Bridging the Delay: Lag-Aware Spatio-Temporal Causal Inference for Microservice Root Cause Analysis

Shenglin Zhang
Nankai University
Tianjin, China

Junhua Kuang
Nankai University
Tianjin, China

Yimeng Zhang
Nankai University
Tianjin, China

Sibo Xia
Nankai University
Tianjin, China

Jintao Feng
Nankai University
Tianjin, China

Jingyu Wang
Nankai University
Tianjin, China

Wenwei Gu
Nankai University
Tianjin, China

Yongqian Sun*
Nankai University
Tianjin, China

Wei Li
Alibaba Group
Hangzhou, China

Liping Zhang
Alibaba Group
Hangzhou, China

Dan Pei
Tsinghua University &
BNRist
Beijing, China

Abstract

Timely root cause analysis (RCA) is essential for stable microservice operations, especially when failures propagate across services. In practice, upstream failures rarely trigger downstream alerts immediately; symptoms at dependent services often emerge seconds or minutes later. However, most existing RCA methods still analyze service interactions synchronously and fail to explicitly account for such multi-lag propagation, which misaligns causes and symptoms and can dilute true upstream culprits while over-ranking downstream victims. In this paper, we present *LagRCA*, a lag-aware spatio-temporal causal inference framework for microservice RCA. It models failure propagation with heterogeneous time lags, aligning upstream causes with lagged downstream symptoms. At the same time, it disentangles whether one service causally affects another from how strongly their service-level metrics co-fluctuate, preventing shared state changes from being misread as direct causal dependencies. It also produces interpretable propagation paths that help operators understand failure dynamics and act on diagnoses. We evaluate *LagRCA* on public microservice benchmarks and large-scale real incident data from Alibaba. Experimental results show that *LagRCA* consistently outperforms state-of-the-art RCA methods, achieving 88.3% top-five localization accuracy (outperforming the best baseline by 21.8 percentage points). Moreover, *LagRCA* has been deployed in Alibaba's production microservice environment, where it improves incident diagnosis efficiency and reduces manual troubleshooting effort.

Keywords

Microservice systems, Root cause analysis, Spatio-temporal modeling, Failure propagation

1 Introduction

Microservice architectures have become the de facto backbone of modern Internet and cloud services [4, 33, 47]. However, rapid growth in service count and intricate inter-service dependencies make microservice systems highly vulnerable to cascading failures, where a local failure can propagate along dependency chains and escalate into large-scale outages. A recent major outage at Microsoft [6] led to the prolonged unavailability of critical cloud services and an estimated $5.4 billion loss for affected enterprises, illustrating the magnitude of this risk. Therefore, quickly and accurately identifying root causes is essential for ensuring business continuity and maintaining user experience.

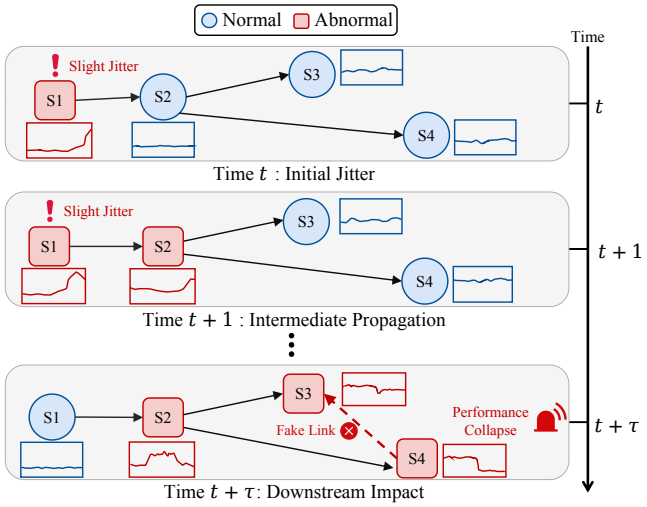

Figure 1: Lagged failure propagation in a microservice system, where each service instance is annotated with a KPI in the small line charts.

In practice, failures in microservice systems rarely manifest as instantaneous, system-wide outages; instead, they propagate gradually along service dependency chains with non-negligible and time-varying lags. As illustrated in Figure 1, each service instance is annotated with a time series of a key performance indicator (KPI), that is, a critical performance metric reflecting the service's operational state (e.g., request latency), and a slight jitter at an upstream service S1 may first affect its direct callee S2, while indirect downstream services S3 and S4 only experience a noticeable performance collapse after an additional time lag τ. However, these propagation

*Yongqian Sun is the Corresponding author.

lags are not stable across incidents: dynamic traffic patterns and runtime control policies (e.g., autoscaling and routing) continually reshape the effective runtime dependency graph and alter how long it takes for disturbances to reach downstream services. When the disturbance finally arrives, multiple downstream services may degrade within a short time window even though they do not directly call each other, and their metric curves may exhibit similar fluctuations. This ambiguity makes it difficult to determine whether such metric co-fluctuations reflect a true causal dependency or are instead driven by common upstream or environmental factors.

To handle lagged and evolving failure propagation, traditional RCA methods mainly fall into two families. Single-dimensional approaches [2, 10, 17, 18, 21, 26, 27] focus on either per-service temporal patterns or short-range spatial dependencies, and thus fail to model how upstream states influence downstream ones under time-varying lags. Joint spatio-temporal methods [11, 13, 15, 19, 23, 35, 41, 44, 45] combine temporal encoders with graphs over microservices to reason over both dimensions, but they typically decouple temporal and spatial modeling and still emphasize local interactions. In practice, such methods typically rely on two graphs extracted from observability data: physical topology graphs describing who calls whom and performance correlation graphs capturing how service metrics co-vary over time. Physical topology graphs offer interpretable execution paths but only cover observed communication links and miss hidden dependencies. In turn, performance correlation graphs are dense but noisy and often confound common-cause or synchronized fluctuations with genuine failure propagation. Some methods [10, 13, 19, 21] exploit both graphs jointly. However, this coupling is limited: the physical topology cannot recover unobserved dependencies, and correlations on this backbone are not corrected for noise or common causes. As a result, models retain the blind spots of both views and often mistake short-range co-fluctuations along the topology for genuine causal influence.

Based on the analysis above, we aim to develop a lag-aware spatio-temporal causal inference framework that can faithfully recover failure propagation in microservice systems. However, this requires addressing two key challenges.

Challenge 1: Modeling Asynchronous Propagation under Dynamic Lags. In microservice systems, disturbances at an upstream service often reach different downstream services after heterogeneous, time-varying lags driven by workload, scheduling, and resource contention. The challenge is to model cross-service causal dependencies while explicitly accounting for these variable time lags; when interactions are instead treated as synchronous or with a fixed lag, models can no longer correctly associate root causes at time t with symptoms that appear at time $t + \tau$.

Challenge 2: Bridging the Gap between Physical Topology and Service Performance Correlation. Physical topology offers sparse but structurally precise call paths, whereas performance correlation graphs encode dense co-fluctuation patterns among service metrics that are vulnerable to common causes and noise. In practice, neither view alone suffices: the physical topology misses hidden dependencies mediated by shared resources, while performance correlation graphs introduce spurious links between services that are not causally related. The challenge is to bridge this gap so that learned propagation paths respect the physical topology while discounting edges suggested only by non-causal performance correlations, yielding failure chains that better match how faults actually spread.

To address these challenges, we propose *LagRCA*, a lag-aware spatio-temporal causal inference framework for root cause analysis in microservice systems. Overall, *LagRCA* consists of several components, and two key modules are specifically designed to address the two challenges above. (1) *Multi-Lag Causal Graph Learning*. This module learns a dynamic multi-lag causal graph that combines physical topology with performance co-fluctuations among service metrics, separating relatively stable call structure from time-varying interaction strengths so that the resulting dependencies respect the physical topology while remaining robust to spurious performance co-fluctuations. (2) *Lag-Aware Spatio-Temporal Representation Learning*. This module leverages the learned multi-lag causal graphs as structural priors to drive a spatio-temporal attention mechanism that aggregates multi-lag contextual signals from causal neighbors, explicitly tying each temporal lag to its corresponding upstream service. As a result, the learned representations better capture how upstream anomalies give rise to delayed downstream deviations. Together with the learned multi-lag causal graphs, these representations provide accurate spatio-temporal cues and a solid basis for subsequent root cause localization. In summary, the main contributions of this paper are as follows:

- We formulate microservice RCA as a lag-aware spatio-temporal causal inference problem and present *LagRCA*, a framework that jointly learns multi-lag causal structures and spatio-temporal representations.
- We design a lag-aware propagation module that couples the learned multi-lag causal graph with spatio-temporal attention, explicitly modeling failure propagation along lagged edges and capturing time-varying multi-lag effects.
- We introduce a dual-branch causal graph learner that separates trace-based structural relations from metric-based interaction strengths and recombines them under joint constraints, yielding inter-service dependencies that better reconcile call topology with observed metric co-fluctuations.
- We conduct systematic experiments on public microservice benchmarks and real incident data from a large Internet enterprise, and deploy *LagRCA* in Alibaba's production microservice environment, where it improves RCA accuracy and incident handling efficiency. To ensure better reproducibility, we have made our code publicly available[1].

2 Background

2.1 Data in Microservice Systems

In microservice systems, operators rely on three main streams of observability data to diagnose incidents: logs, interaction traces, and runtime metrics [1, 24, 42]. Runtime metrics track the performance and resource usage of each instance over time, while interaction traces record how individual requests flow across service instances together with timing and status information. Logs provide rich textual context about individual events in the system. In this work, we focus on traces and metrics because they jointly provide the

[1] https://anonymous.4open.science/r/LagRCA-76FE

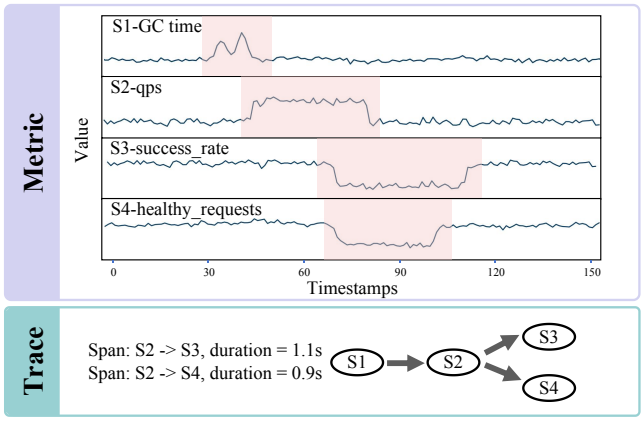

Figure 2: Metrics and traces for a microservice failure; the red-shaded regions mark time intervals affected by the incident.

structural and temporal signals required by our lag-aware spatio-temporal causal model, and we leave the systematic use of logs in the modeling pipeline for future work.

Runtime metrics are collected at fixed intervals from each instance and expose its operational state over time. Typical metrics include request latency, throughput (QPS), error rate, and resource usage such as CPU and memory. Since metrics are observed per instance and logical services are deployed as multiple runtime instances, we study root cause analysis at the granularity of individual instances. In Figure 2, for each of the four services as a time series, and the curves show that performance degradation begins at different times as the failure propagates across services. Synchronous fluctuations among these per-service series can be further aggregated into performance correlation graphs that connect services with strongly co-varying metrics, providing a metric-based view of performance dependencies.

Interaction traces complement metrics by capturing request-level execution paths: each trace consists of a sequence of timestamped spans that record which service instance calls which downstream instance. By aggregating these spans over time, operators obtain the call edges in Figure 2, forming a runtime call topology that reveals which services communicate with each other. In this work, we use traces to recover this time-varying service topology and treat it as a structural prior over candidate propagation paths. Metrics provide the instance-level metric time series along this topology, allowing us to observe when performance anomalies appear on each service and how their effects are delayed and amplified along propagation paths, as illustrated in Figure 1. Together, these two views form the data foundation for the lag-aware spatio-temporal causal modeling developed in the following sections.

2.2 Spatio-temporal Graph Modeling Paradigm

Spatio-temporal graph modeling aims to jointly capture spatial dependencies and temporal dynamics in networked systems. In microservice operations, this task is often formalized as modeling the evolution of per-service metrics on top of a service call dependency graph, so that algorithms can exploit both who depends on whom and how each node's state changes over time. In recent years, spatio-temporal graph neural networks (STGNNs), represented by STGCN [39], Graph WaveNet [34], and DCRNN [14], have become a prevalent modeling choice for such data.

Although specific architectures vary, these methods typically follow a *spatio-temporal decoupling* paradigm. Concretely, a spatial module (e.g., a graph convolution layer) first aggregates information from neighboring nodes within each time slice, and a temporal module (e.g., an RNN or temporal convolution) is then applied along the time axis to capture sequential dynamics. By alternating these spatial and temporal operations, STGNNs factorize the joint spatio-temporal distribution into two more manageable components, which greatly improves scalability and facilitates the reuse of mature graph and sequence modeling blocks.

However, this decoupling design implicitly relies on a *synchronous aggregation* assumption: a node's representation at time t is updated based mainly on the representations of its neighbors at the same time step. This assumption is reasonable in many physical-sensor and traffic applications, where interactions between nodes can be approximated as instantaneous at the chosen sampling rate. In microservice systems, as we discuss in Sec. 3, failure propagation often exhibits pronounced asynchronous and multi-lag behavior across services, which challenges the suitability of purely synchronous spatio-temporal aggregation and motivates the lag-aware causal modeling approach developed in this work.

3 Motivation

To provide insight into the complexities of root cause localization in large-scale production environments, we analyze a representative cascading failure scenario depicted in Figure 1. This case study highlights the distinct spatio-temporal evolution patterns of microservice failures and explains why existing spatio-temporal modeling paradigms often fall short in industrial settings.

3.1 How do failures propagate with multi-lag effects in microservice systems?

Temporal propagation pattern. We first revisit the production incident shown in Figure 1 from a temporal perspective. The failure is triggered by a transient performance jitter at the upstream service S_1, whose latency and error-rate metrics start to deviate at time t. Due to retry mechanisms, message queue buffering, and timeout policies that are inherent to microservice architectures, this disturbance does not immediately manifest downstream. Instead, the direct callee S_2 becomes noticeably abnormal only a few minutes later, and some further downstream services such as S_4 remain stable for an even longer period before their metrics collapse. Instead of all services becoming abnormal at roughly the same time, different instances turn abnormal one after another along the call chain [36].

As discussed in Sec. 2.2, most existing spatio-temporal graph neural networks for root cause analysis follow a slice-based synchronous aggregation paradigm. At each time step, they mainly aggregate information from a node's neighbors at the same time step, so the representation of a downstream service at time t is computed from neighbor representations that are all indexed by t. Even if one enlarges the temporal window or stacks more temporal layers, the model still lacks an explicit mechanism to connect an

Table 1: Distribution of maximum propagation lags $\Delta t_{\max}$ across incidents in D1.

$\Delta t_{\max}$ (minutes)	#Incidents	Ratio
≤ 1 (near-synchronous)	*27*	*18.49%*
2–5 (short lag)	*94*	*64.38%*
≥ 6 (long lag)	*25*	*17.12%*

upstream anomaly at time t with a downstream deviation at a later time $t+\tau$ along the call chain. In our incident, when the core downstream service S_4 raises an alert at $t+\tau$, the early anomaly of the root-cause service S_1 at time t only appears as a past fluctuation in S_1's own history and is not explicitly linked to the state of S_4. A model that only performs such synchronous aggregation therefore tends to treat the anomalies of S_1 and S_4 as separate events instead of a causally connected propagation pattern [8]. This mismatch between synchronous modeling and multi-lag propagation exemplifies Challenge 1 and calls for an explicitly lag-aware causal learner that models cross-time, cross-node dependencies.

Empirical incident statistics. To assess whether such multi-lag propagation is an isolated failure or a prevalent pattern in production, we conduct a systematic analysis of all labeled incidents in our first industrial dataset D1, which we describe in Sec. 5.1.1. In D1, all performance metrics are sampled every 1 minute, and we treat each 1-minute interval as a basic time step t. For modeling, we adopt a sliding window over recent observations and move the window forward every 1 minute. On this 1-minute grid, we mark the anomaly onset time of each instance as the first minute when its metric becomes abnormal. For each incident, we then compute $\Delta t_{\max}$ as the largest difference between the onset time of the annotated root-cause instance and that of any affected instance. Thus, $\Delta t^{\max}$ directly measures how many minutes it takes for the failure to propagate from the root cause to the last affected instance.

Table 1 summarizes the distribution of $\Delta t_{\max}$ across all incidents in D1. Only a minority of incidents exhibit near-synchronous propagation with $\Delta t_{\max} \leq 1$ minute, while a substantial portion falls into short-lag (2–5 minutes) and long-lag (at least 6 minutes) categories. These observations indicate that non-trivial and highly variable propagation lags are the norm rather than the exception in real microservice systems. RCA methods that rely on a single short window or assume near-instantaneous interactions are therefore prone to miss the true temporal alignment between root causes and delayed symptoms.

3.2 Where is the Gap between Physical Topology and Performance Correlation?

Similar patterns can mislead. Beyond lagged propagation, the incident in Figure 1 shows that performance co-fluctuations among service metrics can easily mislead microservice RCA. As the failure escalates to time $t+\tau$ in Figure 1, the intermediate service S_2 forwards anomalous traffic simultaneously to its downstream services S_3 and S_4. Although S_3 and S_4 are physically independent in the service topology, their metrics deteriorate almost in lockstep because they are driven by the same upstream disturbance from S_2. If we only inspect runtime metrics within the incident window, S_3 and S_4 appear to be strongly coupled, even though there is no direct call relationship between them.

This is exactly where topology and metric signals can become misaligned. Trace-derived topology provides a sparse but reliable structural skeleton, whereas metric co-fluctuations form a dense but noisy graph of statistical associations. Existing dynamic graph learning methods [16, 20, 37] typically collapse these two views by learning a single shared weight matrix directly from co-occurrence patterns. When they observe strong statistical correlation between S_3 and S_4, such models tend to interpret the synchronization as a strong direct dependency and introduce a spurious causal edge between them. In reality, however, the high correlation mainly reflects resonance in traffic intensity induced by S_2, rather than a genuine physical connection or latent dependency between S_3 and S_4.

Empirical correlation statistics. Our empirical analysis on D1 confirms that this topology–metric misalignment is common rather than exceptional. For each labeled incident, we compute Pearson correlations ρ between the key performance metrics of every pair of services within a time window around the failure and check whether the pair has a direct call edge in the trace-derived topology. We find that *56.12%* of strongly correlated pairs with $|\rho| > 0.7$ have no direct call relationship, while over *42.86%* of trace neighbors exhibit only weak correlation with $|\rho| < 0.3$. This indicates that graphs learned purely from metric co-fluctuations are prone to spurious edges, whereas graphs built purely from traces can easily miss important latent dependencies that only manifest at runtime.

This mechanistic conflation makes the learned graphs prone to spurious causal links and injects substantial noise into root cause inference. The resulting ambiguity exposes a clear topology–metric causality gap and motivates Challenge 2, namely, to explicitly decouple the relatively stable physical topological skeleton from time-varying runtime interaction strengths and then recombine them under a unified causal modeling framework.

4 Methodology

4.1 Overview

Figure 3 shows the overall architecture of *LagRCA*, which consists of a data preprocessing stage and three core modules arranged in an offline–online workflow.

In our pipeline, data preprocessing first converts raw observability data into unified latent health-state sequences for each microservice instance, which are then used by the subsequent modules throughout the offline–online workflow. In the offline phase, *Module 1: Dynamic Multi-Lag Causal Graph Learning* then learns multi-lag causal graphs that combine physical topology with performance co-fluctuations among service KPIs, separating relatively stable call structure from time-varying interaction strengths so that the learned dependencies respect the topology while remaining robust to spurious performance co-fluctuations. Given these graphs, *Module 2: Lag-Aware Spatio-Temporal Representation Learning* applies a lag-aware spatio-temporal attention mechanism to aggregate contextual signals from causal neighbors across different lags and trains a predictor that uses the resulting causal spatio-temporal representations to support accurate KPI prediction and

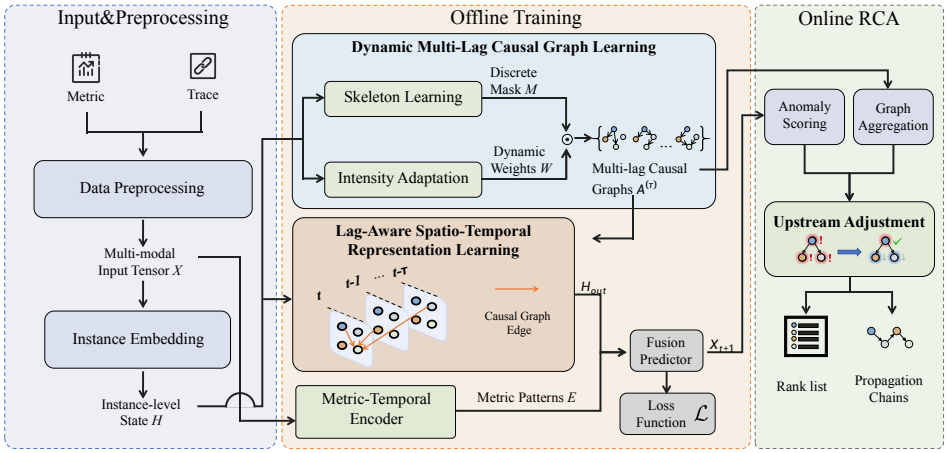

Figure 3: The framework of *LagRCA*

anomaly scoring. In the online diagnosis phase, *Module 3: Upstream-Adjusted Root Cause Inference* reuses the learned causal graphs and predictor to compute reconstruction-based anomaly scores, adjusts these scores according to how well they can be explained by upstream propagation along the causal graphs, and finally produces ranked root cause candidates together with interpretable failure propagation chains.

4.2 Data Preprocessing and Unified Representation

The data preprocessing component of *LagRCA* converts heterogeneous monitoring signals into per-instance latent time series that downstream causal and spatio-temporal modules can directly operate on.

4.2.1 Multimodal time-series construction and alignment. *LagRCA* takes microservice instances as analysis units. For each instance i, we collect infrastructure metrics (e.g., CPU, memory, network I/O) together with statistics aggregated from distributed traces (e.g., request rate, error rate, latency percentiles). All channels are resampled to a common interval, and the timeline is partitioned into windows of length L with stride s. Within each window, we compute the mean of each channel to obtain a temporally aligned tensor $\tilde{\mathbf{X}} \in \mathbb{R}^{T\times N\times M}$. We then apply per-metric z-score normalization using statistics from the offline training set and perform linear interpolation along time to fill remaining missing values, yielding a standardized sequence $\tilde{\mathbf{x}}_{t,i} \in \mathbb{R}^M$ for each instance.

4.2.2 Latent health-state projection. Running causal discovery directly on the high-dimensional metric space is inefficient and brittle, as the search space grows with $N \cdot M$ and noisy or redundant signals may overshadow true cross-instance propagation [28, 46]. Therefore, *LagRCA* adopts a lightweight learnable projection to compress observations into a compact health-state space. For each instance i and time step t, a small multi-layer perceptron (MLP) f_θ maps the standardized observation $\tilde{\mathbf{x}}_{t,i}$ to a D-dimensional latent vector $\mathbf{h}_{t,i} = f_\theta(\tilde{\mathbf{x}}_{t,i}) \in \mathbb{R}^D$, which represents the health state of instance i at time t. Stacking all instances and time steps yields a latent tensor $\mathbf{H} \in \mathbb{R}^{T\times N\times D}$ that fuses multimodal signals while reducing the feature dimension from M to D, lowering the cost of downstream causal structure learning and improving robustness to noisy or redundant metrics. Both Module 1 and Module 2 take $\mathbf{H}$ as their unified input.

4.3 Dynamic Multi-Lag Causal Graph Learning

In microservice incidents, failures propagate across services with different, time-varying lags, so a single static adjacency matrix cannot adequately describe their dependencies. To capture lagged failure propagation, we model the system's latent dependencies as a set of multi-lag causal graphs $\mathcal{A}_t = \{A_t^{(\tau)}\}_{\tau=1}^K$, where each $A_t^{(\tau)} \in \mathbb{R}^{N\times N}$ characterizes causal relationships under lag τ and $A_t^{(\tau)}[i,j]$ quantifies the direct influence of upstream node j at time t on downstream node i at time $t+\tau$.

To disentangle discrete structural changes (e.g., circuit breaking) from continuous traffic fluctuations (e.g., congestion), we decompose each lag-specific graph into a discrete skeleton and a continuous intensity:

$$A_t^{(\tau)} = M_t^{(\tau)} \odot W_t^{(\tau)}, \tag{1}$$

where $M_t \in \{0,1\}^{K\times N\times N}$ encodes which cross-instance edges are present and $W_t \in \mathbb{R}_+^{K\times N\times N}$ captures their effect strengths. Separating the relatively stable call structure in M_t from the runtime statistical dependencies in W_t allows the model to follow the physical topology while flexibly adapting to time-varying interaction patterns.

Directly parameterizing all lag-specific adjacency matrices is intractable for large N, as the number of free parameters grows as $O(N^2)$. We therefore employ a dynamic low-rank parameterization driven by the latent state tensor $\mathbf{H}$. At each time t, two lightweight hypernetworks [9] generate factor matrices $U_t, V_t \in \mathbb{R}^{N\times r}$ ($r \ll N$) that are shared across lags and used to reconstruct full matrices as $U_t V_t^\top$, reducing the complexity to $O(Nr)$. The same factors parameterize both the skeleton M_t and the intensity W_t, and we apply a Gumbel-Sigmoid relaxation [12, 22] to obtain differentiable binary skeletons during training.

To ensure that the learned structures remain physically interpretable, we further impose two structural regularizers. First, we

incorporate a topology prior derived from aggregated traces, encouraging edges that are consistent with observed call paths while still allowing the model to discover hidden dependencies when trace information is missing. Second, we apply a differentiable NOTEARS-based acyclicity constraint [46] to enforce a directed acyclic graph (DAG) over the aggregated causal graph. The concrete loss terms implementing these regularizers are given in Sec. 4.4.

4.4 Lag-Aware Spatio-Temporal Representation Learning

4.4.1 Lag-aware spatio-temporal attention. Having acquired the multi-lag causal graphs in Sec. 4.3, this module leverages them as structural priors to guide lag-aware spatio-temporal information flow and avoid mixing signals from misaligned lags. It builds on attention-based message passing mechanisms [31, 32] and injects the learned causal structures as a bias on cross-node interactions.

Let $\mathbf{H} \in \mathbb{R}^{T\times N\times D}$ denote the latent instance states from Sec. 4.2, with $h_{t,i} \in \mathbb{R}^D$ the state of instance i at time t. Given the multi-lag causal graphs $\mathcal{A}_t = \{A_t^{(\tau)}\}_{\tau=1}^K$, we design a lag-aware cross-node attention layer that aggregates information from causal upstream neighbors at different lags. To encode temporal offsets, we adopt a learnable relative lag embedding $P^{(\tau)} \in \mathbb{R}^D$ and compute query, key, and value vectors as

$$\mathbf{q}_{t,i} = W_Q h_{t,i}, \quad \mathbf{k}_{t-\tau,j} = W_K h_{t-\tau,j} + P^{(\tau)}, \quad \mathbf{v}_{t-\tau,j} = W_V h_{t-\tau,j}, \tag{2}$$

where $W_Q, W_K, W_V \in \mathbb{R}^{D\times d}$ are learnable projections. We then combine feature similarity and structural priors in the attention logits:

$$\mathrm{Logits}_{t,ij}^{(\tau)} = \frac{\mathbf{q}_{t,i}\mathbf{k}_{t-\tau,j}^\top}{\sqrt{d}} + \gamma \log\left(A_{t,ij}^{(\tau)} + \epsilon\right), \tag{3}$$

where γ controls the strength of the structural bias, ϵ is a small constant for numerical stability, and $A_{t,ij}^{(\tau)}$ is the learned multi-lag causal strength from Sec. 4.3. In practice, we apply a softmax over all candidate upstream nodes and lags to turn these logits into attention weights, and use the resulting weights to take a weighted sum of the value vectors $\mathbf{v}_{t-\tau,j}$ across services and lags. This produces a single causal context vector $\mathbf{m}_{t,i}$ for each instance and time step, which summarizes the most relevant multi-lag influences from its causal neighbors.

To mitigate over-smoothing in deep graph models [25, 29], we fuse this causal context with the original latent state via a gated residual connection. Concretely, we first compute a gate by concatenating $h_{t,i}$ and $\mathbf{m}_{t,i}$, passing them through a linear layer and a sigmoid activation, and then use the resulting gate to modulate how much of $\mathbf{m}_{t,i}$ should be injected. The gated context is added back to $h_{t,i}$ as a residual update followed by layer normalization, yielding the final state $h_{t,i}^{\text{out}}$. Intuitively, the gate lets each instance decide how much information to absorb from causal neighbors versus preserving its own dynamics, and $h_{t,i}^{\text{out}}$ serves as the causal spatio-temporal embedding for metric-level prediction and anomaly scoring.

4.4.2 Metric-level temporal encoding and prediction. For each instance i and metric m, we then take a fixed-length sliding window of its standardized time series up to time t and feed it into a shared Transformer-based temporal encoder [31] to obtain a local pattern embedding $\mathbf{e}_{t,i,m}$. Sharing this encoder across all instances and metrics allows it to capture common temporal patterns while keeping the model parameter-efficient. To predict the next-step value of metric m, we concatenate the instance-level causal state $h_{t,i}^{\text{out}}$ with the metric-level embedding and pass them through a small MLP g_ψ:

$$\hat{y}_{t+1,i,m} = g_\psi\big([\, h_{t,i}^{\text{out}} \,\|\, \mathbf{e}_{t,i,m} \,]\big), \tag{4}$$

and define the prediction loss $\mathcal{L}_{\text{task}}$ as the mean squared error between $\hat{y}_{t+1,i,m}$ and the ground truth $y_{t+1,i,m}$, averaged over normal training windows.

4.4.3 Structural regularizers and joint objective. To regularize the multi-lag causal graphs $\mathcal{A}_t$ and encourage simple, topology-consistent structures, we further impose two complementary structural constraints. A sparsity-and-topology regularizer $\mathcal{L}_{\text{sparse}}$ penalizes dense edges and edges that contradict the trace-derived physical skeleton S, while still allowing data-supported latent dependencies to emerge [30]. In addition, to obtain well-formed propagation paths for RCA, we enforce acyclicity using a NOTEARS-style DAG constraint [46] by applying a smooth acyclicity function to an aggregated adjacency matrix, yielding the loss term $\mathcal{L}_{\text{dag}}$. The overall loss of *LagRCA* combines the prediction loss with the two structural regularizers:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{sparse}\mathcal{L}_{\text{sparse}} + \lambda_{dag}\mathcal{L}_{\text{dag}}, \tag{5}$$

where $\lambda_{sparse}, \lambda_{dag} > 0$ balance prediction and structure. This joint objective encourages the multi-lag causal graphs to remain sparse and topology-aware while supporting accurate metric prediction and subsequent anomaly scoring.

4.5 Upstream-Adjusted Root Cause Inference

After training converges, *LagRCA* enters the online diagnosis phase. At each time window t, we use the prediction head in Sec. 4.4 to obtain per-metric residuals and aggregate them into an instance-level anomaly score vector $\mathbf{r}_t = [r_{t,1}, \ldots, r_{t,N}]^\top$, for example by averaging or taking the maximum over metrics. Meanwhile, the dynamic multi-lag causal graphs $\mathcal{A}_t = \{A_t^{(\tau)}\}_{\tau=1}^K$ learned in Sec. 4.3 provide the current causal dependency structure among instances. The goal of this phase is to rank instances by their likelihood of being root causes and to trace plausible propagation paths for explanation.

4.5.1 Upstream-adjusted root cause scoring. A naive strategy would directly rank instances by $r_{t,i}$ and treat those with large reconstruction errors as root causes, as is common in error-based anomaly localization methods [3, 13], but under cascading failures downstream "victim" services may exhibit even larger deviations than the true source due to amplification[5, 43]. To mitigate this victim-dominance issue, *LagRCA* adjusts anomaly scores by explicitly discounting the portion of each instance's deviation that can be explained by upstream propagation. Concretely, we first aggregate the multi-lag graphs into a single influence matrix $G_t \in \mathbb{R}^{N\times N}$ by applying an exponential decay over lags, so that shorter lags receive higher weights. Using W_t, we compute for each instance i an expected propagated error $p_{t,i}$ from its upstream neighbors by combining their anomaly scores according to the corresponding

influence weights. The final upstream-adjusted root cause score is

$$s_{t,i} = \mathrm{ReLU}\big(r_{t,i} - \beta\, p_{t,i}\big), \tag{6}$$

where $\beta \in [0, 1]$ controls how aggressively we discount propagation effects. If most of the deviation of instance i can be explained by upstream abnormalities, $p_{t,i}$ will be close to $r_{t,i}$ and the adjusted score $s_{t,i}$ becomes small, whereas a large residual suggests that i is more likely to be a root cause. We then rank instances in descending order of $s_{t,i}$ to obtain the root cause candidate list.

4.5.2 Propagation tracing for explanation. Beyond providing a ranked list of suspicious instances, *LagRCA* also traces propagation paths to assist SREs in understanding how failures spread through the system. Starting from each top-ranked root cause candidate, we perform a forward traversal on the aggregated influence matrix W_t to construct a localized propagation subgraph: at each step, we follow outgoing edges with large influence weights and significant anomaly scores, and expand until either the influence falls below a threshold or a maximum depth is reached. This greedy tracing procedure yields path-like structures such as $u \to v \to w$ that reflect likely fault propagation chains [38], and in practice we cap both the path length and fan-out to keep the generated subgraphs readable and the online overhead negligible. The resulting root cause scores and propagation paths together form the final output of *LagRCA* for online RCA.

5 Evaluation

In this section, we conduct a comprehensive evaluation of *LagRCA* to answer the subsequent research questions (RQs):

- **RQ1**: Does *LagRCA* outperform existing methods in microservice root cause localization?
- **RQ2**: How much does each key component of *LagRCA* contribute to the overall performance?
- **RQ3**: Is *LagRCA* practical for deployment in terms of computational cost and hyperparameter robustness?

5.1 Experiment Setup

5.1.1 Datasets. We evaluate *LagRCA* on two complementary datasets that together reflect both industrial-scale operational complexity and widely adopted cloud-native benchmarks.

D1 is derived from a production-grade, high-fidelity microservice system deployed on the cloud infrastructure of a top-tier commercial bank. The system consists of 46 microservice instances running across multiple virtual machines and follows realistic e-commerce traffic patterns observed in daily operations. The dataset covers five representative infrastructure failure categories frequently encountered in practice, including CPU Anomaly, Memory Exhaustion, Service Interruption, Storage Capacity, and I/O Contention. Each collected record is annotated with the corresponding root-cause instances and failure categories.

D2 is built on *Online Boutique* [7], a widely used open-source microservice benchmark deployed on a Kubernetes cluster. It contains 41 microservice instances and represents a standard cloud-native architecture. To obtain precise ground truth and controlled failure scenarios, we inject faults spanning the entire stack, comprehensively covering infrastructure-level issues (Network, Storage, Resource Stress, and Pod Lifecycle) as well as application-level anomalies (JVM Runtime and Application Logic errors).

Detailed statistics of both datasets are summarized in Table 2.

Table 2: Detailed information of datasets.

Dataset	# Instances	# Failures	# Time Span	# Types	# Records	
D1	46	146	5 days	5	trace	44,858,388
					metric	20,917,746
D2	41	161	11 days	7	trace	74,060,870
					metric	11,368,258

5.1.2 Metrics. To evaluate the performance of *LagRCA* against baseline methods, we employ *AC@k* and *Avg@k*, which are widely adopted metrics in the relevant literature [13, 23, 40]. *AC@k* denotes the probability that the true root cause is present within the *AC@k* candidates recommended by a method. Given a set of microservice failure cases A, *AC@k* is calculated as follows:

$$AC@k = \frac{1}{|A|} \sum_{a \in A} \frac{\sum_{i \le k} R_a[i] \in V_a}{min(k, |V_a|)} \tag{7}$$

where $R_a[i]$ denotes the ranked list of candidate metrics for microservice failure case a, while V_a represents the set of ground truth root causes for case a. *Avg@k* evaluates the overall performance of each method by calculating the average *AC@k*. *Avg@k* is formally defined as follows:

$$Avg@k = \frac{1}{k} \sum_{1 \le j \le k} AC@j \tag{8}$$

5.1.3 Baselines. We select six representative root cause analysis methods for microservice systems as baselines.

These baselines constitute a representative subset of the most influential recent advancements, demonstrating superior performance and high alignment with the research objectives of this study. First, we selected methods grounded in single-dimension modeling paradigms, specifically BARO [27] for temporal dimension modeling and DyCause [26] for spatial dimension modeling. Second, we selected four representative approaches based on spatiotemporal modeling paradigms, namely CauseLens [19], RCD [11], MicroCause [23], and CausalRCA [35]. For all baselines, we follow the hyperparameter settings recommended in the original papers whenever possible; for dataset-specific configurations such as window length, we tune within the ranges suggested by the authors based on validation performance to ensure a fair comparison.

5.1.4 Implementation Details. We implement our method using Python 3.9.7 with PyTorch 2.1.0, scikit-learn 1.0.2, and DGL 0.9.0. All experiments are conducted on a server equipped with two Intel Xeon Gold 5416S CPUs, 376 GB RAM and 8 × NVIDIA RTX A6000 GPUs. Each experiment is repeated five times and the results are averaged to reduce the effect of randomness.

5.2 RQ1: The Overall Performance

Table 3 shows that *LagRCA* consistently outperforms all baseline methods on both datasets across all evaluation metrics. The gains are especially pronounced on AC@1 and Avg@5, indicating that *LagRCA* can rank true root causes higher and more reliably than

Table 3: Comparison of root cause localization accuracy and time cost on D1 and D2. Train (s) denotes total training time on each dataset, Test (s) is the average inference time per incident, and "–" indicates methods without a separate training phase. Bold numbers indicate the best performance.

Method	D1						D2					
	AC@1	*AC@3*	*AC@5*	*Avg@5*	*Train (s)*	*Test (s)*	*AC@1*	*AC@3*	*AC@5*	*Avg@5*	*Train (s)*	*Test (s)*
BARO [27]	0.246	0.356	0.506	0.360	**162.83**	1.25	0.303	0.424	0.576	0.447	**210.63**	0.83
DyCause [26]	0.343	0.457	0.493	0.434	-	5.74	0.331	0.462	0.569	0.457	-	4.88
MicroCause [23]	0.080	0.104	0.110	0.104	-	15.39	0.111	0.164	0.233	0.170	-	11.52
RCD [11]	0.171	0.371	0.450	0.325	351.42	1.57	0.363	0.546	0.546	0.510	243.32	2.23
CauseLens [19]	0.445	0.514	0.637	0.529	1783.62	**0.34**	0.391	0.497	0.665	0.518	2041.73	**0.23**
CausalRCA [35]	0.245	0.313	0.571	0.405	1643.17	1.34	0.233	0.233	0.433	0.306	1578.41	0.61
LagRCA	**0.667**	**0.767**	**0.858**	**0.763**	821.87	0.93	**0.519**	**0.703**	**0.883**	**0.681**	930.63	0.88

Table 4: Performance by failure type on D1.

Failure Type	Count	*AC@1*	*AC@5*
CPU Anomaly	40	0.675	0.850
Memory Exhaustion	26	0.654	0.846
Service Interruption	12	0.500	0.833
Storage Capacity	12	0.750	0.833
I/O Contention	57	0.684	0.877

competing approaches. On the industrial dataset, we further break down performance by failure category in Table 4; due to space constraints, we report this analysis only on D1. *LagRCA* maintains high AC@5 across all five categories together with reasonably strong AC@1, suggesting that its advantages are robust across heterogeneous incident types rather than being driven by a single dominant failure class. The weaker performance of baselines mainly stems from their limitations in modeling delayed propagation. Temporal-only methods such as BARO rely on anomaly magnitude and thus favor downstream services where deviations are amplified. Spatial-focused approaches like DyCause dynamically update dependency graphs but lack explicit modeling of varying time lags, making them vulnerable to spurious dependencies caused by synchronized fluctuations. Joint spatio-temporal methods (CauseLens, RCD, MicroCause, CausalRCA) combine temporal signals with service graphs, yet their fixed-window or static causal structures implicitly assume near-synchronous interactions and struggle with asynchronous, long-range effects. In contrast, *LagRCA* aligns service states across multiple time lags and adjusts anomaly scores by discounting explainable upstream impacts, effectively suppressing propagated symptoms and consistently prioritizing true failure sources. In terms of computational cost, *LagRCA* incurs moderate offline training overheads that are comparable to or lower than several graph-based causal baselines, while maintaining sub-second inference latency per incident on both datasets. This level of efficiency is sufficient for online diagnosis in large-scale microservice environments.

5.3 RQ2: Ablation Study

To answer RQ2, we conduct an ablation study to quantify the contribution of each core component in *LagRCA*. Based on the full model, we construct four variants, each removing a single module while keeping all other settings unchanged:

Table 5: The evaluation results of ablation study

Variants	D1			D2		
	AC@1	*AC@5*	*Avg@5*	*AC@1*	*AC@5*	*Avg@5*
LagRCA	**0.667**	**0.858**	**0.763**	**0.519**	**0.883**	**0.681**
c1	0.583	0.775	0.665	0.445	0.632	0.547
c2	0.629	0.813	0.715	0.488	0.846	0.653
c3	0.583	0.808	0.692	0.469	0.790	0.623
c4	0.547	0.808	0.679	0.389	0.753	0.560

- **c1**: Removes causal structure learning and performs message passing only on the static call graph.
- **c2**: Disables the structure–intensity decoupling and instead learns a single dynamic adjacency matrix.
- **c3**: Replaces the Lag-aware spatio-temporal attention (L-STA) with a standard temporal Graph Convolutional Network (GCN).
- **c4**: Drops the upstream adjustment strategy and ranks candidates solely by raw reconstruction error.

Table 5 shows that the full *LagRCA* consistently outperforms all variants across AC@1, AC@5 and Avg@5 on both D1 and D2.

Comparing *LagRCA* with **c1** confirms that relying only on static topology fails to recover the true delayed propagation chains, and that a dynamic multi-lag causal graph is essential for capturing how service states influence each other across several time steps. The gap between **c2** and the full model indicates that explicitly decoupling edge existence (topology) from time-varying influence strength yields cleaner causal structures and more stable intensity estimates, thereby improving localization robustness under diverse failure patterns. The degradation in **c3** shows that standard temporal GCNs tend to misinterpret synchronous correlations as causality in the presence of time-lagged failure, whereas the lag-aware attention in L-STA helps *LagRCA* focus on genuine cross-temporal dependencies. Finally, the drop observed in **c4** verifies the necessity of upstream adjustment: without discounting the reconstruction error that can be explained by upstream services, downstream services that are only passively affected and exhibit amplified anomalies are frequently ranked above true root causes.

Overall, these results demonstrate that the four modules of *LagRCA* — multi-lag causal structure learning, structure–intensity

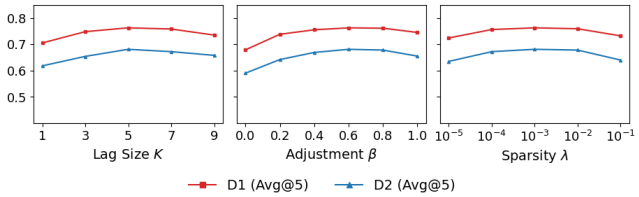

Figure 4: Parameters sensitivity on *LagRCA*.

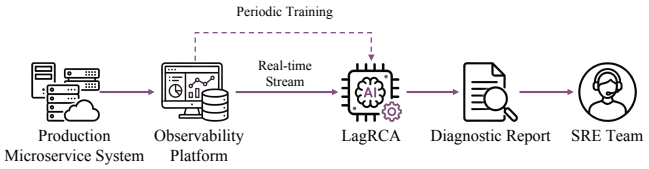

Figure 5: Deployment architecture of *LagRCA*.

decoupling, L-STA, and upstream adjustment — play complementary roles in modeling delayed cascading failures. Removing any single component inevitably weakens the root cause localization performance of *LagRCA* to varying degrees.

5.4 RQ3: Hyperparameters Sensitivity

We study the sensitivity of *LagRCA* to three key hyperparameters: the lag window size K, the upstream adjustment coefficient β, and the sparse regularization weight λ_{sparse}, as shown in Figure 4. Overall, *LagRCA* remains stable within a broad range of settings on both datasets. For the lag window K, the results exhibit an inverted U-shaped trend: a very small window cannot cover delayed propagation and leads to underfitting, while an excessively large window introduces historical noise; setting K around 5 provides a good trade-off. For the adjustment coefficient β, moderate values clearly improve AC@1 by suppressing false positives on downstream services, whereas overly small values reduce *LagRCA* to ranking by raw reconstruction error and overly large values over-correct the anomaly scores of true root-cause services. Finally, *LagRCA* is robust to the sparse regularization weight λ_{sparse} in the range of 10^{-4} to 10^{-2}: too little regularization yields dense graphs with spurious edges, and too strong regularization removes useful propagation paths. These observations suggest that operators can select K, β and λ_{sparse} from the above intervals to obtain reliable performance without heavy tuning effort.

6 Discussion

6.1 Deployment

To assess the industrial viability and business value of *LagRCA*, we integrated the framework into the observability platform of Alibaba for a three-month implementation, where the deployed *LagRCA* service shares the same GPU server configuration as our experimental setup in Section 5.1.4. As illustrated in Figure 5, the deployment follows a dual-stage workflow that combines periodic offline learning with real-time online inference. In the offline stage, the model is regularly updated using historical metrics and trace data from the past three days so that the learned multi-lag causal graphs can track recent changes in microservice dependencies. In the online stage, whenever a high-severity business alert is triggered, the system ingests the corresponding real-time metric streams and trace samples, runs *LagRCA*, and generates a diagnostic report with ranked root-cause services and candidate failure propagation chains. During the stable operation period, the system automatically processed hundreds of incident cases per month without manual intervention, demonstrating that it can handle production alert volumes under high-throughput conditions.

To understand the operational benefits, we benchmarked this automated workflow against the traditional manual troubleshooting process used by SRE teams. Prior to the introduction of *LagRCA*, resolving complex microservice cascading failures typically required an emergency squad of 3–5 senior SREs to collaboratively inspect logs and dashboards across teams, often resulting in a mean time to identify the root cause exceeding 30 minutes. With *LagRCA* acting as an intelligent diagnostic assistant, the end-to-end latency from alert to root-cause ranking is reduced to less than 0.9 seconds, and online inference uses less than 10% of one CPU socket and under 5% of one GPU, which is negligible compared with the existing monitoring and logging workloads on the same server. The visualized propagation chains produced by the system make the causal rationale behind the ranking explicit, helping engineers quickly filter out false suspects and focus on truly influential services, which in turn shortens the overall time to mitigation. Across the three-month deployment period, the system achieved an empirical AC@5 of over 80% when cross-checked against post-mortem root-cause labels.

6.2 Case Study

To demonstrate the interpretability and effectiveness of our framework, we conducted an in-depth analysis of a real-world cascading failure within an online ticketing microservice system. As illustrated in Figure 6, the incident originated from a memory leak (manifesting as frequent Full GC) in the upstream Recommendation Service C, starting at 09:07. However, due to middleware buffering and retry mechanisms, this minor perturbation did not trigger immediate alerts. Instead, it propagated with a significant lag of 3 to 5 minutes, causing a QPS surge in the downstream Product Services B and D, which eventually led to a sharp drop in the success rate of the Frontend Service A at 09:13. During the propagation process, the downstream node B exhibited more severe metric fluctuations than the root cause C.

In contrast, *LagRCA* successfully pinpointed the true origin by leveraging lag-aware causal inference and upstream impact calibration. The model accurately identified that Service C significantly affected Service B after a specific delay, which subsequently induced anomalies in Service A, thereby reconstructing the failure propagation skeleton across time windows without manual intervention. Addressing the intense metric fluctuations in the downstream node B, the algorithm calculated and subtracted the expected error component transmitted from the upstream Service C. This operation significantly suppressed the endogenous anomaly score of B. Conversely, since the anomalous fluctuations in Service C could not be explained by other nodes in the system, its characteristics as an endogenous failure source were amplified, securing its top position in the root cause list. This analysis ultimately produced a propagation

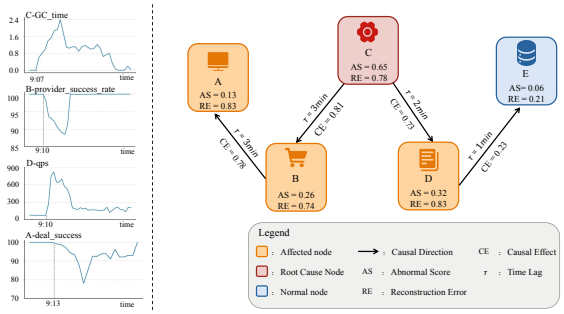

Figure 6: *LagRCA* on a production incident

path consistent with the diagnosis by SRE experts, demonstrating the robustness of the model in handling long-lag cascading failures and suppressing false alarm noise.

6.3 Lessons Learned

6.3.1 Topology-Guided Learning. In online microservice systems, service topology and effective propagation lags change frequently due to canary releases and scheduling policy updates. In our early prototype, call relationships derived from traces and configuration, together with historical lag patterns, were treated as hard constraints in the causal graph learner. However, once the topology was reconfigured or new paths emerged, the model suffered sharp drops in root cause localization performance. A more robust practice is to treat topology and lag patterns as soft priors: they define the search space and regularize multi-lag causal structure learning, while still allowing moderate deviations to capture latent dependencies and remain robust as the system evolves.

6.3.2 Result Interpretability. After deployment, we observed that returning only a single suspected root-cause service with a confidence score was insufficient to support SREs' troubleshooting decisions and to build trust in the model. SREs care more about where a failure originates, along which service paths it propagates, and over what time span it reaches the current alerting point. We therefore changed the system to output a ranked list of candidate propagation chains, accompanied by aligned views of key KPIs along each path. Internal feedback indicates that these path-based, time-aware explanations substantially increase the likelihood that the model's suggestions are adopted and make it easier to reuse the results in ticket documentation and post-mortem analysis.

6.3.3 Incident Workflow Integration. We also learned that an RCA system, even with high offline accuracy, will not be widely adopted unless it fits naturally into existing incident management workflows. On-call engineers already juggle alert dashboards, logging systems, and ticketing tools, and they are reluctant to open another standalone interface during high-pressure incidents. To reduce this friction, we integrated LagRCA's outputs into the observability platform by attaching ranked suspects and propagation chains to the original alerts and by providing short summaries that SREs can reference in incident tickets and post-mortem reports. This tight integration lowered the cognitive and operational overhead of using LagRCA and was repeatedly highlighted by SREs as a key factor for daily adoption.

7 Related Work

Single-dimensional methods : Single-dimensional methods for root cause analysis in microservice systems typically model only a single perspective, describing failure propagation from either a temporal or a topological viewpoint. BALANCE [2] and BARO [27], perform anomaly detection and root cause localization over multivariate metric streams through Bayesian modeling and change point detection. PDiagnose [10] further integrates KPIs, logs, and traces, and localizes root-cause services and metrics by combining unsupervised detection with rule-based diagnosis and voting mechanisms. Another line of work, including Microscope [17], MS-Rank [21], and DyCause [26], emphasizes topology- or causality-aware modeling. These approaches construct dependency or causal graphs from tracing data, monitoring metrics, or time-series causal analysis, and identify root causes through anomaly propagation analysis or backward tracing. Although effective in practice, single-dimensional methods treat temporal modeling and structural propagation in isolation. As a result, they are susceptible to noise and implicit dependencies, which limits the accuracy and robustness of root cause localization in complex production environments.

Joint spatio-temporal methods : Joint spatio-temporal methods characterize failure propagation across both temporal and structural dimensions by integrating temporal modeling with service topologies or causal graphs within a unified framework. ReconRCA [45], which follows an offline reconstruction–online localization paradigm and imputes missing metrics using a spatio-temporal Seq2Seq model before localizing root causes through cross-service attention mechanisms. CauseLens [19] constructs a heterogeneous causal graph from tracing data and jointly infers root causes and their propagation paths by combining structural causal modeling with counterfactual analysis. CausalRCA [35] learns a weighted causal graph from monitoring time-series data and applies graph-based ranking to identify fine-grained root-cause metrics. Although these methods formally integrate temporal information with service topology, they still treat temporal modeling and graph-based reasoning in isolation. This separation makes it difficult to consistently capture cross-service interactions and to distinguish the true causes of synchronized metric fluctuations, thereby limiting localization accuracy and robustness in complex failure scenarios.

8 Conclusion

This work presents *LagRCA*, a lag-aware spatio-temporal causal inference framework for incident diagnosis in large-scale microservice systems, operating on multimodal observability data from metrics and traces to help maintain service availability and user experience. *LagRCA* jointly learns a multi-lag causal graph and a lag-aware propagation encoder that separate trace-derived physical topology from metric-based interaction strengths, yielding propagation paths that better match real failure dynamics, improving instance-level localization accuracy, and providing path-based explanations aligned with SREs' understanding of how faults spread through microservice call chains. Experiments on public microservice benchmarks and real incident data from a large Internet enterprise, together with a three-month deployment in Alibaba's production environment, show consistent gains over strong RCA baselines and substantially reduced manual troubleshooting effort.

References

[1] Alexander Bakhtin, Jesse Nyyssölä, Yuqing Wang, Noman Ahmad, Ke Ping, Matteo Esposito, Mika Mäntylä, and Davide Taibi. 2025. LO2: Microservice API Anomaly Dataset of Logs and Metrics. In *Proceedings of the 21st International Conference on Predictive Models and Data Analytics in Software Engineering*. 1–10.

[2] Chaoyu Chen, Hang Yu, Zhichao Lei, Jianguo Li, Shaokang Ren, Tingkai Zhang, Silin Hu, Jianchao Wang, and Wenhui Shi. 2023. Balance: Bayesian linear attribution for root cause localization. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[3] Ailin Deng and Bryan Hooi. 2021. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 4027–4035.

[4] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering* (2017), 195–216.

[5] Aoyang Fang, Songhan Zhang, Yifan Yang, Haotong Wu, Junjielong Xu, Xuyang Wang, Rui Wang, Manyi Wang, Qisheng Lu, and Pinjia He. 2025. Rethinking the Evaluation of Microservice RCA with a Fault Propagation-Aware Benchmark. *arXiv preprint arXiv:2510.04711* (2025).

[6] Brian Fung. 2024. CrowdStrike outage: We finally know what caused the global tech outage - and how much it cost. https://www.cnn.com/2024/07/24/tech/crowdstrike-outage-cost-cause

[7] Google Cloud Platform. [n. d.]. Online Boutique: Sample Cloud-First Microservices Application (microservices-demo). https://github.com/GoogleCloudPlatform/microservices-demo. Accessed: 2026-01-15.

[8] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 922–929.

[9] David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106* (2016).

[10] Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li, and Jing Han. 2021. Diagnosing performance issues in microservices with heterogeneous data source. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 493–500.

[11] Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. 2022. Root cause analysis of failures in microservices through causal discovery. *Advances in Neural Information Processing Systems* 35 (2022), 31158–31170.

[12] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rkE3y85ee

[13] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2022. Causal inference-based root cause analysis for online service systems with intervention recognition. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3230–3240.

[14] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2017. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926* (2017).

[15] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 1–10.

[16] Zhuo Lin Li, Gao Wei Zhang, Jie Yu, and Ling Yu Xu. 2023. Dynamic graph structure learning for multivariate time series forecasting. *Pattern Recognition* 138 (2023), 109423.

[17] JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *International Conference on Service-Oriented Computing*. Springer, 3–20.

[18] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. 2021. Microhecl: High-efficient root cause localization in large-scale microservice systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 338–347.

[19] Qihan Liu, Pengfei Chen, Guangba Yu, Yuanhao Lai, and Xiaoyun Li. 2025. Causelens: Causality-Based Interpretable Root Cause Analysis for Microservice Systems. In *2025 IEEE/ACM 33rd International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[20] Minbo Ma, Jilin Hu, Christian S Jensen, Fei Teng, Peng Han, Zhiqiang Xu, and Tianrui Li. 2024. Learning time-aware graph structures for spatially correlated time series forecasting. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4435–4448.

[21] Meng Ma, Weilan Lin, Disheng Pan, and Ping Wang. 2019. Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications. In *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 60–67.

[22] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1jE5L5gl

[23] Yuan Meng, Shenglin Zhang, Yongqian Sun, Ruru Zhang, Zhilong Hu, Yiyin Zhang, Chenyang Jia, Zhaogang Wang, and Dan Pei. 2020. Localizing failure root causes in a microservice through causality inference. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[24] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. 2021. A survey of aiops methods for failure management. *ACM Transactions on Intelligent Systems and Technology (TIST)* 12, 6 (2021), 1–45.

[25] Kenta Oono and Taiji Suzuki. 2020. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1ldO2EFPr

[26] Yicheng Pan, Meng Ma, Xinrui Jiang, and Ping Wang. 2023. DyCause: Crowdsourcing to diagnose microservice kernel failure. *IEEE Transactions on Dependable and Secure Computing* 20, 6 (2023), 4763–4777.

[27] Luan Pham, Huong Ha, and Hongyu Zhang. 2024. Baro: Robust root cause analysis for microservices via multivariate bayesian online change point detection. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 2214–2237.

[28] J Runge, P Nowack, M Kretschmer, S Flaxman, and D Sejdinovic. 2019. Detecting causal associations in large nonlinear time series datasets, Sci. Adv., 5, eaau4996.

[29] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. 2023. A Survey on Oversmoothing in Graph Neural Networks. *SAM Research Report* 2023 (2023).

[30] Lei Tao, Xianglin Lu, Shenglin Zhang, Jiaqi Luan, Yingke Li, Mingjie Li, Zeyan Li, Qingyang Yu, Hucheng Xie, Ruijie Xu, et al. 2024. Diagnosing performance issues for large-scale microservice systems with heterogeneous graph. *IEEE Transactions on Services Computing* 17, 5 (2024), 2223–2235.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJXMpikCZ

[33] Yingying Wang, Harshavardhan Kadiyala, and Julia Rubin. 2021. Promises and challenges of microservices: an exploratory study. *Empirical Software Engineering* 26, 63 (2021). doi:10.1007/s10664-020-09910-y

[34] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 1907–1913.

[35] Ruyue Xin, Peng Chen, and Zhiming Zhao. 2023. Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications. *Journal of Systems and Software* 203 (2023), 111724.

[36] Yijun Xiong and Huajun Wang. 2024. Spatio-temporal contextual conditions causality and spread delay-aware modeling for traffic flow prediction. *IEEE Access* 12 (2024), 21250–21261.

[37] Han Yan, Dongliang Chen, Guiyuan Jiang, Bin Wang, Lei Cao, Junyu Dong, and Yanwei Yu. 2025. DGraFormer: Dynamic Graph Learning Guided Multi-Scale Transformer for Multivariate Time Series Forecasting. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*. 3516–3524.

[38] Zhenhe Yao, Changhua Pei, Wenxiao Chen, Hanzhang Wang, Liangfei Su, Huai Jiang, Zhe Xie, Xiaohui Nie, and Dan Pei. 2024. Chain-of-event: Interpretable root cause analysis for microservices through automatically learning weighted event causal graph. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 50–61.

[39] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875* (2017).

[40] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments. In *Proceedings of the Web Conference 2021*. 3087–3098.

[41] Chenxi Zhang, Zhen Dong, Xin Peng, Bicheng Zhang, and Miao Chen. 2024. Trace-based multi-dimensional root cause localization of performance issues in microservice systems. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.

[42] Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Aiwei Liu, Yong Yang, Zhonghai Wu, Xuming Hu, Philip S Yu, and Ying Li. 2024. A survey of aiops for failure management in the era of large language models. *arXiv preprint arXiv:2406.11213* (2024).

[43] Songhan Zhang, Aoyang Fang, Yifan Yang, Ruiyi Cheng, Xiaoying Tang, and Pinjia He. 2025. DynaCausal: Dynamic Causality-Aware Root Cause Analysis for Distributed Microservices. *arXiv preprint arXiv:2510.22613* (2025).

[44] Shenglin Zhang, Sibo Xia, Wenzhao Fan, Binpeng Shi, Xiao Xiong, Zhenyu Zhong, Minghua Ma, Yongqian Sun, and Dan Pei. 2025. Failure diagnosis in microservice systems: A comprehensive survey and analysis. *ACM Transactions on Software Engineering and Methodology* 35, 1 (2025), 1–55.

[45] Zekun Zhang, Jian Wang, Bing Li, and Liuxiaoxiao Zhang. 2025. ReconRCA: Root Cause Analysis in Microservices with Incomplete Metrics. In *2025 IEEE*

International Conference on Web Services (ICWS). IEEE, 116–126.

[46] Xun Zheng, Bryon Aragam, Pradeep K Ravikumar, and Eric P Xing. 2018. Dags with no tears: Continuous optimization for structure learning. *Advances in neural information processing systems* 31 (2018).

[47] Xin Zhou, Shanshan Li, Lingli Cao, He Zhang, Zijia Jia, Chenxing Zhong, Zhihao Shan, and Muhammad Ali Babar. 2023. Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry. *Journal of Systems and Software* 195 (2023), 111521.