AI and Data Science Journal
Online ISSN: 3078-7394
DOI: https://doi.org/10.61784/adsj4045

ADAHEAL: AN ADAPTIVE FRAMEWORK FOR FAULT DETECTION AND AUTOMATED RECOVERY IN LARGE-SCALE CLOUD-NATIVE MICROSERVICES

Xiang He[1*], Lin Zhou[2]
[1]*University of Illinois Urbana-Champaign, Urbana 61801, IL, USA.*
[2]*University of Florida, Gainesville 32611, FL, USA.*
[*]*Corresponding Author: Xiang He*

Abstract: Cloud-native microservice systems fail in ways that are frequent and hard to contain: their telemetry drifts with every deployment and traffic surge, faults propagate along call graphs and surface far from their origin, and the effectiveness of remediation actions changes as the system evolves. This paper presents AdaHeal, an adaptive closed-loop framework that unifies fault detection, root-cause localization, fault-type identification, and automated recovery. AdaHeal detects anomalies with a dual-axis masked reconstruction network (DAMR) that reconstructs every telemetry cell only from its temporal and cross-metric context and keeps alarms calibrated under benign drift through score-gated online recalibration. It localizes faults with calibrated propagation-aware ranking (CPR), which scores each service by the part of its deviation that its dependencies cannot explain under the flow-conservation rules of synchronous RPC and assigns latent evidence to stateful backends without request-level metrics. Recovery is planned by a risk-aware discounted Thompson sampling engine (RA-DTS) that selects (service, action) pairs under an attempt budget and revises its beliefs after every failed attempt. On the public SMD and PSM datasets, DAMR reaches a point-adjusted F1 of 0.893 on SMD (best of nine detectors) and the highest AUPR among the deep sequence models on PSM (0.440). On microservice benchmarks built from the Online Boutique, Sock Shop, and a 100-service Alibaba-style topology, CPR raises top-1 localization accuracy to 0.964, 0.903, and 0.922 (best baseline: 0.858, 0.694, 0.892) at 60 ms per incident for 400 services. In closed-loop experiments with shifting action effectiveness, AdaHeal shortens the mean time to recovery by 35.7% relative to a static detection–localization–runbook pipeline.
Keywords: Microservices; AIOps; Anomaly detection; Root cause analysis; Self-healing systems; Thompson sampling

1 INTRODUCTION

Cloud-native applications are built as graphs of loosely coupled microservices orchestrated by platforms such as Kubernetes. Production deployments reach thousands of services whose call graphs change with request types and releases, and benchmark suites such as DeathStarBench and Train-Ticket show how quickly a local fault spreads through such graphs [1-3]. Every minute an incident stays open translates into violated service-level objectives (SLOs), and the mean time to recovery (MTTR) is dominated by three steps that remain largely manual: noticing the problem, finding the faulty component, and choosing a mitigation that works [4].

Automating these steps is hard for three reasons. *First (C1, non-stationarity)*, the telemetry of a healthy system drifts continuously: traffic surges, rolling deployments, and autoscaling change metric levels and correlations, so detectors calibrated once either flood operators with false alarms or are tuned too insensitive to catch real faults. *Second (C2, propagation and partial observability)*, faults rarely stay local: latency adds up along synchronous calls, timeouts turn latency into errors and retries at the callers, and datastores and message brokers often expose no request-level metrics at all. Per-service anomaly scores therefore tend to blame the loudest symptom rather than the origin, while causal-discovery-based methods become slow and unstable on large graphs [5]. *Third (C3, remediation under uncertainty)*, the mapping from a diagnosis to a successful action is neither fixed nor certain: a restart cures a leaking process until a release turns the leak into a regression that only a rollback fixes, every action consumes minutes of a limited attempt budget, and acting on the wrong service disrupts healthy callers.

Existing work addresses these steps in isolation. Deep multivariate detectors such as TimesNet, iTransformer, and TimeMixer model normal behaviour but are trained once and scored with fixed statistics; root-cause methods such as BARO, CIRCA, and MicroRCA rank services from metric deviations, regression residuals, or random walks; unified frameworks such as Eadro and ART share representations between detection and localization but stop before recovery; and mitigation is usually delegated to static runbooks or to assistants that recommend steps to human engineers [6-14]. A self-healing control loop needs every stage to adapt and to hand calibrated uncertainty to the next one.

This paper presents AdaHeal, an adaptive framework that closes the monitor–analyze–plan–execute (MAPE-K) loop for cloud-native microservices [15]. Its contributions are as follows.

• DAMR, a dual-axis masked reconstruction detector that alternates attention along the time and variate axes, reconstructs each telemetry cell from its context only through complementary checkerboard masking, aggregates calibrated per-channel evidence, and recalibrates online under benign drift.

© By the Author(s) 2024, under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0).

• CPR, a propagation-aware localization method that turns the flow-conservation rules of synchronous RPC into explanation discounts, attributes latent evidence to unobservable backends, and runs in time linear in the size of the call graph.
• RA-DTS, a recovery planner that performs budget- and risk-aware Thompson sampling jointly over target services and actions, revises root-cause and fault-type beliefs after each failed attempt, and keeps learning action effectiveness from postmortems with discounting.
• An evaluation on two public cloud-monitoring datasets and on a telemetry simulator built on real microservice topologies with six fault types, benign drift, retry amplification, and partial observability, covering detection, localization, fault-type identification, and closed-loop recovery under shifting action effectiveness.

2 RELATED WORK

2.1 Multivariate Time-Series Anomaly Detection

Reconstruction- and prediction-based detectors dominate cloud monitoring. OmniAnomaly introduced the Server Machine Dataset (SMD) with a stochastic recurrent model, USAD trains adversarial autoencoders, and RANSynCoders released the eBay PSM dataset; Isolation Forest remains a strong classical reference [16-19]. General-purpose sequence backbones are now routinely used for detection: TimesNet reshapes series by dominant periods, PatchTST and DLinear revisited patching and linear models, iTransformer embeds whole variates as tokens, TimeMixer mixes multiscale decompositions, and KAN-AD fits smooth univariate functions [6-8, 20-22]. Kim *et al.* showed that point adjustment inflates F1 scores, which motivates reporting threshold-free metrics next to it [23]. DAMR borrows masked modelling from vision but uses complementary masks so that every cell is predicted only from its context, and it adds online recalibration, which the above backbones lack [24].

2.2 Root-Cause Localization in Microservices

Metric-based methods rank services by deviation magnitude, e.g., ε-Diagnosis with two-sample statistics and BARO with robust scores after Bayesian change-point detection [9, 25]. Graph-based methods propagate evidence over call graphs, e.g., MicroRCA with personalized PageRank on an attributed graph, whereas CIRCA tests regression residuals on a structural graph derived from the architecture [10-11]. A recent study found that causal-discovery-based methods degrade on large systems [5]. Multi-source approaches combine metrics, logs, and traces (Eadro, Nezha); ART unifies detection, triage, and localization in one unsupervised model; and RCACopilot uses large language models to categorize cloud incidents [12-13, 26-27]. CPR needs neither causal discovery nor training: it encodes how latency, errors, and traffic compose along synchronous calls and handles unobservable components explicitly.

2.3 Automated Mitigation and Self-Adaptation

Autonomic computing frames self-management as a MAPE-K loop [15]. FIRM mitigates SLO violations by learned resource reprovisioning, and LLM-based assistants recommend root causes and mitigation steps to on-call engineers [14, 28]. Contextual bandits such as LinUCB and linear Thompson sampling learn action values online, and discounting adapts bandit policies to abrupt changes [29-31]. RA-DTS differs in three ways: its arms are (service, action) pairs whose success depends on an uncertain diagnosis, its index accounts for a finite attempt budget and the blast radius of actions, and it revises the diagnosis itself after every failure.

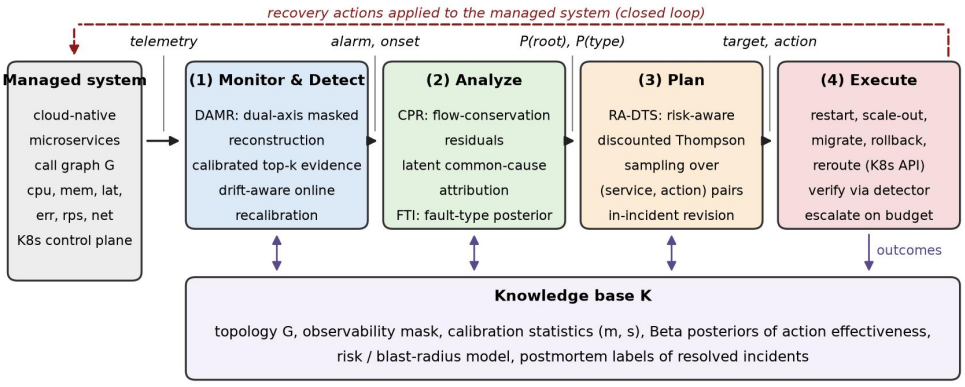


Figure 1 Overview of AdaHeal

Note: The detector (DAMR) raises calibrated alarms, the analyzer (CPR and FTI) produces posteriors over root causes and fault types, the planner (RA-DTS) selects (service, action) pairs, and the executor applies, verifies, or escalates them. The knowledge base keeps topology, calibration statistics, and action-effectiveness beliefs updated from incident outcomes.

3 PROBLEM FORMULATION AND OVERVIEW

A microservice system is a directed call graph $G = (V, E)$, where $(u, v) \in E$ means that service u synchronously calls v. Each service exposes up to M metrics; here $M = 6$: CPU usage, memory usage, response latency, error rate, successful throughput, and network traffic. An observability mask $O \in \{0, 1\}^{|V|\times M}$ marks the metrics that are actually exported—stateful backends such as caches, databases, and brokers usually lack latency and error metrics because their traffic is not instrumented at request level—so the telemetry is a stream $x_t \in \mathbb{R}^N$ with $N = \Sigma O$. An incident is a tuple (r, y, t_0): a fault of type $y \in Y$ that starts in service r at time t_0.

Given the stream, AdaHeal must (i) raise an alarm shortly after t_0 while keeping false alarms within an operator budget, (ii) rank services so that r comes first, (iii) estimate y, and (iv) choose recovery actions $(s_1, a_1), (s_2, a_2), \ldots$ that restore health with minimum expected time, where at most K automated attempts are allowed before the incident is escalated to an on-call engineer at cost H.

Figure 1 maps these tasks onto a MAPE-K loop. The monitor stage runs DAMR on streaming telemetry. On an alarm, the analyze stage runs CPR and the fault-type identifier (FTI) on a short post-alarm window and emits posteriors over root causes and fault types. The plan stage (RA-DTS) turns them into ranked (service, action) pairs; the execute stage applies them through the orchestration API, verifies recovery with the detector, and continues, finishes, or escalates. The knowledge base stores the topology, calibration statistics, action-effectiveness beliefs, and postmortem labels of resolved incidents.

4 METHODOLOGY

4.1 DAMR: Dual-Axis Masked Reconstruction

Tokenization. A window $X \in \mathbb{R}^{L\times N}$ is split per channel into $P = L/p$ non-overlapping patches of length p, and patch (c, j) is embedded as

$$h_{c,j} = W_e\, x_{c,j} + e_j^{pos} + e_c^{ch}, \tag{1}$$

where e^{pos} and e^{ch} are learned positional and channel embeddings, so that every token is identified by its time and by its variate (Figure 2).

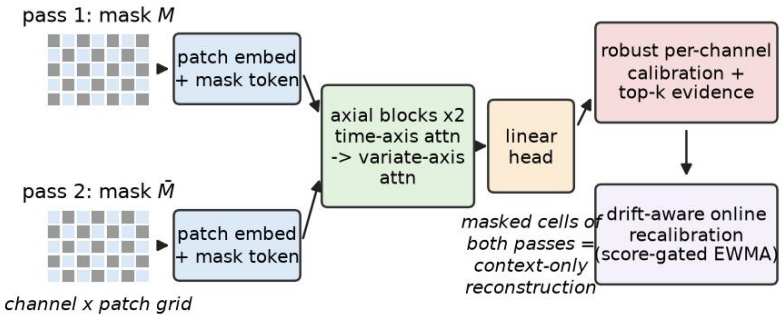


Figure 2 DAMR

Note: Two complementary checkerboard masks hide half of the channel–patch cells; axial blocks attend along time and across channels; each cell is reconstructed in the pass where it is masked, and the resulting errors are calibrated and recalibrated online.

Axial blocks. Each block applies multi-head self-attention along the time axis (over the P patches of one channel) followed by attention along the variate axis (over the N channels at one patch position), each with a pre-norm residual connection and a feed-forward layer. The time axis captures regularities such as diurnal load, the variate axis captures relations such as latency following load, and the cost per window is $O(NP^2 + PN^2)$ instead of $O((NP)^2)$ for full attention.

Masked training. Each cell is masked independently with probability $\rho = 0.5$ and its embedding is replaced by a learned mask token; a linear head maps tokens back to patches, and the loss is computed on masked cells only:

$$\mathcal{L} = \Sigma_{c,j}\, m_{c,j}\, \|\hat{x}_{c,j} - x_{c,j}\|^2 \,/\, \Sigma_{c,j}\, m_{c,j}. \tag{2}$$

Complementary inference. At test time DAMR runs two passes, with the checkerboard mask $M_{c,j} = 1[(c + j) \bmod 2 = 0]$ and with its complement $\bar{M} = 1 - M$, and keeps for every cell the reconstruction of the pass in which it was masked:

$$\hat{x} = M \odot \hat{x}^{(M)} + \bar{M} \odot \hat{x}^{(\bar{M})}. \tag{3}$$

Every cell is thus reconstructed from its temporal neighbours and from other channels at the same time, never from itself. This removes the identity shortcut that lets full-reconstruction models reproduce anomalies, while the checkerboard keeps half of each cell's temporal and cross-variate context visible.

Calibrated evidence. Raw errors $e_{t,c} = (\hat{x}_{t,c} - x_{t,c})^2$ differ by orders of magnitude across channels. DAMR standardizes them with robust validation statistics—the median m_c and the spread $s_c = \max(q_{0.99,c} - m_c, \delta)$, where δ is 10% of the median spread over channels—and averages the k largest standardized errors:

$$z_{t,c} = \mathrm{clip}((e_{t,c} - m_c)/s_c, 0, 50),\ S_t = (1/k)\, \Sigma_{c\in \text{Top-k}(z_t)}\, z_{t,c}. \tag{4}$$

k is set to 10% of the channels of one monitored entity: all channels for a server machine (SMD, PSM) and one of the six metrics of a service in microservice telemetry, where faults leave sparse footprints.

Drift-aware online recalibration. Benign changes shift the error statistics of individual channels. After scoring a window (or a streaming patch), DAMR updates the calibration with a forgetting rate gated by the current score:

$$m_c \leftarrow (1 - r)\, m_c + r \cdot \mathrm{med}(e_{\cdot,c}),\ r = \eta \text{ if } S < \tau, \text{ else } \kappa\eta, \tag{5}$$

where τ is a high percentile (99th–99.5th) of the validation scores and $\kappa = 0.1$. The spread is tracked in the same way; for short streaming patches DAMR tracks the mean absolute deviation and rescales it by the validation ratio between s_c and the mean absolute deviation, which keeps the estimate unbiased. Windows that already look anomalous are absorbed ten times more slowly, so persistent benign shifts are normalized while genuine faults are not.

4.2 CPR: Calibrated Propagation-Aware Ranking

Given an onset estimate $\hat{t}$ (the alarm time minus one patch), CPR compares a normal window N (30 min before $\hat{t}$) with an analysis window A after $\hat{t}$. For every observed metric k of service v it computes robust z-scores of the log-transformed series against the median and inter-quartile range of N, and summarizes A by upward and downward shifts $u_{v,k}$ and $d_{v,k}$ (80th percentiles of the positive and negative parts), which suppresses isolated spikes.

Flow-conservation explanation. In synchronous RPC, a caller's latency contains its callees' latency, callee latency beyond a timeout surfaces as caller errors and retries, and a callee receives fewer requests when its callers fail. CPR turns these rules into explanation discounts. With $C(v)$ and $P(v)$ the callees and callers of v,

$$U_{v,\ell} = u_{v,\ell} \cdot \exp(-E_v^{\downarrow}/\theta),\ E_v^{\downarrow} = \max_{w \in C(v)} \max(u_{w,\mathrm{lat}}, u_{w,\mathrm{err}}), \tag{6}$$

$$U_{v,\mathrm{rps}} = d_{v,\mathrm{rps}} \cdot \exp(-E_v^{\uparrow}/\theta),\ E_v^{\uparrow} = \max_{q \in P(v)} \max(d_{q,\mathrm{rps}}, u_{q,\mathrm{err}}), \tag{7}$$

for $\ell \in \{\mathrm{lat}, \mathrm{err}\}$ and $\theta = 3$. Resource metrics are normalized by the service's own traffic: $U_{v,\mathrm{cpu}}$ and $U_{v,\mathrm{net}}$ are the absolute robust z-scores of log(cpu/rps) and log(net/rps), so that load-proportional changes are not mistaken for local faults, and $U_{v,\mathrm{mem}} = \max(u_{v,\mathrm{mem}}, d_{v,\mathrm{mem}})$.

Latent common-cause attribution. For a backend b without latency and error metrics, CPR collects the unexplained evidence of its callers and assigns a share of it to b:

$$U_{b,\mathrm{lat}} = |P(b)|/(|P(b)| + 1) \cdot \mathrm{mean}_{q \in P(b)} \max(U_{q,\mathrm{lat}}, U_{q,\mathrm{err}}), \tag{8}$$

and subtracts the same amount from each caller. A backend that is the common callee of several anomalous services inherits most of their evidence, whereas a single caller and its backend share it evenly and the remaining metrics break the tie.

Ranking. The score of service v is $S_v = \max_k U_{v,k}$. CPR performs a constant amount of work per service, edge, and metric, i.e., $O((|V| + |E|) \cdot M \cdot |A|)$, and needs no training, causal discovery, or random walk.

4.3 Fault-Type Identification

FTI describes a candidate service by a 28-dimensional topology-agnostic signature: squashed upward and downward shifts of its six metrics, its unexplained evidence U_v, the early-to-late trend inside A (which separates slow leaks from abrupt failures), the strongest error and latency shifts among its callers, the strongest traffic drop among its callees, and an observability flag. A random forest with 300 trees, trained on labelled historical incidents pooled across systems, returns the posterior $p(y|v)$.

4.4 RA-DTS: Risk-Aware Discounted Thompson Sampling

Action model. The catalogue $A = \{\mathrm{restart}, \mathrm{scale\text{-}out}, \mathrm{migrate}, \mathrm{rollback}, \mathrm{reroute}\}$ has durations τ_a and disruption weights ρ_a; acting on service s affects its transitive callers, summarized by the blast radius $\beta_s = 1 + \log(1 + |\mathrm{anc}(s)|)$. The success probability $\theta_{y,a}$ of action a for fault type y is unknown and non-stationary; RA-DTS keeps posteriors $\mathrm{Beta}(\alpha_{y,a}, \beta_{y,a})$ initialized with runbook knowledge worth four observations.

Diagnosis posterior. CPR scores are mapped to a root-cause posterior $\pi_s = \mathrm{softmax}(S_s/T)$ over the three top candidates, with the temperature T fitted by maximum likelihood on historical incidents, and FTI provides $p(y|s)$ for each candidate.

Budget-aware index. At every attempt, RA-DTS draws $\tilde{\theta} \sim \mathrm{Beta}(\alpha, \beta)$ and scores each untried pair by

$$q(s, a) = \pi_s \sum_y p(y|s)\ \tilde{\theta}_{y,a}, \tag{9}$$

$$c(s, a) = \tau_d + \tau_a + \tau_v + \lambda \rho_a \beta_s + (1 - q)\, H/(K - n), \tag{10}$$

and executes the pair with the largest q/c, where τ_d and τ_v are the decision and verification times and n is the number of attempts already made. Without its last term, q/c is the classical optimal search index for unlimited attempts; the term charges each failure its share of the escalation cost that the remaining budget must absorb, which steers the planner away from cheap but unlikely actions when attempts are scarce.

In-incident revision. When (s, a) fails, RA-DTS applies Bayes' rule with the posterior means $\bar{\theta} = \alpha/(\alpha + \beta)$:

$$\pi_s \leftarrow \pi_s(1 - \bar{q}_s)/(1 - \pi_s \bar{q}_s),\ p(y|s) \propto p(y|s)(1 - \bar{\theta}_{y,a}), \tag{11}$$

where $\bar{q}_s = \sum_y p(y|s)\bar{\theta}_{y,a}$, and the other candidates are renormalized. A failed restart therefore lowers both the belief that s is the root cause and the belief in fault types that restarts usually fix.

Postmortem learning. Once an incident is resolved, its root cause and type are known. RA-DTS discounts the beliefs of the true type, $\alpha_y \leftarrow 1 + \gamma(\alpha_y - 1)$ and likewise β_y, with $\gamma = 0.97$, and adds the outcomes of the attempts made on the true root; for escalated incidents, the action applied by the engineer is recorded as a success. Discounting lets the beliefs follow changes in action effectiveness [31].

5 EXPERIMENTAL SETUP

5.1 Datasets

Table 1 summarizes the data. SMD contains 38 metrics of 28 server machines of a large Internet company; following common practice the machines are concatenated [16]. PSM contains 25 metrics of eBay application server nodes [18]. For both, the last 20% of the training split is held out for early stopping and calibration; test labels are used only for evaluation.

Table 1 Datasets Used in the Evaluation

Dataset	Source	Units	Ch.	Train	Test
SMD [16]	real servers	28 machines	38	708,405	708,420 (4.16%)
PSM [18]	real servers	pooled nodes	25	132,481	87,841 (27.76%)
MicroSim-OB	Online Boutique	11 svc / 15 edges	64	11,520	360 + 120 benign
MicroSim-SS	Sock Shop	13 svc / 14 edges	68	11,520	360 + 120 benign
MicroSim-SYN	Alibaba-style	100 svc / 170 edges	576	11,520	360 incidents

Localization and recovery require ground-truth root causes, fault types, and a controllable environment, so we built MicroSim, a telemetry simulator driven by real topologies: Online Boutique (OB, 11 services) and Sock Shop (SS, 13 services), plus SYN-100, a 100-service topology generated with the shallow, heavy-tailed fan-out reported for Alibaba call graphs [1]. Request rates propagate from callers to callees, latency follows a utilization-dependent queueing law and accumulates along synchronous calls, errors propagate upstream, and client timeouts generate retries that inflate caller latency, errors, and CPU. Telemetry is sampled every 15 s with binomial counting noise on error rates, heterogeneous noise levels, propagation lag, garbage-collection pauses, cron-like CPU spikes, and diurnal load with bursts. Stateful backends (e.g., redis-cart, the Sock Shop databases, and RabbitMQ) export no latency or error metrics. Six fault types are injected with random intensity: CPU contention, memory leak, network delay, packet loss, pod failure, and faulty release. Benign drifts—traffic surges of 30–60%, redeployments that slow a service by 15–30%, and scale-outs—occur in 30% of the fault episodes and in 120 fault-free episodes for OB and SS. Each topology has a two-day normal trace, 240 historical incidents (used to fit FTI and the posterior temperature), and 360 held-out test incidents.

5.2 Baselines and Metrics

Detection baselines are IForest, USAD, DLinear, PatchTST, TimesNet, iTransformer, TimeMixer, and KAN-AD, run through the Time-Series-Library implementations with identical windows ($L = 100$), training budget, and threshold rule: the threshold is the $(100 - r)$-th percentile of the combined training and test scores, with $r = 0.5\%$ for SMD and 1% for PSM [6-8, 17, 19-22]. We report F1 after point adjustment (PA), point-wise F1 at the same threshold, and the threshold-free AUPR and AUROC [23]. Streaming detection on MicroSim is evaluated at matched false-alarm budgets: an alarm opens after two consecutive exceeding patches and closes after three quiet ones; recall counts faults with an alarm opened within 10 min of the onset, and false alarms are alarms opened during normal operation, including benign drift.

Localization baselines are NSigma, BARO, ε-Diagnosis, MicroRCA, and CIRCA with the call graph as structural prior; all receive the same onset and windows, and we report AC@k (the fraction of incidents whose root cause is among the top k) and Avg@5 [9-11, 25]. Recovery baselines are manual handling (immediate escalation), restart-only (Kubernetes-style), a static runbook (the two best actions of the predicted type on the two top services), an ε-greedy bandit, LinUCB, and an oracle that knows the root cause, the type, and the current effectiveness [29]. We report MTTR, its 90th percentile (P90), the automated recovery rate, and the number of actions and of wrong-target actions per incident.

5.3 Implementation Details

DAMR uses patches of length $p = 10$, $d = 64$, two axial blocks with four heads, and mask ratio 0.5. All deep detectors are trained for three epochs with Adam (learning rate 10^{-3}, halved per epoch, batch 128) with early stopping on the validation loss. On MicroSim DAMR uses $L = 40$ (10 min), $p = 4$, and $d = 48$; $\eta = 0.05$ everywhere. CPR uses a 30-min normal window, a 10-min analysis window, and $\theta = 3$. The closed loop collects 3 min of post-alarm telemetry for diagnosis and uses $K = 4$, $H = 30$ min, 5-s decisions, 60-s verification, action durations of 40 s (restart), 75 s (scale-out), 110 s (migrate), 210 s (rollback), and 25 s (reroute), disruption weights 1.0, 0.1, 0.6, 0.8, and 0.3, $\lambda = 20$ s, and $\gamma = 0.97$. Incidents missed by the detector are reported by users after 15 min. Before the shift, the most effective actions are scale-out for CPU contention (0.85), restart for memory leaks (0.90), migration for network delay (0.80), rerouting for packet loss (0.80), restart for pod failures (0.85), and rollback for faulty releases (0.95). A *mild* shift makes migration, rollback, and rerouting the best responses to CPU contention, memory leaks, and network delay—actions that are still the runbook's second choice—whereas a *severe* shift additionally turns CPU contention into a release regression (rollback 0.90) and pod failures into node-pool instability (migration 0.85). Each closed-loop run replays 1,000

incidents drawn from the 720 OB and SS test incidents with the shift after incident 500; results are averaged over 20 seeds.

6 RESULTS

6.1 Detection on Real Cloud-Monitoring Data

Table 2 reports detection on the real datasets. On SMD, DAMR with online recalibration attains the best point-adjusted F1 (0.893, precision 0.947, recall 0.844), 2.6 points above the strongest baseline, TimesNet (0.867). The gain is larger on the stricter metrics: point-wise F1 rises from 0.108 to 0.148 and AUPR from 0.164 to 0.210 (+28%), i.e., DAMR ranks anomalous points higher, not only one point per segment; TimeMixer keeps the highest AUROC (0.757), which is dominated by the ranking of the abundant normal points.

PSM behaves differently. Its anomalies are long segments (27.8% of the test points), and PatchTST, DLinear, and iTransformer reach the highest PA-F1 (0.954–0.959). Their threshold-free scores are, however, among the lowest (AUPR 0.360–0.375, AUROC 0.574–0.583), the pattern that Kim *et al.* traced to point adjustment rewarding noisy scores [23]. DAMR with static calibration reaches 0.936 PA-F1 with AUPR 0.440 and AUROC 0.673, the highest AUPR among the deep sequence models, while USAD has the highest threshold-free scores on this dataset (AUPR 0.525, AUROC 0.718).

Table 2 Detection on SMD and PSM (PA: Point-Adjusted; pw: Point-Wise; PR, ROC: Area Under the Precision–Recall and ROC Curves)

	SMD				PSM			
Method	F1 PA	F1 pw	PR	ROC	F1 PA	F1 pw	PR	ROC
IForest [19]	0.469	0.104	0.133	0.656	0.844	0.048	0.441	0.666
USAD [17]	0.720	0.052	0.116	0.690	0.921	0.107	0.525	0.718
DLinear [21]	0.840	0.100	0.145	0.738	0.957	0.049	0.375	0.583
PatchTST [20]	0.837	0.104	0.150	0.744	0.959	0.041	0.360	0.574
TimesNet [6]	0.867	0.108	0.162	0.753	0.947	0.050	0.392	0.600
iTransformer [7]	0.832	0.103	0.149	0.748	0.954	0.046	0.370	0.583
TimeMixer [8]	0.761	0.105	0.164	0.757	0.845	0.039	0.385	0.603
KAN-AD [22]	0.844	0.100	0.137	0.687	0.881	0.045	0.342	0.556
DAMR, static (ours)	0.821	0.090	0.118	0.718	0.936	0.083	0.440	0.673
DAMR, online recal. (ours)	0.893	0.148	0.210	0.700	0.921	0.078	0.378	0.618

6.2 Ablation and Sensitivity of DAMR

Table 3 isolates the scoring components (rows 1–4) and the architecture (rows 5–6). Robust per-channel calibration lifts SMD F1 from 0.721 to 0.822, because SMD channels differ in scale by orders of magnitude; online recalibration adds 7.2 points and raises AUPR from 0.118 to 0.210 by absorbing the shift between the training and test periods of individual machines. On PSM, whose test period stays close to the training distribution, calibration changes little and recalibration at $\eta = 0.05$ trades 1.5 F1 points for drift robustness. Figure 3 (right) shows the sensitivity to η: SMD keeps improving up to $\eta = 0.1$–0.2 (0.920–0.922), while PSM peaks at $\eta = 0.01$–0.02 (0.942), so η can be set from the drift rate observed during validation. For the architecture (rows 5–6), removing the variate-axis attention leaves the recalibrated scores almost unchanged (SMD 0.899, PSM 0.921) but lowers the statically calibrated F1 from 0.821 to 0.792 on SMD and from 0.936 to 0.933 on PSM, i.e., cross-variate context matters most when no online adaptation is available; replacing complementary masking by plain full reconstruction gives 0.864 and 0.925 with recalibration and 0.719 and 0.932 with static calibration.

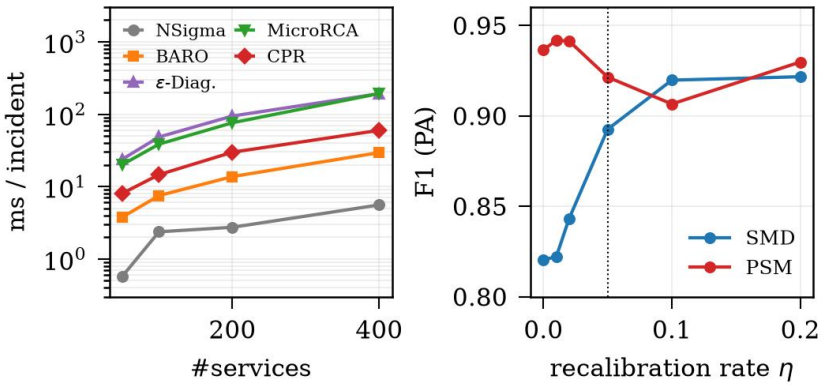


Figure 3 Left: Localization Time per Incident Versus System Size (Log Scale); Right: Sensitivity of DAMR to the Recalibration Rate η on SMD and PSM

Note: The dotted line marks the default $\eta = 0.05$.

Table 3 Ablation of DAMR Scoring and Architecture

	SMD			PSM		
Variant	F1 PA	F1 pw	AUPR	F1 PA	F1 pw	AUPR
Raw MSE, channel mean	0.721	0.051	0.119	0.943	0.070	0.441
+ robust calibration (mean)	0.822	0.104	0.127	0.943	0.080	0.447
+ top-k evidence	0.821	0.090	0.118	0.936	0.083	0.440
+ online recalibration (full)	0.893	0.148	0.210	0.921	0.078	0.378
w/o variate-axis attention	0.899	0.149	0.210	0.921	0.091	0.381
w/o complementary masking	0.864	0.146	0.211	0.925	0.076	0.381

6.3 Streaming Detection Under Benign Drift

Table 4 and Figure 4 compare detectors at matched false-alarm budgets on MicroSim. With at most 0.1 false alarms per hour—one page per ten hours of operation—DAMR with online recalibration detects 76.7% of OB faults and 67.2% of SS faults within 10 min, versus 56.9% and 54.7% for NSigma. The statically calibrated DAMR cannot reach this budget on SS without suppressing almost all alarms, because redeployments and traffic surges keep some calibrated errors elevated for the rest of the episode; recalibration absorbs these shifts within minutes. With looser budgets the static model becomes the most sensitive (0.831 on OB and 0.764 on SS at 0.2 alarms/h), so the two variants span complementary operating ranges, and at every budget in Table 4 the better of the two variants detects more faults than NSigma. At the 0.1 budget used by the closed loop, the median detection delay of the adaptive detector is 1.00 min and its alarm precision 0.914 (OB) and 0.900 (SS); scoring one hour of telemetry takes 396–485 ms on one CPU core.

Table 4 Recall Within 10 min at Matched False-Alarm Budgets (alarms/h) and Median Delay (min)

	Online Boutique				Sock Shop			
Detector	≤0.05	≤0.1	≤0.2	delay	≤0.05	≤0.1	≤0.2	delay
NSigma	0.325	0.569	0.758	1.25	–	0.547	0.675	1.25
DAMR, static calibration	–	0.681	0.831	1.25	0.000	0.000	0.764	–
DAMR, online recalibration	0.542	0.767	0.806	1.00	0.586	0.672	0.717	1.00

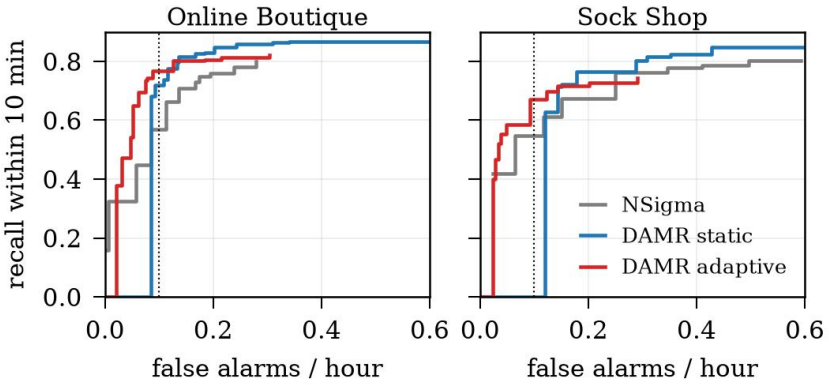


Figure 4 Recall Within 10 min Versus False Alarms per Hour on MicroSim (Upper Envelope over Thresholds)
Note: The dotted line marks the 0.1 alarms/h budget used in the closed loop.

6.4 Root-Cause Localization and Fault-Type Identification

Table 5 shows that CPR improves AC@1 to 0.964, 0.903, and 0.922 on OB, SS, and SYN-100, against 0.858, 0.694, and 0.892 for the strongest baseline (ε-Diagnosis in all three cases); Avg@5 reaches 0.991, 0.963, and 0.965. The largest gains occur where symptoms surface away from the origin: network delay on OB (0.85 vs. 0.50), faulty releases on SS (0.90 vs. 0.50), packet loss on SS (1.00 vs. 0.72), and pod failures on SS (1.00 vs. 0.82). In these cases timeouts and retries make callers look worse than the faulty service, and per-service scores follow the loudest symptom. CIRCA's residual tests struggle with error propagation, because error rates barely vary during healthy operation and the regression coefficients learned from normal data cannot extrapolate; MicroRCA's random walk spreads evidence toward callers.

Table 5 Root-Cause Localization on Held-Out Incidents

	Online Boutique			Sock Shop			SYN-100		
Method	AC@1	AC@3	Avg@5	AC@1	AC@3	Avg@5	AC@1	AC@3	Avg@5
NSigma	0.847	0.939	0.926	0.661	0.864	0.824	0.817	0.900	0.887
BARO [9]	0.828	0.933	0.921	0.669	0.861	0.819	0.775	0.886	0.870
ε-Diagnosis [25]	0.858	0.947	0.934	0.694	0.889	0.853	0.892	0.928	0.922
MicroRCA [11]	0.656	0.897	0.849	0.531	0.589	0.594	0.517	0.681	0.651
CIRCA [10]	0.653	0.875	0.843	0.656	0.856	0.823	0.711	0.825	0.808
CPR (ours)	0.964	0.997	0.991	0.903	0.986	0.963	0.922	0.972	0.965

	Online Boutique			Sock Shop			SYN-100		
w/o flow explanation	0.850	0.947	0.934	0.686	0.894	0.854	0.872	0.936	0.922
w/o latent attribution	0.914	0.981	0.967	0.722	0.922	0.883	0.892	0.936	0.926
w/o traffic normalisation	0.969	0.994	0.989	0.903	0.978	0.959	0.914	0.975	0.962

The ablation rows confirm the design. Without the flow-conservation explanation, AC@1 drops by 11.4, 21.7, and 5.0 points; without latent attribution it drops by 5.0, 18.1, and 3.1 points—most on SS, where five of thirteen services are unobservable backends. Traffic normalization changes AC@1 by less than one point on these benchmarks; it targets load-proportional resource changes, which matter mainly under traffic surges. Figure 3 (left) shows that CPR's runtime grows linearly from 8.0 ms (50 services) to 59.8 ms (400 services), about a third of ε-Diagnosis (192.5 ms) and MicroRCA (193.1 ms) at 400 services.

For fault-type identification at CPR's top-ranked service, FTI reaches an accuracy of 0.953, 0.856, and 0.933 (macro-F1 0.952, 0.847, 0.934); with the true root cause the accuracy is 0.989, 0.989, and 0.967, so most remaining type errors are inherited from localization misses rather than from ambiguous signatures.

6.5 Automated Recovery

Table 6 and Figure 5 report the closed loop. Manual handling and restart-only recovery leave MTTR at 38.40 and 30.36 min. Under the mild shift, all type-aware policies perform similarly (15.87–16.25 min) because the runbook's second action already covers the new best responses, and RA-DTS matches the best of them (15.91 min). Under the severe shift the static runbook degrades to 18.12 min overall and 20.89 min after the shift, whereas RA-DTS returns close to its pre-shift level within roughly 200 incidents (Figure 5) and attains the lowest MTTR among non-oracle policies (16.03 min; 16.30 min after the shift) with an automated recovery rate of 0.895, ahead of LinUCB (16.32) and ε-greedy (16.79). Because it chooses services and actions jointly from the diagnosis posterior, RA-DTS also issues the fewest actions on healthy services (0.40 per incident vs. 0.51 for LinUCB and 0.60 for the runbook), which reduces collateral disruption of callers by 31% relative to LinUCB.

Table 6 Closed-Loop Recovery (MTTR and P90 in Minutes; Severe-Shift Statistics on the Right)

	Mild	Severe shift					
Policy	MTTR	MTTR	post	P90	auto	act.	wrong
Manual (on-call only)	38.40	38.40	38.43	48.00	0.000	0.00	0.00
Restart-only	30.36	32.17	35.47	49.75	0.266	1.00	0.15
Static runbook	15.92	18.12	20.89	44.32	0.837	1.87	0.60
ε-greedy bandit	16.25	16.79	17.85	43.79	0.877	1.71	0.52
LinUCB [29]	15.87	16.32	17.11	39.32	0.889	1.63	0.51
RA-DTS (ours)	15.91	16.03	16.30	39.50	0.895	1.69	0.40
Oracle	12.44	12.66	13.34	22.58	0.966	1.43	0.00
w/o budget-aware index	16.00	16.40	17.40	42.46	0.874	1.94	0.46
w/o in-incident revision	15.69	16.04	16.79	41.44	0.890	1.67	0.39
w/o risk term	15.60	15.94	16.53	38.29	0.897	1.67	0.39
w/o discounting	15.51	15.95	16.67	39.39	0.895	1.67	0.39

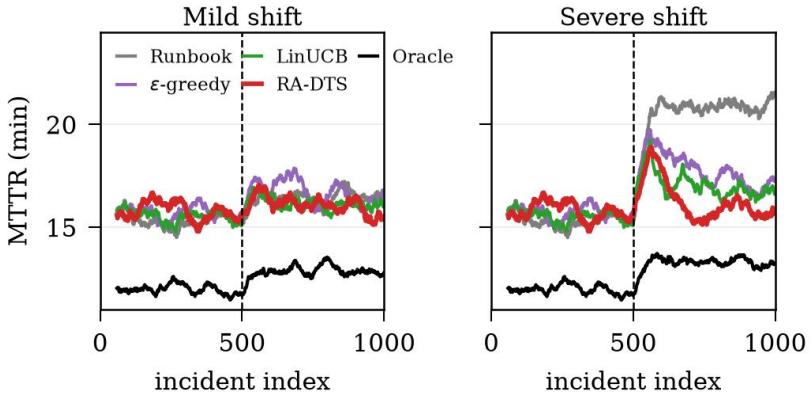


Figure 5 Rolling MTTR (Window of 60 Incidents) under the Mild and Severe Shifts of Action Effectiveness
Note: The shifts occur after incident 500 (dashed line).

The ablation rows of Table 6 identify the budget-aware index as the decisive component: without it, RA-DTS spends attempts on cheap but unlikely actions, its automated recovery rate falls to 0.874, and MTTR rises to 16.40 min under the severe shift. The in-incident revision, the risk term, and discounting each change MTTR by at most 0.10 min in this setting, where postmortem feedback reveals the engineers' fix after every escalation; the oracle (12.66 min) bounds what better diagnosis could still gain.

6.6 End-to-End Contribution of Each Stage

Table 7 replaces one stage at a time, with every detector operated at the same budget of 0.1 false alarms per hour. The full framework reaches 15.91 and 16.03 min under the mild and severe shifts, 30.0% and 35.7% below the static pipeline of NSigma, BARO, and a runbook (22.73 and 24.91 min). Replacing CPR by BARO costs 3.83 min, replacing DAMR by NSigma costs 2.39 min, and replacing RA-DTS by the runbook costs 2.09 min under the severe shift (0.01 min under the mild one). Localization accuracy therefore contributes most, earlier detection second, and adaptive planning becomes decisive once action effectiveness changes.

Table 7 End-to-End MTTR (min) and Automated Recovery Rate

Detect	Localize	Plan	Mild shift		Severe shift	
			MTTR	auto	MTTR	auto
DAMR	CPR	RA-DTS	15.91	0.893	16.03	0.895
DAMR	CPR	Runbook	15.92	0.897	18.12	0.837
DAMR	BARO	RA-DTS	19.68	0.799	19.86	0.800
DAMR	BARO	Runbook	19.68	0.797	21.91	0.736
NSigma	CPR	RA-DTS	18.29	0.894	18.42	0.894
NSigma	CPR	Runbook	18.39	0.897	20.66	0.837
NSigma	BARO	RA-DTS	22.49	0.788	22.90	0.786
NSigma	BARO	Runbook	22.73	0.780	24.91	0.720

7 DISCUSSION

Deployment. AdaHeal consumes standard signals—Prometheus-style metrics and a call graph from the service mesh or tracing system—and its actions map to orchestration primitives: rollout restarts, replica changes, cordon-and-reschedule, rollout undo, and traffic shifting. The attempt budget, the verification gate, the blast-radius term, and escalation keep operators in control of every automated decision, and postmortem labels, which incident processes already produce, are the only supervision the planner requires.

Cost. DAMR scores an hour of telemetry of a 13-service system in under half a second on one CPU core, CPR diagnoses a 400-service system in 60 ms, and RA-DTS decides in well under a millisecond; all are negligible compared with the minutes that recovery actions take.

Tuning. Two quantities adapt AdaHeal to a deployment: the evidence width k, set from the number of channels per monitored entity, and the recalibration rate η, which can be chosen from the drift observed during validation (Figure 3). The remaining parameters were fixed across all experiments.

8 CONCLUSION

AdaHeal closes the detection–diagnosis–recovery loop for cloud-native microservices with components that adapt where production systems change: DAMR keeps detection calibrated under benign drift, CPR localizes faults by explaining symptoms through the flow-conservation rules of synchronous calls and through latent evidence for unobservable backends, and RA-DTS plans recovery under an attempt budget while learning how action effectiveness evolves. Together they shorten MTTR by 30.0–35.7% relative to a static pipeline in closed-loop experiments. Future work will add logs and traces as evidence, learn action catalogues from incident tickets, and evaluate the loop on a production-scale cluster.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Luo S, Xu H, Lu C, et al. Characterizing microservice dependency and performance: Alibaba trace analysis. In: Proc. ACM Symp. Cloud Computing (SoCC), 2021: 412-426.
[2] Gan Y, Zhang Y, Cheng D, et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In: Proc. 24th Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS). Providence, RI, USA, 2019.
[3] Zhou X, Peng X, Xie T, et al. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. IEEE Trans. Softw. Eng., 2021, 47(2): 243-260.
[4] Soldani J, Brogi A. Anomaly detection and failure root cause analysis in (micro)service-based cloud applications: A survey. ACM Comput. Surv., 2022, 55(3): 1-39.
[5] Pham L, Ha H, Zhang H. Root cause analysis for microservices based on causal inference: How far are we? In: Proc. 39th IEEE/ACM Int. Conf. Automated Software Engineering (ASE), 2024: 706-715.
[6] Wu H, Hu T, Liu Y, et al. TimesNet: Temporal 2D-variation modeling for general time series analysis. In: Proc. Int. Conf. Learning Representations (ICLR), 2023.

[7] Liu Y, Hu T, Zhang H, et al. iTransformer: Inverted transformers are effective for time series forecasting. In: Proc. Int. Conf. Learning Representations (ICLR), 2024.
[8] Wang S, Wu H, Shi X, et al. TimeMixer: Decomposable multiscale mixing for time series forecasting. In: Proc. Int. Conf. Learning Representations (ICLR), 2024.
[9] Pham L, Ha H, Zhang H. BARO: Robust root cause analysis for microservices via multivariate Bayesian online change point detection. Proc. ACM Softw. Eng., 2024, 1(FSE): 2214-2237.
[10] Li M, Li Z, Yin K, et al. Causal inference-based root cause analysis for online service systems with intervention recognition. In: Proc. 28th ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD), 2022.
[11] Wu L, Tordsson J, Elmroth E, et al. MicroRCA: Root cause localization of performance issues in microservices. In: Proc. IEEE/IFIP Network Operations and Management Symp. (NOMS), 2020.
[12] Lee C, Yang T, Chen Z, et al. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In: Proc. 45th IEEE/ACM Int. Conf. Software Engineering (ICSE), 2023.
[13] Sun Y, Shi B, Mao M, et al. ART: A unified unsupervised framework for incident management in microservice systems. In: Proc. 39th IEEE/ACM Int. Conf. Automated Software Engineering (ASE), 2024.
[14] Ahmed T, Ghosh S, Bansal C, et al. Recommending root-cause and mitigation steps for cloud incidents using large language models. In: Proc. 45th IEEE/ACM Int. Conf. Software Engineering (ICSE), 2023.
[15] Kephart J O, Chess D M. The vision of autonomic computing. Computer, 2003, 36(1): 41-50.
[16] Su Y, Zhao Y, Niu C, et al. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In: Proc. 25th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining (KDD), 2019: 2828-2837.
[17] Audibert J, Michiardi P, Guyard F, et al. USAD: Unsupervised anomaly detection on multivariate time series. In: Proc. 26th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining (KDD), 2020: 3395-3404.
[18] Abdulaal A, Liu Z, Lancewicki T. Practical approach to asynchronous multivariate time series anomaly detection and localization. In: Proc. 27th ACM SIGKDD Conf. Knowledge Discovery & Data Mining (KDD), 2021: 2485-2494.
[19] Liu F T, Ting K M, Zhou Z H. Isolation forest. In: Proc. 8th IEEE Int. Conf. Data Mining (ICDM), 2008: 413-422.
[20] Nie Y, Nguyen N H, Sinthong P, et al. A time series is worth 64 words: Long-term forecasting with transformers. In: Proc. Int. Conf. Learning Representations (ICLR), 2023.
[21] Zeng A, Chen M, Zhang L, et al. Are transformers effective for time series forecasting? In: Proc. AAAI Conf. Artificial Intelligence, 2023.
[22] Zhou Q, Pei C, Sun F, et al. KAN-AD: Time series anomaly detection with Kolmogorov–Arnold networks. arXiv:2411.00278, 2024.
[23] Kim S, Choi K, Choi H S, et al. Towards a rigorous evaluation of time-series anomaly detection. In: Proc. AAAI Conf. Artificial Intelligence, 2022, 36(7): 7194-7201.
[24] He K, Chen X, Xie S, et al. Masked autoencoders are scalable vision learners. In: Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), 2022: 16000-16009.
[25] Shan H, Chen Y, Liu H, et al. ε-Diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. In: Proc. World Wide Web Conf. (WWW), 2019.
[26] Yu G, Chen P, Li Y, et al. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In: Proc. 31st ACM Joint European Software Engineering Conf. and Symp. Foundations of Software Engineering (ESEC/FSE), 2023: 553-565.
[27] Chen Y, Xie H, Ma M, et al. Automatic root cause analysis via large language models for cloud incidents. In: Proc. European Conf. Computer Systems (EuroSys). Athens, Greece, 2024.
[28] Qiu H, Banerjee S S, Jha S, et al. FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices. In: Proc. 14th USENIX Symp. Operating Systems Design and Implementation (OSDI), 2020: 805-825.
[29] Li L, Chu W, Langford J, et al. A contextual-bandit approach to personalized news article recommendation. In: Proc. 19th Int. Conf. World Wide Web (WWW), 2010: 661-670.
[30] Agrawal S, Goyal N. Thompson sampling for contextual bandits with linear payoffs. In: Proc. 30th Int. Conf. Machine Learning (ICML), PMLR, 2013, 28(3): 127-135.
[31] Garivier A, Moulines E. On upper-confidence bound policies for switching bandit problems. In: Proc. 22nd Int. Conf. Algorithmic Learning Theory (ALT), LNCS 6925, 2011: 174-188.