ROOT CAUSE OR DELAYED SYMPTOM? LAG-CALIBRATED COUNTERFACTUAL INFERENCE FOR MICROSERVICE FAULT LOCALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In microservice systems, fault propagation often involves non-negligible temporal lags, and faults in upstream services may manifest as anomalies in downstream services after several time steps. Therefore, relying solely on anomaly signals observed at the current timestamp often makes it difficult to distinguish the true upstream root cause from delayed downstream responses. Most existing approaches rely on anomaly intensity or statistical correlations at individual timestamps, without explicitly modeling propagation delays or temporal causal relationships. As a result, they are prone to errors when anomaly signals are not synchronized across services. To address this problem, we propose RippleRCA, a lag-calibrated causal inference framework for root cause analysis in microservice systems. Specifically, RippleRCA begins by capturing global cross-service propagation dependencies through edge-level and node-level lag-aware causal priors constructed with PCMCI+ under service call topology constraints. It then performs coarse-grained candidate recall from both the structural anomaly and propagation response perspectives. This is achieved through a dual-channel mechanism based on temporal-difference features. Furthermore, RippleRCA performs lag-calibrated temporal alignment and estimates counterfactual effects via recovery interventions, enabling fine-grained causal root cause ranking. Extensive experiments on two public datasets, AIOps2025 and RCABench, demonstrate the effectiveness of RippleRCA.

1 INTRODUCTION

Microservice architecture has become a dominant paradigm for building large-scale cloud applications. It offers modular design, scalability, and independent deployment of services (Bierzynski et al., 2019; Dragoni et al., 2017; Luo et al., 2021; Wang et al., 2018). In a typical microservices system, hundreds of loosely coupled services are interconnected through complex dependencies. While such a structure improves flexibility and scalability, it also complicates fault propagation paths (Addeen, 2019). Therefore, root cause analysis (RCA) in microservice systems goes beyond identifying where anomalies occur. It also involves modeling how failures propagate along service dependencies and capturing their temporal causal relationships.

In microservice environments, an important but often overlooked fact is that fault propagation usually involves significant time delays (Chen et al., 2023; Brandón et al., 2020). Anomalies rarely propagate from upstream services to downstream services instantly. Instead, factors such as network delay, request queuing, and retry mechanisms often cause the impact of an upstream fault to appear in downstream observations only after several time steps (Li et al., 2021; Mendonca et al., 2020). This means that the true cause of a downstream anomaly is often not a nearby anomaly at the current time step, but a state change in an upstream service at an earlier time step (Lee et al., 2024; Li et al., 2022). Ignoring such a propagation delay can lead to a temporal misalignment of causal relationships over time and confuse true root causes with delayed response nodes.

To verify that temporal lag is an inherent property rather than a modeling assumption, an empirical analysis is conducted on two representative RCA datasets, AIOps2025 (Dataset, 2025) and RCABench (Fang et al., 2025). Clear non-zero lag patterns are observed in both datasets. Although

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

AIOps2025 is dominated by near-synchronous responses, about 47% of service pairs still show non-zero lags. RCABench exhibits a more dispersed lag distribution, with about 65% of service pairs presenting non-zero lags, indicating stronger delayed anomaly propagation. Detailed results are provided in Appendix A. These findings highlight the need to explicitly model temporal lag in microservice root cause analysis.

This temporal misalignment is especially prominent in RCA methods based on causal inference and counterfactual analysis. Many existing methods assume that an upstream intervention affects downstream responses immediately within the same observation window when estimating causal effects (Pham et al., 2024b). However, in real microservice systems, intervention effects often appear only after a propagation delay (Meng et al., 2020; Xiong & Wang, 2024). Estimating causal effects within an uncalibrated synchronous window can underestimate the impact of the true upstream root cause and overestimate the contribution of downstream response nodes. This introduces a systematic bias into counterfactual attribution. Therefore, RCA needs a causal attribution mechanism that aligns interventions and responses according to propagation delay, so that it can distinguish fault sources from propagated responses more accurately.

Besides lag modeling, RCA also faces a granularity gap between service-level and instance-level diagnosis. Service-level analysis can quickly narrow the suspicious scope from a global dependency view. However, it is often difficult to further pinpoint specific faulty instances (Wu et al., 2020; Chen et al., 2014; Wang et al., 2018). Instance-level analysis is more fine-grained, but it faces a larger search space and stronger noise interference. Existing methods often treat these two levels separately. As a result, service-level results cannot effectively guide instance-level localization (Zhu et al., 2024). Therefore, microservice RCA requires a two-stage diagnosis mechanism that connects service-level inference with instance-level attribution and allows the two levels to constrain each other (Bento et al., 2021; Soldani et al., 2023).

Based on these observations, we propose **RippleRCA**, a two-stage lag-calibrated causal inference framework for microservice RCA. RippleRCA first constructs topology-constrained lag priors using PCMCI+ to capture delayed dependencies along service invocation paths. Guided by these priors, Stage 1 combines structural anomaly and propagation response evidence for coarse-grained candidate recall. Stage 2 aligns the candidates with their dominant lags and performs fine-grained counterfactual ranking using a learned lagged SCM. Experiments on AIOps2025 and RCABench demonstrate the effectiveness of RippleRCA. On AIOps2025, it achieves a mean AC@1 of 0.313 and a mean MRR of 0.476. On RCABench, it achieves a mean AC@1 of 0.370 and a mean MRR of 0.513. Ablation studies further validate the contributions of lag priors, lag alignment, the propagation response channel, and the two-stage design.

The main contributions of this work are summarized as follows:

1) We propose RippleRCA, a two-stage lag-calibrated causal inference framework for microservice root cause analysis. The framework jointly models temporal dependencies in fault propagation, service topology constraints, and cross-granularity diagnosis requirements. Our experiments show that this design improves the accuracy of root cause analysis in complex microservice environments.

2) In RippleRCA, we design a topology-constrained lag prior and a two-stage diagnosis mechanism. The former jointly models service call topology and temporal dependencies to explicitly characterize lag dependency patterns in cross-service propagation. The latter combines service-level coarse-grained candidate recall with instance-level fine-grained localization to progressively narrow the root cause search space.

3) We design lag-calibrated counterfactual inference, which calibrates the temporal alignment between interventions and responses according to propagation delay. This helps distinguish true root cause nodes from downstream nodes that exhibit delayed responses to the propagated faults.

2 Related Work

RCA for microservices. Existing RCA methods for microservices primarily focus on service dependency models and fault propagation localization. RCD (Ikram et al., 2022) improves scalability in complex microservice environments through two-stage local causal learning. Eadro (Lee et al., 2023) and M-RCA (Wang et al., 2024) further integrate multimodal monitoring data to enhance

diagnostic performance. DynaCausal (Zhang et al., 2025) models evolving service dependencies via dynamic causal representation learning, thereby improving the robustness of root cause localization. However, these approaches mainly emphasize dependency modeling or dynamic representation learning, and they typically do not explicitly capture the lagged propagation patterns of faults along service chains. In contrast, this work models cross-service lagged propagation structures and uses temporal delay relationships as a discriminative signal, thereby improving the accuracy of root cause localization.

Counterfactual RCA. Counterfactual reasoning provides a more interpretable causal approach to root cause analysis. CauseLens (Liu et al., 2025) performs root cause localization by learning structural causal models and combining counterfactual analysis with reconstruction errors, while also revealing fault propagation paths. CausalRCA (Xin et al., 2023) learns a structural causal model. It then uses intervention-based causal effect estimation to assess how different components contribute to system anomalies and identify the root cause. However, existing counterfactual RCA methods mainly rely on static causal structures and lack explicit modeling of cross-service delayed propagation mechanisms. In contrast, this work incorporates temporal alignment into the counterfactual reasoning process, enabling root cause refinement by jointly considering temporal propagation structure.

Temporal Causal Discovery. Temporal causal discovery exploits temporal order to infer lagged dependencies in dynamic systems. Granger causality and its extensions (Tank et al., 2021; Barnett & Seth, 2015) capture lagged causal directions, while RUN (Lin et al., 2024) applies neural Granger causality to root cause analysis. However, such methods can be affected by spurious associations in high-dimensional time series with strong autocorrelation or instantaneous dependencies. LagRCA (Zhang et al., 2026) explicitly models heterogeneous propagation delays through lag-aware spatio-temporal causal inference for microservice RCA. RippleRCA instead employs topology-constrained PCMCI+ (Runge, 2020) to construct statistically supported lag priors while suppressing unrelated causal associations through conditional independence tests (Runge et al., 2017; Székely et al., 2007). These priors guide candidate localization and lag-calibrated counterfactual inference in a learned SCM to distinguish root causes from propagated symptoms.

3 METHOD

This section presents the design of RippleRCA, a lag-aware two-stage causal root cause analysis framework. The overall architecture of RippleRCA is illustrated in Figure 1. The framework consists of two stages. In the first stage, lagged causal skeleton discovery and coarse-grained localization are performed, where temporal causal discovery is used to identify potential lagged propagation relationships and to filter candidate services at a coarse level. In the second stage, lag-calibrated counterfactual refinement is conducted to achieve fine-grained localization at the instance level.

3.1 PROBLEM STATEMENT

Consider a microservice system consisting of N services, where each service is deployed as multiple pod instances. Let the multivariate time series of the system be denoted as $X = \{X(t)\}_{t=1}^{T} \in \mathbb{R}^{T\times N\times F}$, where T, N, and F represent the number of time steps, the number of services, and the feature dimension of each service, respectively (Jiang et al., 2023; Shan et al., 2019; Sun et al., 2024; Yu et al., 2023b; Zhang et al., 2023). The service call topology is represented by a directed graph $G_{\text{trace}} = (V, E_{\text{trace}})$, where the node set V corresponds to all services in the system, and the directed edge set E_{trace} encodes the invocation dependencies between services.

3.2 STAGE 1: CAUSAL SKELETON DISCOVERY AND COARSE-GRAINED CANDIDATE RECALL

Topology-constrained Lag Prior Construction. To characterize cross-service lag propagation in microservice systems, a topology-constrained lag prior is constructed based on PCMCI+. Unlike conventional methods that search for causal relationships over the full variable space without structural restrictions (Constantinou et al., 2023), G_{trace} is introduced as a hard constraint, and PCMCI+ is executed only on node pairs $(u, v) \in E_{\text{trace}}$ with invocation dependencies. This design structurally removes spurious causal associations across unrelated service paths while significantly reducing computational complexity.

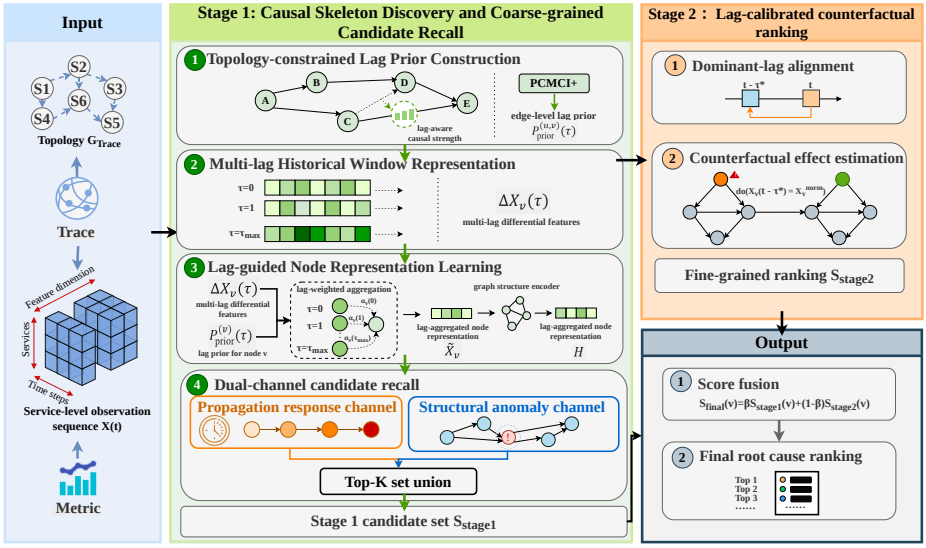


Figure 1: The two-stage RippleRCA framework: coarse-grained candidate recall based on lag-aware causal priors and fine-grained root cause ranking based on lag-calibrated counterfactual inference.

For each candidate causal edge, an edge-level lag prior $P_{\text{prior}}^{(u,v)}(\tau)$ is derived to characterize the relative propagation strength from upstream service u to downstream service v at different lags. By normalizing lag-specific causal strengths estimated by PCMCI+, this prior captures a distribution over the lag space rather than a single fixed delay, thereby preserving propagation uncertainty for subsequent lag-aware representation learning and candidate diagnosis. Details are provided in Appendix B.

Multi-lag Historical Window Representation. Relying only on features at the current time step is insufficient to characterize the fault propagation process. To address this issue, this work constructs a multi-lag differential feature sequence for each node v to preserve its state variation within the historical window. The historical window difference representation measures the deviation from the Stage 1 sample mean at each lag position:

$$\Delta X_v(\tau) = X_v(t-\tau) - \mu_v \quad \tau \in [0, \tau_{\max}] \tag{1}$$

Here, μ_v denotes the mean feature vector of node v, computed across the input samples used in Stage 1. This representation measures the deviation of each historical observation from μ_v and provides the basis for subsequent lag weight learning and node representation construction.

Lag-guided Node Representation Learning. For lag-guided node representation learning, the edge-level priors are aggregated over the incoming neighbors of each node:

$$P_{\text{prior}}^{(v)}(\tau) = \sum_{u \in N_{\text{in}}(v)} w_{(u,v)} \cdot P_{\text{prior}}^{(u,v)}(\tau) \tag{2}$$

where $N_{\text{in}}(v)$ denotes the set of upstream neighbors of node v, and $w_{(u,v)}$ is the normalized weight.

The node-level lag prior $P_{\text{prior}}^{(v)}(\tau)$ and historical window difference representation $\Delta X_v(\tau)$ are then combined to construct the lag-aggregated node representation. Specifically, the response score $r_v(\tau)$ is first computed from $\Delta X_v(\tau)$ to measure the anomaly response intensity at each lag position. It is then fused with the node-level lag prior to obtain the lag weights:

$$\alpha_v(\tau) = \frac{\exp\Big(r_v(\tau) + \log\Big(P_{\text{prior}}^{(v)}(\tau) + \epsilon\Big)\Big)}{\sum_{\tau'} \exp\Big(r_v(\tau') + \log\Big(P_{\text{prior}}^{(v)}(\tau') + \epsilon\Big)\Big)} \tag{3}$$

where ϵ is a smoothing term introduced to avoid numerical instability. Based on these weights, the historical differential features are aggregated to obtain the lag-aware node representation:

$$\tilde{X}_v = \sum_{\tau=0}^{\tau_{\max}} \alpha_v(\tau) \cdot \Delta X_v(\tau) \tag{4}$$

The lag with the largest weight is then defined as the dominant propagation lag of the node,$\tau_v^* = \arg\max_\tau \alpha_v(\tau)$. Finally, let $\tilde{X} = \{\tilde{X}_v\}_{v=1}^N$ denote the lag-aggregated representations of all nodes. Combined with the service call topology G_{trace}, the graph structure encoder $\Phi_G(\cdot)$ integrates node states with neighborhood structural information (Gan et al., 2023) and produces the graph-aware node representation matrix $H = \Phi_G(G_{\text{trace}}, \tilde{X})$, where h_v denotes the final representation of node v in H. This representation jointly incorporates call topology, historical propagation responses, and lagged causal priors, thereby capturing the temporal characteristics of anomaly propagation along service dependencies more accurately.

Dual-channel Candidate Recall for Coarse-grained Diagnosis. Based on the graph-aware node representations h_v, candidate recall is formulated as a cross-sample statistical estimation problem for identifying high-confidence root cause candidates. For each sample i, a structural anomaly channel score is defined from the perspectives of structural representation and observed anomalies:

$$S_A^{(i)}(v) = \frac{1}{2}\,\text{norm}\Big(S_{\text{gnn}}^{(i)}(v)\Big) + \frac{1}{2}\,\text{norm}\Big(S_{\text{anom}}^{(i)}(v)\Big)\,. \tag{5}$$

Here, $S_{\text{gnn}}^{(i)}(v) = \left\|h_v^{(i)}\right\|$ and $S_{\text{anom}}^{(i)}(v) = \left\|X_v^{(i)}(t) - \mu_v\right\|$ denote the graph representation magnitude and the magnitude of the current observation's deviation from the mean feature vector μ_v, respectively.

Second, to characterize abnormal temporal responses induced by fault propagation, a propagation response channel score is defined as

$$S_B^{(i)}(v) = \phi\Big(\Big\{\Delta X_v^{(i)}(\tau)\Big\}_{\tau=0}^{\tau_{\max}}\Big)\,, \tag{6}$$

where $\phi(\cdot)$ computes the propagation response score from the historical difference sequence, measuring the strength of propagation-related anomalies of node v across different lag positions.

For each sample, Top-K candidate nodes are retained from the structural anomaly channel and the propagation response channel, respectively. The two candidate lists are then merged by taking their union, with duplicated nodes retained only once. No additional reward is assigned to nodes retrieved by both channels.

Across multiple fault samples, candidate nodes are further aggregated statistically. Their occurrence frequency $p_{\text{vote}}(v)$, average local score $\bar{S}(v)$, and peak local score $S_{\max}(v)$ are jointly used to assess their global stability as root cause candidates. Based on these statistics, the Stage 1 global score is defined as

$$S_{\text{stage1}}(v) = \frac{1}{3}\left(p_{\text{vote}}(v) + \text{norm}\big(\bar{S}(v)\big) + \text{norm}(S_{\max}(v))\right). \tag{7}$$

The final candidate set is obtained as $\mathcal{S}_{\text{cand}} = \text{TopK}(S_{\text{stage1}})$. This stage effectively reduces the search space and improves the coverage of true root causes by combining complementary local retrieval signals with cross-sample statistical aggregation. Detailed definitions of sample-level candidate construction and statistical aggregation are provided in Appendix C.

3.3 Stage 2: Lag-calibrated Counterfactual Inference

Although Stage 1 substantially narrows the search space, its coarse-grained localization scores may not reliably distinguish an upstream root cause from downstream affected nodes when both exhibit strong anomalies. Stage 2 therefore first aligns each candidate with its dominant propagation lag and then performs counterfactual inference using a learned lagged structural causal model (SCM). Specifically, RippleRCA follows the abduction-action-prediction procedure to estimate how restoring a candidate to its normal state changes its downstream responses.

Dominant-Lag-Based Feature Alignment: For each candidate node $v \in \mathcal{S}_{\text{cand}}$, we use the dominant lag τ_v^* obtained in Stage 1 to identify the historical state that is more likely to correspond to the onset of fault propagation. Rather than directly using the feature at the current time t, we construct the lag-aligned difference representation

$$\Delta X_v(\tau_v^*) = X_v(t - \tau_v^*) - \mu_v \tag{8}$$

This representation captures the deviation of a node from its normal state at the dominant propagation lag, and compared with the raw feature at the current time step, it better aligns with the true temporal and causal location of the fault.

Lag-Calibrated Counterfactual Effect Inference: Lag-aligned anomaly magnitude alone cannot determine whether a candidate is the source of an anomaly or a downstream response node (Liu et al., 2021). We therefore learn a lagged SCM from normal operating windows:

$$X_v(t) = f_v(X_v(t-1), \{X_u(t - \tau_{uv}) \mid u \in \mathrm{Pa}(v)\}) + U_v(t), \tag{9}$$

where f_v is the node-specific structural mechanism, τ_{uv} is the propagation lag associated with edge $u \to v$, $\mathrm{Pa}(v)$ is the parent set of v, and $U_v(t)$ denotes its exogenous disturbance.

For each faulty sample, RippleRCA first abducts the sample-specific exogenous residuals $\widehat{U}$ from the observed trajectory. It then performs a normal-state restoration intervention at the lag-aligned time $t_v = t - \tau_v^*$ (Pearl, 2009):

$$\mathrm{do}(X_v(t_v) = \mu_v^{\text{norm}}\,). \tag{10}$$

During prediction, the factual and counterfactual worlds share the same abducted residuals $\widehat{U}$, while the downstream states in the counterfactual world are regenerated by solving the structural equations. This design controls for sample-specific disturbances unrelated to the candidate intervention.

Let $\mathrm{De}(v)$ denote the causal descendants of v in the directed SCM graph, $\mathcal{T}_r$ the response-time range, and d the feature dimension. We define the counterfactual effect as

$$S_{\text{cf}}(v) = \sqrt{\frac{\displaystyle\sum_{s \in \mathcal{T}_r} \sum_{j \in \mathrm{De}(v)} \left\| X_j^{\mathrm{F}}(s) - X_j^{\mathrm{CF}(v)}(s) \right\|_2^2}{|\mathcal{T}_r|\,|\,\mathrm{De}(v)|\,d}}, \tag{11}$$

where X^{F} and $X^{\mathrm{CF}(v)}$ denote the factual and counterfactual trajectories, respectively. A larger $S_{\text{cf}}(v)$ provides stronger model-based evidence that restoring v substantially changes its downstream responses. Details of exogenous-residual abduction, factual replay, fixed-point solution for zero-lag relations, and counterfactual validity checks are provided in Appendix D.

Fine-Grained Ranking and Final Score Fusion: Stage 2 combines the lag-aligned local anomaly score with the SCM-based counterfactual effect. We define the local anomaly score as

$$S_{\text{loc}}(v) = \sigma(\|\Delta X_v(\tau_v^*)\|_2)\, \|\Delta X_v(\tau_v^*)\|_2\,, \tag{12}$$

where $\sigma(\cdot)$ is the sigmoid function. The Stage 2 score is then computed as

$$S_{\text{stage2}}(v) = \lambda_1\, \mathrm{Norm}(S_{\text{loc}}(v)) + \lambda_2\, \mathrm{Norm}(\log(1 + S_{\text{cf}}(v)))\,, \tag{13}$$

where $\mathrm{Norm}(\cdot)$ denotes min–max normalization over the candidate set, and λ_1 and λ_2 control the contributions of local anomaly evidence and counterfactual evidence, respectively.

Finally, we combine the global localization evidence from Stage 1 with the fine-grained evidence from Stage 2:

$$S_{\text{final}}(v) = \beta\, \mathrm{Norm}(S_{\text{stage1}}(v)) + (1 - \beta)\, \mathrm{Norm}(S_{\text{stage2}}(v))\,, \tag{14}$$

where $\beta \in [0, 1]$ controls the relative contributions of the two stages. Candidates are ranked in descending order of $S_{\text{final}}(v)$. The directional source terms, validity controls, and implementation-specific SCM fusion configurations are detailed in Appendix D.

4 EXPERIMENTS

To systematically evaluate the proposed RippleRCA framework, we design experiments to answer the following research questions:

RQ1: Overall effectiveness. How does RippleRCA perform in root cause analysis for microservice systems compared with state-of-the-art RCA methods?

RQ2: Ablation study. How do lagged causal priors, lag alignment, the propagation response channel, and two-stage fusion contribute to the effectiveness of RippleRCA?

RQ3: Robustness and sensitivity. How robust is RippleRCA to incomplete or noisy service topologies and variations in the maximum lag window, candidate size, and relative stage weighting?

4.1 EXPERIMENTAL SETUP

Datasets and Baselines RippleRCA is evaluated on two representative public datasets for microservice root cause analysis, namely AIOps2025 and RCABench, which represent a real-world complex setting and a controlled benchmark setting, respectively. AIOps2025 (Dataset, 2025) is collected from a real production environment and contains 400 fault cases across 5 services, covering 6 resource-related fault types. RCABench (Fang et al., 2025), built upon the classical Train-Ticket system (Ticket, 2024), contains 1,430 valid fault cases and covers 25 fault types.

To evaluate the effectiveness of RippleRCA, comparisons are conducted against six state-of-the-art RCA methods, including MicroRank (Yu et al., 2021), Baro (Pham et al., 2024a), Eadro (Lee et al., 2023), Nezha (Yu et al., 2023a), CauseLens (Liu et al., 2025) and LagRCA (Zhang et al., 2026). These baselines cover trace-based, metric-based, end-to-end, multimodal, and causal-graph-based RCA paradigms. More details of the baselines are provided in Appendix E.

Evaluation Metrics and Implementation Following existing evaluation protocols (Liu et al., 2025; Fang et al., 2025), AC@1, AC@3, AC@5, Avg@5, and MRR are used to evaluate localization performance. AC@k measures the top-k hit rate, Avg@5 averages AC@k over k=1 to 5, and MRR averages the reciprocal rank of the first true root cause. Additionally, downstream error rate (DER) measures the proportion of all cases where a downstream node is ranked first instead of any true root cause. Downstream nodes are identified along the caller-to-callee direction in the original call graph. Lower DER indicates fewer such errors; its formal definition is provided in Appendix E.

All experiments were conducted on a single machine with an Intel Core i5-14400 processor (2.50 GHz) and 16 GB RAM. The implementation is based on Python 3.10.19 and PyTorch 1.13.1. We empirically set $(\tau_{\max}, K, \beta) = (5, 8, 0.35)$ as the shared configuration for both datasets. This configuration is used in all subsequent experiments unless otherwise specified. Parameter sensitivity is further examined in RQ3.

4.2 RQ1: OVERALL EFFECTIVENESS

Table 1 summarizes the performance of RippleRCA and the baselines on both datasets. MicroRank, Baro, and Nezha produce fixed results under the evaluated setting and are therefore reported as single values; the remaining methods are reported as mean $\pm$ standard deviation across repeated runs. RippleRCA achieves the best reported performance across all five metrics on both datasets, demonstrating consistent improvements in root cause identification and ranking quality.

On AIOps2025, Eadro is the strongest baseline across all metrics. RippleRCA improves AC@1 from 0.217 to 0.313 and MRR from 0.435 to 0.476, corresponding to relative improvements of 44.2% and 9.4%, respectively. It also achieves AC@3, AC@5, and Avg@5 scores of 0.598, 0.779, and 0.564. The larger gains at AC@1 and AC@3, compared with the smaller improvement at AC@5, indicate that RippleRCA primarily improves the placement of true root causes near the top of the ranking. This pattern suggests that the main challenge on AIOps2025 is not merely including the root cause among the candidates, but distinguishing it from competing anomalous services and assigning it a higher rank.

On RCABench, Baro is the strongest baseline across all five metrics. RippleRCA increases MRR from 0.366 to 0.513, a relative improvement of 40.2%. It also improves AC@1 from 0.295 to 0.370,

AC@3 from 0.410 to 0.635, and AC@5 from 0.513 to 0.756, while Avg@5 increases from 0.406 to 0.587. Compared with the gains on AIOps2025, the improvements in MRR, AC@3, and AC@5 are more pronounced on RCABench. Together, these results demonstrate its effectiveness across both benchmarks. The contributions of lag-aware modeling and counterfactual ranking are further examined in the ablation study.

Table 1: Overall performance comparison between RippleRCA and SOTA baselines on AIOps2025 and RCABench. Metrics include AC@1, AC@3, AC@5, Avg@5 and MRR. Best results are highlighted in bold.

	AIOps2025					RCABench				
Method	AC@1	AC@3	AC@5	Avg@5	MRR	AC@1	AC@3	AC@5	Avg@5	MRR
MicroRank	0.143	0.427	0.427	0.332	0.238	0.013	0.092	0.118	0.074	0.051
Baro	0.156	0.393	0.437	0.327	0.265	0.295	0.410	0.513	0.406	0.366
Nezha	0.054	0.071	0.071	0.065	0.061	0.092	0.171	0.243	0.169	0.264
CauseLens	0.153±0.014	0.309±0.018	0.470±0.012	0.307±0.007	0.300±0.012	0.184±0.028	0.239±0.018	0.269±0.010	0.230±0.014	0.228±0.023
Eadro	0.217±0.021	0.538±0.025	0.743±0.007	0.499±0.002	0.435±0.009	0.153±0.056	0.268±0.058	0.379±0.089	0.267±0.065	0.282±0.055
LagRCA	0.064±0.007	0.233±0.025	0.462±0.018	0.253±0.011	0.182±0.005	0.041±0.008	0.121±0.006	0.180±0.004	0.114±0.005	0.089±0.006
RippleRCA	**0.313±0.012**	**0.598±0.007**	**0.779±0.007**	**0.564±0.002**	**0.476±0.005**	**0.370±0.020**	**0.635±0.069**	**0.756±0.045**	**0.587±0.040**	**0.513±0.030**

4.3 RQ2: Ablation Study

Table 2 evaluates the contribution of each component. Removing the lag prior reduces AC@1 and MRR on both datasets. On AIOps2025, AC@1 drops from 0.313 to 0.177, while DER rises from 0.141 to 0.301. RCABench exhibits the same trend, although the changes are smaller. These results suggest that the lag prior helps distinguish true root causes from downstream non-root services, particularly on AIOps2025.

Removing lag alignment also lowers AC@1 and MRR and increases DER on both datasets, while changes in AC@3 and AC@5 are comparatively small. The degradation in AC@1 and MRR is more pronounced on RCABench, where AC@1 decreases from 0.370 to 0.316 and DER increases from 0.173 to 0.232. This pattern is consistent with the more prevalent non-zero lags on RCABench compared with the predominantly near-synchronous responses on AIOps2025 in Appendix A. These results support the value of aligning counterfactual effect estimation with propagation delays to improve root cause prioritization.

Removing the propagation response channel has different effects across the two datasets. On RCABench, AC@3 decreases from 0.635 to 0.534 and AC@5 from 0.756 to 0.643, supporting its contribution to retaining true roots near the top of the ranking. On AIOps2025, AC@3 improves slightly and AC@5 remains unchanged, while AC@1 and MRR decline. Despite a small decrease in DER, more errors involve other non-root services, resulting in lower overall Top-1 accuracy.

The stage-only variants further support the two-stage design. On AIOps2025, Stage 1 alone frequently ranks downstream non-root services first, while Stage 2 alone achieves slightly higher AC@3 and AC@5 but lower AC@1 and MRR than the full model. On RCABench, Stage 1 alone produces constant Top-1 predictions; its zero DER reflects errors outside the downstream-node category rather than accurate localization. Stage 2 alone exhibits frequent downstream-node errors. The full model achieves the highest mean AC@1 and MRR on both datasets, outperforming either stage alone. These results support the effectiveness of combining candidate recall with counterfactual refinement.

Table 2: Ablation results. DER: downstream-node error rate; †: constant Top-1 prediction on RCABench.

	AIOps2025					RCABench				
Method	AC@1↑	AC@3↑	AC@5↑	MRR↑	DER↓	AC@1↑	AC@3↑	AC@5↑	MRR↑	DER↓
RippleRCA	**0.313±0.012**	0.598±0.007	0.779±0.007	**0.476±0.005**	0.141±0.007	**0.370±0.020**	**0.635±0.069**	**0.756±0.045**	**0.513±0.030**	0.173±0.006
w/o lag prior	0.177±0.007	0.570±0.007	0.759±0.021	0.384±0.006	0.301±0.000	0.327±0.022	0.592±0.016	0.729±0.023	0.472±0.011	0.222±0.061
w/o lag alignment	0.289±0.001	0.590±0.012	**0.783±0.000**	0.462±0.002	0.157±0.000	0.316±0.026	0.615±0.050	0.739±0.010	0.472±0.024	0.232±0.000
w/o propagation response	0.273±0.014	**0.614±0.000**	0.779±0.007	0.461±0.005	0.133±0.000	0.340±0.034	0.534±0.027	0.643±0.044	0.446±0.012	0.222±0.037
Stage1 only	0.108±0.000	0.402±0.014	0.590±0.021	0.273±0.005	0.458±0.000	0.122±0.000	0.524±0.010	0.609±0.017	0.328±0.006	0.000±0.000†
Stage2 only	0.265±0.000	0.602±0.000	**0.783±0.000**	0.452±0.000	0.217±0.000	0.038±0.000	0.192±0.000	0.269±0.000	0.122±0.000	0.468±0.000

4.4 RQ3: Robustness and Sensitivity

Parameter sensitivity. As shown in Figure 2, RippleRCA remains relatively stable within a reasonable range of hyperparameter settings on AIOps2025. The corresponding RCABench results are provided in Appendix F. For the maximum lag window $\tau_{\max}$, performance is stable from 0 to 5, while AC@1, AC@3, and MRR decline at $\tau_{\max} = 8$, suggesting that an excessively large temporal window may introduce redundant information. For the candidate size K, performance improves markedly from 3 to 5 and changes only slightly from 5 to 8. At $K = 8$, AC@1 and AC@5 improve slightly, AC@3 decreases slightly, and MRR remains nearly unchanged. For the Stage 1 fusion weight β, performance is strongest around 0.25-0.35, while larger values lead to noticeable degradation in AC@1, AC@3, and MRR.

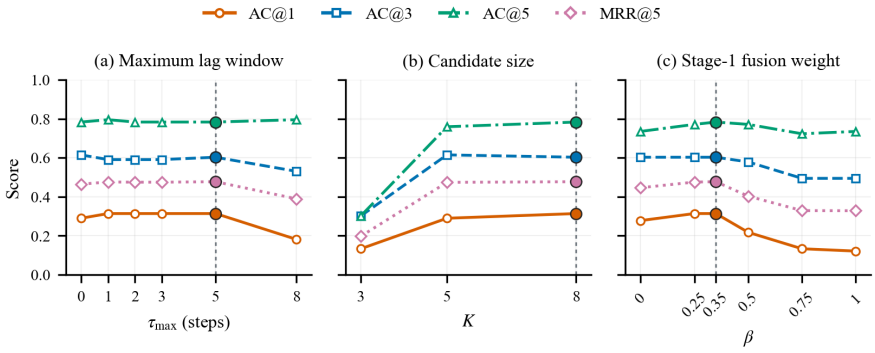


Figure 2: Parameter sensitivity of $\tau_{\max}$, K, and β on AIOps2025.

Robustness to topology perturbations. Robustness is evaluated on AIOps2025 by removing existing edges or adding spurious directed edges at rates of 10% and 20%. All topology-dependent components are recomputed for each perturbed topology. The detailed experimental setup and the corresponding RCABench results are provided in Appendix G. As shown in Table 3, RippleRCA remains relatively stable under moderate topology perturbations on AIOps2025. Removing 20% of the existing edges reduces MRR from 0.476 to 0.423, whereas adding 20% spurious edges results in a smaller decrease to 0.465. A similar pattern is observed for AC@1 and AC@3. Meanwhile, AC@5 remains close to the clean result under missing edges and reaches 0.783 and 0.775 under the 10% and 20% spurious-edge settings, respectively. These results indicate that missing true dependencies has a greater effect on ranking performance than introducing a moderate number of spurious connections.

Table 3: Robustness to topology perturbations on AIOps2025.

Dataset	Condition	AC@1	AC@3	AC@5	MRR
AIOps2025	Clean	**0.313±0.012**	**0.598±0.007**	0.779±0.007	**0.476±0.005**
AIOps2025	Missing 10%	0.265±0.036	0.570±0.049	0.763±0.007	0.438±0.030
AIOps2025	Missing 20%	0.253±0.021	0.542±0.032	0.735±0.021	0.423±0.013
AIOps2025	Spurious 10%	0.289±0.043	0.574±0.028	**0.783±0.000**	0.461±0.034
AIOps2025	Spurious 20%	0.293±0.025	0.586±0.039	0.775±0.007	0.465±0.022

5 Conclusion

In this paper, we propose RippleRCA, a two-stage lag-calibrated causal inference framework for root cause analysis in microservice systems. The framework models fault propagation from three aspects, namely topology-constrained lag priors, two-stage design, and lag-calibrated counterfactual inference. It progressively narrows the root cause search space through service-level candidate recall and instance-level fine-grained localization. Experimental results on RCABench and AIOps2025 show that RippleRCA achieves strong performance, demonstrating its effectiveness.

For reproducibility, The source code and experimental data used in this study are available in our anonymous repository at `https://anonymous.4open.science/r/RippleRCA-373C/`.

AI USE STATEMENT

In this work, We used generative AI tools to assist with language polishing, translation, and LaTeX formatting. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work aims to improve the reliability of microservice systems through root cause analysis. All experiments use publicly available datasets.

REPRODUCIBILITY STATEMENT

Section 4.1 describes the public datasets, baseline methods, evaluation metrics, hardware configuration, software environment, and hyperparameter settings used in our experiments. Additional details of the baseline methods are provided in Appendix E. The sources of AIOps2025 and RCABench are cited to facilitate access to the experimental datasets.

REFERENCES

Hajar Hameed Addeen. *A dynamic fault tolerance model for microservices architecture*. South Dakota State University, 2019.

Lionel Barnett and Anil K Seth. Granger causality for state-space models. *Physical Review E*, 91 (4):040101, 2015.

Andre Bento, Jaime Correia, Joao Duraes, João Soares, Luís Ribeiro, António Ferreira, Rita Carreira, Filipe Araujo, and Raul Barbosa. A layered framework for root cause diagnosis of microservices. In *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, pp. 1–8. IEEE, 2021.

Kay Bierzynski, Pavel Lutskov, and Uwe Assmann. Supporting the self-learning of systems at the network edge with microservices. In *Smart Systems Integration; 13th International Conference and Exhibition on Integration Issues of Miniaturized Systems*, pp. 1–8. VDE, 2019.

Álvaro Brandón, Marc Solé, Alberto Huélamo, David Solans, María S Pérez, and Victor Muntés-Mulero. Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software*, 159:110432, 2020.

Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 1887–1895. IEEE, 2014.

Yuhua Chen, Dongqi Xu, Ningjiang Chen, and Xu Wu. Frl-mfpg: Propagation-aware fault root cause location for microservice intelligent operation and maintenance. *Information and Software Technology*, 153:107083, 2023.

Anthony C Constantinou, Zhigao Guo, and Neville K Kitson. The impact of prior knowledge on causal structure learning. *Knowledge and Information Systems*, 65(8):3385–3434, 2023.

AIOps Competition Dataset. `https://www.aiops.cn/gitlab/aiops-live-benchmark/aiopschallenge2025`, 2025. Accessed: 2025-11-20.

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

Aoyang Fang, Songhan Zhang, Yifan Yang, Haotong Wu, Junjielong Xu, Xuyang Wang, Rui Wang, Manyi Wang, Qisheng Lu, and Pinjia He. Rethinking the evaluation of microservice rca with a fault propagation-aware benchmark. *arXiv preprint arXiv:2510.04711*, 2025.

Yu Gan, Guiyang Liu, Xin Zhang, Qi Zhou, Jiesheng Wu, and Jiangwei Jiang. Sleuth: A trace-based root cause analysis system for large-scale microservices with graph neural networks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, pp. 324–337, 2023.

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

Xinrui Jiang, Yicheng Pan, Meng Ma, and Ping Wang. Look deep into the microservice system anomaly through very sparse logs. In *Proceedings of the ACM Web Conference 2023*, pp. 2970–2978, 2023.

Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1750–1762. IEEE, 2023.

I-Ting Angelina Lee, Zhizhou Zhang, Abhishek Parwal, and Milind Chabbi. The tale of errors in microservices. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 8 (3):1–36, 2024.

Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering*, 27(1):25, 2022.

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

Cheng-Ming Lin, Ching Chang, Wei-Yao Wang, Kuang-Da Wang, and Wen-Chih Peng. Root cause analysis in microservice using neural granger causal discovery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 206–213, 2024.

Chao Liu, Kin Gwn Lore, Zhanhong Jiang, and Soumik Sarkar. Root-cause analysis for time-series anomalies via spatiotemporal graphical modeling in distributed complex systems. *Knowledge-Based Systems*, 211:106527, 2021.

Qihan Liu, Pengfei Chen, Guangba Yu, Yuanhao Lai, and Xiaoyun Li. Causelens: Causality-based interpretable root cause analysis for microservice systems. In *2025 IEEE/ACM 33rd International Symposium on Quality of Service (IWQoS)*, pp. 1–10. IEEE, 2025.

Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM symposium on cloud computing*, pp. 412–426, 2021.

Nabor C Mendonca, Carlos M Aderaldo, Javier Cámara, and David Garlan. Model-based analysis of microservice resiliency patterns. In *2020 IEEE International Conference on Software Architecture (ICSA)*, pp. 114–124. IEEE, 2020.

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

Judea Pearl. *Causality*. Cambridge university press, 2009.

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

Luan Pham, Huong Ha, and Hongyu Zhang. Root cause analysis for microservice system based on causal inference: How far are we? In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 706–715, 2024b.

Jakob Runge. Discovering contemporaneous and lagged causal relations in autocorrelated nonlinear time series datasets. In *Conference on uncertainty in artificial intelligence*, pp. 1388–1397. Pmlr, 2020.

Jakob Runge, Peer Nowack, Marlene Kretschmer, Seth Flaxman, and Dino Sejdinovic. Detecting causal associations in large nonlinear time series datasets. *arXiv preprint arXiv:1702.07007*, 2017.

Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. ϵ -diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. In *The World Wide Web Conference*, pp. 3215–3222, 2019.

Jacopo Soldani, Stefano Forti, and Antonio Brogi. yrca: An explainable failure root cause analyser. *Science of Computer Programming*, 230:102997, 2023.

Yongqian Sun, Binpeng Shi, Mingyu Mao, Minghua Ma, Sibo Xia, Shenglin Zhang, and Dan Pei. Art: A unified unsupervised framework for incident management in microservice systems. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1183–1194, 2024.

Gábor J Székely, Maria L Rizzo, and Nail K Bakirov. Measuring and testing dependence by correlation of distances. 2007.

Alex Tank, Ian Covert, Nicholas Foti, Ali Shojaie, and Emily B Fox. Neural granger causality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4267–4279, 2021.

Train Ticket. `https://github.com/FudanSELab/train-ticket`, 2024. Accessed: 2025-07-01.

Ping Wang, Jingmin Xu, Meng Ma, Weilan Lin, Disheng Pan, Yuan Wang, and Pengfei Chen. Cloudranger: Root cause identification for cloud native systems. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 492–502. IEEE, 2018.

Yidan Wang, Zhouruixing Zhu, Qiuai Fu, Yuchi Ma, and Pinjia He. Mrca: Metric-level root cause analysis for microservices via multi-modal data. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1057–1068, 2024.

Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. Microrca: Root cause localization of performance issues in microservices. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2020.

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

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

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

Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM joint European software engineering conference and symposium on the foundations of software engineering*, pp. 553–565, 2023a.

Qingyang Yu, Changhua Pei, Bowen Hao, Mingjie Li, Zeyan Li, Shenglin Zhang, Xianglin Lu, Rui Wang, Jiaqi Li, Zhenyu Wu, et al. Cmdiagnostor: An ambiguity-aware root cause localization approach based on call metric data. In *Proceedings of the ACM web conference 2023*, pp. 2937–2947, 2023b.

Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, et al. Robust failure diagnosis of microservice system through multimodal data. *IEEE Transactions on Services Computing*, 16(6):3851–3864, 2023.

Shenglin Zhang, Junhua Kuang, Yimeng Zhang, Sibo Xia, Jintao Feng, Jingyu Wang, Wenwei Gu, Yongqian Sun, Wei Li, Liping Zhang, et al. Bridging the delay: Lag-aware spatio-temporal causal inference for microservice root cause analysis. In *Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering*, pp. 461–472, 2026.

Songhan Zhang, Aoyang Fang, Yifan Yang, Ruiyi Cheng, Xiaoying Tang, and Pinjia He. Dynacausal: Dynamic causality-aware root cause analysis for distributed microservices. *arXiv preprint arXiv:2510.22613*, 2025.

Yuhan Zhu, Jian Wang, Bing Li, Yuqi Zhao, Zekun Zhang, Yiming Xiong, and Shiping Chen. Microirc: Instance-level root cause localization for microservice systems. *Journal of Systems and Software*, 216:112145, 2024.

A Temporal Lag

To verify whether anomaly propagation delays genuinely exist in the two datasets, we further conduct an empirical lag analysis on the raw data. Specifically, the original metric observations are first aggregated into service-level anomaly intensity sequences. We then compare the consistency of anomaly responses across different services under varying temporal offsets. If the anomaly variation of one service is more strongly correlated with that of another service after several time steps, an empirical lagged response is inferred between the two services. In the experiments, the maximum analysis window is set to 10 time steps, where the temporal resolution is 10 seconds for RCABench and 1 minute for AIOps2025.

Figure 3 presents the distribution of lagged anomaly responses across service pairs. The horizontal axis denotes the time delay at which each service pair reaches the highest anomaly correlation, and the vertical axis denotes the proportion of such pairs among all service pairs. The results show that non-zero lagged responses exist in both datasets, indicating that anomalies do not always occur synchronously, but often propagate with delays along service dependencies. In AIOps2025, synchronous responses dominate, yet approximately 47% of service pairs still exhibit non-zero lags. In contrast, RCABench shows a more dispersed lag distribution, with about 65% of service pairs presenting non-zero lags, suggesting more pronounced lagged anomaly propagation.

These findings confirm that temporal propagation delays are a real phenomenon in microservice fault localization. If lagged dependencies are ignored, models may struggle to distinguish true root cause services from downstream affected services. Therefore, it is necessary to explicitly model temporal lags in the anomaly propagation process.

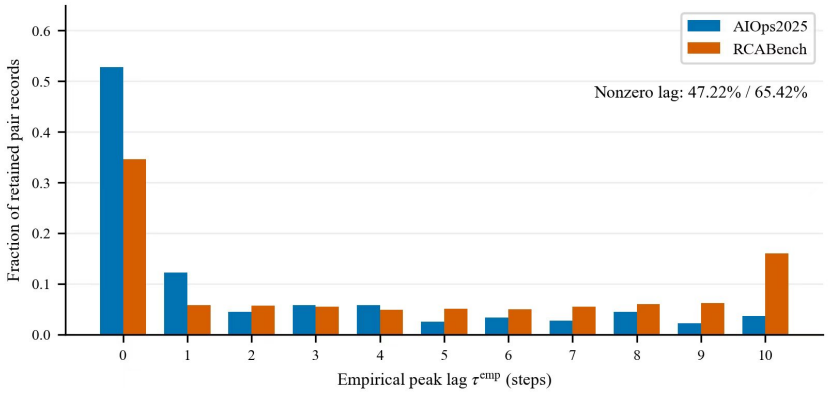


Figure 3: Temporal lag distribution across service pairs.

B PCMCI+ AND EDGE-LEVEL LAG PRIOR CONSTRUCTION

B.1 PCMCI+ BACKGROUND

PCMCI+ is a constraint-based causal discovery algorithm designed for multivariate time series and is well suited to identifying temporal causal relationships in high-dimensional and strongly autocorrelated settings. Unlike methods that infer causal directions mainly from correlations or regression coefficients, PCMCI+ progressively removes spurious causal associations through conditional independence tests and identifies lagged dependency structures with statistical support. Since monitoring metrics in microservice systems often exhibit strong temporal dependence and cross-service propagation effects, PCMCI+ provides an appropriate foundation for modeling lagged causal relationships among services.

B.2 TOPOLOGY-CONSTRAINED ADAPTATION

In microservice systems, anomaly propagation typically follows service invocation paths, making the service call graph an important source of structural prior knowledge. Based on this observation, the service call graph G_{trace} is introduced as a hard constraint, and PCMCI+ is executed only on node pairs $(u, v) \in E_{\text{trace}}$. In other words, temporal causal relationships are tested only when services u and v are connected by an invocation dependency in the call topology.

Such a topology constraint provides two main benefits. First, it significantly reduces the search space of causal discovery and therefore lowers computational complexity. Second, it structurally suppresses spurious causal associations between unrelated services, reducing false inferences caused by high-dimensional noise or synchronous fluctuations. As a result, topology-constrained PCMCI+ is better suited to modeling cross-service fault propagation in microservice environments.

B.3 EDGE-LEVEL LAG PRIOR CONSTRUCTION

After obtaining candidate causal edges with statistical support, an edge-level lag prior is further constructed for each edge to characterize the relative propagation strength of faults from an upstream service to a downstream service at different lag positions. Specifically, for each candidate edge $(u \to v, \tau')$, PCMCI+ computes lag-specific causal strengths within a predefined maximum lag window $[0, \tau_{max}]$, thereby capturing the potential delayed influence of service u on service v.

Let $MCI(u \to v, \tau)$ denote the causal strength of edge (u, v) at lag τ. The edge-level lag prior distribution is then defined as

$$P_{prior}^{(u,v)}(\tau) = \frac{MCI(u \to v, \tau)}{\sum_{\tau'=0}^{\tau_{max}} MCI(u \to v, \tau')} \tag{15}$$

where $P_{prior}^{(u,v)}(\tau)$ denotes the relative causal strength of edge (u, v) at lag τ. This definition normalizes lag-specific causal strengths into a distribution, yielding an edge-level lag prior over the lag space.

Rather than assigning a single optimal delay to each edge, the proposed formulation provides a distributional characterization over the entire lag space. This design is motivated by the fact that fault propagation in microservice systems is often influenced by request queueing, network latency, retry mechanisms, and other system factors, so propagation delay does not necessarily concentrate on a single time step. The resulting edge-level lag prior therefore preserves the uncertainty of propagation delay and provides richer prior information for subsequent lag-aware representation learning and candidate diagnosis.

C STAGE 1 CANDIDATE RECALL AND STATISTICAL AGGREGATION

This section provides the detailed definitions of sample-level candidate construction and cross-sample statistical aggregation in Stage 1.

For each fault sample i, Top-K candidate nodes are selected according to the structural anomaly channel score $S_A^{(i)}$ and the propagation response channel score $S_B^{(i)}$, yielding two sample-level can-

didate sets:

$$C_A^{(i)} = \mathrm{TopK}\Big(S_A^{(i)}\Big), \qquad C_B^{(i)} = \mathrm{TopK}\Big(S_B^{(i)}\Big). \tag{16}$$

The final candidate set for sample i is defined as the union of the two:

$$C^{(i)} = C_A^{(i)} \cup C_B^{(i)}. \tag{17}$$

If a node is retrieved by both channels, it is included only once in the sample-level candidate set, and no additional reward is assigned for cross-channel overlap. The local score of node v in sample i is therefore defined as

$$S_{\mathrm{local}}^{(i)}(v) = \max\Big(S_A^{(i)}(v), S_B^{(i)}(v)\Big), \qquad v \in C^{(i)}. \tag{18}$$

Across multiple fault samples, candidate nodes are further aggregated statistically. Assume that there are M fault samples, each corresponding to an independent fault instance. Let

$$I_v = \{\, i \mid v \in C^{(i)} \,\} \tag{19}$$

denote the set of sample indices in which node v is retrieved. The occurrence frequency of node v is defined as

$$p_{\mathrm{vote}}(v) = \mathbb{E}_i\left[\mathbb{I}\Big(v \in C^{(i)}\Big)\right]. \tag{20}$$

For each node retrieved in at least one sample, the average local score and the peak local score are defined as

$$\bar{S}(v) = \frac{1}{|I_v|}\sum_{i \in I_v} S_{\mathrm{local}}^{(i)}(v), \qquad S_{\max}(v) = \max_{i \in I_v} S_{\mathrm{local}}^{(i)}(v). \tag{21}$$

These statistics are combined into the Stage 1 global score according to Eq. (7).

D Stage2 Implementation Details of Lagged-SCM Counterfactual Inference

The main text defines the lagged SCM, lag-aligned intervention, and descendant-level counterfactual effect. This section provides the additional implementation details required to reproduce Stage 2.

SCM fitting. For each node v, the predictor input concatenates its previous state with the lagged states of its parents. In our implementation, f_v is instantiated as a ridge-regularized linear mechanism and learned from normal operating windows:

$$(\widehat{W}_v, \widehat{b}_v) = \arg\min_{W_v, b_v} \sum_{s \in \mathcal{T}_{\mathrm{norm}}} \big\|X_v(s) - W_v^\top Z_v(s) - b_v\big\|_2^2 + \rho\|W_v\|_F^2, \tag{22}$$

where $Z_v(s)$ contains $X_v(s-1)$ and $\{X_u(s-\tau_{uv}) \mid u \in \mathrm{Pa}(v)\}$. The intervention lag τ_v^* is obtained from the Stage 1 lag weights as $\tau_v^* = \arg\max_\tau \alpha_v(\tau)$. The edge lags τ_{uv} determine the SCM inputs, whereas τ_v^* determines the intervention time.

Residual abduction and factual replay. Given an observed faulty trajectory, the sample-specific exogenous residuals are inferred by

$$\widehat{U}_v(s) = X_v^{\mathrm{obs}}(s) - \widehat{f}_v\big(Z_v^{\mathrm{obs}}(s)\big). \tag{23}$$

The inferred residuals are then used to replay the factual trajectory. This replay is computed once for each fault sample and shared by all candidate interventions.

Lag-aligned intervention and prediction. For candidate v, the intervention time is $t_v = t - \tau_v^*$, where t denotes the current observation time. We retain the observed history before t_v and apply the normal-state intervention defined in the main text at t_v. For $s \geq t_v$, the counterfactual trajectory is computed using the fitted structural mechanisms while keeping the abducted residuals fixed.

$$X_j^{\mathrm{CF}(v)}(s) = \begin{cases} \mu_v^{\mathrm{norm}} & j = v \text{ and } s = t_v, \\ \widehat{f}_j\Big(Z_j^{\mathrm{CF}(v)}(s)\Big) + \widehat{U}_j(s), & \text{otherwise.} \end{cases} \tag{24}$$

Thus, the factual and counterfactual trajectories differ only through the candidate intervention and its propagated consequences. The default intervention is applied at t_v only, rather than being clamped at all later time steps.

Zero-lag relations and validity checks. Contemporaneous relations with $\tau_{uv} = 0$ are solved by fixed-point iteration within each time slice. We use at most 25 iterations and declare convergence when the maximum element-wise update is no greater than 10^{-6}. A counterfactual is used only when both factual replay and counterfactual prediction converge and the factual reconstruction error is below the implementation threshold. Otherwise, its counterfactual effect is set to zero. The effect is also set to zero when the candidate has no causal descendant. With the default future horizon of zero, the response range in equation 11 contains the final observed time step.

E Experimental Setup

E.1 Evaluation Metrics

Downstream Error Rate (DER). DER measures the fraction of all evaluated fault cases in which the top-ranked prediction is a downstream node that is not an annotated root cause. A downstream node is defined as a node reachable from at least one annotated root cause along the caller-to-callee direction in the original service call topology. Formally,

$$\mathrm{DER} = \frac{1}{M}\sum_{i=1}^{M} \mathbb{I}\left[\hat{v}_1^{(i)} \notin \mathcal{R}^{(i)} \ \wedge \ \hat{v}_1^{(i)} \in \bigcup_{u \in \mathcal{R}^{(i)}} \mathrm{Reach}^{+}_{G^{(i)}_{\mathrm{trace}}}(u)\right]. \tag{25}$$

Here, M is the total number of evaluated fault cases, $\hat{v}_1^{(i)}$ is the top-ranked prediction for case i, $\mathcal{R}^{(i)}$ is its set of annotated root causes, and $\mathbb{I}[\cdot]$ is the indicator function. $G^{(i)}_{\mathrm{trace}}$ denotes the original directed service call topology for case i, with edges directed from callers to callees. $\mathrm{Reach}^{+}_{G}(u)$ denotes the set of nodes reachable from u through a directed path containing at least one edge. Predictions, annotated root causes, and topology nodes use the same service-level identifiers.

Cases with no reachable nodes outside the annotated root cause set remain in the denominator and contribute zero to the numerator. A lower DER indicates a smaller proportion of cases in which a downstream node outside the root cause set is incorrectly ranked first. This metric does not measure verified fault propagation relationships or propagation delays.

E.2 Datasets

To comprehensively evaluate the performance of RippleRCA in microservice root cause analysis, two representative public datasets are adopted, namely AIOps2025 and RCABench. These two datasets represent a real-world complex setting and a controlled benchmark setting, respectively, enabling evaluation of the model in terms of both effectiveness and generalization.

AIOps2025: This dataset is collected from a real production environment and contains multiple types of microservice failure scenarios (Dataset, 2025), such as CPU anomalies and network latency faults, together with ground-truth root cause labels. It includes 400 fault cases across 5 services and covers 6 resource-related fault types, reflecting substantial real-world complexity.

RCABench: The RCABench dataset (Fang et al., 2025) used in this work is built upon the classical Train-Ticket system (Ticket, 2024). It contains 1,430 valid fault cases and covers 25 fault types, providing a systematically extended benchmark for root cause analysis.

Table 4: Statistics of the datasets.

Dataset	Modal	Services	# Types	# Faults
AIOps2025	Metric, Trace	8	6	400
RCABench	Metric, Trace, Log	50	25	1430

E.3 BASELINES

To comprehensively evaluate the effectiveness of RippleRCA, we compare it with six state-of-the-art (SOTA) RCA methods that cover diverse technical paradigms based on metrics, traces, and multi-modal observability data.

MicroRank (Yu et al., 2021) A trace-based method that analyzes the discrepancy between normal and abnormal execution traces to construct an anomaly propagation graph, and ranks potential root causes using an improved PageRank algorithm.

Baro (Pham et al., 2024a) A metric-based approach that leverages multivariate Bayesian online change point detection to accurately identify anomaly occurrence time points, and infers root causes by analyzing metric propagation patterns.

Eadro (Lee et al., 2023) An end-to-end RCA framework that integrates anomaly detection and root cause analysis, and is designed to handle microservice topology structures together with multidimensional monitoring metrics.

Nezha (Yu et al., 2023a) A fine-grained and interpretable multimodal RCA method that localizes fault sources by identifying inconsistencies among anomalous patterns across heterogeneous signals.

CauseLens (Liu et al., 2025) A method that constructs a heterogeneous causal graph from metric and trace data. It captures complex inter-service dependencies and introduces counterfactual analysis to reveal fault propagation paths.

LagRCA (Zhang et al., 2026) A lag-aware causal RCA method that models delayed fault propagation across microservices. It learns multi-lag causal dependencies to capture spatio-temporal relationships and identify root causes.

F PARAMETER SENSITIVITY

As a complement to the AIOps2025 sensitivity analysis presented in the main text, Figure 4 reports the corresponding results on RCABench. For the maximum lag window $\tau_{\max}$, performance varies non-monotonically across the tested settings. At $\tau_{\max} = 5$, AC@3 reaches its highest value. Increasing the window from 5 to 8 reduces AC@1, AC@3, and MRR, while AC@5 improves slightly. These results indicate that a larger lag window does not consistently improve localization performance.

For the candidate size K, performance varies moderately across the tested settings. $K = 8$ achieves the highest AC@1 and MRR and shares the highest AC@3 with $K = 10$. Further increasing K to 10 slightly improves AC@5 but reduces AC@1 and MRR. Thus, $K = 8$ provides a balanced configuration for candidate coverage and root cause prioritization.

For the Stage 1 fusion weight β, all four metrics remain unchanged across the tested values from 0 to 1. This indicates that the reported localization performance on RCABench is insensitive to the fusion weight under the evaluated configuration. The value $\beta = 0.35$ is retained for consistency with AIOps2025.

G ROBUSTNESS TO TOPOLOGY PERTURBATIONS

To complement the AIOps2025 results reported in the main text, we evaluate the robustness of RippleRCA on RCABench under incomplete and noisy service topologies. We consider two perturbation settings: removing existing edges and adding previously absent directed edges. For each

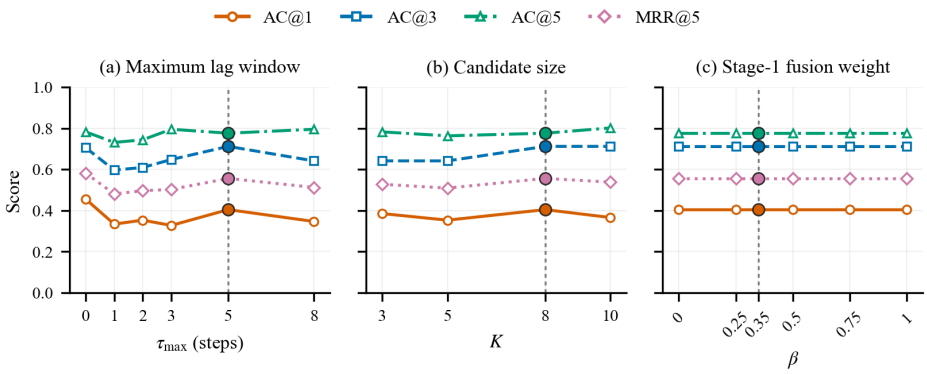


Figure 4: Parameter sensitivity of $\tau_{\max}$, K, and β on RCABench.

setting, 10% and 20% of the original edge count are perturbed. Each condition, including the clean topology, is evaluated using seeds 42, 52, and 72, and the mean and standard deviation are reported.

Importantly, we do not reuse the lag priors or SCM outputs obtained from the clean topology. For each run, we reconstruct the Stage 1 graph, the topology-constrained PCMCI+ lag priors, and the parent and descendant relations of the lagged SCM from the perturbed topology, while keeping all other configurations unchanged.

Table 5: Robustness to topology perturbations on RCABench.

Dataset	Condition	AC@1	AC@3	AC@5	MRR
RCABench	Clean	0.370±0.020	0.635±0.069	0.756±0.045	0.513±0.030
RCABench	Missing 10%	0.346±0.019	0.590±0.039	0.714±0.023	0.483±0.020
RCABench	Missing 20%	0.316±0.039	0.517±0.019	0.688±0.026	0.447±0.028
RCABench	Spurious 10%	0.359±0.032	0.632±0.041	0.769±0.029	0.508±0.038
RCABench	Spurious 20%	0.335±0.027	0.611±0.023	0.737±0.013	0.485±0.018

As shown in Table 5, RippleRCA remains relatively stable under moderate topology perturbations on RCABench. Removing 10% of the existing edges has only a minor effect on MRR, which changes from 0.513 to 0.483, while removing 20% reduces it to 0.447. In contrast, adding spurious edges causes smaller degradation: MRR reaches 0.508 under 10% spurious edges and remains at 0.485 under 20% perturbation. Similar trends are observed for AC@3 and AC@5, where missing edges lead to larger performance drops than spurious connections. These results indicate that RippleRCA is reasonably robust to topology noise, while missing true service dependencies has a greater impact on localization performance than introducing a moderate number of spurious edges.