Microservice Root Cause Analysis With Limited Observability Through Intervention Recognition in the Latent Space

Zhe Xie[1], Shenglin Zhang, Yitong Geng, Yao Zhang, Minghua Ma, Xiaohui Nie, Zhenhe Yao, Longlong Xu, Yongqian Sun, Wentao Li, Dan Pei

1. Presenter. Email: xiez22@mails.tsinghua.edu.cn

Background

Search Engine

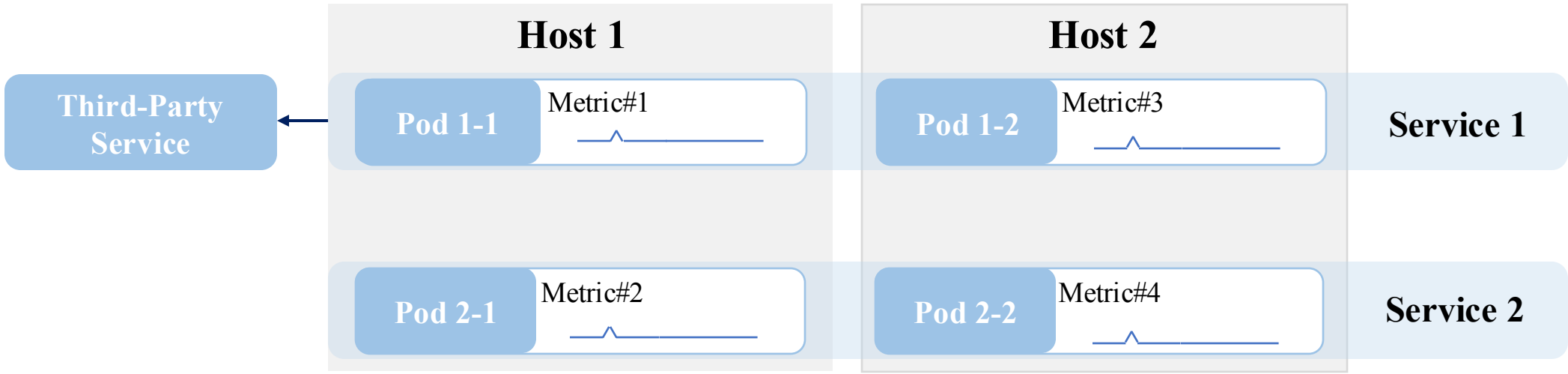
Online Shopping

Social Network

Videos

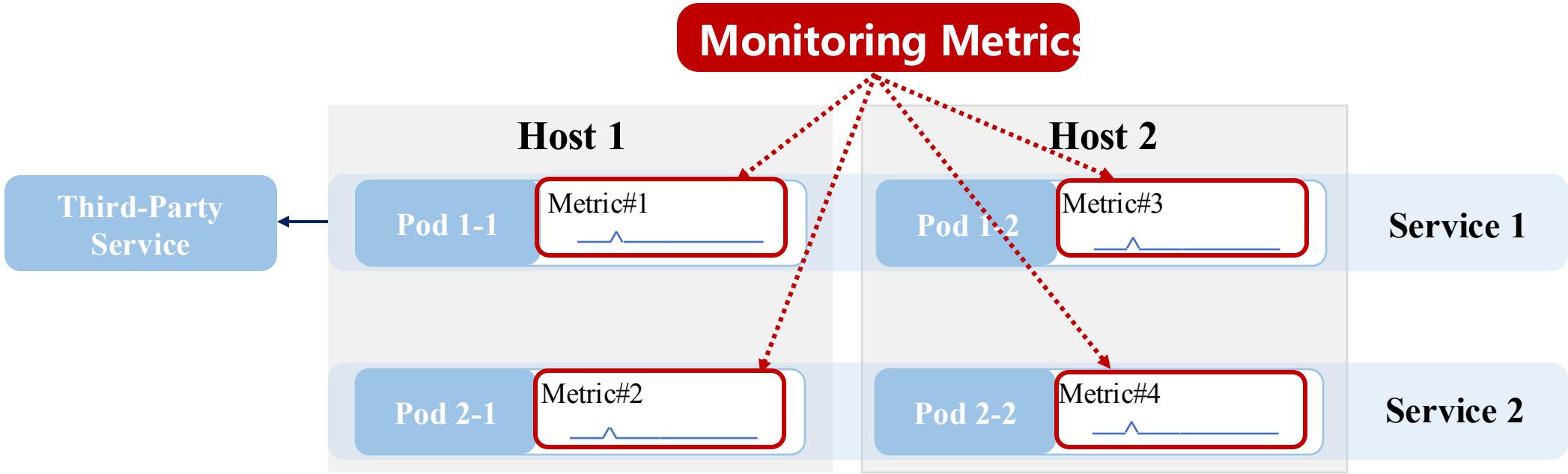
Online applications have been widely used in our daily life

Microservices



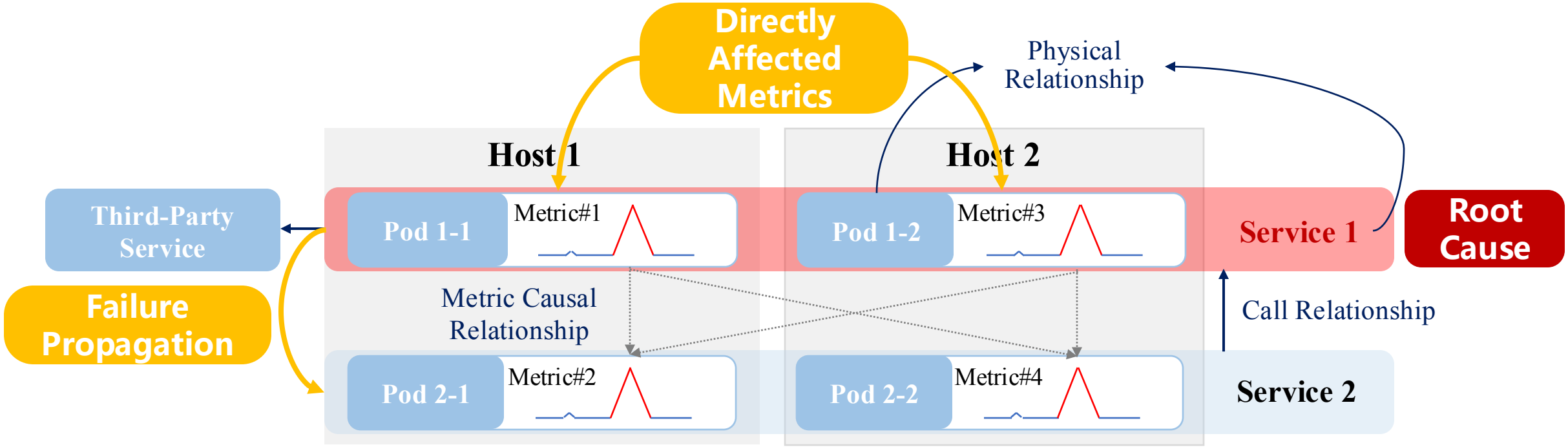
Microservice systems are increasingly being used in online service systems due to their scalability and flexibility

Observability in Microservices



- However, system failures are inevitable due to frequent change and scale expansion of microservices.
- In microservices systems, observability tools are deployed to monitor the system status by collecting data (like metrics)

Failure Propagation in Microservices Systems

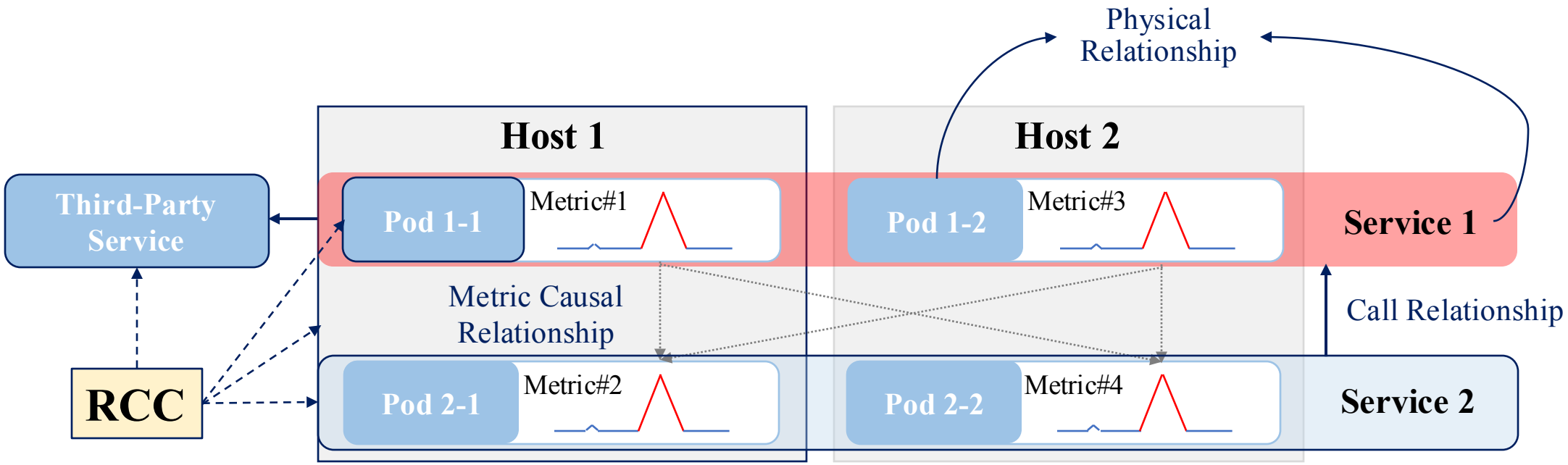


Due to the complex, multi-layered, and heterogeneous components in microservice systems, the situations that may arise when a failure occurs become very complicated.

All metrics are affected by the failure in Service 1:

- Metric#1 and Metric#3 are affected because of the **failure**
- Metric#2 and Metric#4 are affected by **failure propagation**

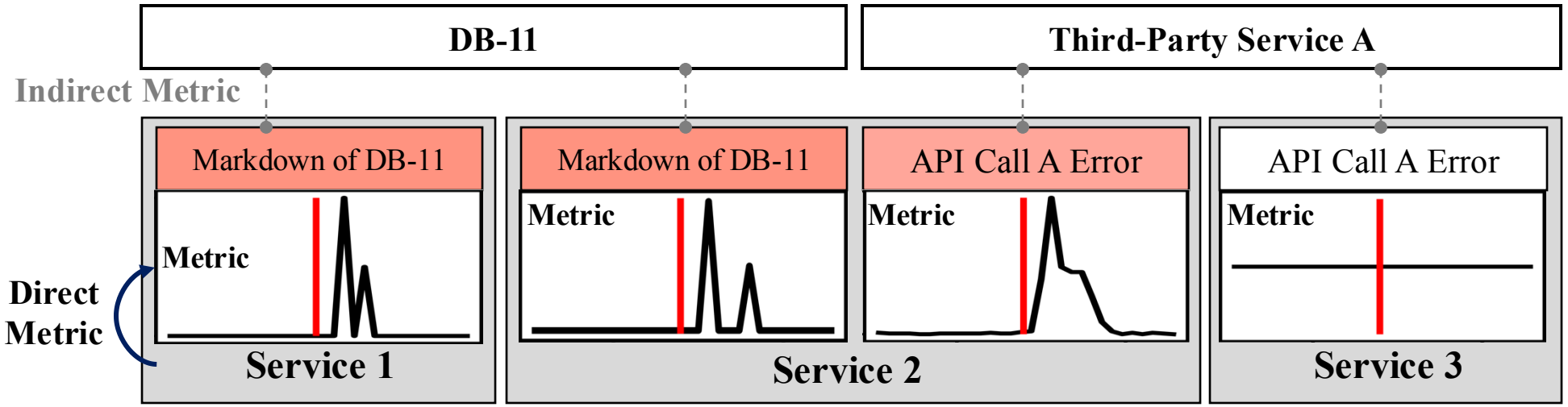
Root Cause Analysis



Root Cause Analysis (RCA)

- Rank the **Root Cause Candidates (RCCs)** according to the observations to identify **the root cause** of the failure

Indirect Metrics of RCC



The same metric might be associated with multiple RCCs at different levels

Direct Metrics of RCC are metrics monitoring on the location or event of RCC **(e.g., "API Call A" in Service 3)**

Indirect Metrics of RCC are direct metrics of other RCCs that can be used to infer its **status (e.g., "API Call A" and Third-Party Service A)**

Limited Observability in Microservices System

Table: Some Common Categories of Root Causes in eBay

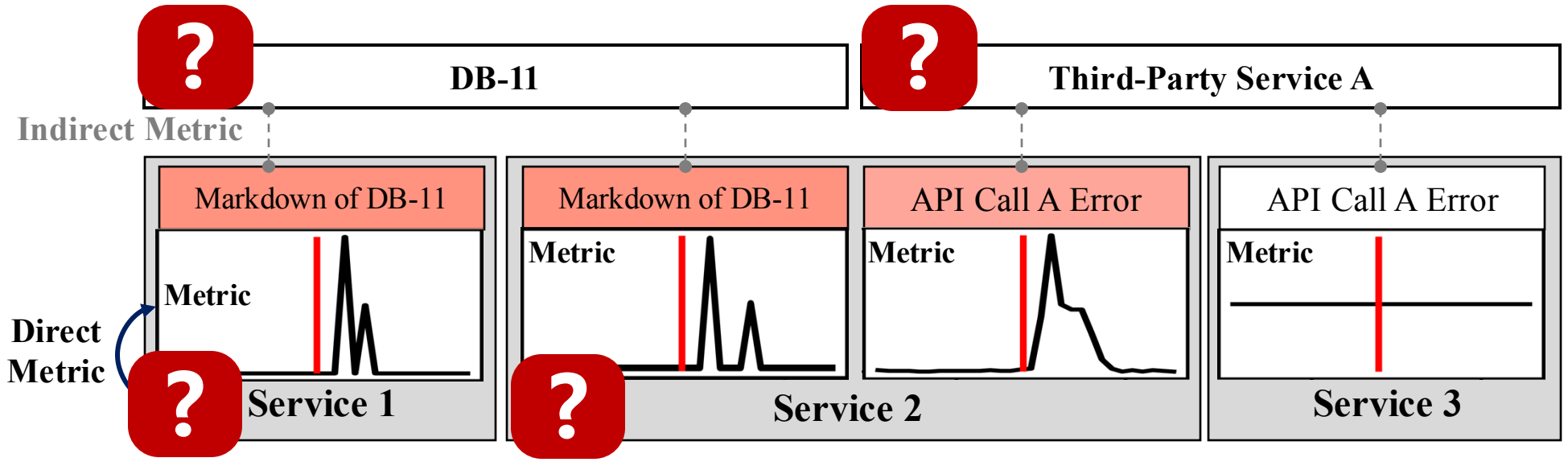
Category	Percentage	Typical Related Metrics
Third-Party Services	63.59%	Third-Party API Error*
Internal Services	8.76%	Runtime Error
Software Change	7.83%	Change Process, Runtime Error*
Database	5.53%	Markdown Error*

(*) Indirect Metrics

Indirect metrics are important for RCA

Many failures (e.g., third-party service failures) require indirect metrics to be used in RCA due to their **limited observability**

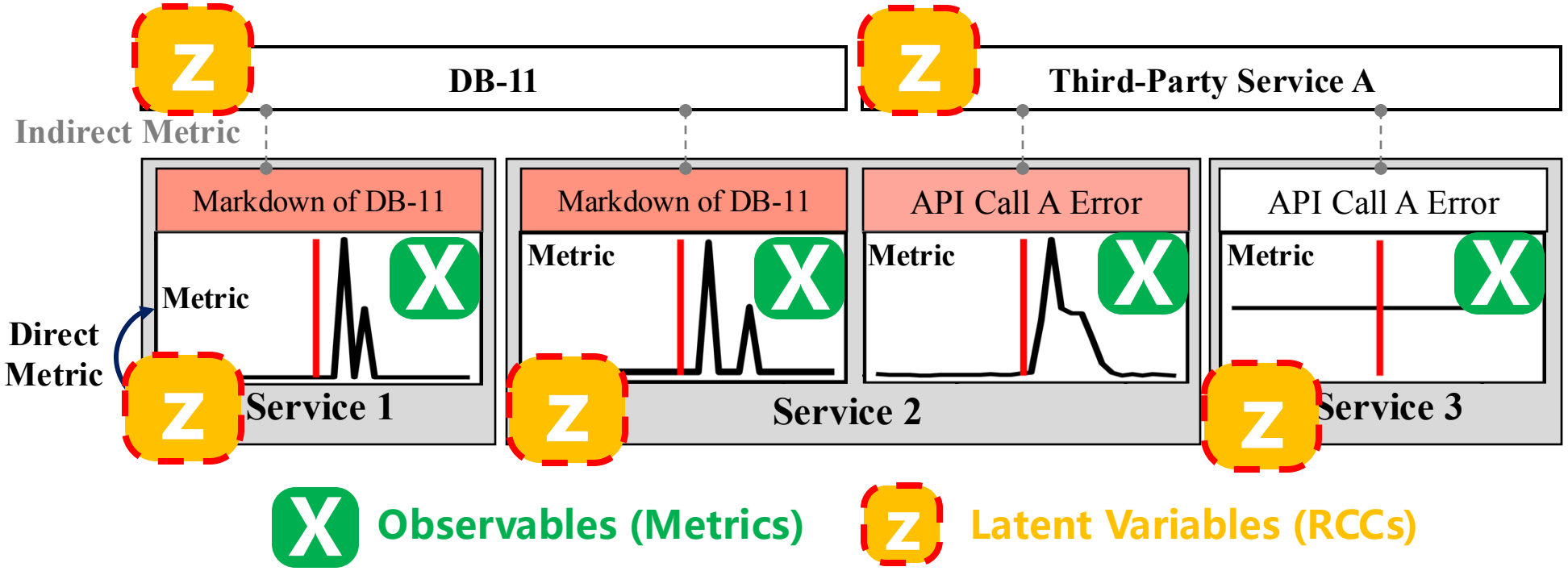
Indirect Metrics of RCC



The status of RCCs are not always visible!

- Indirect Metrics are not accurate enough. In failures, many RCCs are associated with anomalous metrics. **Due to indirect metrics, anomalous metrics are associated with multiple RCCs. Their causal relationships are ambiguous.**

Core Idea: Modeling RCCs as Latent Variables



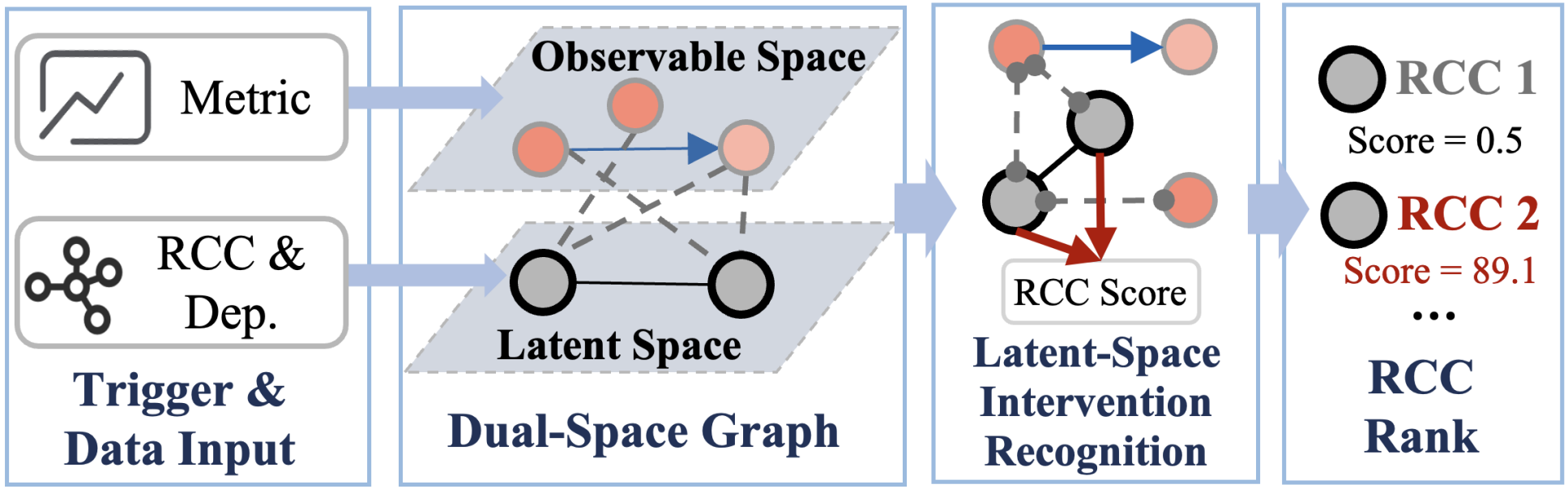
Modeling RCCs as Latent Variables

- Latent variables bypasses the need for complete observability of RCCs
- Transforms metric features extraction into **inferring latent variables**, enabling effective RCC modeling even when complete observability is not feasible

Challenges

- **Challenge 1: The graph construction for both latent variables (RCCs) and the observable variables (metrics).** The relationship between RCCs can be heterogeneous.
- **Challenge 2: Modeling latent RCC variables under limited observability.** Latent variables can only be inferred through related observable variables.
- **Challenge 3: Inference the unobservable latent variables with observable data and rank the RCCs.** An efficient algorithm is required to implement the inference.

LatentScope Overview

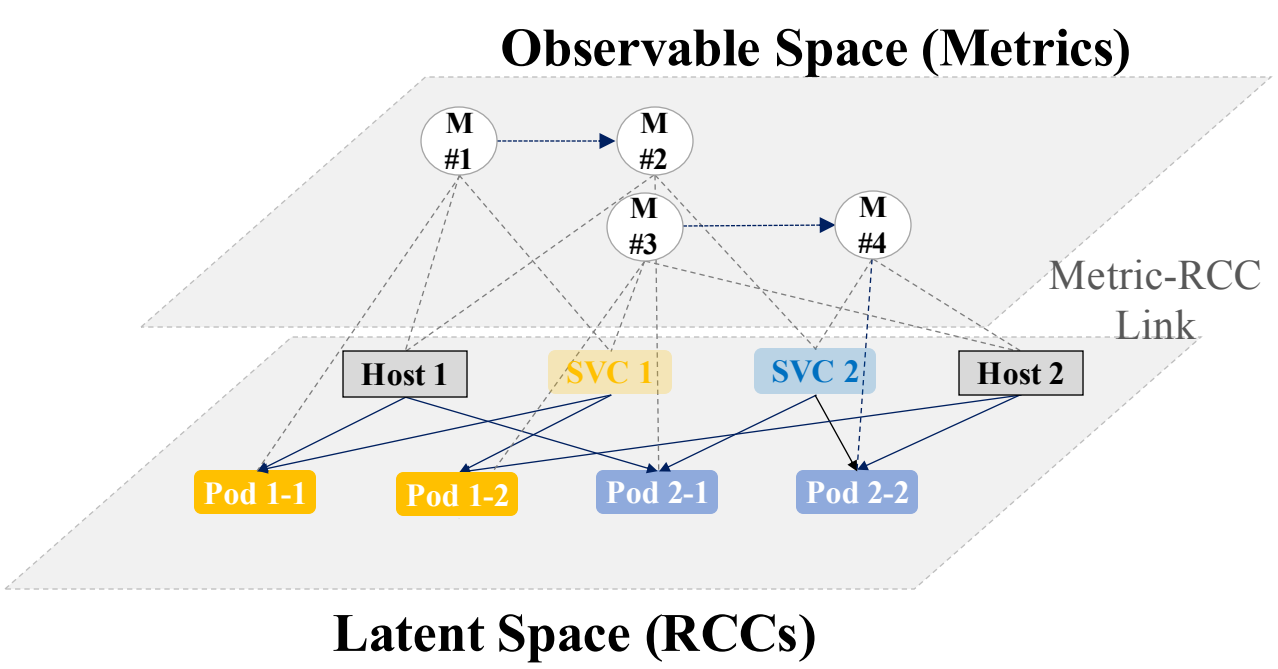


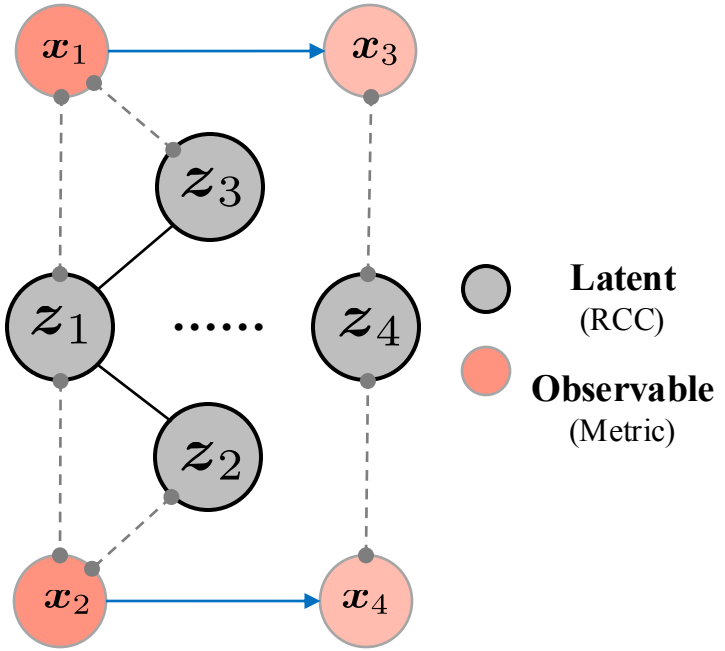
- **LatentScope:** An unsupervised RCA algorithm for microservices system through latent-space intervention recognition on a dual-space graph.

Key designs in LatentScope:

- Dual-Space Graph Construction and Modeling
- Latent-Space Intervention Recognition

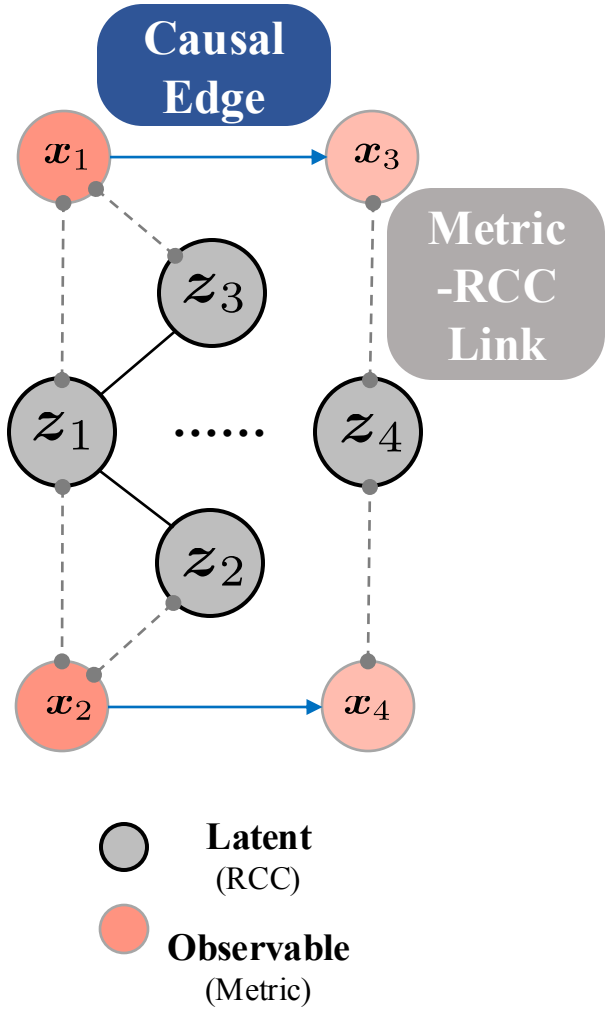
Dual-Space Graph in LatentScope





- **Observable Space:** Metrics and their **causal relationships** **(Pre-Defined Rules, etc.)**
- **Latent Space:** RCCs and their **physical relationships** **(e.g., Pod 1 runs on Host A)**
- **Metric-RCC Link: Many-to-many relationships** between spaces **(Direct & Indirect)**

Modeling of Variables in LatentScope



A monitoring metric may be affected by:

- **Parent Metrics** (in Observable Space, represented by the causal edges)
- **Related RCCs** (from Latent Space, represented by the Metric-RCC links)

$$\boldsymbol{x}_i^{(t)} = f_i(\boldsymbol{pa}(\boldsymbol{x}_i)^{(t)}, \boldsymbol{rcc}(\boldsymbol{x}_i)^{(t)})$$

Current Metric | Parent Metrics | Related RCCs

How to find the root cause?

Intervention Recognition

Intervention Recognition From CIRCA (KDD 2022) [1]

A few faulty data are available

Fault-free data may not cover the given condition

Core Idea: $V_i \in \mathbf{M} \Leftrightarrow P(V_i \mid \mathbf{pa}(V_i), do(\mathbf{m})) \neq P(V_i \mid \mathbf{pa}(V_i))$

Hypothesis Testing

Regression

$\mathbf{H}_0(V_i \notin \mathbf{M})$: $V_i^{(t)} \sim P(V_i^{(t)} \mid \mathbf{pa}^{(t)}(V_i))$

The need for faulty data is reduced

$\approx$

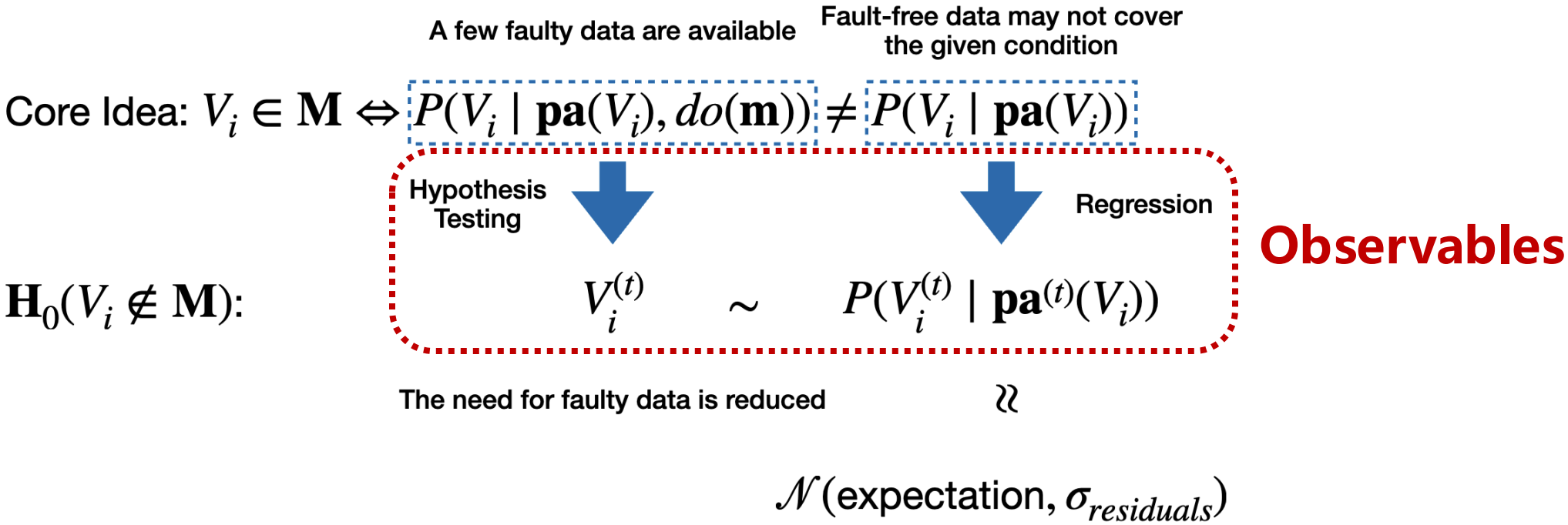
$\mathcal{N}(\text{expectation}, \sigma_{residuals})$

Intervention recognition finds the root cause by identifying interventions through changes in probability distributions, using methods like **Regression-based Hypothesis Testing (RHT).**

[1] Li, Mingjie, et al. "Causal inference-based root cause analysis for online service systems with intervention recognition." *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022.

Intervention Recognition

Intervention Recognition From CIRCA (KDD 2022)[1]



However, RHT (Regression-based Hypothesis Testing) **requires variables to be observable**, which cannot be used for intervention recognition in latent space.

[1] Li, Mingjie, et al. "Causal inference-based root cause analysis for online service systems with intervention recognition." *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022.

Latent-Space Intervention Recognition

- **Theorem:** The **intervention score** of RCC z_a can be calculated with:

$$score_e(z_a) = \max_{i, z_a \in rcc_i} \max_t M_{a,i}^{(t)}$$

$$= \max_{i, z_a \in rcc_i} \left(score_m(x_i) \cdot (1 - \max_{j, \exists z_b \in rcc_j - rcc_i} L_{i,j}) \right)$$

High Time Complexity

where $L_{i,j} = \max_t \frac{L(\hat{x}_i, \hat{x}_j)^{(t)}}{\hat{x}_i^{(t)}}$ **Sensitive to noise**

$$M_{a,i} = \hat{x}_i \cdot \left(1 - \max_{j, \exists z_b \in rcc_j - rcc_i} L_{i,j}\right)$$

Too slow and too sensitive to noise in practice!

LatentRegressor

Algorithm 1: LatentRegressor

Data: RCC Relationship $G_{RCC} = (V_r, E_r(a, b, \{\boldsymbol{x}_i | a, b \in \boldsymbol{rcc}(\hat{\boldsymbol{x}}_i)\})$, Metric Relationship $G_{metric} = (V_m, E_m(\boldsymbol{x}_i, \boldsymbol{x}_j))$

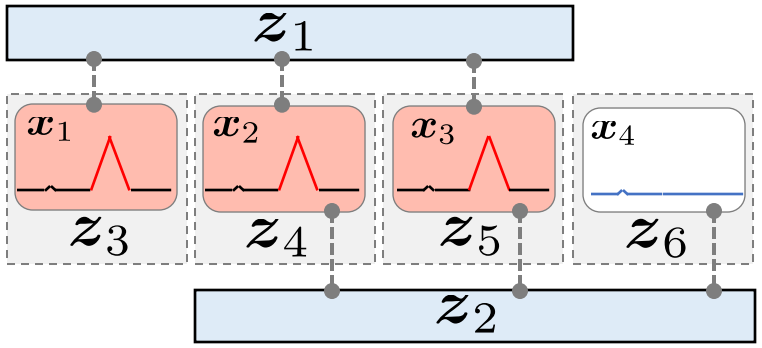
Result: Root Cause Score for RCCs

$a\_scores \leftarrow \{\}, b\_scores \leftarrow \{\}, results \leftarrow \{\}$;
foreach $b \in V_r$ **do**
  $\hat{\boldsymbol{x}}_{max} \leftarrow \max(\{\hat{\boldsymbol{x}}_i | b \in \boldsymbol{rcc}(\boldsymbol{x}_i), \forall c \in next(b), c \notin \boldsymbol{rcc}(\boldsymbol{x}_i)\})$;
  $b\_scores[b][\boldsymbol{x}_{max}] \leftarrow score_m(\boldsymbol{x}_{max})$;
end
foreach $a \in V_r$ **do**
  foreach $b \in next(a)$ **do**
    //Up Step
    $\hat{\boldsymbol{x}}_{amax} \leftarrow \max(\{\hat{\boldsymbol{x}}_i | a \in \boldsymbol{rcc}(\boldsymbol{x}_i), b \notin \boldsymbol{rcc}(\boldsymbol{x}_i))\})$;
    $\hat{\boldsymbol{x}}_{bmax} \leftarrow \max(\{\hat{\boldsymbol{x}}_i | a \in \boldsymbol{rcc}(\boldsymbol{x}_i), b \in \boldsymbol{rcc}(\boldsymbol{x}_i)\})$;
    $score_a \leftarrow score_m(\boldsymbol{x}_{bmax}) \cdot L_{amax,bmax}$;
    $score_b \leftarrow score_m(\boldsymbol{x}_{bmax}) \cdot (1 - L_{amax,bmax})$;
    //Down Step
    foreach $c \in prev(b) - \{a\}$ **do**
      $\hat{\boldsymbol{x}}_{cmax} \leftarrow \max(\{\hat{\boldsymbol{x}}_i | b \in \boldsymbol{rcc}(\boldsymbol{x}_i), c \in \boldsymbol{rcc}(\boldsymbol{x}_i)\})$;
      $score_a \leftarrow score_a \cdot (1 - L_{bmax,cmax})$;
    end
    $a\_scores[a] \leftarrow \max(a\_scores[a], score_a)$;
    if $\boldsymbol{x}_{bmax} \in b\_scores[b]$ **then**
      $b_scores[b][\boldsymbol{x}_{bmax}] \leftarrow \min(b\_scores[b][\boldsymbol{x}_{bmax}], score_b)$;
    end
  end
end
foreach $rcc \in V_r$ **do**
  $\text{results}[rcc] \leftarrow \max(a\_scores[rcc], \max(b\_scores[rcc]))$;
end
return *results*;

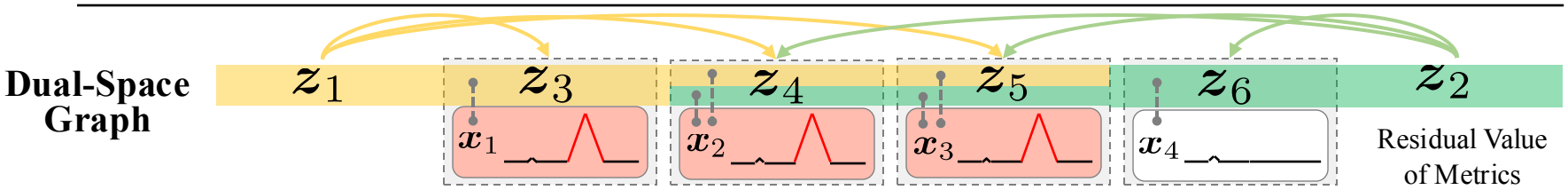
LatentRegressor:

- **Reduce the number of regression calculations (Faster)**
- **Use Ridge regressions instead of linear regressions (More Robust)**

LatentRegressor

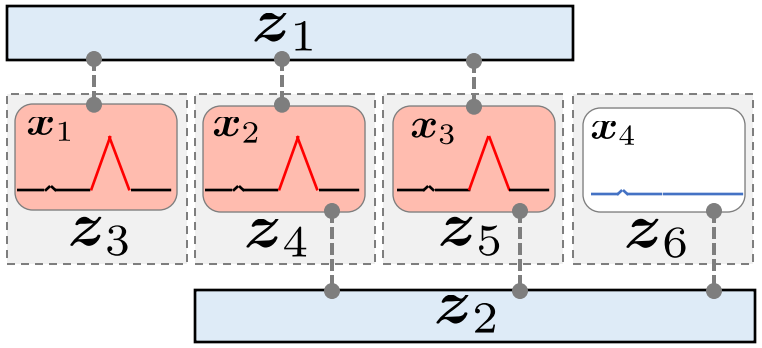


- Z_1 is the root cause
- Only indirect metrics for Z_1
- Some indirect metrics are related to 3 RCCs

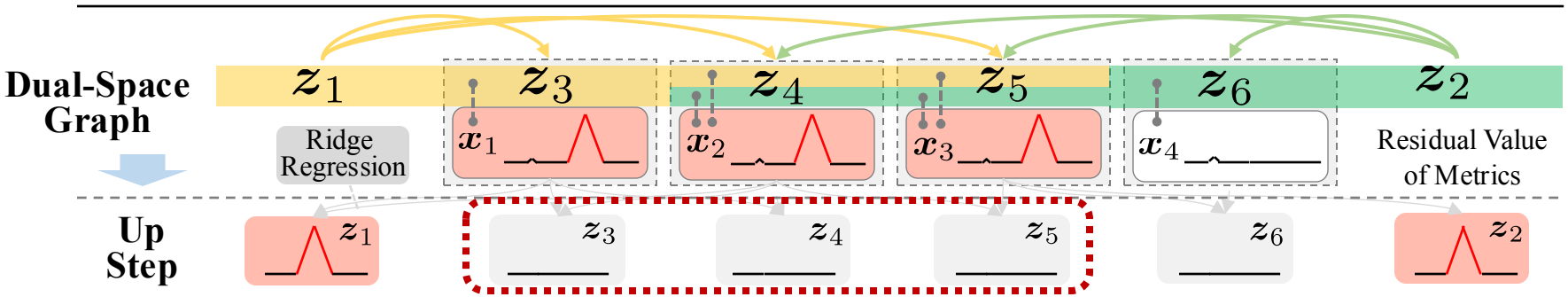


Step 1: Build the dual-space graph and calculate the residual value of metrics

LatentRegressor



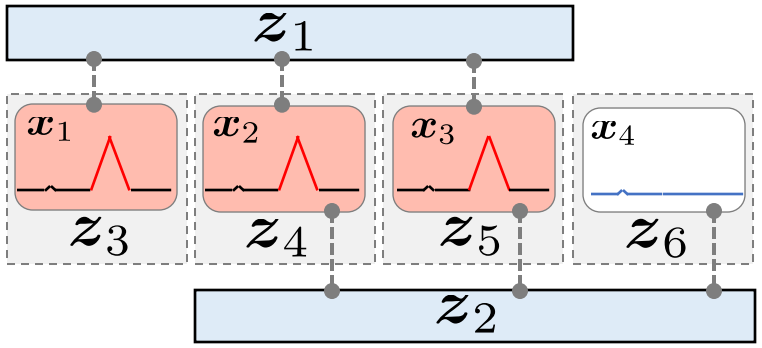
- Z_1 is the root cause
- Only indirect metrics for Z_1
- Some indirect metrics are related to 3 RCCs



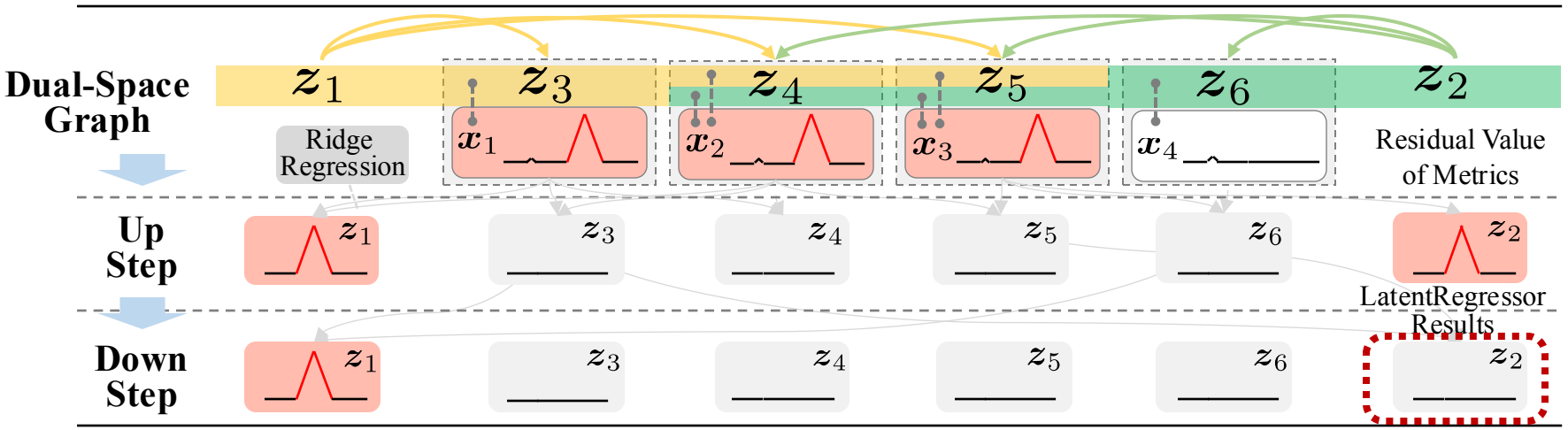
The scores of z3, z4 and z5 are eliminated according to their regression results of each other

Step 2: Conduct ridge regression iteratively on the related metrics of the RCCs with the same parent RCCs (on the Dual-Space Graph)

LatentRegressor



- Z_1 is the root cause
- Only indirect metrics for Z_1
- Some indirect metrics are related to 3 RCCs



Step 3: Conduct ridge regression iteratively on the related metrics of the RCCs with the same child RCCs and get the final results

Datasets and Evaluation Metrics

- **Dataset A:**
  - **Deployed and evaluated in eBay’ s real-world microservices system**
  - Containing over 300 microservices and 10+ third-party services
  - 66 real-world failure cases over 6 months
- **Dataset B:**
  - **Collected from Testbed (Online Boutique[1]) with failure injection**
  - 88 injected failure cases
- **Evaluation Metrics:**
  - **Top@k:** Root causes are ranked into Top k
  - **MRR: T**he average of the reciprocal ranks of the root cause
  - **Macro Values:** Balancing the weights of different categories of root cause cases

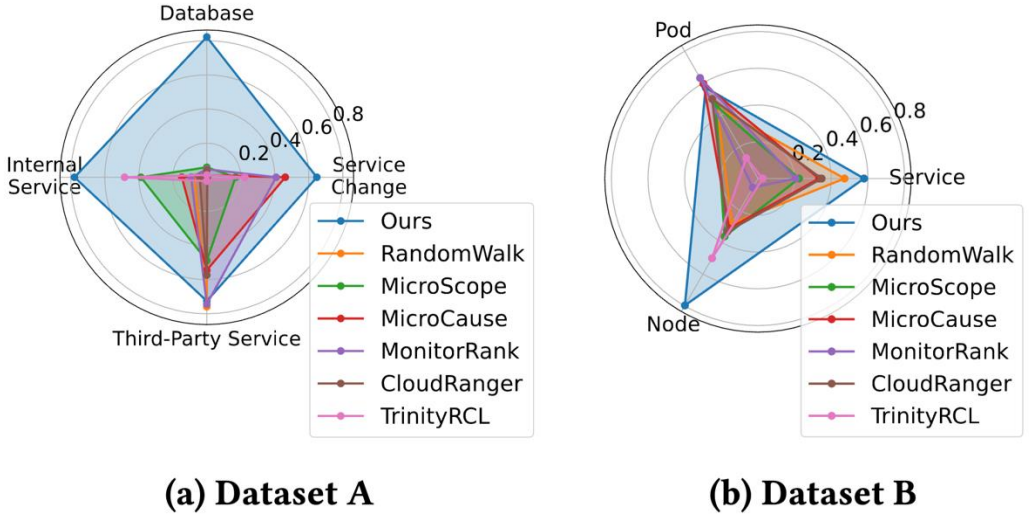
[1] https://github.com/GoogleCloudPlatform/microservices-demo

Evaluation Results

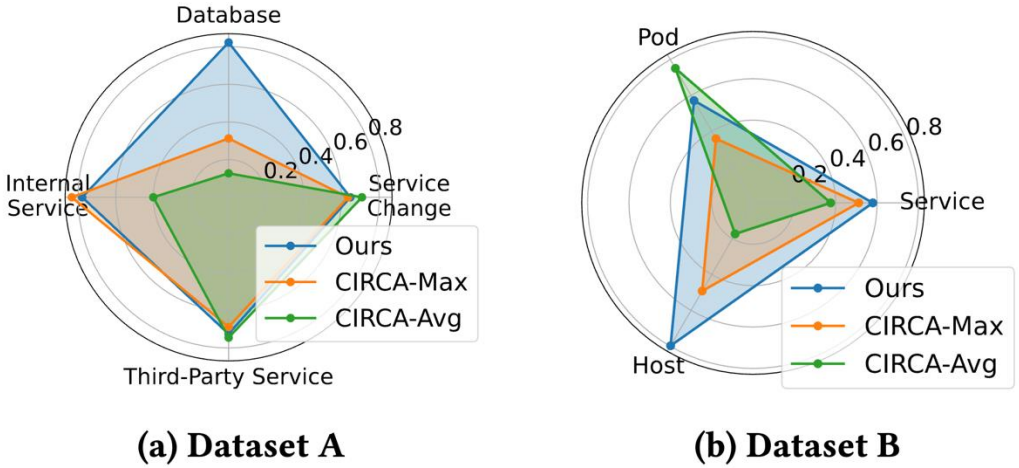
Dataset	Category	Model	Micro				Macro				Avg. Time (s)
			Top@1	Top@5	MRR	MRR@5	Top@1	Top@5	MRR	MRR@5	
A	Baselines	RandomWalk	0.5606	0.5606	0.5683	0.5606	0.1888	0.1888	0.2117	0.1888	7.1
		MonitorRank	0.5000	0.6970	0.5993	0.5745	0.2415	0.3443	0.3228	0.2666	5.9
		MicroScope	0.3030	0.5303	0.4030	0.3851	0.2168	0.2934	0.2759	0.2445	5.4
		CloudRanger	0.2632	0.5802	0.4045	0.3813	0.0939	0.2094	0.1564	0.1358	603.3
		MicroCause	0.2391	0.8478	0.5163	0.5116	0.1433	0.8158	0.3841	0.3707	302.5
		TrinityRCL	0.0303	0.1212	0.0607	0.0494	0.1250	0.3486	0.1858	0.1788	14.6
	Study of Latent Variables	CIRCA-Avg	0.5152	0.8333	0.6512	0.6288	0.2832	0.8335	0.4952	0.4869	11.3
		CIRCA-Max	0.4697	**0.8923**	0.6473	0.6402	0.4243	**0.9118**	0.6228	0.6167	11.3
	Ours	**LatentScope (RLIR Only)**	0.5606	0.8615	0.6828	0.6641	0.4133	0.8699	0.5789	0.5729	136.4
		LatentScope	**0.6154**	**0.8923**	**0.7324**	**0.7205**	**0.6302**	**0.9118**	**0.7430**	**0.7372**	11.6
B	Baselines	RandomWalk	0.1379	0.4912	0.2902	0.2498	0.1273	0.5795	0.3210	0.2841	3.0
		MonitorRank	0.1724	0.5632	0.3146	0.2816	0.1923	0.4313	0.2983	0.2587	5.1
		MicroScope	0.0920	0.5977	0.3123	0.2739	0.1026	0.6404	0.3495	0.3118	0.2
		CloudRanger	0.1707	0.6849	0.3906	0.3703	0.1740	0.7452	0.3936	0.3786	3.2
		MicroCause	0.1724	0.7241	0.4101	0.3771	0.1862	0.7908	0.4151	0.3851	71.9
		TrinityRCL	0.0517	0.1264	0.1169	0.0795	0.1189	0.2809	0.2185	0.1775	6.1
	Study of Latent Variables	CIRCA-Avg	0.2045	0.8295	0.4589	0.4186	0.2246	0.7698	0.4332	0.3841	10.7
		CIRCA-Max	0.2159	0.8636	0.4633	0.4393	0.2804	0.8847	0.4537	0.4344	10.7
	Ours	**LatentScope (RLIR Only)**	0.3258	**0.9299**	0.4882	0.4706	0.3063	**0.9353**	0.4745	0.4586	586.3
		LatentScope	**0.3750**	0.9205	**0.6064**	**0.5953**	**0.4337**	0.9287	**0.6491**	**0.6394**	10.9

- **RLIR Only: Replace LatentRegressor with the vanilla Latent-Space IR**
- **Improvements were achieved in all evaluation metrics compared with baselines**

Evaluation Results



Comparison with Baselines



Study of Latent Variables

The above figures show the evaluation results in different categories of root causes

- **Baselines can only achieve good performance on some of the categories, but LatentRegressor achieves good performance on almost all categories**
- **Even if the observability of the root cause is limited (e.g., Database cases in Dataset A), LatentRegressor is still able to perform accurate RCA**

Conclusion

- Modeling root cause candidates (RCCs) as latent variables for heterogeneous RCCs under limited observability
- Propose LatentScope, using a dual-space graph to model RCCs and metrics
- Introduce Regression-based Latent-space Intervention Recognition (RLIR) algorithm and LatentRegressor enhancement for real-world adaptation
- Deploy and evaluate LatentScope in eBay's cluster, showing superior performance across different root cause categories

Thank you !

Microservice Root Cause Analysis With Limited Observability Through Intervention Recognition in the Latent Space

Source Code & Data:
https://github.com/eBay/LatentScope