using observability and machine learning cots solutions," *Journal of Internet Services and Applications*, vol. 14, no. 1, pp. 189–199, 2023.

[14] D. Soldani *et al.*, "ebpf: A new approach to cloud-native observability, networking and security for current (5g) and future mobile networks (6g and beyond)," *IEEE Access*, vol. 11, pp. 57 174–57 202, 2023.

[15] A. Khichane *et al.*, "5gc-observer: a non-intrusive observability framework for cloud native 5g system," in *NOMS 2023-2023 IEEE/IFIP*. IEEE, 2023, pp. 1–10.

[16] Y. Tan *et al.*, "Zoom-inrcl: Fine-grained root cause localization for b5g/6g network slicing," *Computer Networks*, vol. 256, p. 110893, 2025.

[17] S. Zhang *et al.*, "Robust failure diagnosis of microservice system through multimodal data," *IEEE Transactions on Services Computing*, vol. 16, no. 6, pp. 3851–3864, 2023.

[18] C. Lee *et al.*, "Eadro: An end-to-end troubleshooting framework for microservices on multi-source data," in *2023 IEEE/ACM 45th ICSE*. IEEE, 2023, pp. 1750–1762.

[19] H. Wang *et al.*, "Groot: An event-graph-based approach for root cause analysis in industrial settings," in *2021 36th IEEE/ACM ASE*. IEEE, 2021, pp. 419–429.

[20] G. Yu *et al.*, "Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data," in *Proceedings of the 31st ACM ESEC/FSE*. ACM, 2023, pp. 553–565.

[21] J. Qiu *et al.*, "A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications," *Applied Sciences*, vol. 10, no. 6, p. 2166, 2020.

[22] D. Wang *et al.*, "Incremental causal graph learning for online root cause analysis," in *Proceedings of the 29th ACM SIGKDD*, 2023, pp. 2269–2278.

[23] L. Wang *et al.*, "MULAN: Multi-modal causal structure learning and root cause analysis for microservice systems," in *Proceedings of the ACM WWW 2024*. ACM, 2024.

[24] Y. Chen *et al.*, "Automatic root cause analysis via large language models for cloud incidents," in *Proceedings of the 19th EuroSys*. ACM, 2024, pp. 674–688.

[25] K. Ranjitha *et al.*, "A case for cross-domain observability to debug performance issues in microservices," in *2022 IEEE 15th CLOUD*. IEEE, 2022, pp. 244–246.

[26] R. Rangaiyengar *et al.*, "Multi-layer observability for fault localization in microservices based systems," in *2023 IEEE SANER*. IEEE, 2023, pp. 733–737.

[27] R. Li *et al.*, "Deepstitch: Deep learning for cross-layer stitching in microservices," in *Proceedings of the 2020 6th International Workshop on Container Technologies and Container Clouds*, 2020, pp. 25–30.

[28] J. Wang *et al.*, "Multilayered fault detection and localization with transformer for microservice systems," *IEEE Transactions on Reliability*, vol. 73, no. 3, pp. 1502–1515, 2024.

[29] Q. Liu *et al.*, "Teqa: Temporal knowledge graph enhanced question answering," *Knowledge-Based Systems*, vol. 325, p. 113916, 2025.

[30] A. S. Yaro *et al.*, "Outlier detection performance of a modified z-score method in time-series rss observation with hybrid scale estimators," *IEEE Access*, vol. 12, pp. 12 785–12 796, 2024.

[31] F. Liu *et al.*, "Microcbr: Case-based reasoning on spatio-temporal fault knowledge graph for microservices troubleshooting," in *International Conference on Case-Based Reasoning*. Springer, 2022, pp. 224–239.

[32] M. Usman *et al.*, "A survey on observability of distributed edge & container-based microservices," *IEEE Access*, vol. 10, pp. 86 904–86 919, 2022.

[33] "PRISM," https://github.com/spear-lab /5G-anomaly-detection, 2026.