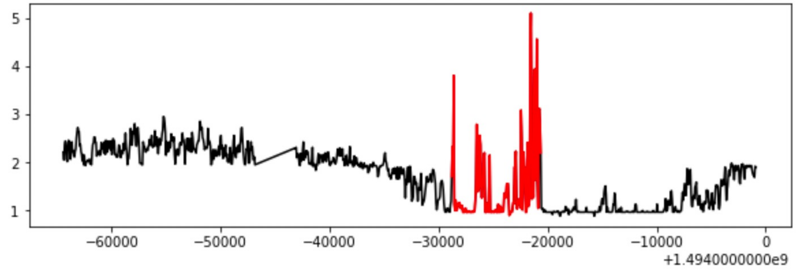
From Point-wise to Group-wise: A Fast and Accurate Microservice Trace Anomaly Detection Approach

Zhe Xie[1], Changhua Pei, Wanxue Li, Huai Jiang, Liangfei Su, Jianhui Li, Gaogang Xie, Dan Pei

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

Anomaly Detection in Microservices

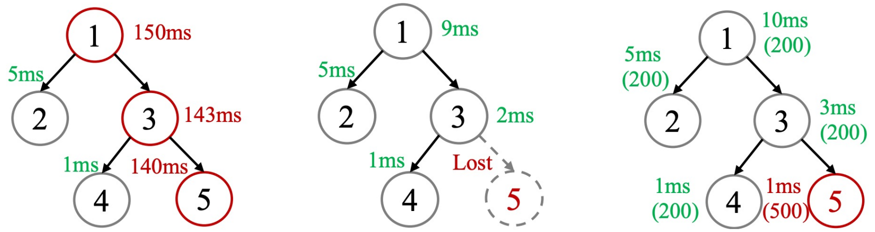


Time-Series Based

[00:00:01] [Info] checking if there are any updates...
[00:00:11] [Error] Connection Timeout.
[00:00:12] [Info] Time cost: 10.00s
[00:00:15] [Info] Time cost: 0.02s

Log Based

Only information about a single service or a single calling relationship

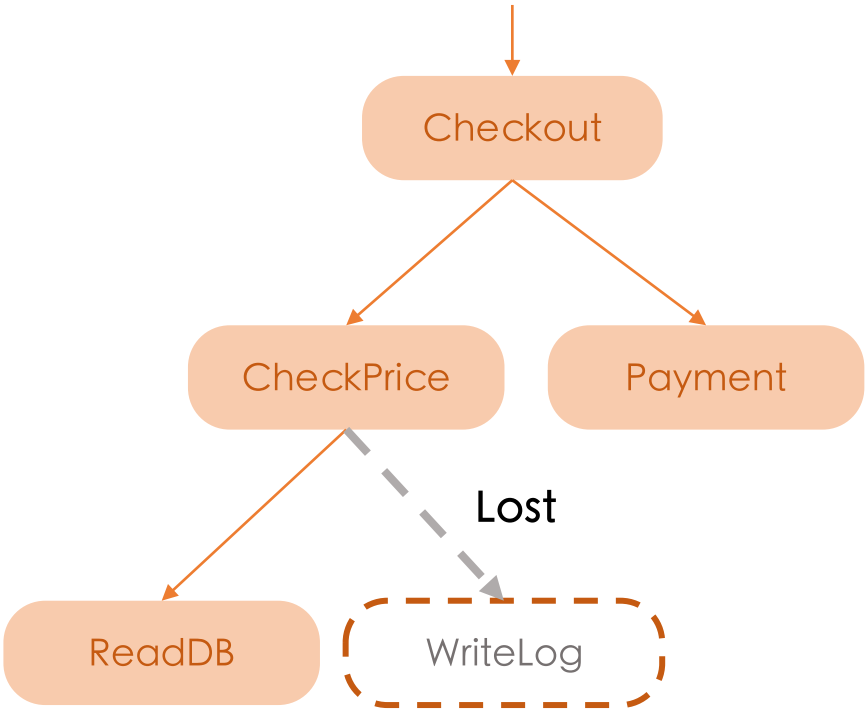


Trace Based

Records the complete call

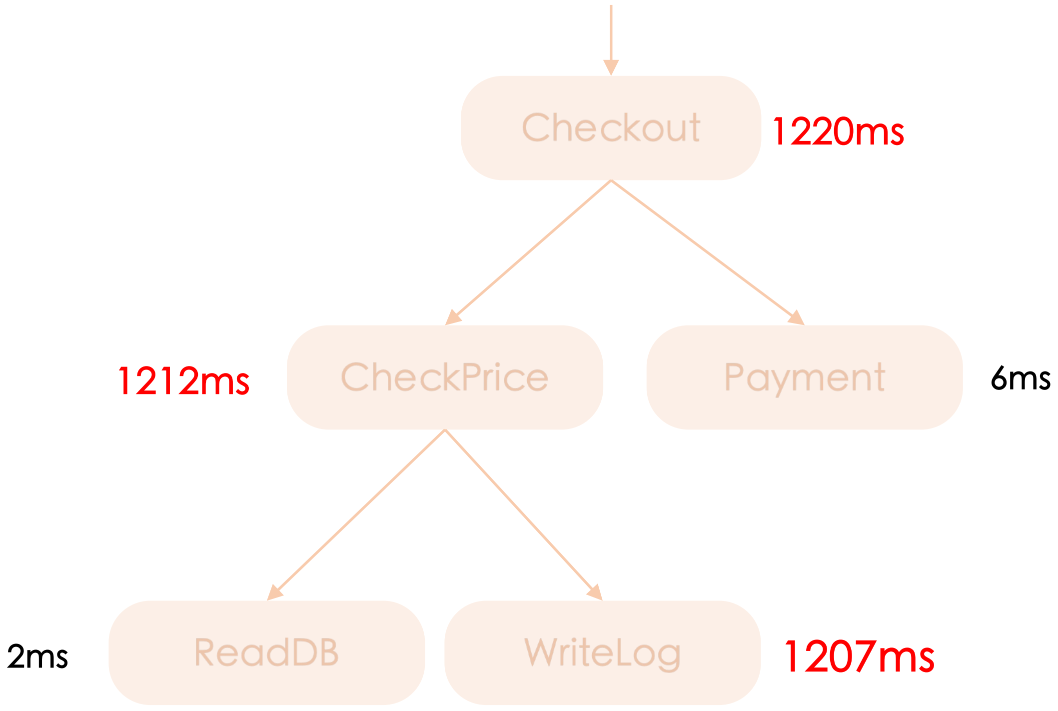
Traces record all these requests along with some additional information, such as the return code and response time of each invocation.

Types of Trace Anomalies



Structural Anomaly

(API, invocation relationship, return code)



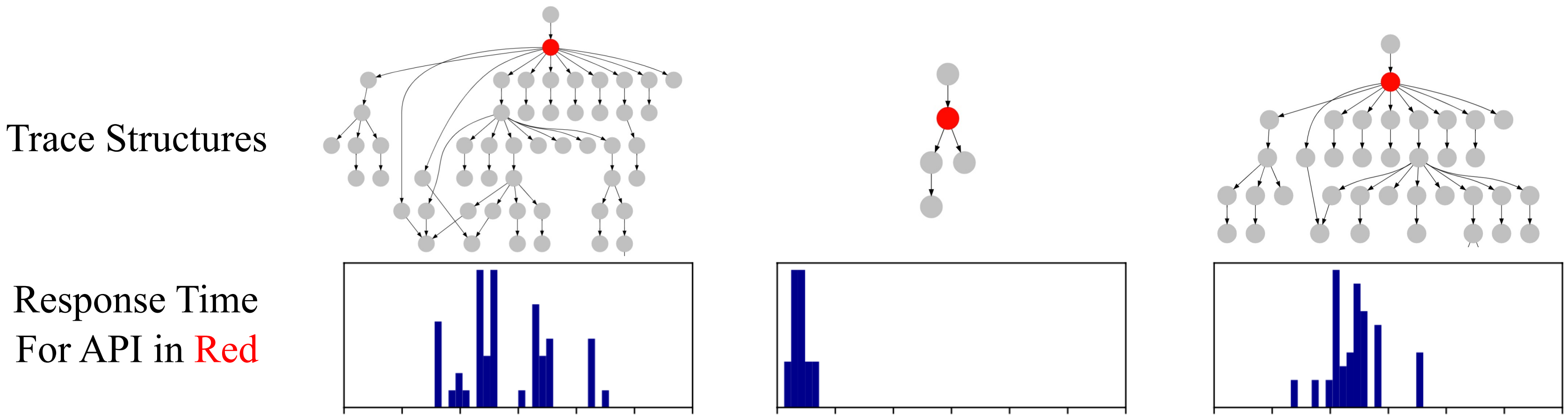
Latency Anomaly

(Response time)

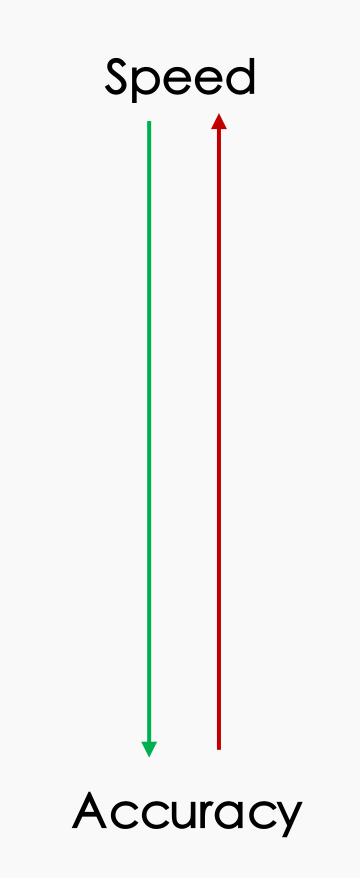
- 2 major types of anomalies in traces

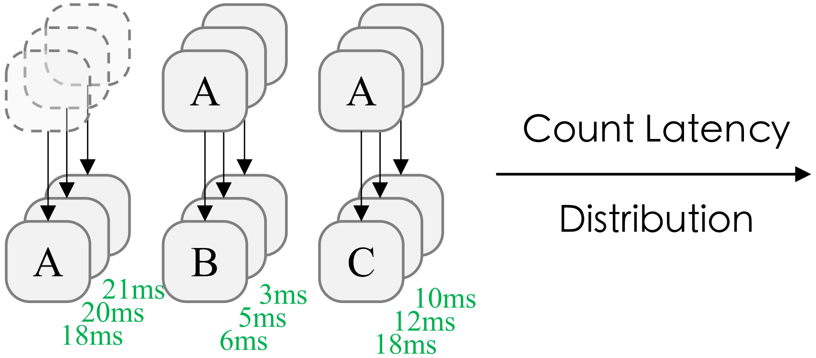
Challenges

- The vast quantity of traces produced by the system requires highly efficient detection methods.
- The diverse structures require our approach to be adaptable to them.
- The variability in response latency for the same API across different downstream call structures also presents significant modeling challenges.



Existing Approaches



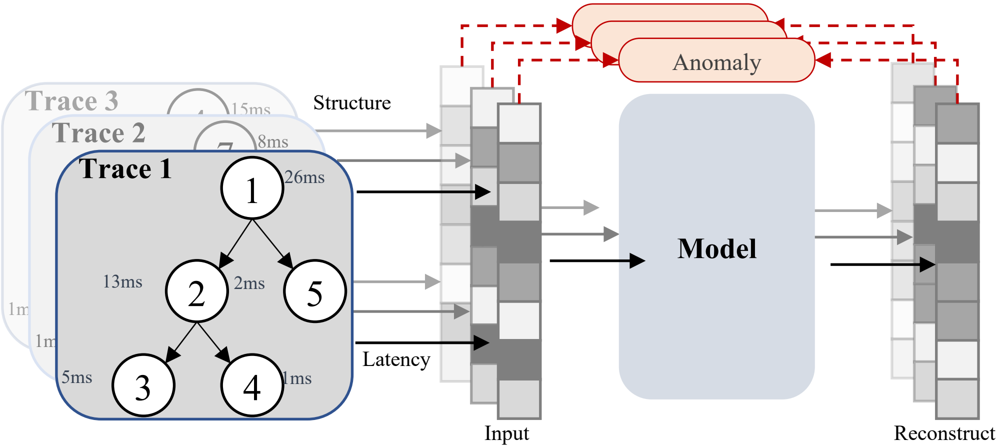


Caller	Callee	Avg. Latency	Latency Std.
-	A	20	2
A	B	5	1
A	C	12	5

Statistic Based

Efficient

Loss of Accuracy

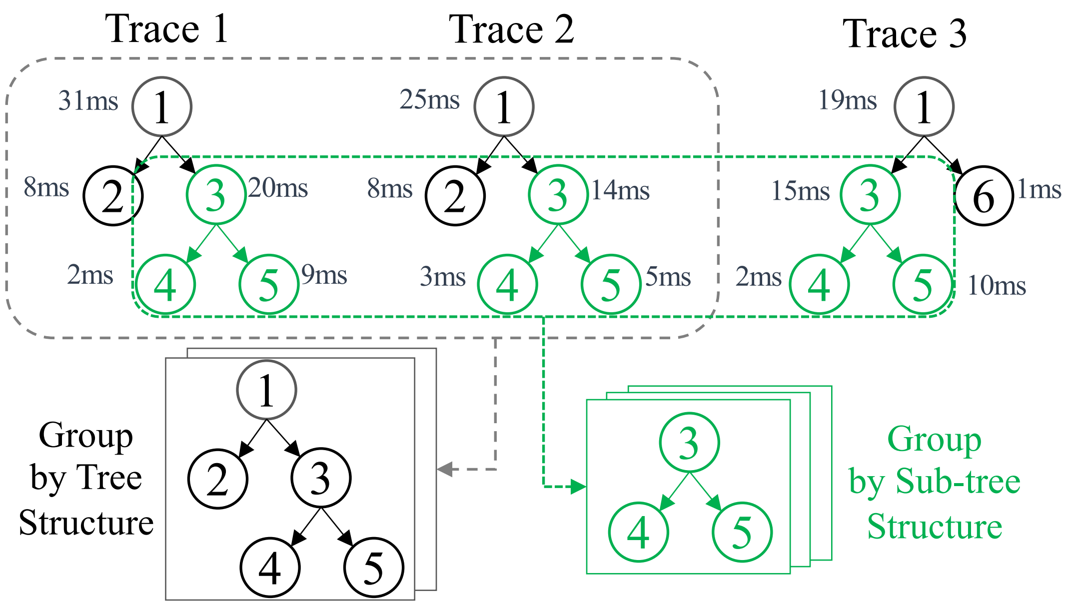


Model Based

Precise Modeling of Traces

One-by-One Inference

Grouping of Traces



- Many traces share the same tree/sub-tree structure.
- Feature distributions in shared structures are usually similar.
- Can traces or sub-trees of traces be grouped according to their structures?

Selection of Grouping Strategy

Table 1: Empirical Study on Grouping Strategies

Group Strategy	Description	Count	$\overline{\sigma(\log(lat))}$
None	No group	**1**	1.96
API	Group by API ID	413	0.48
STV	Group by stv [19]	51373	0.20
Tree	Group by tree [12]	40370	0.24
Sub-tree	Group by sub-tree	3311	**0.18**

(~1M trace nodes before grouping)

Count - #Groups (Small is better)

$\overline{\sigma(\log(lat))}$ - Variance of features (Small is better)

- Requirements for Grouping Strategy:
- With the right granularity and quantity
- Samples within the group share a similar feature distribution (latency distribution for traces)

Selection of Grouping Strategy

Table 1: Empirical Study on Grouping Strategies

Group Strategy	Description	Count	$\overline{\sigma(\log(lat))}$
None	No group	**1**	1.96
API	Group by API ID	413	0.48
STV	Group by stv [19]	51373	0.20
Tree	Group by tree [12]	40370	0.24
Sub-tree	Group by sub-tree	3311	**0.18**

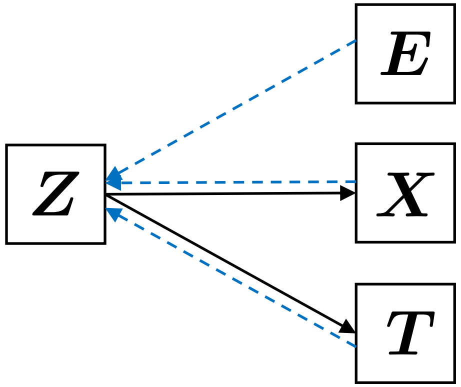
(~1M trace nodes before grouping)

Count - #Groups (Small is better)

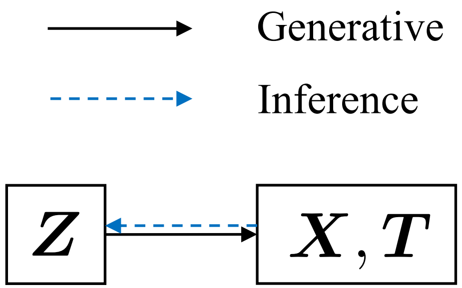
$\overline{\sigma(\log(lat))}$ - Variance of features (Small is better)

- Requirements for Grouping Strategy:
- With the right granularity and quantity
- Samples within the group share a similar feature distribution (latency distribution for traces)
- We use **sub-tree** as the grouping strategy. i.e., we divide each trace into sub-trees and group the sub-trees with the same structure into a group.

Modeling for Grouped Sub-trees

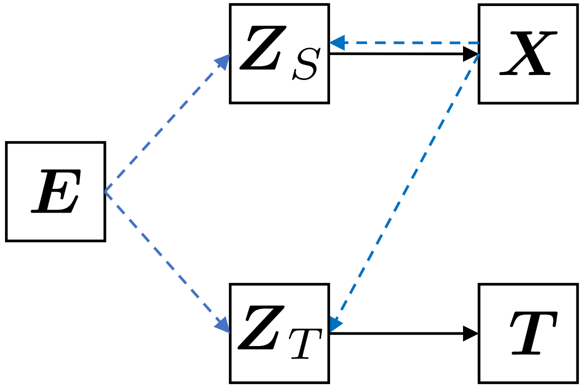


TraceVAE (WWW’ 23)[1]



TraceAnomaly (ISSRE’ 20)[2]

- Latency feature (T) is coupled with structural features (X, E) while inference
- Cannot be used in grouped trace modeling

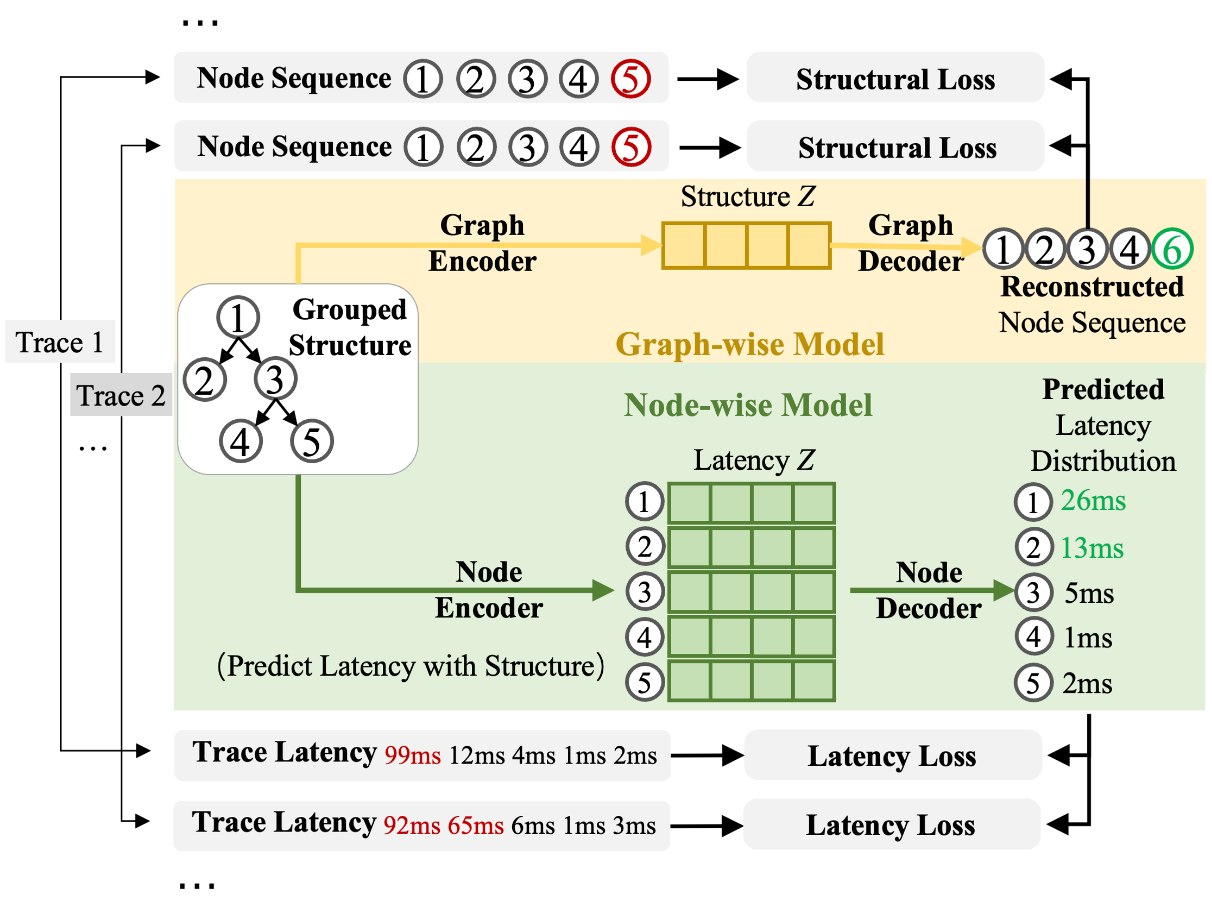


GTrace Model (Ours)

- Predict latency with structure
- Can be used in grouped trace modeling

[1] Xie, Z., Xu, H., Chen, W, et al. Unsupervised Anomaly Detection on Microservice Traces through Graph VAE. In Proceedings of the ACM Web Conference 2023 (pp. 2874-2884).
[2] Liu P, Xu H, Ouyang Q, et al. Unsupervised detection of microservice trace anomalies through service-level deep Bayesian networks. 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2020: 48-58.

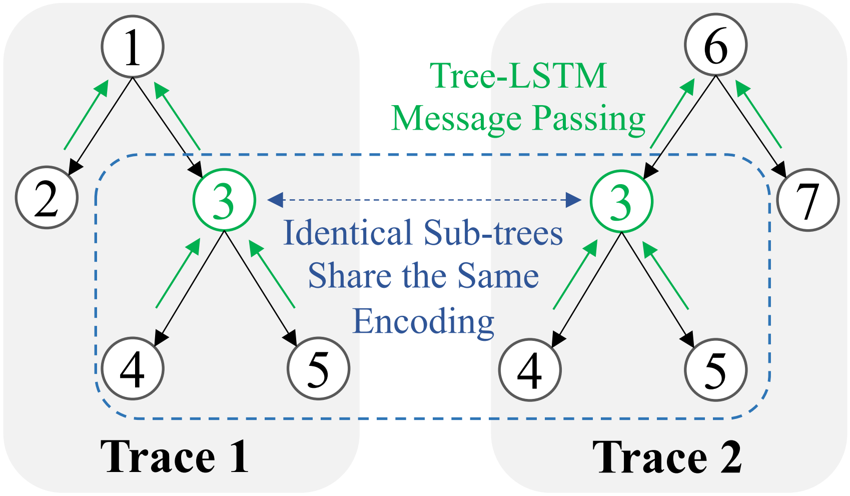
Modeling for Grouped Sub-trees



- Based on Tree-LSTM
- Encoder & Decoder Structure
- Predict latency with structure
- Produce reusable encoding for each sub-tree group

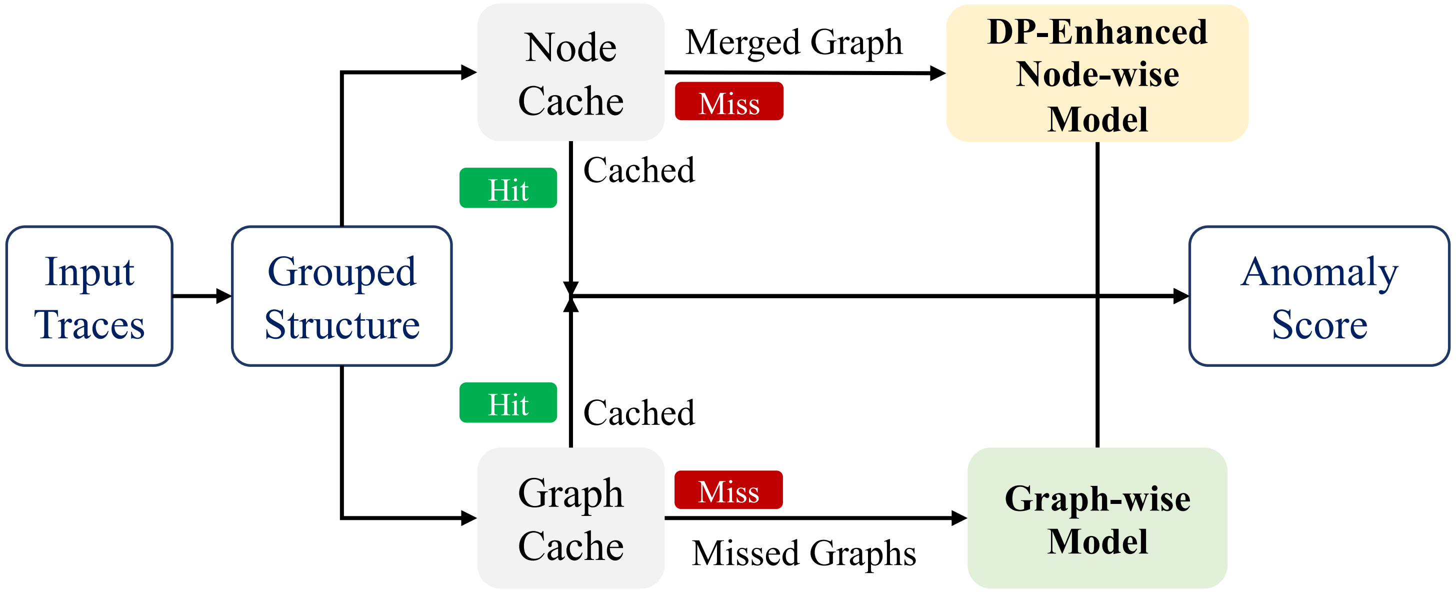
- Still many groups for model inference
- Further acceleration with reusable encoding?

Dynamic Programming Inference



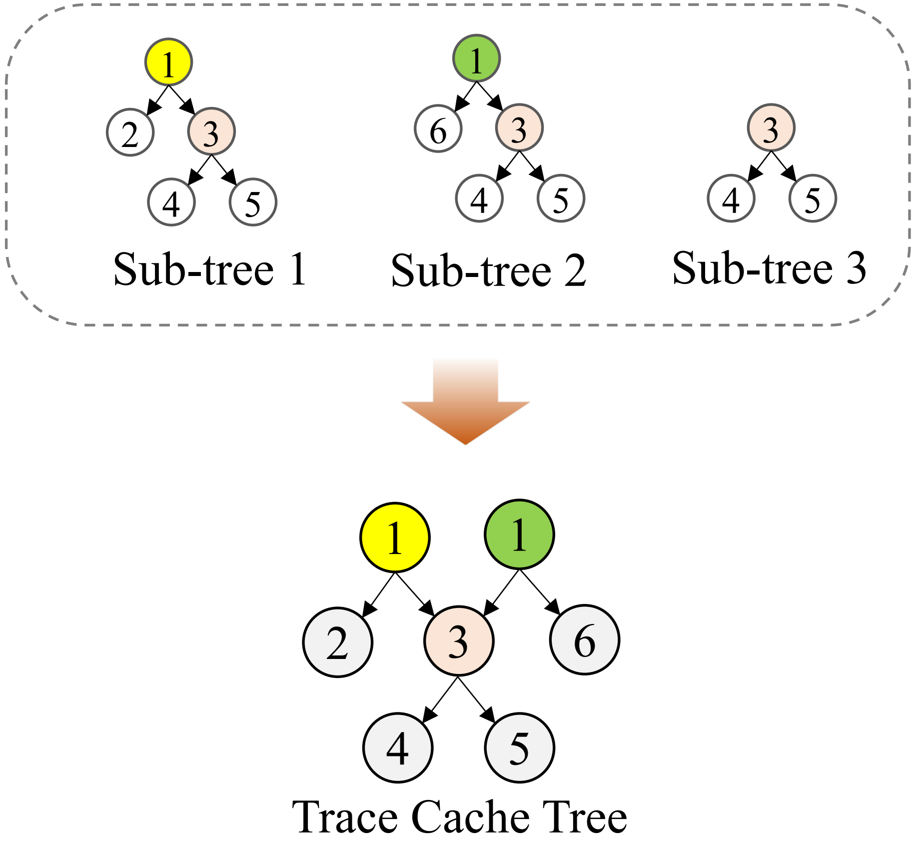
- Messages are passed from bottom to top, producing the same encoding for identical sub-trees
- Dynamic Programming (DP) can be used in the inference
- Encoding can be used by the subsequent inferences (How to store the encoding?)

Inference Acceleration with Cache



- A tree-like cache to store sub-tree encodings.
- The core idea is to store the information through a Trace Cache Tree (TCT) and maintain the nodes in an LRU way.
- A batch of traces is input into the cache for querying. The cache will return a merged graph, including the missed sub-trees for model inference.

Trace Cache Tree (TCT)

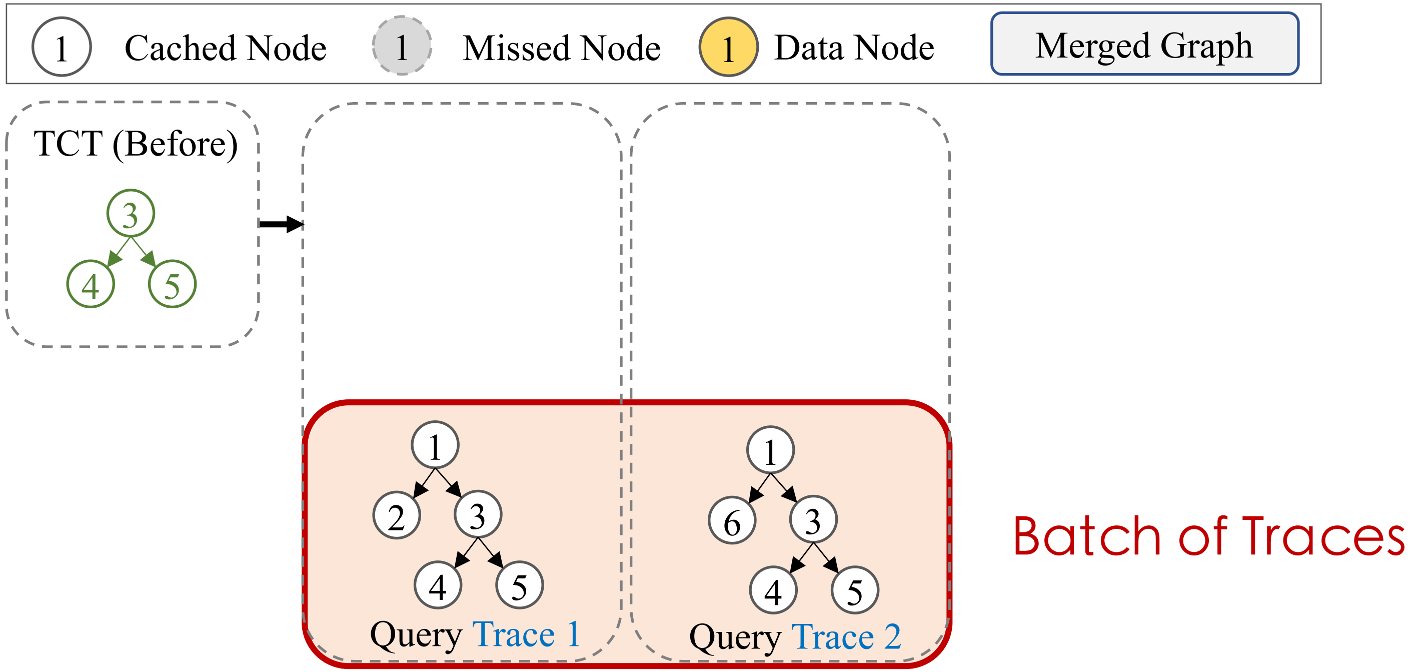


How to store the reusable encodings for different sub-trees?

Trace Cache Tree (TCT)

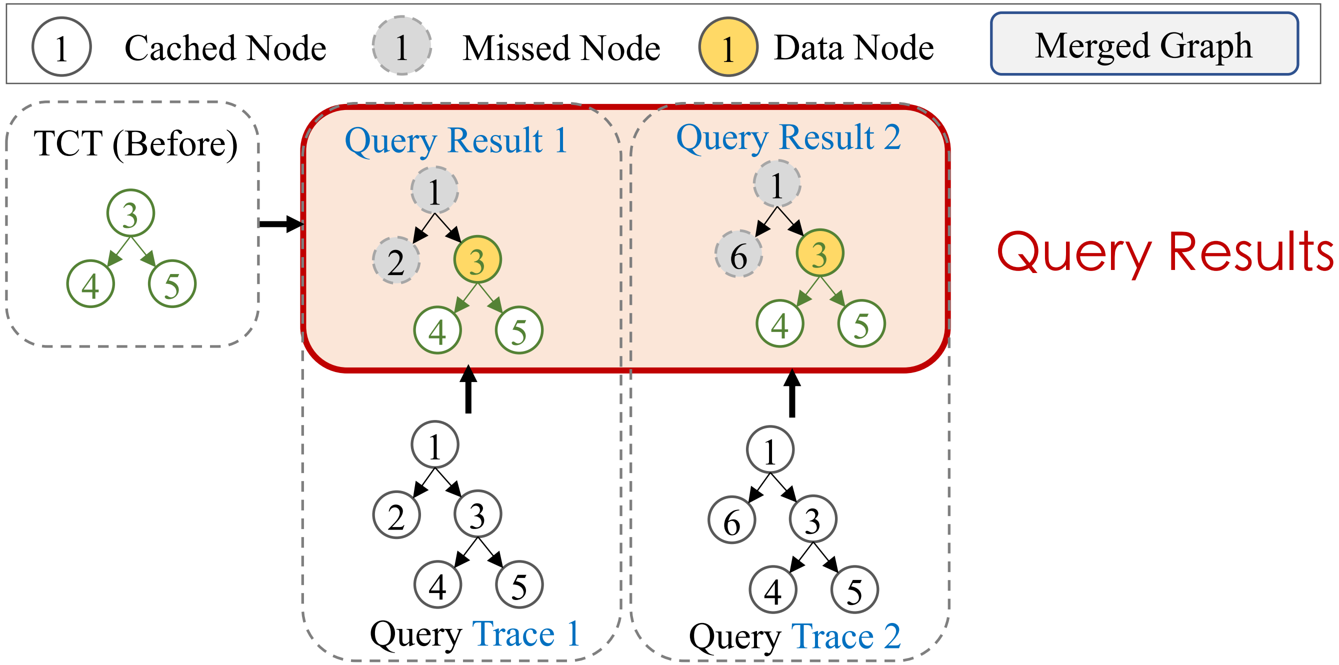
- A merged graph from many sub-trees
- Encodings are shared among different sub-trees
- TCT can be queried with a batch of traces and returns a sub-graph of it

Inference with Node Cache (TCT)



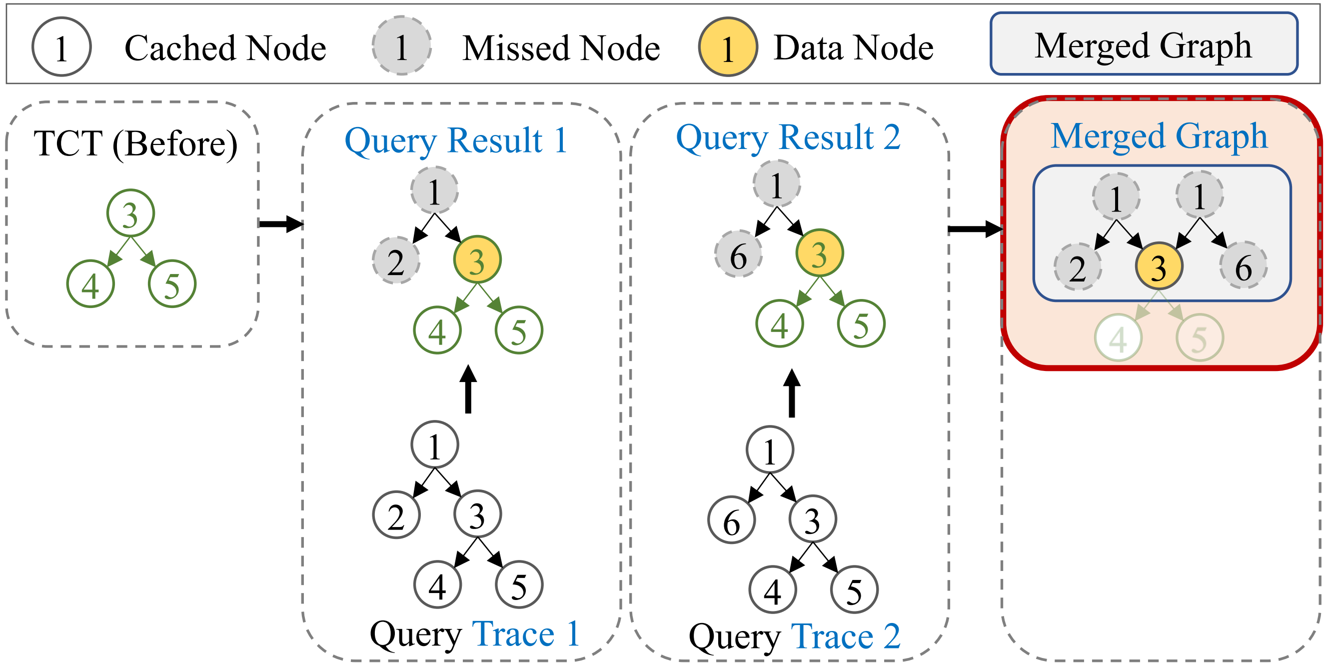
- **Step 1:** Query a batch of traces.

Inference with Node Cache (TCT)



- Step 2: Query each sub-tree in the traces in the Trace Cache Tree (TCT).

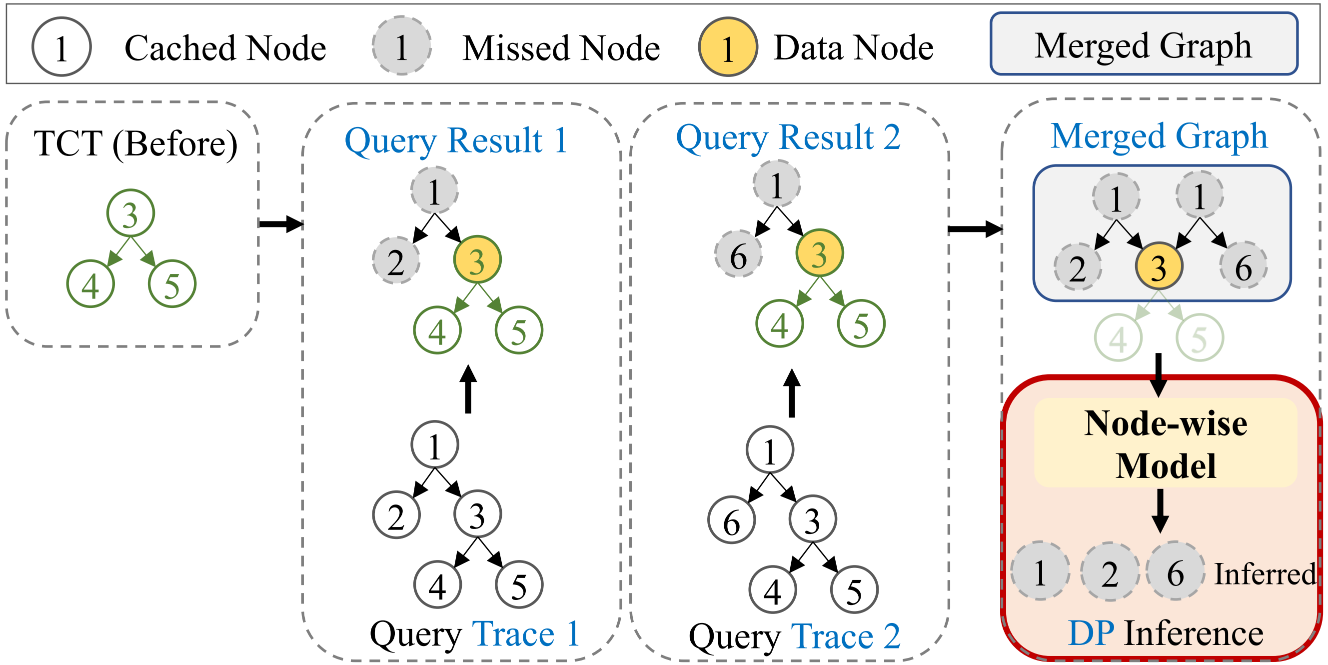
Inference with Node Cache (TCT)



Merged graph

- Step 3: Output the subgraph composed of missed sub-trees (nodes) and data nodes as a merged graph.
  - The merged graph contains all the nodes in this batch to be inferred. Only one model inference is required for this graph.
  - For an entire batch of traces, we only need to run model inference once for the missed sub-trees, thereby further reducing the inference overhead.

Inference with Node Cache (TCT)

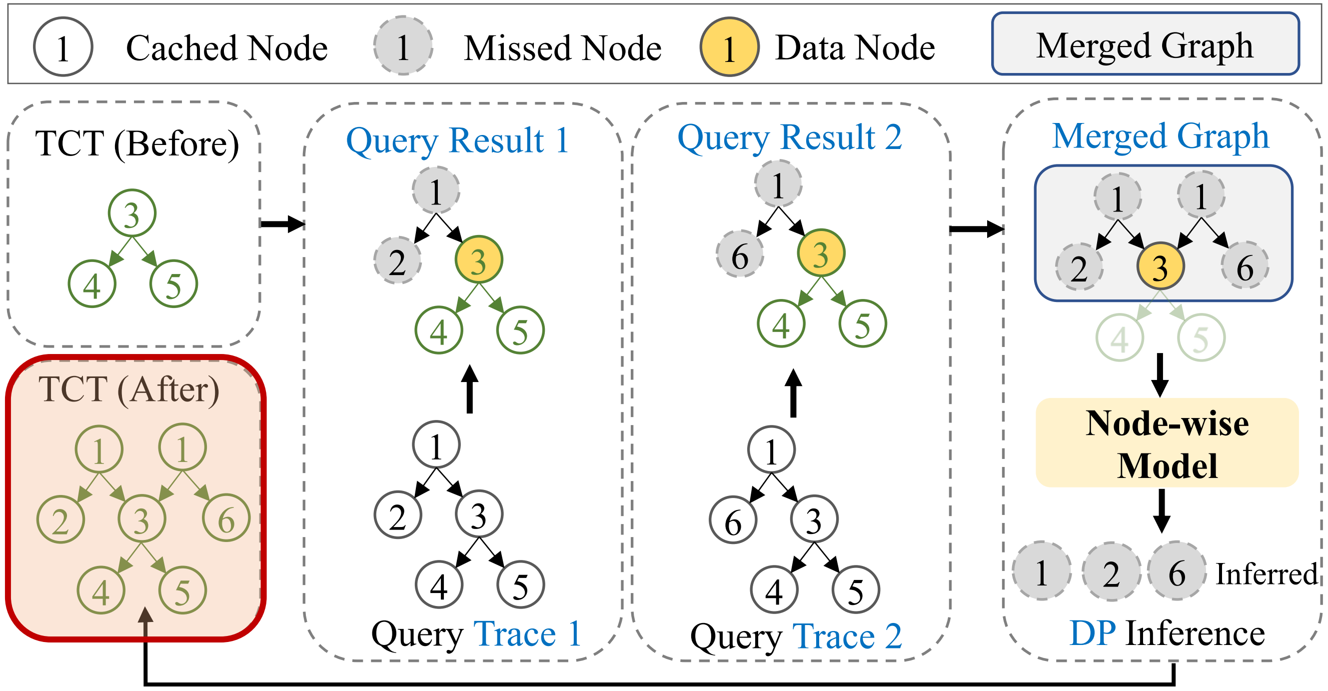


DP-Enhanced Model Inference

- Step 4: **Perform DP-Enhanced** model inference.

Inference with Node Cache (TCT)

Update TCT
in Cache



- Step 5: Insert new nodes into TCT to update the node cache.

Anomaly Detection

- Detecting anomalies by calculating $p_\theta(G)$ with Monte Carlo importance sampling:

$$p_\theta(G) \approx \frac{1}{n_Z} \sum_{i=1}^{n_Z} \frac{p_\theta(G|\boldsymbol{Z}^{(i)}) p_\theta(\boldsymbol{Z}^{(i)})}{q_\phi(\boldsymbol{Z}^{(i)}|G)}$$

- Use negative log-likelihood (NLL) as anomaly score

- Detecting structural anomalies and latency anomalies respectively:

$$p_\theta(\boldsymbol{X}) \approx \frac{1}{n_Z} \sum_{i=1}^{n_Z} \frac{p_\theta(\boldsymbol{X}|\boldsymbol{Z}_S^{(i)}) p_\theta(\boldsymbol{Z}_S^{(i)})}{q_\phi(\boldsymbol{Z}_S^{(i)}|\boldsymbol{X}, \boldsymbol{E})} \stackrel{\text{def}}{=} L_{S,\theta}$$

$$p_\theta(T_k) \approx \frac{1}{n_Z} \sum_{i=1}^{n_Z} \frac{p_\theta(T_k|\boldsymbol{Z}_T^{(i)}) p_\theta(\boldsymbol{Z}_T^{(i)})}{q_\phi(\boldsymbol{Z}_T^{(i)}|\boldsymbol{X}, \boldsymbol{E})} \stackrel{\text{def}}{=} L_{T,k,\theta}$$

Datasets

Dataset	#Traces	P99 Latency	P99 #Spans	P99 Depth
$\mathcal{A}$	125k	7580ms	90	10
$\mathcal{B}$	140k	263ms	96	4

- Dataset A:
  - Collected from eBay
  - Including 314 microservices and 1487 APIs
- Dataset B:
  - Collected from Testbed (**Online Boutique**[1])
  - Including 11 microservices and 65 APIs

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

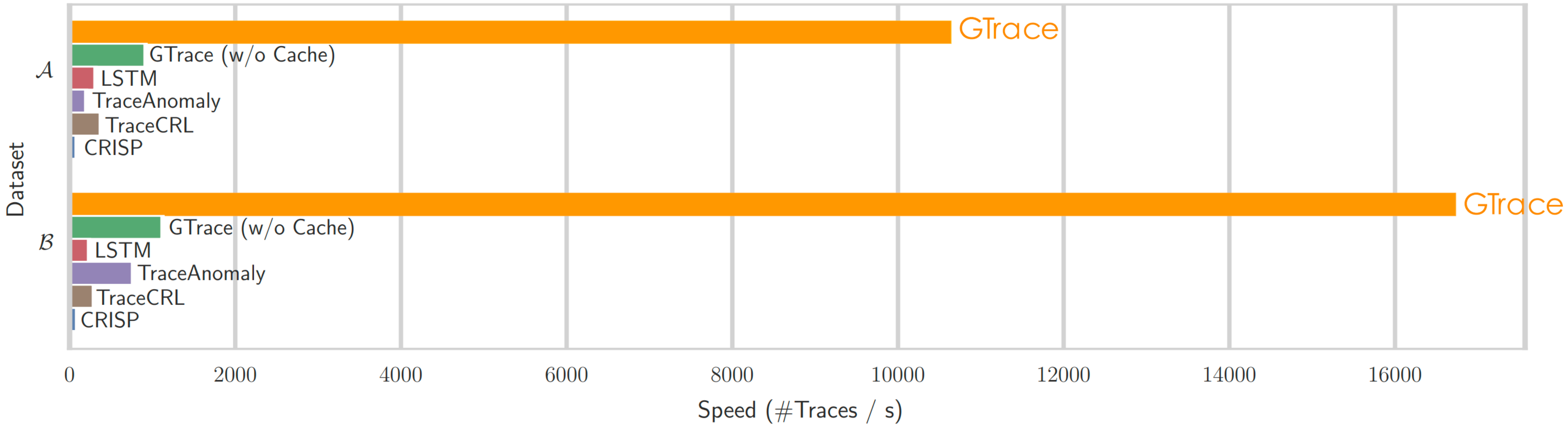
Evaluation of Accuracy

A: AUC **F:** F1-Score

Dataset	Model	Node Latency				Trace Latency				Trace Structure			
		A	⇑A	F	⇑F	A	⇑A	F	⇑F	A	⇑A	F	⇑F
$\mathcal{A}$	CFG	0.795	9.06%	0.757	14.80%	0.880	6.25%	0.803	11.58%	0.192	340.10%	0.314	174.20%
	FSA	0.277	213.00%	0.446	94.84%	0.303	208.58%	0.472	89.83%	0.050	1590.00%	0.105	720.00%
	TA	0.250	246.80%	0.305	184.92%	0.337	177.45%	0.504	77.78%	0.286	195.45%	0.611	40.92%
	LSTM	0.052	1567.31%	0.121	618.18%	0.398	134.92%	0.394	127.41%	0.163	418.40%	0.254	238.98%
	CRISP	0.183	373.77%	0.278	212.59%	0.294	218.03%	0.318	181.76%	0.011	7581.82%	0.054	1494.44%
	TraceCRL	0.022	3840.91%	0.077	1028.57%	0.074	1163.51%	0.114	685.96%	0.062	1262.90%	0.176	389.20%
	GTrace	**0.867**	-	**0.869**	-	**0.935**	-	**0.896**	-	**0.845**	-	**0.861**	-
$\mathcal{B}$	CFG	0.698	12.18%	0.663	10.41%	0.671	16.10%	0.717	5.02%	0.206	306.31%	0.501	60.68%
	FSA	0.392	99.74%	0.616	18.83%	0.384	102.86%	0.569	32.34%	0.124	575.00%	0.221	264.25%
	TA	0.275	184.73%	0.446	64.13%	0.337	131.16%	0.504	49.40%	0.286	192.66%	0.611	31.75%
	LSTM	0.147	432.65%	0.244	200.00%	0.759	2.64%	0.736	2.31%	0.123	580.49%	0.342	135.38%
	CRISP	0.143	447.55%	0.261	180.46%	0.336	131.85%	0.482	56.22%	0.295	183.73%	0.611	31.75%
	TraceCRL	0.023	3304.35%	0.072	916.67%	0.437	78.26%	0.552	36.41%	0.072	1062.50%	0.227	254.63%
	GTrace	**0.783**	-	**0.732**	-	**0.779**	-	**0.753**	-	**0.837**	-	**0.805**	-

- Improvements were achieved in all evaluation metrics
- "Predicting latency with structure" brings better generalization performance to the model

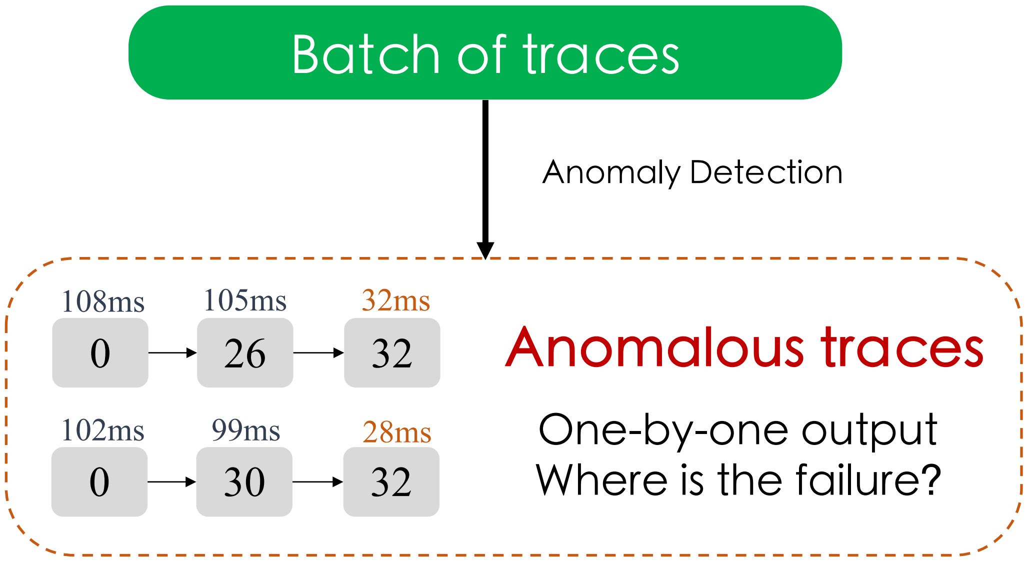
Evaluation of Time Efficiency

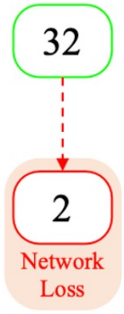


- Full process evaluation: from span data to anomaly detection results
- GTrace achieves a large advantage in time efficiency over other model-based methods

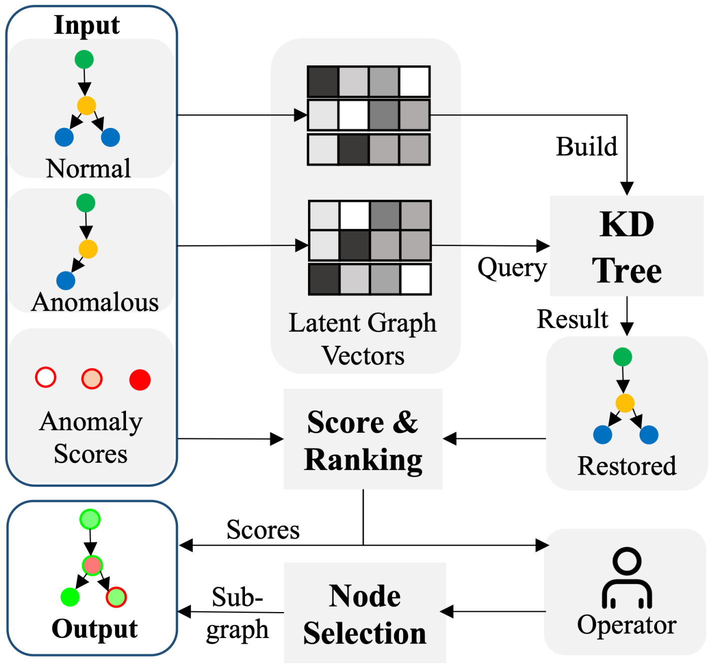
Visualization Tool

- How can the results be clearly understood by operators?





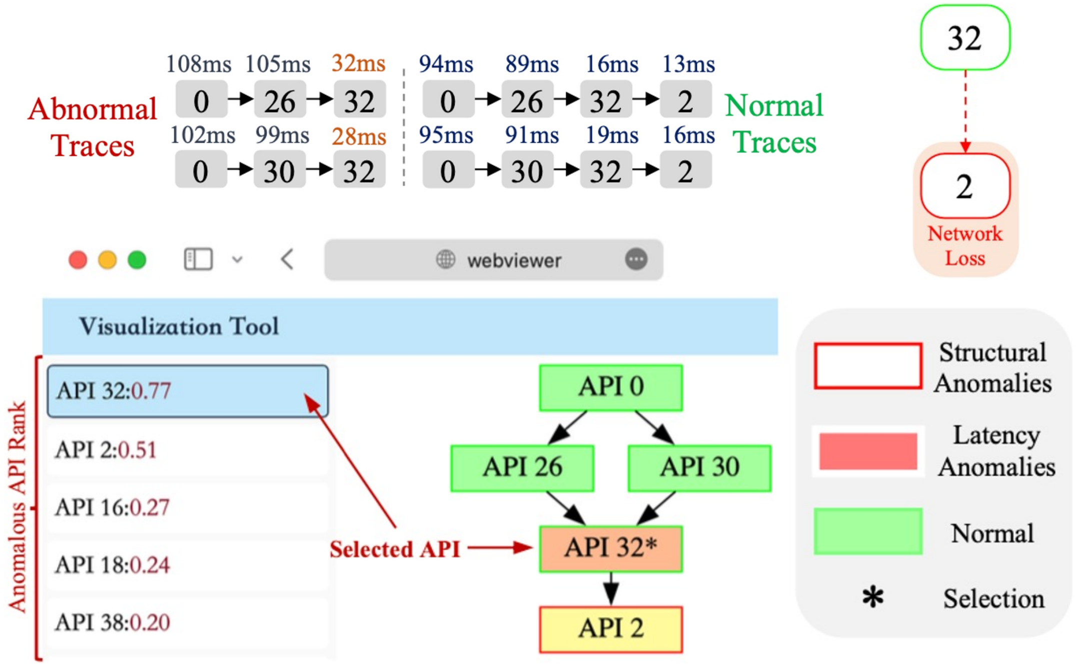
- The network failure of API 2 resulted in many traces with structural anomalies
- However, API 2 was not recorded in these anomalous traces



A visualization tool to help operators get an overview of the failure

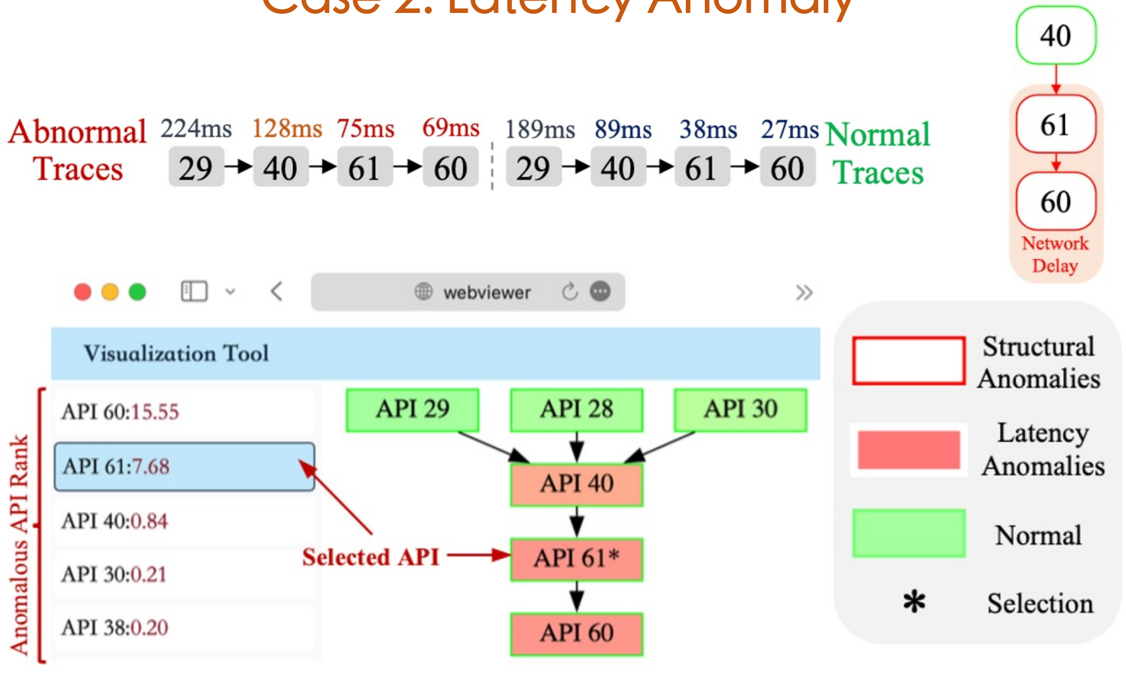
Visualization Tool

Case 1: Structural Anomaly



API 2 is lost in abnormal traces but can be reconstructed by the tool

Case 2: Latency Anomaly



Efficient analysis of fault propagation

- Provide operators with a summary of detected anomalies.
- Reconstruct lost nodes in structural anomalies.

Conclusion

- We propose GTrace, the first group-wise trace anomaly detection method

- A group-wise VAE model which models trace latency in a novel "predicting latency with structure" way

- Inference acceleration through DP inference, TCT and merged graph

- A visualization tool to show a summary of detected trace anomalies in the form of a graph

Thank you !

From Point-wise to Group-wise: A Fast and Accurate Microservice Trace Anomaly Detection Approach

Paper: https://doi.org/10.1145/3611643.3613861

Source Code & Dataset & Demo: https://github.com/NetManAIOps/GTrace.git