KLD: KINETIC LOG-AWARE JOINT LEARNING FOR DYNAMIC MICROSERVICE DIAGNOSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Dynamic microservice diagnosis enables timely anomaly detection and rapid fault remediation, laying essential foundations for the availability, performance and reliability of microservice systems. Nevertheless, frequent changes of service instances and invocation topologies, noisy and sparse monitoring data, and pervasive metric drift pose significant diagnostic challenges. We present KLD, an evolution- and runtime-dependency-aware framework for joint anomaly detection and root-cause localization in dynamic microservices. Instead of treating multi-modal observations as static system states, KLD explicitly models the temporal evolution of runtime signals and fault evidence propagation over execution-aware dependency structures. KLD extracts multi-order temporal and trajectory features from metrics and traces, aligns these features with log semantics via cross-modal interaction and cooperative gating, and propagates fused evidence over runtime dependency graphs reconstructed from distributed traces to model fault cascading. Leveraging margin-based contrastive representation learning, KLD distinguishes root-cause nodes corresponding to injected faults from downstream affected services. Evaluations on SN and TT datasets show that our method achieves detection F1 scores of 0.995 / 0.993 and localization HR@1 of 0.995 / 0.994, outperforming state-of-the-art baselines. Ablation studies validate the effectiveness of each component, and robustness experiments demonstrate stable performance under controlled perturbations on logs, traces and metric baselines.

1 INTRODUCTION

Cloud computing fuels wide adoption of microservices via elastic scaling, yet tangled service dependencies bring heavy maintenance burdens. Faults spread fast and create alert storms, requiring automated RCA for system availability Soldani & Brogi (2022); Pu et al. (2025). Static diagnosis built on fixed scale, topology and stable data cannot resolve three spatiotemporal microservice issues: elastic scaling invalidates static topology priors Wang et al. (2024); volatile traffic distorts invocation graphs and hinders fault dependency-aware reasoning Zhang et al. (2025); workload shifts trigger lasting metric drift Liang et al. (2026). Dynamic RCA Zhang et al. (2026a) demands a unified framework to model variable instances, evolving graphs and drifting signals, a demand unmet by static diagnostic paradigms.

Logs, metrics and traces serve as primary monitoring modalities, yet each has inherent limitations for industrial RCA. Logs produce trivial latency alerts lacking direct semantic connections to real faults; metric fluctuations often remain within SLO bounds, invalidating simple threshold detection Yao et al. (2024). Distributed traces adopt sampling to reduce overhead, yielding fragmented sparse data under low traffic Zhang et al. (2024). The challenge is therefore not merely to fuse heterogeneous modalities, but to reconcile two complementary forms of evidence: semantic fault cues carried by logs and evolving physical behavior captured by metrics and traces. A diagnosis model should preserve this semantic–physical correspondence while remaining sensitive to temporal transitions and runtime dependency changes. Furthermore, the loosely coupled multi-instance microservice architecture complicates modeling complex dependencies and capturing cross-instance dependency relationships. Separating true root nodes from cascaded victim services introduces additional difficulties. As illustrated in Appendix D.5, Figure 3,existing methods can confuse injected root causes with downstream affected services, motivating more explicit root–victim discrimination.

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.

To address the above limitations, we propose KLD, a kinetic-aware cross-modal graph framework for multimodal RCA. We treat logs as semantic fault cues, traces and metrics as runtime physical signals, and service call graphs as dynamic fault propagation structures. Instead of static topology or fixed threshold heuristics, our model captures temporal signal dynamics, aligns heterogeneous modalities via cross-attention, filters noise through co-gating modules, and enables precise root localization by combining dependency-aware graph with margin-based contrastive learning.

The main contributions of this work are outlined as follows:

- We introduce an evolution- and runtime-dependency-aware view of dynamic microservice diagnosis, moving beyond state-centric multimodal fusion by jointly modeling how monitoring signals evolve over time and how fault evidence propagates over the dependency structure observed at runtime.
- We develop KLD to operationalize this view through kinetic and trajectory-aware representations of metrics and traces, log-conditioned cross-modal co-gating, runtime dependency propagation, and margin-based root–victim discrimination, enabling joint anomaly detection and root-cause localization.
- Extensive experiments on two microservice benchmarks show that this design consistently improves both anomaly detection and root-cause ranking. Controlled ablations isolate the contributions of kinetic modeling and runtime-observed dependencies, while event-level and perturbation experiments characterize the reliability and operating boundaries of KLD.

2 Related Work

2.1 Single-Modal Diagnosis Methods

Single-modal RCA approaches diagnose faults solely based on either logs or traces. MicroIRC builds heterogeneous weighted topologies and adopts personalized random walks to realize fine-grained instance-level localization. AetherLog Cui et al. (2025) is a log-only root cause analysis framework that does not explicitly exploit multi-modal complementary evidence to capture hidden cascading faults. LogReasoner Ma et al. (2026) extracts fault semantics from raw logs via pre-trained large language models. However, these methods only access partial system signals and overlook the complementary information carried by logs, traces and metrics, resulting in poor capability to distinguish subtle cascading faults.

2.2 Multi-Modal Diagnosis Methods

To address information limitations of single-modal inputs, existing methods fuse logs, traces and metrics under multi-task or self-supervised learning to learn unified representations. FAMOS Duan et al. (2025) achieves full three-modality fusion yet assumes static service topologies. It lacks modules to model dynamic invocation shifts and metric drift and struggles to separate genuine root faults from cascaded secondary anomalies. AMER-RCL adopts multi-agent recursive LLM reasoning and agentic memory to reuse historical diagnosis results. However, its memory only stores textual records without structured multi-modal feature cache and cannot fully exploit latent fault propagation paths. AMuLSys Wang et al. (2025) leverages basic kinetic temporal modeling to partially ease these issues, but remains limited in precise root cause discrimination.

2.3 Graph-Aware Causal RCA Methods

Graph-based RCA techniques explicitly model service invocation topology to infer fault propagation paths. Eadro introduces graph attention networks into multi-modal pipelines for joint fault detection and localization. MODIFy designs dedicated feature encoders to align multi-modal signals before graph reasoning. LagRCA is an inference framework built upon multi-latency causal graphs as structural priors, yet it lacks sufficient robustness for topology adaptation and fails to capture emerging runtime-observed dependency paths in a timely manner. Despite their progress on causal inference, these approaches lack temporal kinetic modeling for drifting KPIs and neglect the sparsity and noise inherent to real-world sampled traces and incomplete logs.

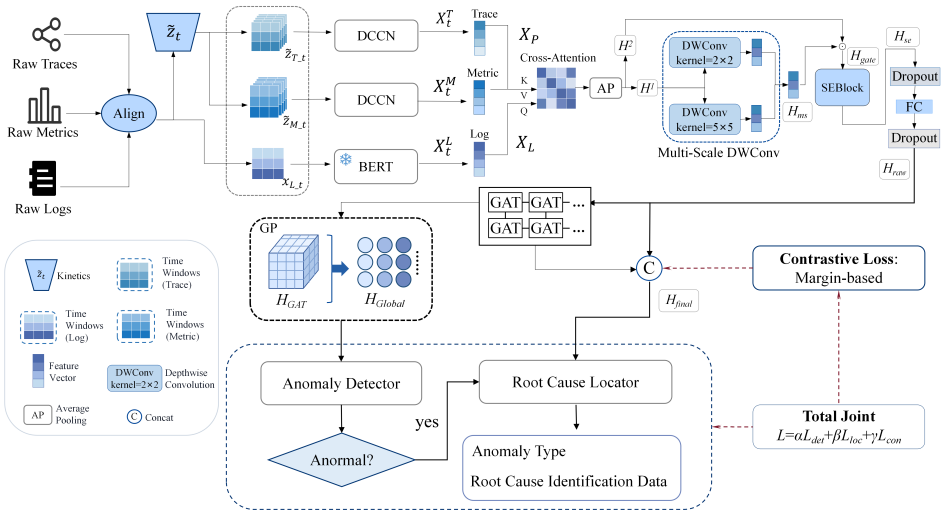


Figure 1: Overview of KLD. Logs, traces, and metrics are aligned, fused, propagated over a dynamic graph, and jointly optimized for detection and RCA ranking.

3 METHOD

As shown in Figure 1, KLD implements an evolution- and dependency-aware diagnosis pipeline that progressively models temporal signal evolution, cross-modal semantic–physical consistency, runtime dependency propagation, and root–victim discrimination.

3.1 KINETIC-AWARE FEATURE LEARNING

3.1.1 LOG AND METRIC PREPROCESSING

We use a frozen BERT-large Devlin et al. (2019) encoder for logs, representing each sanitized message with its normalized 1024-dimensional CLS embedding and truncating inputs beyond 512 WordPiece tokens. Log embeddings are frequency-weighted and averaged within each service-level 10-second window, then projected to 16 dimensions; empty log windows use zero vectors. We retain seven KPIs, fill missing log/trace entries with zeros, and keep metric windows only when all ten second-level samples are available, without interpolation or imputation.

3.1.2 KINETIC FEATURE CONSTRUCTION

Microservice runtime data suffers severe dynamics from frequent instance scaling, shifting workloads, baseline drift Liang et al. (2026) and evolving service dependencies. We align logs, traces, metrics, and fault records onto a unified second-level timeline. Logs replace addresses, numeric IDs, and UUIDs with placeholders. Metrics retain seven CPU, memory, and network KPIs, which are standardized. Traces provide second-level latency, caller–callee relationships, and RPC counts for graph construction.

For an m-dimensional time series

$$X = \{x_1, x_2, \ldots, x_T\},\ x_t \in \mathbb{R}^m.$$

the first-order temporal difference captures velocity-style fluctuations:

$$\Delta x_t = x_t - x_{t-1}. \tag{1}$$

The second-order difference models acceleration dynamics and pinpoints sudden state shifts:

$$\Delta^2 x_t = \Delta x_t - \Delta x_{t-1} = x_t - 2x_{t-1} + x_{t-2}. \tag{2}$$

We concatenate three components to form the final kinetic embedding at time t:

$$z_t = [x_t; \Delta x_t; \Delta^2 x_t]. \tag{3}$$

Here $[\,;\,]$ denotes concatenation. The state, trend, and transition intensity emphasize sharp fault responses over smooth low-order drift Sun et al. (2026).

3.1.3 Trajectory Geometry Representation

To complement local differences with trajectory geometry Zha et al. (2026), we treat X as a feature-space path and compute its normalized tangent:

$$u_t = \frac{x_t - x_{t-1}}{\|x_t - x_{t-1}\|_2 + \epsilon}. \tag{4}$$

where ϵ is a small constant for numerical stability. The tangent displacement is then defined as

$$d_t = \|u_t - u_{t-1}\|_2. \tag{5}$$

which serves as a curvature surrogate. A large d_t indicates an abrupt turning point of the temporal trajectory, often corresponding to state transitions such as traffic surges, resource contention, timeout propagation, or recovery.

Combining kinetic and geometric information, the enhanced temporal representation is

$$\tilde{z}_t = [x_t; \Delta x_t; \Delta^2 x_t; u_t; d_t]. \tag{6}$$

These position, velocity, acceleration, direction, and curvature features distinguish smooth drift from sharp fault evolution.

We apply the same feature extraction pipeline to KPI metrics and trace latency sequences to ensure cross-modal consistency. The resulting representations are encoded by dilated causal convolution:

$$X_t = \mathrm{DCCN}_\theta(\tilde{z}_{\leq t}). \tag{7}$$

where causality preserves temporal ordering and dilation enlarges the receptive field without reducing temporal resolution.

For log events, we use a semantic encoder followed by a linear projection:

$$X_t^L = W \cdot \mathrm{Encoder}_{LM}(x_{L\text{-}t}). \tag{8}$$

where W maps log representations into the shared latent space for subsequent multimodal alignment and interaction.

3.2 Enhanced Cross-Modal Co-Gating

Cross-modal attention aligns log semantics X_L with trace and metric representations $X_P = [X_T; X_M]$ Chowdhury et al. (2026); Lin et al. (2024):

$$Q = X_L W_Q, \quad K = X_P W_K, \quad V = X_P W_V. \tag{9}$$

$$H_{\mathrm{att}} = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V. \tag{10}$$

H_{att} quantifies the correlation between trace/metric fluctuations and log semantics. We aggregate attention outputs via average pooling:

$$H_{\mathrm{ap}} = \mathrm{AP}(H_{\mathrm{att}}). \tag{11}$$

The pooled representation branches into two cooperative streams:

$$H^1 = f_1(H_{\mathrm{ap}}), \quad H^2 = f_2(H_{\mathrm{ap}}). \tag{12}$$

where H^1 retains primary fault-related feature responses, and H^2 supplies auxiliary modulation signals for feature selection.We further encode H^1 with multi-scale depthwise convolutions:

$$H_2 = \mathrm{DWConv2} \times 2(H^1), \tag{13}$$

$$H5 = \mathrm{DWConv5} \times 5(H^1). \tag{14}$$

The 2×2 kernel captures short-range burst anomalies such as sudden latency spikes and log bursts, while the 5×5 kernel models long-range temporal dependencies like inter-service fault propagation. Multi-scale features are fused as:

$$H\mathrm{ms} = \mathrm{Fuse}(H_2, H_5). \tag{15}$$

where $\mathrm{Fuse}(\cdot)$ is implemented as channel concatenation followed by linear projection. H^1 preserves dominant cross-modal fault embeddings, and H^2 generates soft modulation weights for collaborative feature filtering. The gated feature representation is formulated as:

$$H_{\mathrm{gate}} = \phi(H_{\mathrm{ms}}) \odot \sigma(H^2). \tag{16}$$

Here $\odot$ denotes element-wise multiplication, $\sigma(\cdot)$ represents the sigmoid activation, and $\phi(\cdot)$ is a learnable nonlinear projection. This gate amplifies feature channels jointly validated by log semantics and physical runtime signals, while suppressing noise originating from routine logs and sparse traces Li et al. (2024); Zhang et al. (2026b).

A squeeze-and-excitation (SE) block suppresses residual channel redundancy and strengthens fault-relevant responses:

$$H\mathrm{se} = \mathrm{SE}(H_{\mathrm{gate}}). \tag{17}$$

Finally, a lightweight feed-forward layer with dropout regularizes and projects features to obtain the final multi-modal embedding:

$$H_{\mathrm{raw}} = \mathrm{Dropout}\left(\mathrm{FC}\left(\mathrm{Dropout}(H_{\mathrm{se}})\right)\right). \tag{18}$$

3.3 Runtime Dependency Graph Propagation

Each node has a raw multimodal representation $r_i \in \mathbb{R}^d$. At each time window t, we reconstruct the runtime-observed service dependency graph $G_t = (V_t, E_t)$ directly from parent–child trace references. KLD does not learn or infer the graph topology itself; rather, it propagates multimodal node representations over the dependency structure observed in the current window. Multi-head graph attention learns neighbor importance, aggregates projected neighbor features, and applies max pooling across $K = 4$ heads. Appendix B.2 gives the complete equations and variable definitions. After stacked graph layers, global attention pooling supports detection, while concatenation retains local evidence for localization:

$$H_{\mathrm{Global}} = \sum_{i=1}^{N} \pi_i h_i, \quad \pi_i = \frac{\exp(\mathbf{w}_g^\top h_i)}{\sum_{j=1}^{N} \exp(\mathbf{w}_g^\top h_j)}, \quad H_{\mathrm{final}} = H_{\mathrm{GAT}} \| H_{\mathrm{raw}}. \tag{19}$$

Here $H_{\mathrm{GAT}} \in \mathbb{R}^{N \times d_g}$, $H_{\mathrm{raw}} \in \mathbb{R}^{N \times d_r}$, and $H_{\mathrm{final}} \in \mathbb{R}^{N \times (d_g + d_r)}$. The detector uses $\hat{y}_{\mathrm{anom}} = \mathrm{MLP}_{\mathrm{det}}(H_{\mathrm{Global}})$; the localization head computes $s_i = \mathrm{MLP}_{\mathrm{loc}}(H_{\mathrm{final},i})$. Concatenation preserves weak local fault evidence that message passing may attenuate.

3.4 Joint Multi-Task Contrastive Optimization

Built upon node-level H_{final} and graph-level H_{Global}, KLD unifies system anomaly detection and node-level root localization under one multi-task training paradigm Xin et al. (2025). A lightweight MLP takes H_{Global} to predict system anomaly probability $\hat{y}_t$. We supervise global fault classification with binary cross-entropy:

$$\mathcal{L}_{\mathrm{det}} = -\frac{1}{N} \sum_{t=1}^{N} \big(y_t \log \hat{y}_t + (1 - y_t) \log(1 - \hat{y}_t) \big). \tag{20}$$

where N is the sample count, $y_t \in \{0, 1\}$ denotes the ground-truth system label, and $\hat{y}_t$ is the predicted anomaly likelihood. For root localization, we feed H_{final} into a scoring head, $s_i = \mathrm{MLP}_{\mathrm{loc}}(H_{\mathrm{final},i})$. During inference, raw node scores are normalized with fixed temperature $\tau = 0.2$:

$$p_i = \mathrm{softmax}\left(\frac{s_i}{\tau}\right). \tag{21}$$

During training, the localization cross-entropy is computed from temperature-scaled logits and maximizes the probability of ground-truth root i^*:

$$\mathcal{L}_{\text{loc}} = -\frac{1}{N}\sum_{t=1}^{N}\log\frac{\exp(s_{t,i_t^*}/\tau)}{\sum_{j\in\mathcal{V}_t}\exp(s_{t,j}/\tau)}. \tag{22}$$

where s_{t,i_t^*} is the raw score of the true root node. To resolve ambiguous cascaded faults, we introduce margin-based contrastive loss over the propagated node embeddings H_{GAT} to enlarge latent-space distances between the root and non-root nodes Chopra et al. (2005); Lin et al. (2024):

$$\widehat{\mathbf{h}}_{t,v} = \frac{\mathbf{h}_{t,v}}{\|\mathbf{h}_{t,v}\|_2 + \epsilon}, \qquad v \in \mathcal{V}_t. \tag{23}$$

$$\mathcal{L}_{\text{con}} = \frac{1}{|\mathcal{B}_f|}\sum_{t\in\mathcal{B}_f}\frac{1}{|\mathcal{V}_t|-1}\sum_{\substack{j\in\mathcal{V}_t\\ j\neq y_t}}\left[m - \left\|\widehat{\mathbf{h}}_{t,y_t} - \widehat{\mathbf{h}}_{t,j}\right\|_2\right]_+, \tag{24}$$

Before computing the margin-based separation loss, we apply L_2 normalization to all active node representations. This removes embedding-scale ambiguity and prevents the objective from being trivially minimized by increasing representation norms. Consequently, the loss enforces directional separation between the root-cause node and active non-root nodes on the unit hypersphere. We aggregate all losses into a unified multi-task objective:

$$\mathcal{L} = \alpha\mathcal{L}_{\text{det}} + \beta\mathcal{L}_{\text{loc}} + \gamma\mathcal{L}_{\text{con}}. \tag{25}$$

where fixed weights $\alpha = 0.5$, $\beta = 0.4$, and $\gamma = 0.1$ balance global anomaly identification, per-node root ranking, and contrastive separation, respectively.

4 Experiments

To comprehensively evaluate the proposed collaborative optimization framework, we conduct extensive experiments to answer the following research questions:

- **RQ1:** Does joint evolution- and dependency-aware modeling improve anomaly detection and root-cause ranking over existing diagnosis methods?
- **RQ2:** How do multimodal inputs, cross-modal co-gating, dependency-aware graph propagation, and contrastive optimization contribute to the overall performance?
- **RQ3:** How robust and efficient is KLD under controlled perturbations that emulate noisy logs, sparse traces, topology variations, and metric drift?

4.1 Experimental Setup

4.1.1 Dataset

Our datasets (Appendix C.1, Table 5) are built on two reproducible open-source microservice systems, SocialNetwork Delimitrou & Sanchez (2019) and TrainTicket Zhou (2020). We consider CPU load, network delay, and packet loss, injected through computational stress or network emulation. Each fault is applied once to every candidate root-cause service per round, giving 36 injections for SN and 81 for TT; two independent rounds yield 72 and 162 injections, respectively. Our evaluation targets run-level generalization across independent injections, not zero-shot transfer to unseen services or fault categories.

Implementation and leakage control. We use Ubuntu 22.04.4, eight A800-SXM4-40GB GPUs, batch size 256, and learning rate 0.001. Complete source runs are split 70/15/15 before generating 10-second windows with 10-second stride; normalization uses training data only. Five repetitions fix the partition and vary training randomness. Baselines share partitions, labels, graph metadata, and evaluation protocols with modality-appropriate inputs. Appendix C.3 gives all hyperparameters and leakage controls.

Table 1: Window-level anomaly detection performance.

Level	Method	SN			TT		
		Precision	Recall	F1	Precision	Recall	F1
Window	AMuLSys	0.874±0.021	0.845±0.019	0.859±0.017	0.814±0.023	0.784±0.021	0.799±0.020
	Eadro	0.904±0.015	0.909±0.014	0.906±0.013	0.834±0.019	0.825±0.018	0.829±0.017
	MODIFy	0.914±0.014	0.937±0.012	0.925±0.013	0.860±0.018	0.824±0.019	0.842±0.017
	AetherLog	0.942±0.011	0.930±0.012	0.936±0.011	0.975±0.009	0.963±0.010	0.969±0.009
	FAMOS	0.964±0.010	0.954±0.011	0.958±0.010	0.874±0.016	0.788±0.020	0.829±0.017
	KLD	**0.994±0.005**	**0.997±0.003**	**0.995±0.004**	**0.991±0.006**	**0.995±0.004**	**0.993±0.005**

4.1.2 Evaluation Metrics

We evaluate anomaly detection using Precision, Recall, and F1 Du et al. (2017), which provide a balanced assessment of binary anomaly detection performance and using Hit Ratio@K (HR@K) and Normalized Discounted Cumulative Gain (NDCG@K), which measure the ranking quality of fine-grained root-cause candidate Lee et al. (2023).

4.2 Overall Performance (RQ1)

4.2.1 Anomaly Detection Results

Table 1 summarizes window-level anomaly-detection performance. Under window-level evaluation, KLD achieves F1 scores of 0.995 ± 0.004 on SN and 0.993 ± 0.005 on TT. The strongest baselines obtain 0.958 (FAMOS) and 0.969 (AetherLog), respectively; KLD therefore improves F1 by 3.7 and 2.4 percentage points. On SN, its precision and recall exceed the corresponding best baseline values by 3.0 and 4.3 points. On TT, the gains over AetherLog are 1.6 points in precision and 3.2 points in recall. The simultaneous improvement of both quantities indicates that the F1 gain is not obtained by merely shifting the detection threshold toward either fewer false alarms or more aggressive positive predictions. The small standard deviations across five repetitions further show that this behavior is stable under different model initializations. To account for correlations among windows from an injection episode, Appendix D.1 further reports a separate event-level aggregation evaluation.

4.2.2 Root Cause Localization Results

Table 2 further evaluates whether the detected anomaly can be traced to the injected root service. At window level, LagRCA is the strongest competing method in HR@1, reaching 0.921 on both datasets, while KLD obtains 0.995 ± 0.004 on SN and 0.994 ± 0.005 on TT. These correspond to absolute improvements of 7.4 and 7.3 percentage points. The advantage remains across larger candidate sets: KLD reaches HR@3/HR@5 of 0.997/0.997 on SN and 0.995/0.996 on TT. Its NDCG@3 improvements over LagRCA are 6.5 points on SN and 6.8 points on TT, showing that the true root is not only covered by the candidate set but is consistently promoted toward its top. This result is consistent with the intended roles of runtime dependency propagation and contrastive separation: the graph transmits cross-service evidence, while the margin objective discourages downstream victim nodes from occupying the same representation region as the injected root. Their individual contributions are examined separately in RQ2. Appendix D.1 provides the complementary event-level aggregation result.

4.3 Ablation Study (RQ2)

4.3.1 Ablation on Single Modalities

As illustrated in Table 3, dropping logs yields SN HR@1=0.877 (F1=0.979) and TT HR@1=0.892 (F1=0.971), as log semantics supply critical contextual evidence for fault localization. Excluding traces leads to mild performance drops. SN achieves HR@1=0.880 (F1=0.980) and TT HR@1=0.893 (F1=0.951), since traces capture invocation dependencies and fault propagation paths. Removing metrics incurs the most severe degradation, with SN HR@1 falling to 0.789 (F1=0.965) and TT HR@1 plummeting to 0.471 (F1=0.923), suggesting that metrics provide important quan-

Table 2: Window-level root cause localization performance on the SN and TT datasets.

Dataset	Level	Method	HR@1	HR@3	HR@5	NDCG@3	NDCG@5
SN	Window	MicroIRC	0.759±0.022	0.864±0.018	0.928±0.014	0.827±0.020	0.884±0.016
		Eadro	0.810±0.020	0.878±0.017	0.900±0.016	0.850±0.018	0.899±0.015
		AMER-RCL	0.865±0.017	0.944±0.012	0.963±0.010	0.927±0.013	0.931±0.012
		MODIFy	0.878±0.016	0.917±0.015	0.965±0.009	0.902±0.014	0.921±0.013
		LagRCA	0.921±0.012	0.939±0.011	0.984±0.007	0.932±0.010	0.954±0.009
		KLD	**0.995±0.004**	**0.997±0.003**	**0.997±0.002**	**0.997±0.003**	**0.997±0.002**
TT	Window	MicroIRC	0.704±0.024	0.822±0.021	0.904±0.016	0.755±0.023	0.878±0.017
		Eadro	0.757±0.021	0.833±0.020	0.896±0.017	0.803±0.020	0.824±0.019
		AMER-RCL	0.760±0.020	0.891±0.016	0.935±0.012	0.857±0.017	0.884±0.015
		MODIFy	0.816±0.018	0.863±0.017	0.923±0.013	0.841±0.018	0.917±0.014
		LagRCA	0.921±0.013	0.948±0.011	0.953±0.010	0.927±0.012	0.936±0.011
		KLD	**0.994±0.005**	**0.995±0.004**	**0.996±0.003**	**0.995±0.004**	**0.996±0.003**

Table 3: Modality and core-module ablation results on SN and TT.

Dataset	Metric	Modality			Module			
		w/o Log	w/o Trace	w/o Metric	w/o Dynamic Graph	w/o Graph	w/o L_{con}	w/o CMCG
SN	Pre	0.981±0.004	0.983±0.002	0.972±0.006	0.884±0.004	0.885±0.002	0.976±0.003	0.924±0.006
	Rec	0.977±0.003	0.978±0.004	0.958±0.007	0.879±0.005	0.849±0.003	0.970±0.002	0.898±0.007
	F1	0.979±0.003	0.980±0.003	0.965±0.005	0.882±0.003	0.867±0.001	0.973±0.002	0.911±0.005
	HR@1	0.877±0.001	0.880±0.002	0.789±0.012	0.887±0.003	0.858±0.003	0.948±0.002	0.891±0.002
TT	Pre	0.975±0.002	0.958±0.004	0.936±0.001	0.892±0.004	0.888±0.002	0.983±0.005	0.927±0.035
	Rec	0.967±0.002	0.944±0.003	0.910±0.002	0.870±0.003	0.864±0.002	0.977±0.004	0.905±0.029
	F1	0.971±0.001	0.951±0.003	0.923±0.000	0.881±0.003	0.876±0.001	0.980±0.004	0.916±0.032
	HR@1	0.892±0.001	0.893±0.001	0.471±0.007	0.894±0.001	0.835±0.001	0.957±0.001	0.960±0.002

titative evidence for localization. Overall, eliminating any modality hurts performance, confirming logs, traces and metrics offer complementary fault information.

4.3.2 Ablation on Core Model Components

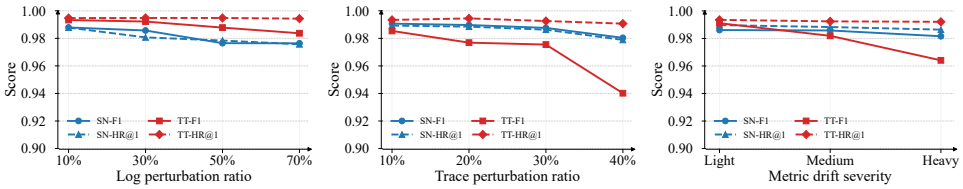
Removing the cross-modal cooperative gating (CMCG) breaks multi-modal alignment and fusion. Removing the contrastive loss L_{con} substantially degrades localization performance, supporting its role in discriminating genuine root-cause services from downstream affected services. Replacing the dynamic graph with a static one (w/o Dynamic Graph) degrades performance since static topology fails to capture time-varying service invocation relations; further removing the entire graph-propagation branch (w/o Graph, no-graph setting) discards all structural dependency evidence for fault-cascade modeling. As shown in the ablation table, static dependency graph performs better than the graph-free baseline, whereas the proposed dynamic graph G_t achieves superior results by modeling time-evolving service dependencies.

4.4 Robustness Analysis (RQ3)

Figure 2 evaluates controlled corruption of model-ready log and trace tensors and gradual metric drift, rather than full production reconfiguration. Log and trace perturbations are isolated, resampled per window and training iteration, and preserve tensor dimensions; trace perturbations affect only non-root nodes and replace edges without changing their count. Appendix C.4 specifies the full procedure.

4.4.1 Robustness against Noisy Log Inputs

KLD is stable under severe log corruption. On SN, F1 remains between 0.976 and 0.987 as the log-noise ratio increases from 10% to 70%. On TT, F1 remains between 0.984 and 0.993. HR@1 also stays high, with SN consistently above 0.975 and TT above 0.994. This result suggests that dependency-aware node representations remain reliable for root-cause ranking.



(a) Log noise (b) Trace perturbation (c) Metric drift

Figure 2: Robustness trends on SN and TT.

Table 4: Inference speed comparison on the Train-Ticket dataset.

Method	Seconds / Query	GPU Mem(MB)	Train/Epoch(s)
AMER-RCL	53.86±1.37	346.01±0.00	64.58±2.71
AetherLog	2.30±0.14	321.67±1.04	56.33±3.69
KLD	0.20±0.01	272.12±0.00	38.82±1.28

4.4.2 Robustness to Dynamic Topology Changes

Trace perturbation is the most challenging setting, especially on TT. SN remains robust, with F1 above 0.980 and HR@1 above 0.978 under all trace perturbation ratios. TT keeps strong localization robustness, with HR@1 above 0.990 even at 40% perturbation, but detection F1 drops from 0.985 at 10% to 0.940 at 40%. This suggests that stronger perturbations to the trace-observed dependency structure mainly affect the anomaly decision boundary, while root-cause ranking remains comparatively stable under the evaluated perturbation setting.

4.4.3 Robustness on Non-Stationary Workload Drift

We simulate daily and promotional traffic drift by applying smooth continuous offsets to normal KPIs while preserving sharp fault spikes, where light, medium and heavy drift correspond to offsets with amplitudes of 10%, 25% and 50% of each metric's standard deviation, respectively. Metric drift brings negligible performance loss: SN sustains F1 over 0.981, and TT reaches 0.991 / 0.981 / 0.964 under light, medium and heavy drift. These results are consistent with the intended role of multi-order kinetic modeling in reducing sensitivity to gradual baseline variations while preserving sharp fault-related changes.

4.4.4 Efficiency Result

We compare the training overhead, per-query inference latency and GPU memory consumption of representative approaches on the Train-Ticket dataset, with results summarized in Table 4. Among the evaluated representative approaches, KLD achieves the lowest inference latency, GPU memory consumption, and per-epoch training time. AetherLog exhibits moderate computational cost, while AMER-RCL incurs substantial inference latency. These results demonstrate the computational efficiency of KLD and indicate its potential suitability for latency-sensitive online diagnosis.

5 Conclusion

This work shows that dynamic microservice diagnosis can benefit from jointly modeling temporal signal evolution and runtime-observed service dependencies. KLD operationalizes this view through kinetic-aware multimodal representation, cross-modal interaction, dependency-aware propagation, and contrastive root–victim discrimination for joint anomaly detection and root-cause localization. Experiments on SN and TT demonstrate consistent gains over the evaluated baselines, while ablations and controlled perturbation studies support the contributions and robustness of the proposed design. The current evaluation is limited to run-disjoint generalization over two systems and three injected fault types; generalization to unseen faults, services, and systems remains future work.

AI USE STATEMENT

In this work, we used generative AI tools for manuscript polishing, grammar correction, and mathematical formula typesetting. We have not used generative AI tools for generating core research ideas, designing model architectures, conducting experimental measurements, or interpreting experimental results, and human-subject-related disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for paraphrasing several paragraphs to improve readability of the draft. We have reviewed all AI-assisted work. All AI-revised manuscript passages were manually cross-checked by all authors to guarantee technical accuracy, logical consistency and avoid unintended plagiarism. No model architecture, experimental code or key conclusions were directly produced by generative AI. 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 paper presents KLD, a multimodal diagnostic framework for dynamic microservice fault detection and root-cause localization. No human subjects are involved in this research. All experimental evaluations are conducted on two publicly released open-source microservice benchmark datasets, SocialNetwork (SN) and TrainTicket (TT). The datasets consist of system monitoring artifacts including logs, metrics and distributed traces, which contain no personally identifiable information, private user data or sensitive real-world production records.

Our fault-injection experimental workflow strictly follows standard cloud-system simulation protocols. The proposed technique targets automated operational diagnosis for cloud microservice platforms; it does not generate harmful outputs, manipulate system behaviours or introduce security risks when deployed as intended. We have carefully examined potential fairness and bias risks: the model performance is evaluated across multiple fault types and services without intentional demographic or system-specific discrimination. There are no conflicts of interest, external commercial sponsorships or proprietary data licensing constraints that could bias the experimental outcomes or research conclusions. All experiments, metric computations and result reporting adhere to standard academic research integrity norms.

REPRODUCIBILITY STATEMENT

To support full reproducibility of our empirical results, key experimental configurations, hyper-parameter settings, evaluation metrics and dataset statistics are documented within the main text (Section 4). The SN and TT benchmarks we adopt are publicly available open-source microservice systems. Full preprocessing pipelines for logs, metrics and traces, together with run-level train-validation-test splitting rules, are described in Appendix C.3. We submit anonymous source code implementing the KLD model as supplementary material, including model architecture, loss-function implementations, evaluation scripts and ablation pipelines. Additional preprocessing, graph propagation, implementation, and perturbation details are provided in Appendices C.2–C.4. All reported numerical results are averaged over five repeated experimental runs with fixed data partitions to mitigate random training stochasticity. Researchers can refer to the above-mentioned main-body sections, appendix and supplementary code to replicate our anomaly-detection and root-cause-localization experiments.

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

REFERENCES

Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pp. 539–546, 2005.

Abdul Monaf Chowdhury, Rabeya Akter, and Safaeid Hossain Arib. T3time: Tri-modal time series forecasting via adaptive multi-head alignment and residual fusion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 20597–20605, 2026.

Tianyu Cui, Ruowei Fu, Changchang Liu, Yuhe Ji, Wenwei Gu, Shenglin Zhang, Yongqian Sun, and Dan Pei. Aetherlog: Log-based root cause analysis by integrating large language models with knowledge graphs. In *2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 49–60. IEEE, 2025.

Christina Delimitrou and Daniel Sanchez. Deathstarbench: A benchmark suite for distributed microservice applications. In *Proceedings of the 14th European Conference on Computer Systems*, pp. 1–16, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1285–1298, 2017.

Chiming Duan, Yong Yang, Tong Jia, Guiyang Liu, Jinbu Liu, Huxing Zhang, Qi Zhou, Ying Li, and Gang Huang. Famos: Fault diagnosis for microservice systems through effective multi-modal data fusion. In *IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 2613–2624, 2025.

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 *IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1750–1762, 2023.

Jiacheng Li, Xinyu Zhang, and Zhi Wang. Logformer: A pre-train and tuning pipeline for log anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 135–143, 2024.

Daojun Liang, Jing Chen, Xiao Wang, Yinglong Wang, and Shuo Li. Deepboots: Dual-stream residual boosting for drift-resilient time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 23391–23400, 2026.

Yu Lin, Kai Feng, and Yang Liu. Gala: Can graph-augmented large language model agentic workflows elevate root cause analysis? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 206–213, 2024.

Lipeng Ma, Yixuan Li, Weidong Yang, Mingjie Zhou, Xinyi Liu, Ben Fei, Shuhao Li, Xiaoyan Sun, Sihang Jiang, and Yanghua Xiao. Logreasoner: Empowering llms with expert-like coarse-to-fine reasoning for automated log analysis. *ACM Transactions on Software Engineering and Methodology*, 2026. doi: 10.1145/3801747. Just Accepted (March 2026).

Junsong Pu, Yichen Li, Zhuangbin Chen, Jinyang Liu, Zhihan Jiang, Jianjun Chen, Rui Shi, Zibin Zheng, and Tieying Zhang. Errorprism: Reconstructing error propagation paths in cloud service systems. *arXiv preprint arXiv:2509.26463*, 2025.

Jacopo Soldani and Antonio Brogi. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys*, 55(3):59:1–59:39, 2022. doi: 10.1145/3501297.

Xin Sun, Heng Zhou, Yuhao Wu, and Chao Li. State-derivative-aware neural controlled differential equations for multivariate time series anomaly detection and diagnosis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 25727–25735, 2026.

Peipeng Wang, Xiuguo Zhang, and Zhiying Cao. Anomaly detection for microservice system via augmented multimodal data and hybrid graph representations. *Information Fusion*, 118:103017, 2025.

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.

Ruyue Xin, Wujian Zhang, and Peng Chen. Chase: A causal heterogeneous graph based framework for root cause analysis in multimodal microservice systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 2025.

Zhenhe Yao, Haowei Ye, Changhua Pei, Guang Cheng, Guangpei Wang, Zhiwei Liu, Hongwei Chen, Hang Cui, Zeyan Li, Jianhui Li, et al. Sparserca: Unsupervised root cause analysis in sparse microservice testing traces. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 391–402. IEEE, 2024.

Yaohua Zha, Xue Yuerong, Chunlin Fan, Yuansong Wang, Tao Dai, Ke Chen, and Shu-Tao Xia. Casl: Curvature-augmented self-supervised learning for 3d anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 12340–12348, 2026.

Shenglin Zhang, Yongxin Zhao, Sibo Xia, Shirui Wei, Yongqian Sun, Chenyu Zhao, Shiyu Ma, Junhua Kuang, Bolin Zhu, Lemeng Pan, et al. No more data silos: Unified microservice failure diagnosis with temporal knowledge graph. *IEEE Transactions on Services Computing*, 17(6): 4013–4026, 2024.

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, 2026a.

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.

Wujian Zhang, Ruyue Xin, Peng Chen, Xi Li, Ang Bian, Yibin Zhao, and Zhiming Zhao. Modify: A multi-modal anomaly diagnosis framework with diffusion-enhanced adaptive fusion in microservices. *Journal of Systems and Software*, 237:112844, 2026b.

Xiang Zhou. Trainticket: An open-source microservice benchmark system. `https://github.com/FudanSELab/train-ticket`, 2020.

A Appendix

B Method Details

B.1 Kinetic and Trajectory Feature Learning

For the synchronized physical signal $x_t \in \mathbb{R}^m$ at window t, KLD combines the observed value with temporal velocity and acceleration:

$$\Delta x_t = x_t - x_{t-1}, \qquad \Delta^2 x_t = \Delta x_t - \Delta x_{t-1}, \qquad z_t = [x_t \| \Delta x_t \| \Delta^2 x_t]. \tag{26}$$

The representation reduces reliance on absolute KPI baselines. To capture the overall shape of a multivariate trajectory, we additionally compute the normalized tangent and its displacement:

$$u_t = \frac{x_t - x_{t-1}}{\|x_t - x_{t-1}\|_2 + \epsilon}, \qquad d_t = \|u_t - u_{t-1}\|_2. \tag{27}$$

Here, d_t is a magnitude-invariant turning surrogate. The same construction is applied to trace latency sequences, and dilated causal convolution encodes the resulting features without accessing future windows.

B.2 Dependency-Aware Graph Propagation

At window t, trace parent–child references define the observed invocation graph $G_t = (\mathcal{V}_t, \mathcal{E}_t)$. Given node features $h_i^{(l)}$, the k-th GATv2 head calculates

$$e_{ij}^{(k)} = \mathbf{a}^{(k)\top} \operatorname{LeakyReLU}\Big(W^{(k)}[h_i^{(l)} \| h_j^{(l)}]\Big), \qquad \alpha_{ij}^{(k)} = \frac{\exp(e_{ij}^{(k)})}{\sum_{u \in \mathcal{N}(i)} \exp(e_{iu}^{(k)})}, \tag{28}$$

followed by neighbor aggregation and max pooling across heads:

$$\tilde{h}_i^{(l+1,k)} = \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(k)} W^{(k)} h_j^{(l)}, \qquad h_i^{(l+1)} = \operatorname{MaxPool}_{k=1}^{4}\left(\tilde{h}_i^{(l+1,k)}\right). \tag{29}$$

The graph-level detector input and node-level localization input are respectively

$$H_{\text{Global}} = \sum_{i \in \mathcal{V}_t} \pi_i h_i, \quad \pi_i = \frac{\exp(w_g^\top h_i)}{\sum_{j \in \mathcal{V}_t} \exp(w_g^\top h_j)}, \quad H_{\text{final}} = H_{\text{GAT}} \| H_{\text{raw}}. \tag{30}$$

Thus, H_{Global} drives anomaly detection, while $H_{\text{final},i}$ preserves both graph context and raw local evidence for service i.

B.3 Joint Optimization

The detector and locator are $\hat{y}_{\text{anom}} = \text{MLP}_{\text{det}}(H_{\text{Global}})$ and $s_i = \text{MLP}_{\text{loc}}(H_{\text{final},i})$. The joint objective is

$$\mathcal{L} = \alpha \mathcal{L}_{\text{det}} + \beta \mathcal{L}_{\text{loc}} + \gamma \mathcal{L}_{\text{con}}, \qquad \mathcal{L}_{\text{con}} = \frac{1}{|\mathcal{V}_t| - 1} \sum_{j \neq y} \max\big(0, m - \|h_y - h_j\|_2\big), \tag{31}$$

where y is the true root service. The selected configuration is kernels $\{2, 5\}$, $\alpha/\beta/\gamma = 0.5/0.4/0.1$, $\tau = 0.2$, and $m = 0.5$.

C Experimental Details

C.1 Datasets and Run-Level Partition

The experiments use SocialNetwork (SN) and TrainTicket (TT), which provide synchronized logs, traces, and metrics under three controlled fault types. Complete independent source runs are split

before window generation, so temporally adjacent windows never cross train, validation, and test sets. Table 5 summarizes the benchmark observations.

Table 5: Statistics of the experimental datasets.

Dataset	Fault types	Logs (K)	Templates	Traces (K)	Metrics
SN	3	98.5	7	2.6	7
TT	3	444.6	60	48.3	7

C.2 Multimodal Preprocessing

We use a frozen BERT-large log encoder with WordPiece tokenization and a maximum length of 512. The normalized 1024-dimensional CLS embedding represents each sanitized message; embeddings are averaged by service and 10-second window, with repeated messages weighted by frequency. Empty log windows and missing trace entries use zero vectors. Seven metric streams—`cpu_usage_system`, `cpu_usage_total`, `cpu_usage_user`, `memory_usage`, `memory_working_set`, `rx_bytes`, and `tx_bytes`—are retained only when all ten second-level samples exist; no interpolation is applied.

C.3 Implementation, Runtime, and Metrics

We train with batch size 256 and learning rate 0.001. All methods share the same run-level split, labels, graph metadata, and evaluation protocol; each repeated experiment varies training randomness while retaining the split. Table 6 lists the runtime environment.

Table 6: Hardware and software environment.

Component	Configuration
Operating system	Ubuntu 22.04.4 LTS
CPU	2× AMD EPYC 7K62, 48 cores each
System memory	1.0 TB RAM
GPU	8× NVIDIA A800-SXM4, 40 GB each
Python	3.10.12
PyTorch / DGL	2.9.1+cu128 / 1.1.3+cu118
NumPy / pandas	1.26.4 / 2.3.3
SciPy / scikit-learn	1.15.3 / 1.5.2
NetworkX	3.4.2

For anomaly detection, we report

$$\text{Precision} = \frac{TP}{TP+FP}, \quad \text{Recall} = \frac{TP}{TP+FN}, \quad \text{F1} = \frac{2\,\text{Precision}\,\text{Recall}}{\text{Precision}+\text{Recall}}. \tag{32}$$

For localization, with true-root rank r_n, we use

$$\text{HR@K} = \frac{1}{N}\sum_{n=1}^{N}\mathbb{I}(r_n \le K), \qquad \text{NDCG@K} = \frac{1}{N}\sum_{n=1}^{N}\frac{\mathbb{I}(r_n \le K)}{\log_2(r_n+1)}. \tag{33}$$

C.4 Perturbation Protocol

All robustness perturbations are applied online to model-ready tensors after parsing, encoding, normalization, and window alignment. Rather than reproducing complete production-level system reconfigurations, we perturb model-ready observations to emulate three common sources of uncertainty: corrupted log representations, incomplete or altered trace dependencies, and gradual metric baseline drift. All perturbations are applied on-the-fly to preprocessed model-ready tensors (after parsing, encoding, normalization and time-window alignment), without regenerating raw logs or

trace spans. For log-noise ratio r, sampled feature entries are split into three groups and corrupted via Gaussian noise, feature augmentation and zero-filling to mimic garbled text, redundant output and missing stack information. For trace-perturbation ratio r, only non-root-cause nodes are perturbed: half get trace features zero-masked, while others are overlaid with mean-value traces plus 5% Gaussian noise to simulate transient node appearance/disappearance. We also replace $r|E_t|$ RPC edges with equal-count non-duplicate random edges, keeping tensor dimensions and total edge count unchanged but rewriting invocation topology. Log and trace perturbations are tested in isolation while other modalities remain untouched; perturbations are freshly resampled per time window and per training iteration. For each perturbation setting, the corresponding corruption is applied during both training and evaluation. Perturbations are independently resampled online, so evaluation uses newly sampled corruptions rather than those observed during training. Accordingly, these experiments assess stability under perturbation-aware training and controlled distribution shifts, rather than zero-shot robustness of a model trained only on clean observations.

D Supplementary Experiments

D.1 Incident-Level Event Aggregation

Window-level scores are the primary benchmark measure because all methods are evaluated on aligned windows. As a complementary check, temporally consecutive windows with the same fault state and root label within a held-out source run are aggregated into one event. Detector and node probabilities are averaged within an event before thresholding and ranking. Table 7 reports mean $\pm$ sample standard deviation over three independently trained seed checkpoints; baseline event predictions are not available. Window scores nevertheless give many votes to a single long fault episode because adjacent 10-second windows overlap heavily. We therefore form an event from temporally contiguous windows of the same source run, average their anomaly probabilities, and apply the same decision threshold to the resulting event score. This stricter unit of analysis gives each independent episode one vote and prevents long faults from dominating the test metric. As reported in Table 7, KLD attains event-level F1 of 0.980 ± 0.008 on SN and 0.969 ± 0.001 on TT, respectively 1.5 and 2.4 percentage points below the window-level scores. On SN, aggregation removes isolated normal-window alarms, yielding precision of 1.000 ± 0.000 with recall of 0.963 ± 0.014. On TT, precision and recall are 0.972 ± 0.001 and 0.965 ± 0.002. This precision–recall trade-off is expected because weak onset and recovery windows can dilute the mean score of a true fault episode. Since equivalent event-level outputs are unavailable for the baselines, these rows assess KLD's own episode-level reliability rather than establish a direct event-level ranking against them.

For event-level localization, node scores are averaged over all windows in an episode before ranking. KLD obtains HR@1/HR@3/HR@5 of 0.943 ± 0.008/0.954 ± 0.006/0.982 ± 0.002 on SN and 0.938 ± 0.012/0.979 ± 0.002/0.986 ± 0.004 on TT. Temporal aggregation therefore introduces uncertainty in the exact first rank, while retaining nearly all root services within the top five. Overall, the results support episode-level reliability under the evaluated run-level split; the claim is limited to independently executed injections of the three evaluated fault types and does not imply zero-shot recognition of unseen services or unseen fault categories.

Table 7: Incident-level KLD evaluation over three seed checkpoints. SN contains 18 events (9 fault and 9 normal); TT contains 36 events (18 fault and 18 normal). Note that incident-level results serve as complementary reference due to limited event samples.

Dataset	Precision	Recall	F1	HR@1	HR@3	HR@5
SN	1.000 ± 0.000	0.963 ± 0.014	0.980 ± 0.008	0.943 ± 0.008	0.954 ± 0.006	0.982 ± 0.002
TT	0.972 ± 0.001	0.965 ± 0.002	0.969 ± 0.001	0.938 ± 0.012	0.979 ± 0.002	0.986 ± 0.004

D.2 Kinetic Modeling Ablation

Removing first-order, second-order, and trajectory-geometric features while retaining the remaining architecture lowers F1 by 1.31 percentage points and HR@1 by 1.33 points on SN (Table 8).

Table 8: Kinetic modeling ablation on SN over five seeds.

Model	Precision	Recall	F1	HR@1	HR@3	HR@5
Kinetic	0.9942±0.0032	0.9973±0.0009	0.9957±0.0012	0.9954±0.0030	0.9972±0.0006	0.9973±0.0009
w/o Kinetic	0.9832±0.0014	0.9820±0.0011	0.9826±0.0009	0.9821±0.0015	0.9824±0.0010	0.9827±0.0009

D.3 Dynamic Graph Ablation

Dynamic G_t is recovered per window from trace-observed parent–child relations; static G_{static} reuses fixed metadata; and w/o Graph retains only self-loops. Table 9 confirms that dynamic invocation evidence consistently surpasses static and graph-free propagation.

Table 9: Ablation of dynamic, static, and no-graph propagation.

Dataset	Graph	Precision	Recall	F1	HR@1	HR@3
SN	Static G_{static}	0.8836	0.8794	0.8815	0.8872	0.8894
SN	Dynamic G_t	0.9938	0.9968	0.9953	0.9948	0.9971
SN	w/o Graph	0.8829	0.8515	0.8669	0.8579	0.8915
TT	Static G_{static}	0.8782	0.8841	0.8811	0.8936	0.8941
TT	Dynamic G_t	0.9909	0.9952	0.9930	0.9942	0.9946
TT	w/o Graph	0.8767	0.8749	0.8758	0.8348	0.8349

D.4 Hyperparameter Sensitivity

The dual-scale kernels $\{2, 5\}$ and selected $\alpha/\beta/\gamma = 0.5/0.4/0.1$, $\tau = 0.2$, and $m = 0.5$ achieve the best balanced result (Table 10).

Table 10: Hyperparameter sensitivity on SN.

Group	Setting	Best F1	HR@1 at best F1	Best HR@1
Kernel	2	0.9878	0.9878	0.9878
Kernel	3	0.9907	0.9900	0.9900
Kernel	5	0.9893	0.9893	0.9893
Kernel	2,5	**0.9954**	0.9940	**0.9943**
Kernel	2,3,5	0.9946	0.9940	0.9930
Loss	0.5/0.4/0.1	**0.9954**	0.9940	**0.9943**
Loss	0.5/0.1/0.4	0.9886	0.9857	0.9857
Loss	0.1/0.5/0.4	0.9882	0.9857	0.9864
Loss	0.1/0.4/0.5	0.9872	0.9857	0.9857
Loss	0.4/0.5/0.1	0.9871	0.9828	0.9893
Temperature	0.1	0.9865	0.9900	0.9900
Temperature	0.2	**0.9954**	0.9940	**0.9943**
Temperature	0.5	0.9871	0.9835	0.9871
Temperature	1.0	0.9857	0.9828	0.9850
Margin	0.5	**0.9954**	0.9940	**0.9943**
Margin	1.0	0.9886	0.9893	0.9893
Margin	1.5	0.9872	0.9821	0.9864
Margin	2.0	0.9865	0.9828	0.9864

D.5 Qualitative Diagnosis and Explanations

Figure 3 compares missed root-cause nodes in one SN trial. Colored points denote correctly localized abnormal services; red crosses denote missed roots. KLD has the fewest missed roots among the methods shown.

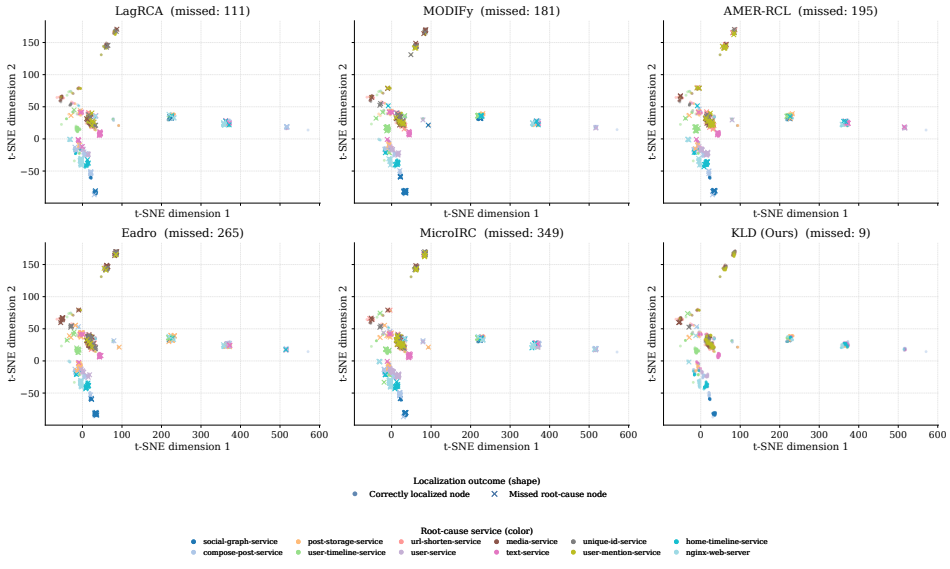


Figure 3: Aggregated comparison of missed root-cause nodes on SN from one experimental run.

For every detected abnormal event, KLD produces the structured record

$$\{\text{id},\ \text{abnormal_type},\ \text{log},\ \text{trace},\ \text{metric},\ \text{reason}\}.$$

The output identifies the service and interval, presents modality-specific evidence, and gives a concise diagnostic rationale. Figure 4 shows the operator summary, modality drill-down, and structured explanation inspector.



Figure 4: Operator-facing diagnosis summary, modality evidence drill-down, and structured explanation inspector.