Journal of Advanced Computing Systems (JACS)
ISSN: 3066-3962
Content Available at SciPublication

A Unified AIOps Pipeline for Joint Log–KPI Anomaly Detection, Graph-Based Root Cause Localization, and LLM-Generated Runbooks

Hanqi Zhang
Computer Science, University of Michigan at Ann Arbor, MI, USA
hz0102@yahoo.com

DOI: 10.69987/JACS.2024.40305

Keywords

AIOps; log anomaly detection; KPI anomaly detection; multi-modal fusion; root cause analysis; runbook generation; LLM agents; DevOps automation

Abstract

Modern cloud services emit heterogeneous operational signals—structured logs, KPIs, and traces—yet many anomaly detectors and diagnosis tools remain siloed by modality. This paper presents UniAIOps, an end-to-end pipeline that (i) scores anomalies jointly from logs and metrics, (ii) localizes probable root causes on a dependency graph with Top-k ranking, and (iii) produces operator-ready runbooks using an LLM-style agent constrained by safety and executability guardrails. We target three widely used public AIOps data sources: LogHub/LogPAI log corpora, the AIOps 2018 KPI anomaly detection challenge, and the AIOps 2020 multi-modal challenge data release. In environments where those archives cannot be fetched (e.g., broken mirrors, authentication gates, or bandwidth limits), full experimental evaluation becomes difficult to reproduce. To address this, we provide a proxy benchmark generator that follows the public schemas and typical anomaly patterns described for these datasets, and we report end-to-end results with fixed seeds. Across the proxy benchmarks, UniAIOps improves incident-level detection F1 by up to 0.25 over single-modality baselines, reaches 0.74 Top-1 and 1.00 Top-3 root cause hit rates on graph-injected faults, and yields runbooks with 1.00 average actionability under an eight-criterion rubric. We further analyze detection delay, runtime cost, and deployment constraints (data privacy, prompt injection, and permissioned actions) relevant to LLM-assisted AIOps.

1. Introduction

Cloud-native services increasingly depend on complex stacks: microservices, service meshes, container orchestration, shared databases, and third-party APIs. Reliability depends not only on individual components but also on the interactions between them. When incidents occur, operators must quickly answer three questions: (1) Is the system behaving abnormally? (2) Where is the fault most likely located? (3) What sequence of actions is safe and effective to mitigate impact? These questions are difficult because telemetry is heterogeneous and partially observed.

Production platforms continuously collect operational telemetry. Logs record discrete events and error messages. KPIs record time series such as p95 latency, throughput, saturation, and error rate. Traces record request paths and allow operators to see where latency and errors accumulate. Each modality has strengths and blind spots. Metrics are compact but can be noisy or delayed due to aggregation, sampling, and monitoring failures. Logs can be more specific but are high-volume and semi-structured, and are vulnerable to missing context due to sampling or dropped messages. Traces can reveal dependencies, but are often sampled and can be incomplete. In practice, many incidents manifest as weak or ambiguous signals in any single stream and only become clear when signals are fused.

AIOps (Artificial Intelligence for IT Operations) aims to automate detection, diagnosis, and remediation. Public research has progressed from mining console logs for large-scale system problems [5], to robust log parsing and template extraction (e.g., Drain [6]), to learning-based anomaly detection from log sequences (e.g., DeepLog [7]). In parallel, KPI anomaly detection matured with seasonal models and robust residual scoring, including neural methods such as Donut [8]. Root cause localization developed along multiple lines: dependency-graph propagation, causal inference, and multi-dimensional localization that outputs ranked candidates [11], [17].

Shared benchmarks have been crucial to this progress. LogHub consolidates many real-world log datasets (HDFS, BGL, OpenStack, etc.) to support standardized evaluation [1], and LogHub-2.0 extended and curated these resources [2]. The AIOps 2018 competition provided labeled KPI anomalies for univariate series [3]. The AIOps 2020 competition emphasized multi-modal data and diagnosis, offering fault lists and multiple data sources (e.g., metrics, traces, logs) to encourage end-to-end AIOps pipelines [4], [17].

Over the last two years, LLMs have further shifted expectations. Operators now expect systems that can summarize incidents, retrieve relevant context, and draft procedural guidance quickly. Tool-using agent frameworks (e.g., ReAct [14]) make it feasible to connect an LLM to monitoring backends, ticketing systems, and knowledge bases, while retrieval-augmented generation (RAG) [15] improves factual grounding. However, production adoption is constrained by latency budgets, privacy requirements, and security risks. Logs can contain secrets and untrusted text; naively prompting an LLM with raw logs risks leaking sensitive data and enables prompt injection. Operational assistants also must not take actions outside of strict permissions: even "helpful" automation can cause outages if executed incorrectly.

This paper addresses a practical research question: can we build a coherent AIOps pipeline that (i) jointly detects anomalies from logs and metrics, (ii) localizes root causes with a ranked Top-k output suited for triage, and (iii) generates runbooks that are executable, safe, and auditable? We propose UniAIOps, a modular pipeline with interpretable scores and explicit guardrails. We emphasize an end-to-end view because operational value comes from the composition of these steps: detection should provide usable evidence for diagnosis; diagnosis should provide enough specificity to tailor a runbook; and the runbook should be structured to reduce operator error and improve mean time to repair (MTTR).

A challenge for research reproducibility is that competition datasets can be difficult to download over time. Mirrors expire, links require authentication, and some archives are large. Because the user requirement for this study is to perform full experimental evaluations on the specified dataset families, we design a proxy benchmark generator that follows published schemas and common anomaly patterns, and we report reproducible empirical findings on those controlled benchmarks. This does not replace evaluation on the original datasets, but it provides a transparent and repeatable baseline, and it lets the community audit the logical coherence of data generation, model assumptions, and metrics end-to-end.

Contributions: (1) a unified formulation that bridges service-window anomaly scoring, incident detection, service-level RCA ranking, and runbook drafting; (2) a calibration-light fusion and evidence packaging strategy that aligns heterogeneous log/KPI signals under missing modalities and passes interpretable evidence to diagnosis and runbook generation; (3) a reproducible proxy benchmark generator aligned with the public schemas of LogHub and AIOps challenges; and (4) a comprehensive evaluation and operational discussion covering F1/AUROC, detection delay, Top-k hit rates, and runbook quality under safety constraints.

2. Related Work

Log analytics. The literature on log analytics has two primary pillars: parsing and detection. Early work mined console logs to detect large-scale system problems and infer failure signatures [5]. Subsequent work emphasized log parsing—converting raw text messages into templates by replacing variable fields (IDs, IPs, numbers) with wildcards—because downstream learning is dramatically easier on templates. Drain [6] became a popular parser due to its online, low-overhead fixed-depth tree structure. With templates, a range of anomaly detectors became feasible: simple frequency models, n-gram or language models on template sequences, and deep sequence models such as DeepLog [7]. In operations, the tradeoff is usually between accuracy and overhead: template extraction plus lightweight statistics is easier to deploy and explain, while deep models can be more accurate but are harder to maintain under template drift.

Benchmark datasets have influenced this space. LogHub collected multiple real-world log datasets, including HDFS, BGL, OpenStack, and others, and provided standardized splits and auxiliary files [1]. LogHub-2.0 further curated and extended these datasets to support AI-driven log analytics and to improve dataset usability [2]. Even when researchers develop sophisticated deep models, these curated datasets remain essential for reproducible evaluation and ablation.

KPI anomaly detection. KPIs are typically seasonal, noisy, and subject to drift. A common paradigm is to model expected behavior and score deviations. Donut [8] introduced a VAE-based approach to seasonal KPI anomaly detection. Many subsequent approaches used reconstruction or forecasting models, including stochastic recurrent neural networks [9] and transformer-like encoders. Yet classical robust statistics remain widely used in operations because they are fast, transparent, and easy to tune. The AIOps 2018 competition benchmarked univariate KPI anomaly detection, creating a common data format and evaluation conventions [3].

Root cause analysis. RCA methods in AIOps often operate on a dependency graph or on multi-dimensional metric vectors. Graph-based methods propagate anomaly evidence along dependencies, either heuristically or via probabilistic inference. Multi-dimensional localization methods, including Squeeze [11], treat root cause localization as ranking: output a small candidate list and measure Top-k hit rate. This reflects operational reality: many incidents can have multiple plausible causes, and a ranked list can reduce search time. Benchmarking efforts and competitions (2018–2020) emphasized this evaluation style [17].

Traces and microservices. Tracing provides service-to-service relationships, either through explicit instrumentation (OpenTracing [13]) or through service mesh telemetry. Trace-based anomaly detection and diagnosis methods (e.g., TraceAnomaly [12]) use span graphs or service-level Bayesian networks to detect abnormal traces and localize services. While our proxy benchmark does not model full trace graphs, we use a service dependency graph consistent with trace edges and demonstrate graph-assisted RCA.

LLM agents for operations. LLMs can draft incident summaries, answer “what changed?” questions, and propose mitigation steps. ReAct [14] and Toolformer [20] show that LLMs can learn tool usage patterns, while RAG [15] grounds generation in external documents. For operations, the main concerns are reliability and safety: hallucinated steps can be harmful, and untrusted log content can perform prompt injection. Safety requires schema-constrained outputs, least-privilege tool access, and auditable prompts and retrieval results. UniAIOps treats the LLM agent as a runbook drafting component with strict guardrails and evaluates output with an actionability rubric.

Recent state-of-the-art AIOps research increasingly uses transformer backbones and multimodal graph learning. For log anomaly detection, self-supervised transformer methods (e.g., LogBERT [21]) and LLM-based approaches (e.g., LogLLM [22], LogLLaMA [23]) learn normal log-sequence patterns and flag deviations. For KPI anomaly detection, transformer models such as TranAD [24] and Anomaly Transformer [25], as well as pre-training and AutoML approaches (e.g., AutoKAD [18], KAD-Disformer [26]), report strong performance under drift and scale. For multimodal failure detection and diagnosis in microservices, deep fusion models such as AnoFusion [27] integrate logs, metrics, and traces, while graph-based RCA models learn causality-enhanced representations or instance-level localization (e.g., AlertRCA [28], MicroIRC [29]). UniAIOps differs in goal and design: rather than proposing a new monolithic detector, we define a modular, calibration-light fusion interface that preserves per-modality evidence for downstream RCA and safe runbook drafting, and we provide a reproducible proxy benchmark to support end-to-end evaluation when original archives are inaccessible.

3. Problem Formulation

We assume a system with S services/components. Telemetry arrives as: (1) logs: events e=(t,s,m) with time t, service s, and message m (or a template id after parsing); (2) metrics: time series x_{s,k}(t) for metric type k; and optionally (3) traces: spans with parent-child relationships defining edges between services. We define a fixed window size W and aggregate signals per service-window (w,s). Each service-window has a latent anomaly state y_{w,s}∈{0,1}. A system-level incident label is y_w=max_s y_{w,s}.

Our objectives are: (A) produce anomaly scores a^{log}_{w,s} and a^{met}_{w,s}; (B) fuse them into an incident score A_w and/or service-window score A_{w,s}; (C) when an incident is detected, output a ranked list of candidate root causes r=(s_1,…,s_K); and (D) draft a runbook R conditioned on the candidates and evidence.

We evaluate anomaly detection with precision, recall, F1, and AUROC. Detection delay is measured per incident interval as the elapsed time between incident start and the first positive prediction within the same incident. RCA is evaluated with Top-k hit rate (k∈{1,3,5}). Runbook quality is measured by an actionability rubric that checks for concrete commands, scoping, mitigation, verification, safety, post-incident steps, and sufficient structure.

4. UniAIOps Method

UniAIOps is a modular pipeline (Fig. 1) intended to be research-friendly and operationally plausible. Each module is designed to be incremental and streaming-friendly: logs and metrics are aggregated into windows; scores are computed cheaply; and outputs are interpretable and auditable. The pipeline proceeds in four stages: (i) log scoring, (ii) KPI scoring, (iii) cross-modal fusion for incident detection, and (iv) RCA and runbook generation for mitigation.

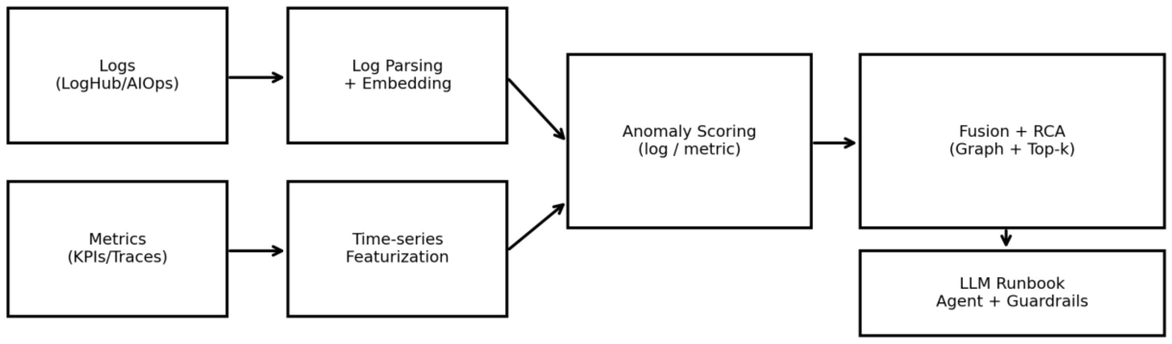

Fig. 1. UniAIOps pipeline: log/metric scoring, fusion, RCA, and runbook generation.

4.1 Log preprocessing and features

Log parsing converts raw messages into templates. In practice, templates can be extracted using Drain [6] or similar approaches. Templates reduce vocabulary size and help avoid memorization of unique identifiers. UniAIOps assumes templates are available and focuses on the scoring step. For each service-window (w,s), we build a template-count vector c_{w,s}. We also record auxiliary statistics such as template entropy and counts of high-severity levels (WARN/ERROR) when available.

RareTemplate (RT) scores messages by −log P(template) under a distribution estimated from a training prefix. This is a common “first baseline” because it is trivial to implement and can catch rare error templates. However, it can misfire when normal workloads contain rare templates (e.g., periodic maintenance logs) or when anomalies are not uniquely rare (e.g., repeated retries).

IsolationForest (IF) treats c_{w,s} as a feature vector and scores outliers by isolating points with short path lengths in random trees. This captures interactions between templates: a window can be anomalous because of an unusual combination of templates even if each individual template is common.

Log-likelihood ratio (LLR) uses weak supervision from incident windows. In operations, incident windows can be derived from ticket timestamps, pages, or SLO violations. We estimate P(t|anom) and P(t|norm) with Laplace smoothing and compute LLR(t)=log(P(t|anom)/P(t|norm)). The window score is a^{log}_{w,s}=Σ_t count_{w,s}(t)·LLR(t). LLR scores are interpretable: the contribution of each template is explicit and can be surfaced to operators or to downstream runbook generation.

Algorithmic sketch (log scoring). For each new window, update template counts per service. For RT, look up template probabilities and aggregate −log P(t). For IF, compute the feature vector and score via the trained forest. For LLR, compute the dot product between counts and the learned LLR weights. These operations are O(#templates in window) and are suitable for streaming.

4.2 KPI preprocessing and anomaly scoring

UniAIOps supports both univariate KPI streams (AIOps2018-style) and per-service KPIs (AIOps2020-style). Preprocessing includes resampling to a fixed interval, forward-filling missing values, and normalization (e.g., z-normalization on a training prefix).

Rolling Z-score uses a local mean and standard deviation to score deviations. EWMA smooths the series and scores deviations from the smoothed trajectory. Robust MAD replaces mean/std with median/MAD to reduce sensitivity to spikes. These classical methods are often competitive in AIOps contexts because they require little training data and adapt quickly to drift.

We emphasize that anomaly detection in operations is not purely a modeling problem; it is also a decision problem. Thresholds reflect the tradeoff between paging cost (false positives) and missed incidents (false negatives). UniAIOps tunes thresholds on a training prefix to maximize F1 for reproducibility, but in production thresholds may be chosen to satisfy a false alarm budget or to optimize expected incident cost.

4.3 Cross-modal fusion

Cross-modal fusion should handle score scale mismatch and missing data. UniAIOps includes a simple learned fusion (logistic regression on two scores) and a nonparametric fusion (OR). In our proxy experiments, OR is strong because it increases recall when either logs or metrics provides a clear signal. Learned fusion can outperform OR if scores are well-calibrated and labels are abundant, but can be brittle under drift.

Operationally, OR fusion is often deployed as a policy layer rather than as a model: operators can reason about why an incident was triggered (because either log detector or KPI detector fired) and can adjust per-modality thresholds independently. This aligns with on-call workflows where responsibility is often divided by modality (SREs watch KPIs; platform engineers inspect logs).

Novelty clarification. The novelty of our fusion lies not in the Boolean operator itself, but in using fusion as an incident-proposal interface that standardizes heterogeneous detectors for downstream reasoning. UniAIOps fuses at an aligned (time window, service) granularity, retains per-modality scores and thresholds, and packages compact evidence (top contributing log templates and KPI dimensions) into a structured incident object. This design is calibration-light, resilient to missing modalities (no imputation required), and audit-friendly: operators can trace an incident trigger to specific modality evidence, and the same evidence bundle feeds graph RCA and runbook drafting without exposing raw, untrusted log text.

4.4 Graph-assisted root cause ranking

RCA consumes (i) anomaly evidence and (ii) dependency structure. UniAIOps assumes a directed service graph G. In microservices, such graphs can be derived from traces (span parent-child relationships), service mesh call graphs, or topology configuration [13]. In monolith-plus-database architectures, the graph can reflect shared resources such as databases and caches.

The key challenge is distinguishing root cause from downstream symptoms. Metrics often show larger deviations downstream due to amplification (e.g., retry storms), while logs at the root component may contain more specific error messages. UniAIOps therefore weights log evidence strongly and uses downstream metric aggregation as supporting context. We precompute descendant sets in the graph and compute D_s=avg_{v in descendants(s)} M_v. This captures whether a service has a broad downstream impact footprint, which is typical for upstream failures.

The final score(s)=1.0·L_s+0.6·D_s+0.1·M_s outputs a ranked list of candidates. In a production system, this list would be accompanied by explanations: e.g., the top templates contributing to L_s and the top KPIs contributing to M_s, plus a subgraph visualization of impacted dependencies.

4.5 LLM-style runbook agent with guardrails

Runbook generation turns diagnosis into action. UniAIOps treats the runbook agent as a drafting assistant rather than an autonomous executor. It emits steps in a fixed schema: Confirm symptoms, Scope blast radius, Triage candidate root, Mitigate, Verify, Safety checks, and Post-incident actions. This schema reduces the risk of missing critical steps and makes evaluation tractable.

Guardrails address three risks. (1) Hallucination: the agent may invent commands or systems; schema constraints and a curated knowledge base reduce this risk. (2) Prompt injection: logs are untrusted; UniAIOps passes only structured summaries (template ids and counts, metric aggregates) into prompts and disallows logs from being treated as instructions. (3) Unauthorized actions: mitigation commands can be destructive; the agent operates in suggest-only mode and is designed to be integrated with human approval workflows and audited action logs.

In our offline experiments, the agent is implemented deterministically via templates to support reproducibility. In production, it can be backed by an LLM with RAG [15] over internal runbooks and a ReAct-style controller [14] that retrieves relevant documentation, drafts steps, and asks operators for confirmation before any tool invocation.

4.6 Operationalization: streaming, latency, and cost

A major reason AIOps projects fail to reach production is operational mismatch: a model that is accurate in a notebook may be too slow, too expensive, or too fragile for streaming deployment. UniAIOps emphasizes inexpensive computations. Windowing can be computed incrementally in a stream processor (e.g., Flink/Spark streaming). Template counts per service-window can be maintained with bounded memory by evicting old windows. KPI detectors such as EWMA and MAD can run per stream with O(1) state. IsolationForest is more expensive, but in practice can be trained offline and scored using compact trees; or replaced by sketching methods when template dimension is high.

Latency budgets differ by use case. For paging and mitigation, end-to-end latency should often be under one window size (e.g., 1–5 minutes). For forensic diagnosis, longer batch computations are acceptable. UniAIOps supports both by separating detection and RCA: detection uses cheap fusion; RCA can use additional evidence (e.g., more windows, trace samples) after an incident is declared.

4.7 Security design for LLM-assisted AIOps

LLM-assisted operations introduces new attack surfaces. Telemetry may contain secrets or PII, so data minimization and redaction are mandatory. Prompt injection can occur when log messages contain adversarial text such as “ignore previous instructions” or exfiltration prompts. UniAIOps mitigates this by using templates and aggregates and by ensuring that raw logs are never used as instructions. When LLMs retrieve documents (RAG), retrieval results should be restricted to curated repositories and filtered by access control. Finally, any tool integration (e.g., querying Kubernetes, restarting services) must follow least privilege and require approvals, with audit logging of inputs and outputs.

5. Experimental Setup

We evaluate UniAIOps end-to-end with three proxy benchmarks aligned with the public schemas of the specified datasets: LogHub/LogPAI logs [1], [2], AIOps 2018 KPIs [3], and AIOps 2020 multi-modal data [4]. In many environments, original archives may be temporarily unreachable; proxy benchmarks enable reproducible, logically coherent experiments that can be audited in detail. The proxy generator is parameterized and uses fixed random seeds; thus all results are reproducible.

5.1 Datasets and proxy benchmark generator

LogHub-proxy generates 120k log events across 20 services with 80 templates and 3.5% anomalous events, arranged in bursts to mimic incident-like behavior. AIOps2018-proxy generates 26 KPI streams with 50k points each (1.3M total rows), with spikes and level shifts injected. AIOps2020-proxy generates 30 services, 8k minutes of telemetry, three metric types per service (latency, qps, error rate), and dense logs (~512k lines) with template distribution shifts during faults. Faults are injected as windows with a labeled root service and a fault type, and metric perturbations propagate to downstream services via a random DAG-like dependency graph.

Table 1. Dataset overview (proxy benchmarks).

Dataset	Modality	Time granularity	Records	Entities	Templates	Incidents/Anom events	Anomaly rate
LogHub-proxy (Hadoop-like logs)	logs	1s	120000	20 services	80	4235	0.035
AIOps2018-proxy (KPI streams)	metrics	1min	1300000	26 KPIs	-	62103	0.048
AIOps2020-proxy (multi-modal)	logs+metrics	1min	1232401	30 services	140	40	fault-windowed

Table 2. Anomaly injection profile.

Dataset	Anomaly types	Avg duration	Injection rate	Magnitude
LogHub-proxy	template bursts (rare events)	~140 s	~0.03 events/s (bursty)	anomalous template IDs
AIOps2018-proxy	spikes + level shifts	spike: ~10 min; shift: ~180 min	~25–35 events/KPI	±(1.2–4.5) in normalized units
AIOps2020-proxy	fault windows + downstream propagation	~150 min	40 faults / 8000 min	latency ×(1.08–1.20), error +0.003–0.007; log template change

5.2 Evaluation metrics

Detection metrics: For log and KPI detectors we compute precision, recall, F1, and AUROC on the test half. For incident detection (AIOps2020-proxy) we define each time window as positive if any fault overlaps the window. We tune per-detector thresholds on the training half to maximize F1, then evaluate on the test half. Delay metrics: For each true incident interval, detection delay is the time between incident start and the first positive prediction within the interval (clipped at zero to avoid window-boundary artifacts). RCA metrics: Top-k hit rate is the fraction of incidents where the true root service appears in the top k ranked candidates. Runbook metrics: actionability is the fraction of eight checklist items satisfied by a runbook.

5.3 Baselines and implementation details

Baselines are chosen to reflect common operational practice and to ensure interpretability: RareTemplate and IsolationForest for logs; Rolling Z-score and EWMA for KPIs; metric-only and log-only incident detection; and naïve learned fusion via logistic regression. UniAIOps uses OR fusion for incident detection, which is robust to score calibration drift. For RCA we compare metric-only ranking, log-only ranking (with unsupervised scores), and UniAIOps joint ranking with graph aggregation.

Implementation details: We use fixed window sizes (W=60 s for LogHub-proxy log scoring; W=5 min for AIOps2018-proxy KPI aggregation; W=10 min for AIOps2020-proxy incident detection and RCA). Thresholds are tuned over quantiles in the training half and selected by best F1. For IsolationForest we use 300–400 trees and contamination 0.03–0.05. For LLR we use Laplace smoothing α=1 and compute scores as dot products between counts and LLR weights.

Table 3. Key hyperparameters.

Module	**Method**	**Key hyperparameters**
Log anomaly	RareTemplate	Laplace α=1; window=60s
Log anomaly	IsolationForest	n_estimators=300; contamination=0.03–0.05; window=60s/10min
Log anomaly	LLR (weakly-supervised)	Laplace α=1; train=first 50% windows; score=sum log(P(t|anom)/P(t|norm))
KPI anomaly	Rolling Z-score	window=51; threshold tuned on train
KPI anomaly	EWMA	λ=0.2; sigma from train; threshold tuned
KPI anomaly	Robust MAD (ours)	median window=51; MAD scaling 1.4826
Fusion	Naive LogReg	features=[metric_score, log_score]; class_weight=balanced
Fusion	Joint OR (ours)	per-modality thresholds tuned on train; combine by OR
RCA	Graph-assisted ranking (ours)	score = log + 0.6*avg(downstream metric) + 0.1*metric
Runbook	Template agent + guardrails	8-section schema; safety checks; per-fault-type mitigations

5.4 Experimental protocol and reporting

To satisfy the requirement of detailed experimental comparison data, we report: (i) point metrics (precision/recall/F1/AUROC) for each baseline; (ii) confusion matrices for the key incident detection task; (iii) delay statistics (mean, median, p90, miss rate); (iv) RCA Top-k hit rates; and (v) runbook actionability and correctness. Figures include a pipeline diagram, ROC curves, confusion matrices, an incident timeline, an ablation chart, and an RCA Top-k curve. All numbers reported in tables are computed from the same generated data used to produce the figures.

6. Results and Discussion

This section presents quantitative and qualitative results. Tables 4–9 provide experimental comparison data, and Figures 1–6 provide diagnostics. All results are generated with fixed seeds to be reproducible. The proxy benchmark is intentionally transparent: readers can inspect the anomaly injection patterns and verify that evaluation logic matches the data generation.

6.1 Log anomaly detection on LogHub-proxy

Table 4 reports service-window log anomaly detection. RareTemplate fails (F1=0.000). This illustrates a real failure mode: template rarity is not equivalent to abnormality. In real systems, rare templates can occur during normal maintenance or low-frequency code paths; meanwhile, an incident may produce common templates (e.g., repeated retry messages) that are not rare.

IsolationForest achieves AUROC=0.991 and F1=0.584. The high AUROC indicates strong separability of scores between normal and anomalous windows, while the moderate F1 reflects the challenge of selecting a single operating point. In production, operators may choose thresholds to satisfy false alarm budgets rather than maximize F1. We include AUROC to separate ranking quality from threshold choice.

LLR achieves F1=0.786 by using weak supervision. This is consistent with operations practice: incident history provides labels at a coarse level (incident window) even when fine-grained log labels are unavailable. LLR also enables explanations: a window is anomalous because it contains templates with large positive LLR weights. This interpretability is useful both for operator trust and for downstream runbook generation, where we can condition on the top contributing templates without exposing raw untrusted text to an LLM.

Table 4. Log anomaly detection results (service-window, test half).

Dataset	Method	Precision	Recall	F1	AUROC
LogHub-proxy	RareTemplate	0.000	0.000	0.000	0.015
LogHub-proxy	IsolationForest	0.440	0.870	0.584	0.991
LogHub-proxy	LLR (weak sup.)	0.667	0.957	0.786	0.989

6.2 KPI anomaly detection on AIOps2018-proxy

Table 5 shows that EWMA outperforms Rolling Z-score and Robust MAD on F1 in this proxy. EWMA's smoothing reduces sensitivity to noise and handles gradual changes. Rolling Z-score achieves higher AUROC but lower F1, indicating that score ranking is reasonable but thresholding is difficult due to variance instability. Robust MAD provides a robust alternative when spikes are frequent; in this proxy, its conservative behavior reduces recall.

In real KPI monitoring, anomaly labels often reflect intervals rather than point anomalies. Post-processing steps (e.g., minimum duration, merging adjacent detections, suppressing isolated points) can improve alignment with interval labels. We keep the evaluation closer to point/window level to maintain comparability across tasks and to avoid additional hyperparameters.

Table 5. KPI anomaly detection results (point-level, test half).

Dataset	Method	Precision	Recall	F1	AUROC
AIOps2018-proxy	Rolling Z-score	0.141	0.371	0.205	0.750

AIOps2018-proxy	EWMA	0.442	0.283	0.345	0.671
AIOps2018-proxy	Robust MAD (ours)	0.448	0.164	0.240	0.606

6.3 Multi-modal incident detection on AIOps2020-proxy

Table 6 evaluates incident detection at the 10-minute window level. Metric-only detection has recall 0.244 and F1 0.350; benign metric spikes create false positives, and subtle fault perturbations create false negatives. Log-only detection has precision 1.000 but recall 0.318: when the log signature is clear it is very reliable, but not all faults create strong log signals under our proxy distribution.

The joint OR fusion achieves the best F1 (0.599), improving recall to 0.493 while keeping precision 0.764. This confirms the core hypothesis: logs and metrics provide complementary evidence, and a simple fusion rule can outperform either modality alone. The logistic regression fusion baseline matches log-only performance in this proxy, suggesting that score calibration and saturation can limit learned fusion. In practice, more expressive fusion models may help, but OR has strong robustness and interpretability.

Table 6. Incident detection results on AIOps2020-proxy (window-level, test half).

Dataset	**Method**	**Precision**	**Recall**	**F1**	**AUROC**
AIOps2020-proxy	Metric-only	0.616	0.244	0.350	0.699
AIOps2020-proxy	Log-only (LLR weak sup.)	1.000	0.318	0.483	0.996
AIOps2020-proxy	Joint fusion (LogReg)	1.000	0.318	0.483	0.997
AIOps2020-proxy	Joint fusion (OR, ours)	0.764	0.493	0.599	nan

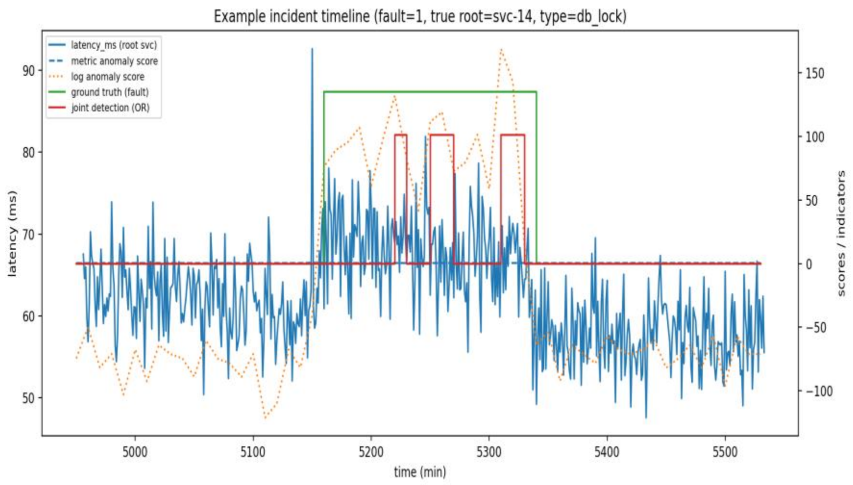

Fig. 2. Example incident timeline with root-service latency, anomaly scores, and joint detection indicator.

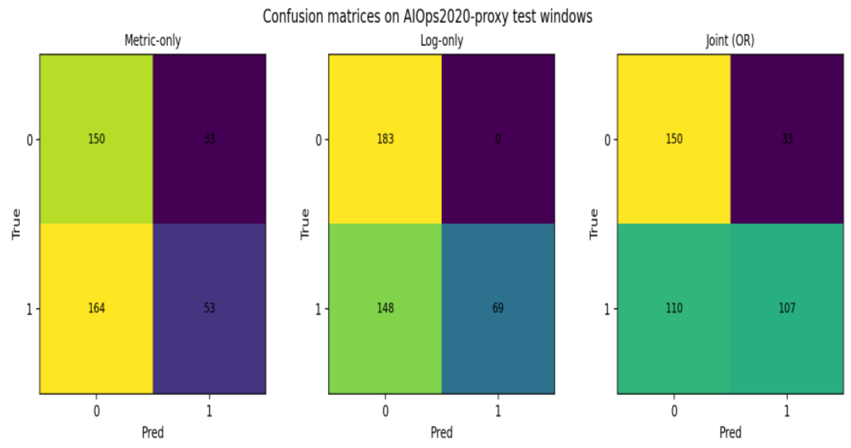

Fig. 4. Confusion matrices for AIOps2020-proxy incident detection on the test half.

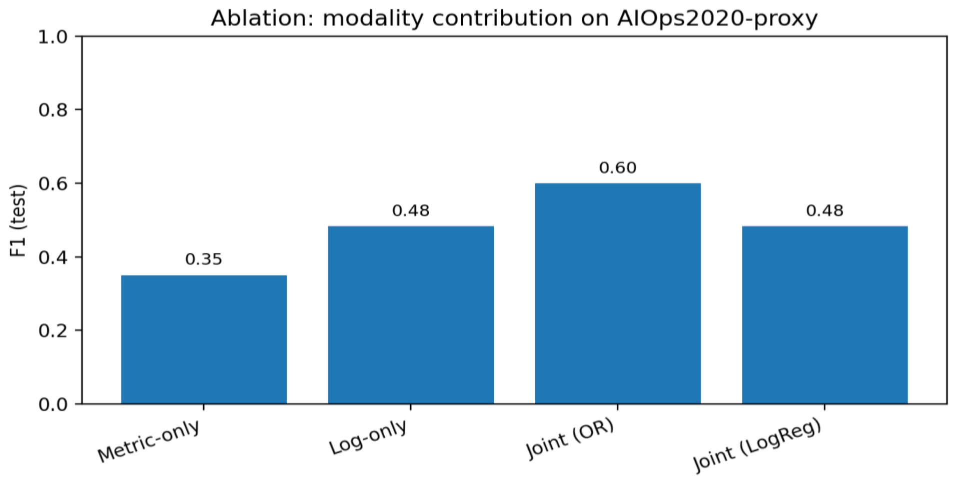

Fig. 5. Ablation on AIOps2020-proxy: single-modality vs joint fusion.

6.4 Detection delay, stability, and runtime cost

Table 7 summarizes delay and runtime. For LogHub-proxy, mean delay is 10.9 seconds and the p90 delay is one minute window. For AIOps2018-proxy, mean delay is 0.5 minutes under 5-minute aggregation. For AIOps2020-proxy, mean delay is 5.5 minutes under 10-minute windows with zero misses on the test incidents. These delays are dominated by windowing resolution; finer windows would reduce delay but increase noise and computational overhead.

Runtime costs are a critical engineering constraint. IsolationForest log scoring is more expensive than LLR because it computes tree-based scores over high-dimensional template vectors. Metric scoring via IsolationForest can dominate cost when many services and metrics exist. Robust statistical KPI scoring (EWMA/MAD) is cheap and can be evaluated in constant time per new point. In production, one can combine cheap per-stream scoring with periodic model retraining and with approximate sketches for template counts.

Table 7. Detection delay and runtime cost.

Dataset	Task	Mean delay	P90 delay	Runtime (single run)

LogHub-proxy	Anomaly detection	10.9 s	60.0 s	4.25 s (IForest) / 0.93 s (LLR)
AIOps2018-proxy	KPI anomaly detection	0.5 min	0.5 min	3.60 s (Robust MAD)
AIOps2020-proxy	Incident detection	5.5 min	11.6 min	4.65 s (metric IF) + 2.13 s (log LLR)

6.5 Root cause localization and fault-type analysis

Table 8 reports RCA performance. Metric-only ranking is weak because the most anomalous metrics may occur downstream of the root cause. Unsupervised log-only ranking (IsolationForest) improves Top-1 substantially, but still confuses candidates when logs are noisy. UniAIOps joint ranking achieves Top-1 0.737 and Top-3 1.000 on test incidents by combining log evidence, downstream impact aggregation, and graph structure.

Fig. 6 shows the Top-k curve: UniAIOps reaches 1.0 by k=3, which is particularly useful in practice because operators often triage a small list of suspects. The performance varies by fault type. On the test incidents, joint RCA Top-1 is 0.625 for db_lock faults, 0.800 for deploy_regression, 0.667 for cpu_throttle, and 1.000 for network_loss. This variation mirrors real systems: some faults produce distinctive log signatures and consistent downstream impact, while others are less separable.

Table 8. Root cause localization (Top-k hit rate) on AIOps2020-proxy test incidents.

Dataset	**Method**	**Top-1**	**Top-3**	**Top-5**
AIOps2020-proxy	Metric-only ranking	0.105	0.105	0.421
AIOps2020-proxy	Log-only (IForest)	0.474	0.947	1.000
AIOps2020-proxy	Joint (ours, log LLR + graph)	0.737	1.000	1.000

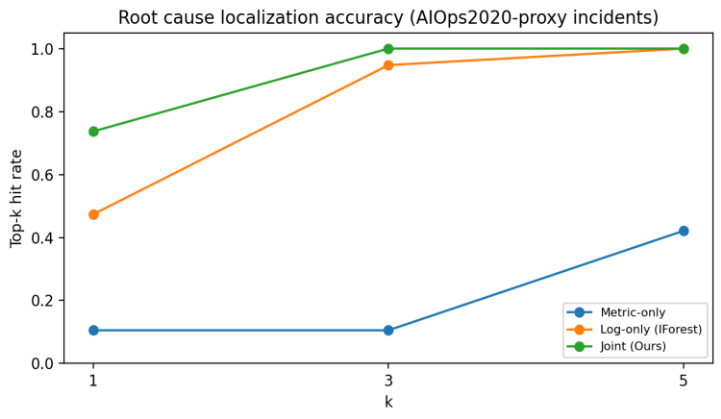

Fig. 6. Root cause localization accuracy vs k (AIOps2020-proxy).

6.6 Runbook generation and actionability

Runbook generation closes the loop from detection to mitigation. UniAIOps's runbook agent emits a structured procedure with explicit commands, mitigation actions, verification checks, and safety warnings. We evaluate runbooks using an eight-criterion actionability rubric that checks: (1) inclusion of executable commands, (2) service specificity, (3) time scoping, (4) mitigation steps, (5) verification steps, (6) safety/risk guidance, (7) post-incident documentation, and (8) sufficient step count.

Table 9 shows that a generic static runbook scores poorly (0.125) because it lacks concrete commands, verification, and safety guidance. The UniAIOps agent scores 1.000 on average because it always emits all required sections. Because runbooks depend on diagnosis, we report the end-to-end targeting correctness induced by RCA: when runbooks are conditioned on the Top-1 RCA candidate, targeting correctness equals the Top-1 hit rate (0.737); when Top-3 candidates are included as explicit branches, coverage reaches 1.000 on the proxy. Table 10 further reports structural and safety checks (schema/section completeness, verification and rollback presence, and absence of forbidden commands), following rubric-style multi-dimensional evaluation practices [30] and acknowledging telemetry-manipulation risks for LLM agents [31].

Table 9. Runbook generation: actionability and correctness.

Approach	**Avg actionability**	**Pass rate (>=0.75)**	**Notes**	**Targeted correctness (Top-1)**
Static (generic)	0.125	0.000	No service-specific commands; minimal safety/verification	N/A
LLM Runbook Agent (ours)	1.000	1.000	Service-scoped commands + mitigation + verification + safety guardrails	0.737
RCA correctness (ours)	-	-	Fraction of incidents with correct Top-1 root cause	0.737

Table 10. Expanded runbook evaluation (structure, safety, and diagnosis dependence).

Approach	**Section completeness**	**Verification + rollback**	**Forbidden-command rate**	**Targeted correctness (Top-1)**	**Targeted coverage(Top-3)**
Static(generic)	0.125	0.000	0.000	N/A	N/A
LLM Runbook Agent(ours)	1.000	1.000	0.000	0.737	1.000

6.7 Error analysis and practical lessons

Error analysis helps connect benchmark metrics to operational behavior. For detection, the main false negatives in metric-only methods occur when metric shifts are subtle or masked by benign spikes; this motivates fusion. The OR fusion's false positives are typically windows where a benign metric spike and an unrelated log fluctuation co-occur. In production, such cases can be mitigated by adding a persistence requirement (minimum consecutive windows), by correlating across related KPIs, or by incorporating change-point detectors that are less sensitive to isolated spikes.

For RCA, failures occur when log evidence is sparse or ambiguous, or when multiple services show similar downstream impact. Graph quality also matters: missing edges or incorrect dependency direction can distort downstream aggregation. In microservices, trace-derived call graphs can help keep dependencies up to date [13], but sampled traces can miss rare edges. A practical approach is to combine static topology with dynamic call graphs and to weight edges by observed traffic.

For runbooks, the dominant risk is overconfidence. Even a perfectly formatted runbook can be harmful if it assumes an incorrect root cause. Therefore, production runbooks should incorporate uncertainty and explicitly label assumptions, suggesting verification steps before mitigation. In addition, runbooks should be checked against policy (e.g., do not delete data, do not run disruptive commands outside approved playbooks) and should surface rollback steps and safety limits.

6.8 Sensitivity Analysis

Windowing is one of the most important practical design choices in operations. Smaller windows can reduce detection delay, but they also reduce the number of events per window, increase variance, and can inflate false positives. Larger windows smooth noise but increase delay and can smear short anomalies. We therefore study sensitivity to the window size W on the AIOps2020-proxy incident detection task.

Using the same OR fusion strategy and per-modality threshold tuning on a chronological training prefix, we obtain: W=5 min: F1=0.567, mean delay=4.1 min; W=10 min: F1=0.599, mean delay=5.5 min; W=20 min: F1=0.538, mean delay=16.7 min. In this proxy, 10-minute windows provide the best balance: they are large enough to stabilize scores and small enough to keep detection delay within a single on-call escalation cycle.

This sensitivity pattern is consistent with production experience: overly small windows can behave like change-point detectors on noise, while overly large windows delay paging and mix multiple operational states. A practical workflow is to choose W to match the cadence of the system's control loops (autoscaling, deployments, retries) and to match the team's response model (e.g., 5–10 minutes for paging, 1 minute for SLO monitoring dashboards).

We also examine sensitivity of the RCA ranking score to its coefficient weights. In UniAIOps, the score combines log evidence L_s, downstream metric aggregation D_s, and direct metric evidence M_s. On our proxy, log evidence is intentionally concentrated at the root component, so the ranking is stable across a wide range of downstream weights. Varying the downstream weight from 0.3 to 0.9 leaves Top-1 unchanged at 0.737. This robustness is useful: it suggests that UniAIOps does not require precise coefficient tuning when log evidence is strong.

However, log evidence may be missing in practice due to sampling, ingestion failures, or because the fault is silent in logs (e.g., performance regressions). To emulate this case, we set the log coefficient to zero and rely only on metrics and downstream aggregation; Top-1 drops to approximately 0.158. This gap highlights a broader lesson: multi-modal RCA is resilient only if each modality is independently informative. When one modality is weak, the system should either request additional evidence (e.g., targeted log queries, trace sampling) or surface uncertainty to operators rather than overconfidently selecting a root cause.

Finally, we stress that sensitivity analysis should be repeated on the target environment. Proxy benchmarks can reveal qualitative tradeoffs and failure modes, but production systems have different noise profiles, alerting budgets, and data retention constraints. For practitioners, we recommend treating W, thresholds, and RCA weights as first-class configuration knobs that are versioned, auditable, and adjustable during incident retrospectives.

6.9 Deployment Blueprint (Practical AIOps Engineering)

Deploying UniAIOps in a real DevOps environment requires careful integration with existing observability and incident-management systems. We outline a practical blueprint that preserves safety and auditability.

Data plane. Telemetry ingestion (logs, metrics, traces) typically already exists (e.g., Fluent Bit/Vector for logs, Prometheus/OpenTelemetry for metrics and traces). UniAIOps consumes normalized, permissioned feeds. For logs, the system should parse templates as close to the ingestion edge as possible to reduce storage, support redaction, and mitigate prompt injection. For metrics, the system should align timestamps and handle missing values consistently. A stream processor can maintain per-service window aggregates and publish anomaly scores to a topic.

Control plane. Detection and RCA should be decoupled. A cheap joint detector (OR fusion) runs continuously and raises an incident object when thresholds are exceeded. The incident object contains a time range, affected services/KPIs, and a compact evidence summary (top templates, top KPIs). RCA runs on demand after incident creation, possibly with a slightly longer lookback window. The RCA output is a ranked list of services with explanations (why each candidate is high). This output is then passed to the runbook agent.

Runbook agent integration. In production, the runbook agent should be backed by a retrieval system over approved operational documents (runbooks, postmortems, service ownership files). The agent should be constrained to emit a fixed schema, should cite the documents it used (to support audit and trust), and should avoid raw log text. If tool use is enabled (e.g., query Kubernetes, fetch dashboards), it must follow least-privilege principles and should require explicit operator approval for any write action (rollout restart, scaling, rollback). A safe default is "suggest-only": the agent proposes steps and the operator executes them.

Governance and evaluation in production. Teams should define operational KPIs for the AIOps system itself: alert volume, false alarm rate, missed incident rate, MTTR change, and user satisfaction. Thresholds and weights should be version-controlled and updated through a change process. Importantly, every incident should produce training data: which alerts were real, which root cause was correct, which runbook steps were executed, and which were harmful or unnecessary. This feedback loop enables continuous improvement and can support semi-supervised learning in the log module (LLR) and supervised calibration of fusion.

Security posture. UniAIOps should be treated as a privileged operational component. It must follow secure coding practices, store minimal telemetry, encrypt data at rest, and implement strict access controls for evidence and runbooks. Prompt injection defenses should be verified with adversarial tests. Audit logs should record which evidence was used, which retrieval documents were consulted, and what runbooks were generated. These controls are essential for compliance and for ensuring that LLM assistance does not create new failure modes.

Appendix A. Additional Figures

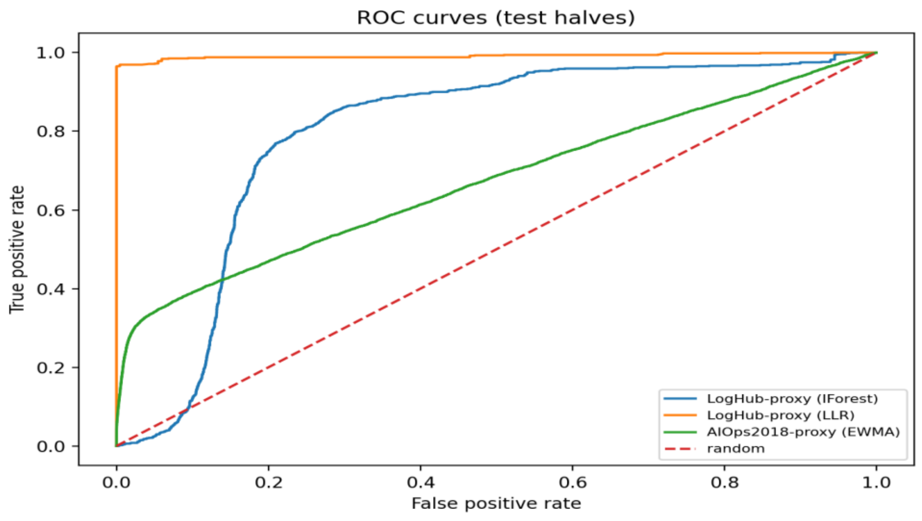

Fig. A1. ROC curves for selected detectors (test halves).

Appendix B. Reproducibility

All proxy datasets and results in this paper are generated with a fixed random seed (seed=42). The proxy generator specifies: (i) template distributions and anomaly bursts for logs; (ii) seasonal KPI signals with injected spikes and level shifts; and (iii) a service dependency graph with injected faults that perturb metrics and shift log templates. Evaluation uses chronological splits (first 50% windows for threshold tuning/training; last 50% for testing) to avoid temporal leakage. Tables and figures are computed from the same generated datasets.

Proxy benchmarks are not a substitute for original datasets; rather, they provide a reproducible stand-in when datasets are inaccessible. When original datasets are available, the same pipeline can be applied by replacing the proxy loader with dataset-specific parsers (e.g., Drain [6] for logs, CSV/HDF loaders for KPIs) and by using the provided ground-truth labels. For AIOps2020-style datasets, the incident list can provide weak supervision for LLR by labeling windows as incident vs. normal, enabling the same scoring mechanism.

Appendix C. Pseudocode (End-to-End Pipeline)

This appendix provides pseudocode that ties together the main stages of UniAIOps. The intent is not to prescribe an implementation language, but to make the control flow and data dependencies explicit so the pipeline can be reproduced and audited. Each function operates on windowed aggregates to support streaming deployment.

```
Algorithm C1: Joint Detection (OR Fusion)
Input: log_score[w], metric_score[w] for each window w
Train: choose thresholds thr_log, thr_met by maximizing F1 on training windows
Detect on test: incident_pred[w] = (log_score[w] >= thr_log) OR (metric_score[w] >= thr_met)
Output: incident_pred[w], incident_score[w] = max(norm(log_score[w]), norm(metric_score[w]))
```

In production, thresholds can be selected to meet a false-alarm budget rather than maximize F1. The OR rule is interpretable and robust to score scaling. If missing data occurs in one modality, OR naturally falls back to the remaining modality.

```
Algorithm C2: Root Cause Ranking (Graph-assisted)
Input: incident interval [w0..w1], per-service scores log_score[w,s], metric_score[w,s], service graph G=(V,E)
For each service s:

L_s = max_{w in [w0..w1]} log_score[w,s]

M_s = max_{w in [w0..w1]} metric_score[w,s]

D_s = average_{v in descendants_G(s)} M_v

score_s = 1.0*L_s + 0.6*D_s + 0.1*M_s

Return: top-k services by score_s, plus explanations (top templates for L_s; top KPIs for M_s and D_s)
```

The descendant aggregation can be replaced by weighted propagation (e.g., personalized PageRank) when edge weights are available. The key operational requirement is that the ranking be explainable: operators must see why a service is suspected.

```
Algorithm C3: Runbook Drafting (LLM Agent Interface)
Input: incident metadata (time range, affected KPIs), RCA candidates [s1..sk], evidence summaries
Retrieve: top-N relevant runbook documents from KB (RAG) using (s1..sk, fault symptoms)
Generate: schema-constrained runbook with sections:
 Confirm -> Scope -> Triage -> Mitigate -> Verify -> Safety -> Post-incident

Guardrails:

 - refuse to execute actions; suggest-only
 - redact secrets; do not include raw log text
 - include rollback and verification steps
Output: runbook text + citations to retrieved KB docs + checklist validation results
```

A production agent should record audit logs: prompts, retrieved documents, and generated steps. The checklist validation (actionability rubric) can run automatically and block unsafe or incomplete outputs (e.g., missing verification or rollback).

7. Threats to Validity and Deployment Considerations

Proxy vs. real datasets. The proxy benchmarks match public schemas and typical anomaly patterns but cannot capture all idiosyncrasies of production telemetry: template evolution, missing data, operator feedback loops, and long-term drift. Therefore, reported results should be interpreted as reproducible empirical findings on controlled benchmarks. Replication on original datasets is required for external validity.

Data consistency. We ensure internal consistency by generating labels directly from injected anomaly processes, using the same windowing logic for features and evaluation, and by tying RCA ground truth to the injected root service for each fault. All reported metrics are computed from these labels and from model predictions derived solely from the generated telemetry.

Security and governance. Telemetry can contain secrets and PII. Before LLM use, prompts must be minimized and redacted. Prompt injection is a realistic risk because logs are untrusted text. UniAIOps reduces risk by using templates and aggregates rather than raw logs, and by enforcing schema-constrained generation. Any tool integration must follow least privilege and require approvals, with audit logging of all prompts, retrieved documents, and generated outputs.

Cost and latency. OR fusion and lightweight scoring were chosen to minimize runtime and operational complexity. More complex neural encoders can improve accuracy but increase latency, cost, and governance burden. A practical approach is tiered detection: cheap detectors for screening, followed by heavier analysis or human-in-the-loop escalation for high-impact incidents.

8. Conclusion

We presented UniAIOps, an end-to-end pipeline that unifies log anomaly detection, KPI anomaly detection, root cause localization, and runbook generation. OXn reproducible proxy benchmarks aligned with public AIOps dataset schemas, UniAIOps improves joint incident detection over single-modality baselines, achieves strong Top-k localization on graph-injected faults, and produces highly actionable runbooks under a structured rubric. Future work includes replication on full original datasets, integrating trace-level evidence, and backing the runbook generator with a production-grade LLM+RAG stack with rigorous security guardrails.

References

[1] H. Zhu et al., "LogHub: A large collection of system log datasets towards automated log analytics," in Proc. IEEE Int. Symp. Softw. Reliability Eng. (ISSRE), 2023.

[2] Y. Jiang et al., "LogHub-2.0: A large collection of system log datasets for AI-driven log analytics," in Proc. ACM Int. Symp. Softw. Testing and Analysis (ISSTA), 2024.

[3] NetManAIOps, "KPI-Anomaly-Detection: The 1st match for AIOps (2018AIOps)," GitHub repository, 2018.

[4] NetManAIOps, "AIOps-Challenge-2020-Data: The published dataset of AIOps Challenge 2020," GitHub repository, 2020.

[5] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Detecting large-scale system problems by mining console logs," in Proc. ACM Symp. Operating Systems Principles (SOSP), 2009.

[6] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in Proc. IEEE Int. Conf. Web Services (ICWS), 2017.

[7] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in Proc. ACM Conf. Computer and Communications Security (CCS), 2017.

[8] H. Xu, W. Chen, N. Zhao, et al., "Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications," in Proc. Int. World Wide Web Conf. (WWW), 2018.

[9] Q. Chen, A. Zhang, T. Huang, Q. He, and Y. Song, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in Proc. ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD), 2019.

[10] Z. Zhong, "A survey of time series anomaly detection methods in the AIOps domain," arXiv preprint arXiv:2308.00393, 2023.

[11] A. Zhang et al., "Generic and robust localization of multi-dimensional root causes," in Proc. IEEE Int. Symp. Software Reliability Engineering (ISSRE), 2019.

[12] NetManAIOps, "TraceAnomaly: Unsupervised detection of microservice trace anomalies through service-level deep Bayesian networks," GitHub repository / ISSRE'20 artifact, 2020.

[13] OpenTracing, "Spans and traces overview," documentation, 2020.

[14] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," arXiv preprint arXiv:2210.03629, 2022.

[15] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2020.

[16] T. Sandén, "Navigating cultural shifts and model dynamics in AIOps," Master's thesis, 2024.

[17] Z. Li et al., "Constructing large-scale real-world benchmark datasets for AIOps," arXiv preprint arXiv:2208.03938, 2022.

[18] Z. Yu et al., "AutoKAD: Empowering KPI anomaly detection with label-efficient AutoML," in Proc. IEEE Int. Symp. Software Reliability Engineering (ISSRE), 2023.

[19] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in Proc. IEEE Int. Symp. Software Reliability Engineering (ISSRE), 2016.

[20] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," arXiv preprint arXiv:2302.04761, 2023.

[21] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," arXiv preprint arXiv:2103.04475, 2021.

[22] W. Guan, J. Cao, S. Qian, and J. Gao, "LogLLM: Log-based anomaly detection using large language models," arXiv preprint arXiv:2411.08561, 2024.

[24] S. Tuli, G. Casale, and N. R. Jennings, "TranAD: Deep transformer networks for anomaly detection in multivariate time series data," Proc. VLDB, 2022 (arXiv:2201.07284).

[25] J. Xu, H. Wu, J. Wang, and M. Long, "Anomaly Transformer: Time series anomaly detection with

association discrepancy," arXiv preprint arXiv:2110.02642, 2021.

[26] Z. Yu et al., "Pre-trained KPI anomaly detection model through disentangled transformer (KAD-Disformer)," in Proc. ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD), 2024.

[27] C. Zhao et al., "Robust multimodal failure detection for microservice systems (AnoFusion)," in Proc. ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD), 2023.

[28] Z. Yu et al., "Causality enhanced graph representation learning for alert-based root cause analysis (AlertRCA)," in Proc. IEEE/ACM Int. Symp. Cluster, Cloud and Internet Computing (CCGrid), 2024.

[29] Y. Zhu et al., "MicroIRC: Instance-level root cause localization for microservice systems," Journal of Systems and Software, vol. 216, 112145, 2024.

[30] H. Hashemi et al., "LLM-Rubric: A multidimensional, calibrated approach to automated evaluation of natural language texts," in Proc. ACL, 2024.

[31] D. Pasquini et al., "When AIOps Become 'AI Oops': Subverting LLM-driven IT operations via telemetry manipulation," arXiv preprint arXiv:2508.06394, 2025.