TS3D: A Temporal Multimodal Dataset for Distributed Database System Analysis

Yuanyuan Yao[†], Yuhan Shi[†], Yian Wei[†], Lu Chen[†], Mourad Khayati[‡], Cheng Long[§], Tianyi Li[¶],
[†]Zhejiang University, China [‡]University of Fribourg, Switzerland
[§]Nanyang Technological University, Singapore [¶]Aalborg University, Denmark
{yoyoyao, shiyuhan, yianwei, luchen}@zju.edu.cn, mourad.khayati@unifr.ch, c.long@ntu.edu.sg, tianyi@cs.aau.dk

***Abstract*—Distributed databases have become foundational infrastructure across a wide range of industries. Unfortunately, the strict data privacy and security regulations often prevent practitioners from accessing real operational data, limiting the reliability of their analysis. While a few public log datasets for distributed databases are available, relying solely on them for downstream tasks still introduces performance bottlenecks. A major limitation is that they typically provide either textual logs or temporal values, but rarely both, making it difficult to fully understand system behavior. Being able to leverage the multimodal information inherent in distributed databases can drastically improve the accuracy of downstream analysis.**

In this paper, we propose TS3D, a temporal multimodal dataset derived from a distributed database. The dataset is the largest to date, with 300 million numerical data points and 90 million contextual data records. We showcase its utility through two representative downstream tasks. First, we propose a multimodal anomaly detection framework that fuses logs and numerical time series data, thereby enhancing both existing log-based models and time series–based models, and we design a hybrid evaluation metric capable of jointly assessing both point-level and segment-level anomalies. Second, we present a SQL-level analysis framework that constructs temporal-dependency and table-dependency graphs to trace anomaly root causes and performs pod-level SQL optimization across a database cluster. We further show how to formally validate these auxiliary methods derived from the dataset. Our empirical results show that, using our multimodal dataset, the efficacy of distributed anomaly detection improves by up to 26% while distributed root cause localization improves by up to 35%.

***Index Terms*—multimodal data, distributed databases, anomaly detection, root cause localization.**

I. Introduction

In recent years, distributed storage has emerged as a fundamental component in many commercial database systems, including PingCAP's TiDB [1], Apache IoTDB [2], and others [3], [4]. These systems typically employ a database autonomy service (DAS) [5] to monitor database metrics, log records, and SQL statements, aiming to optimize performance, improve availability, and ensure data security and integrity. Despite the widespread use of real-world operational data in industry, strict privacy and security constraints make it largely inaccessible to researchers, significantly hindering academic advances in distributed database system management.

Although several existing public datasets in the AIOps domain [6]–[8] contain database-related information (e.g., Oracle performance metrics), they primarily focus on system-level tasks such as service monitoring [9], KPI failure detection [10], and fault diagnosis [11]. Consequently, these datasets are ill-suited for database-specific tasks such as SQL-level root-cause analysis or query optimization. Dedicated datasets for distributed databases are extremely limited. For instance, the popular MultiLog dataset [12] records only log events without accompanying metrics, which are crucial for capturing fine-grained performance dynamics such as resource usage, latency spikes, and throughput bottlenecks. Its logs typically contain only execution results or status messages, omitting the SQL statements, which are essential for understanding anomalous behavior and diagnosing its root causes.

Motivated by this gap, we build a temporal multimodal dataset for distributed databases. Collected from a distributed cluster, it contains over 300 million numerical time series and 90 million textual log entries with a total size of 32 GB. The dataset records 27 real-time metrics at both system (CPU, memory, disk I/O, network) and database levels (slow queries, full joins, table locks), with each log entry enriched with detailed context, node, anomaly type, SQL statement, return message, and execution status. Beyond the recorded anomalies, we also simulate diverse system-and database-level anomalies through 13 stress-test scenarios, including CPU-intensive tasks, I/O-heavy workloads, and slow SQL executions. All numerical values and logs are precisely labeled for anomalies, enabling benchmarking for multimodal learning and correlation analysis.

Applications. Our multimodal dataset is comprehensive and applicable to a variety of real-world tasks. Below, we describe two applications that benefit from this dataset. We empirically evaluate its utility on these applications in Section VI.

Application 1: Multimodal Anomaly Detection. Anomaly detection in distributed database systems aims to identify abnormal states or performance degradations caused by multiple factors, such as high concurrency, transaction conflicts, or node failures. These data quality issues can significantly affect system stability and increase query latency. In distributed environments, anomalies often manifest as correlated deviations across multiple nodes due to violated inter-node dependencies. They are temporally asynchronous and propagative: anomalies may emerge at different times on different nodes and propagate through the system with delays. Such anomalies typically arise from cascading interactions among local faults rather than from an isolated failure on a single machine.

Temporal misalignment naturally arises between log data and numerical values due to their distinct collection mechanisms: logs are event-driven and recorded immediately upon each operation, whereas time series are periodically sampled at fixed intervals. As a result, numerical values often lag behind logs, since system performance series (e.g., CPU or I/O fluctuations) emerge gradually following prior log events. For example, in slow-query scenarios, each query is instantly logged by the database, while resource metrics such as CPU utilization or I/O latency only reflect the accumulated impact a few seconds later, after multiple queries have piled up.

Application 2: Distributed SQL-level Root Cause Localization (RCL). RCL in distributed databases aims to trace and identify the underlying causes of anomalies, such as performance degradation, transaction conflicts, or deadlocks, by analyzing the interactions among SQL statements, system components, and execution dependencies. Specifically, SQL-level RCL focuses on determining which SQL queries or table operations trigger abnormal behaviors, enabling precise fault diagnosis and targeted performance optimization. Most RCL methods focus on scenarios, such as high concurrency [13], often overlooking database anomalies caused by table-level dependencies or deadlock issues. However, with the widespread adoption of modern distributed systems, especially in cloud computing and microservice systems, the complexity of the SQL query execution order has significantly increased, making table-level dependencies more common. As a result, traditional RCL methods are not effective in addressing the complex table dependencies in distributed environments.

Proposed Solutions. To address the aforementioned issues in distributed anomaly detection, we propose a multimodal framework that enables cross-modal alignment and fusion. As anomaly labels from logs and numerical time series are often temporally misaligned, we use labels from each modality as anchors to align and annotate the other modality. Building on this alignment, we integrate log-based and time–series–based anomaly detectors into a unified pipeline. Additionally, we introduce an adaptive masking mechanism that dynamically selects informative features, ensuring that only the most relevant series contribute to anomaly characterization. By fusing log semantics with time-series representations [14], the framework improves distributed anomaly detection across a broad range of anomaly types [15].

We further enhance the multimodal framework with a novel evaluation metric specifically designed for our dataset. TS3D includes both point and segment anomalies, ranging from transient spikes caused by network jitters to prolonged performance degradations due to CPU saturation. Existing metrics are insufficient for such mixed-type anomalies: point-level metrics (e.g., F1) overlook temporal continuity, while segment-level metrics (e.g., PA-F1 [16]) fail to capture isolated anomalies. To overcome these limitations, we introduce an evaluation metric that accounts for both point and segment anomalies, offering a more comprehensive measure of anomaly detection performance in distributed environments.

Regarding the second application, we design a SQL-level root cause localization method based on the characteristics of distributed databases. We employ a two-step strategy. First, we construct a temporal dependency graph that represents the dependencies between nodes based on the execution times of SQL queries. Second, we use this graph to identify the longest predecessor node for each node and trace the backtracking path of any timeout nodes. For complex SQL queries, we further identify the longest execution dependency path, based on which we build a table-level SQL dependency graph to deeply analyze the root causes of SQL link issues. The SQL optimization occurs at the pod level, optimizing SQL according to the load of distributed nodes.

In summary, we make the following contributions.

- We curate and publicly release TS3D, the first temporal multimodal dataset based on a distributed database. We evaluate the effectiveness of the dataset in two key tasks: anomaly detection and root cause localization.
- We propose a multimodal anomaly detection framework that jointly leverages numerical time-series signals and logs by integrating their respective anomaly detectors, and introduce a hybrid evaluation metric combining point and segment-level anomalies.
- We devise a root cause localization method for SQL programs, which leverages time and table-dependency graphs to identify anomaly causes, along with Pod-level SQL optimization informed by the load on distributed nodes.
- Extensive experiments on TS3D show that the multimodal dataset notably enhances anomaly detection over log-only data. They also show that the proposed metric offers more comprehensive evaluation and that the RCL method effectively captures table-level dependency anomalies.

The remainder of this paper is organized as follows: Section II introduces preliminaries; Section III presents the TS3D dataset and collection procedure; Section IV describes the multimodal anomaly detection framework and evaluation metric; Section V outlines the R-SQLs discovery algorithm; Section VI discusses the empirical results; Section VII reviews related work; and Section VIII concludes.

II. PRELIMINARIES

In this section, we first formalize the concept of temporal multimodal data and then define the key concepts employed in the two tasks: anomaly detection and root-cause analysis.

Definition 1 (Temporal Multimodal Data)**.** *A temporal multimodal dataset consists of multiple data types organized sequentially in time:* $X = \langle X_1, X_2, \ldots, X_L \rangle$*, where* L *is the sequence length and* X_i $(1 \leq i \leq L)$ *is the observation value at time* T_i*. Specifically,* $X_i = \{X_i^1, X_i^2, \ldots, X_i^d\}$ *contains different modalities, such as textual and numerical data, and* d *is the dimension of each piece of data.*

We focus on collecting temporal multimodal data, which includes both textual and numerical data. These numerical values include system-level performance metrics, including CPU usage, memory consumption, and disk I/O, alongside

TABLE I
DATASET DETAILS

Dataset	Size	#Records
SQL statements	5.96GB	42,695,4304
Execution logs	22.20GB	48,980,998
Numerical data	2.60GB	300,593,315

database-specific metrics such as query execution time, slow queries, and connection statistics. We monitor and analyze metrics to detect anomalies, then use log data for precise localization and explanation.

Definition 2 (SQL Template)**.** *The SQL Template is a structured representation of the key attributes of an SQL query extracted from log entries. It is defined as an object* $Q_i = (t_i, P_i, O_i, T_i, D_i)$*, where* t_i *represents the execution timestamp, indicating the precise time when the query was executed;* P_i *represents the execution pod in the distributed database;* O_i *refers to the type of operation (e.g., SELECT, INSERT, UPDATE), specifying the action performed by the query;* T_i *is the list of tables involved, detailing the data sources the query interacts with; and* D_i *represents the response time or duration of the query, capturing how long it takes to execute.*

For example, given the log entry: "2025-01-06 11:35:22.115622, database_abnormal, select_1 query successful, SELECT COUNT(*) AS total_drivers FROM drivers, response time:0.00603", the SQL template derived from this log can be represented as Q_1 = (2025-01-06 11:35:22.115622, select, drivers, 0.00603), where t_1 is the execution timestamp (2025-01-06 11:35:22.115622), O_1 represents the operation type (SELECT), T_1 indicates the table involved (drivers), and D_1 denotes the response time (0.006038188934326172 sec). This structured representation encapsulates the key details of the SQL query and its associated performance.

Definition 3 (Root Cause SQLs)**.** *Given an anomaly case* $A = (L, s, e)$*, this task aims to analyze the SQL execution log* L *to identify the root cause of the anomaly. Specifically, we filter out the responsible SQL statements (R-SQLs) that exhibit characteristics such as deadlocks or table dependency issues within the time interval* (s, e)*.*

By detecting SQL statements that exhibit anomalous execution behavior, such as lock contention or cyclic table dependencies, we can explicitly link SQL execution patterns to observed anomalies. This approach provides deeper insight into how particular query structures or table interactions trigger abnormal performance variations. These insights support fine-grained root cause identification and inform adaptive optimization strategies, enabling timely mitigation and resolution of performance anomalies in distributed databases arising from table-level dependencies.

III. THE TS3D DATASET

In this section, we present the construction pipeline of our multimodal dataset and discuss its salient properties.

A. Dataset Construction

We build a distributed database with the goal of capturing the key characteristics of distributed anomaly detection techniques. The architecture is based on MySQL 5.7 with a master–slave replication setup. This design has several merits. First, it is compatible with most MySQL-based systems (e.g., MySQL Replication [17], Aurora-MySQL [18]). Second, it favors a stability–performance trade-off. Third, it is simpler to handle compared to strongly consistent multi-leader designs [19]. The choice of MySQL is primarily motivated by its widespread adoption, as well as its ease of use, integration, and adaptability. Figure 1 describes the pipeline.

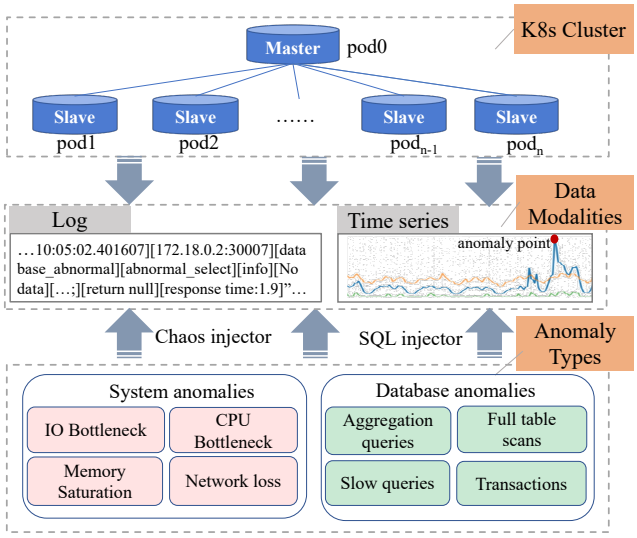

Fig. 1. Pipeline of TS3D Construction.

At the technical level, the cluster consists of seven independently running pod instances organized in a 1-master–6-slave configuration, orchestrated by Kubernetes (K8s) [20] for automated deployment, scaling, and containerized management. The master node (Pod0) serves as the data hub, handling all write operations and coordinating data synchronization through Kubernetes service discovery. Six slave nodes (Pod1–Pod6), distributed across different cluster zones, maintain data consistency by replicating the master's binary logs in real time while processing partial read workloads. Each pod runs an Exporter service to expose performance metrics, which are collected and aggregated by Prometheus [21] into a unified monitoring system for comprehensive performance tracking.

We collect SQL statements, execution logs, and monitoring metric time series data from the distributed database. Detailed information is shown in Table I.

Execution logs & SQL statements. The log dataset records the running status of the distributed database and captures execution details whenever the database performs an operation.

TABLE II
NUMERICAL METRIC DATA DETAILS

Category	**Metrics**
Basic Metrics	cpu, memory, disk_usage, memory_usage, iops, qps
Connection-related Metrics	connections, max_used_connections, aborted_connects, aborted_clients
Query & Thread Metrics	threads_created, slow_queries, select_full_join, select_scan
Lock & Transaction Metrics	table_locks_waited, innodb_row_lock_waits
Buffer Usage Metrics	innodb_buffer_pool_read_requests, innodb_buffer_pool_reads, innodb_buffer_pool_write_requests
Replication Metrics	seconds_behind_master
Cache Efficiency Metrics	qcache_hits, qcache_inserts, qcache_lowmem_prunes
Temporary Object Metrics	created_tmp_tables, created_tmp_disk_tables
Binlog Metrics	binlog_cache_use, binlog_cache_disk_use

TABLE III
DISTRIBUTED DATABASE ANOMALIES

No.	Root Cause	Type (Method)	Example	Solution
1	CPU Intensive Workload	Database (SQL)	Aggregation queries; Transactions	Scale up CPU
2	I/O Intensive Workload	Database (SQL)	Complex queries; Full table scans	Scale up I/O
3	Correlated Slow Query	Database (SQL)	Slow queries	Limit slow queries
4	Resource-Intensive Compaction	Database(SQL)	Complex queries; Full table scans	Scale up CPU and I/O
5	External Operations	Database (SQL)	Frequent access to external APIs	Limit external operations
6	Database Internal Problem	Database (SQL)	Configuration issues	Optimize database
7	Host I/O Bottleneck	System (Chaos)	IO stress	Scale out host I/O
8	Host CPU Bottleneck	System (Chaos)	CPU stress	Scale out host CPU
9	Host Memory Saturation	System (Chaos)	Memory stress	Scale out host Memory
10	Host Network Bottleneck	System (Chaos)	Network loss or delay	Optimize network bandwidth
11	Network Partition Arise	System (Chaos)	Machine-Chaos	Optimize network node
12	Machine Down	System (Chaos)	Machine-Chaos	Fault Repair
13	Unknown Problem	-	External issues	Further diagnosis

Each row in the dataset contains a log entry, and a log entry in the TS3D dataset follows a unified structure:

[Timestamp][Pod in cluster (IP:Port)][Anomaly label&type]
[SQL type][Log type (info/error)][Execution status]
[SQL command][Response time]

Time series data. Here, we include 27 performance indicators across both system-level and database-level dimensions. We adopt a 2-second collection interval to guarantee an optimal trade-off between capturing fine-grained system dynamics and limiting runtime overhead. More frequent sampling would require repeatedly pulling and aggregating internal database/system statistics, increasing CPU/I/O/network cost; it also tends to produce highly correlated, redundant readings for counter- or aggregation-based numerical values with limited additional information. Moreover, 2-second measurements can be readily downsampled to coarser resolutions (e.g., 14 seconds) during post-processing.

Table II describes the nine major aspects that are highly relevant to database performance and anomalies, including resources and workload (CPU, memory, disk usage, IOPS, QPS), connections and concurrency, query execution and thread behavior, locks and transactions, cache and buffer pool usage, replication status, and binlog. Together, these metrics capture the overall system behavior from the perspectives of resource utilization, execution dynamics, concurrency control, and replication mechanism.

B. Anomaly Augmentation

While the original dataset contains anomalies, we have increased their number to mimic a real-world operational environment of distributed databases. We inject two types of anomalies during dataset construction: database-level and system-level anomalies. The injected anomalies cover a wide range of failure scenarios, including SQL performance degradation and transaction conflicts at the database layer, as well as hardware, network, and resource faults at the system layer.

Database-level anomalies. Database-level exceptions are primarily triggered by malicious SQL operations. The database-level anomalies are listed in Table III (No.1–No.6). Aggregation queries, transactions, complex queries, full table scans, and slow queries (No.1 - No.4) simulate high CPU and I/O and limited resources. Frequent API calls (No.5) simulate

system overhead. Internal issues such as lock contention and deadlocks (No.6) simulate inconsistencies. These anomalies are injected using Locust [22], which executes predefined SQL scripts to simulate SQL requests. These scripts trigger various database-level anomalies during designated periods, while Locust controls user concurrency, ramp-up rate, and execution duration to emulate resource exhaustion under high-concurrency workloads.

System-level anomalies. Those anomalies emulate abnormal conditions in the hardware, network, and host environment of a distributed database. As listed in Table III (No.7–No.13), they include resource constraints (I/O delay, CPU contention, and memory pressure), network issues (latency and partitions), and node failures (pod faults). These anomalies are injected using Chaos Mesh, a Kubernetes-based chaos framework [20], and are only meaningful when interpreted through database execution semantics. For instance, OS-level memory pressure typically manifests as lower buffer-pool hit rates, higher disk I/O, and increased query latency. Analyzing such anomalies inside the database enables more accurate and efficient diagnosis of database failures and their root causes.

Using those two types of anomalies, we define four injection scenarios: injecting a single anomaly type into a single pod (Single2Single), injecting a single anomaly type into multiple pods (Single2Multi), injecting multiple anomaly types into a single pod (Multi2Single), and injecting multiple anomaly types into multiple pods (Multi2Multi). To simulate real-world operational anomalies, we randomly inject anomalies from the thirteen types listed in Table III into the database during peak hours (10:00-12:00, 14:00-17:00, and 19:00-21:00) each day. The proportions of injected anomalies in these four scenarios are 25%, 25%, 17.86%, and 32.14%, respectively.

We provide five dataset versions to support different evaluation scenarios. Four datasets correspond to individual anomaly injection settings (Single2Single, Single2Multi, Multi2Single, and Multi2Multi), isolating distinct anomaly propagation patterns for controlled, reproducible analysis. Additionally, an integrated dataset injects all anomaly types concurrently using multi-threaded execution, simulating complex environments with co-occurring anomalies. The dataset is publicly available as a Google Drive document.[1]

C. Anomaly Labeling

In TS3D, log and numerical data are labeled differently due to their temporal misalignment and semantic differences. Logs are discrete and event-driven, while numerical data are continuous time-series aggregated over fixed intervals, causing anomalies to occur at different timestamps.

Log labeling. Logs are labeled as anomalies either when error codes are detected or, in the case of latency-sensitive workloads, when response times exceed a predefined threshold (e.g., 300 ms), indicating performance degradation.

[1]https://drive.google.com/drive/folders/1DptJNiqgXF4DlZ5rx-FXRKz_NaMyS34t

Numerical data labeling. Fixed-threshold labeling does not apply to numerical data, as it lacks response-time values. Instead, we propose an adaptive labeling strategy that partitions an input time series $\{T_i\}_{i=0}^{N}$ of length N into equal-length and non-overlapping windows of size w, denoted as $T_{t,t+w}$. For each window, we compute the upper and lower quantiles using only the non-anomalous points, denoted as $q_{q_\uparrow}(T_{t,t+w})$ and $q_{q_\downarrow}(T_{t,t+w})$. We treat these quantiles as supervision signals and perform least-squares fitting on them, obtaining two dynamic threshold curves that adapt over time. Specifically, we solve over the set of cubic polynomial functions $\mathcal{P}$ [23]:

$$f^{\uparrow}(t) = \arg\min_{f\in\mathcal{P}} \sum_{i} \Big(f(\tilde{t}_i) - q_{q_\uparrow}(T_{i,i+w})\Big)^2, \quad (1)$$

$$f^{\downarrow}(t) = \arg\min_{f\in\mathcal{P}} \sum_{i} \Big(f(\tilde{t}_i) - q_{q_\downarrow}(T_{i,i+w})\Big)^2, \quad (2)$$

where $\tilde{t}_i$ represents the time coordinate difference of the right endpoint t_{i+w} of the window $T_{i,i+w}$ relative to the start time.

The fitted curves $f^{\uparrow}(t)$ and $f^{\downarrow}(t)$ are evaluated at each timestamp to obtain the dynamic upper and lower boundaries at each time step. Data points falling outside the interval $[f^{\downarrow}(t), f^{\uparrow}(t)]$ are labeld anomalies.

IV. Multimodal Anomaly Detection

In this section, we present our multimodal anomaly-detection framework and the accompanying evaluation metric, both designed using the TS3D dataset.

A. Mutimodal Anomaly Detection Framework

The proposed framework consists of three components: temporal multimodal alignment, which synchronizes logs and numerical data; adaptive data fusion, which integrates the log and numerical data information; and a self-adaptive mask, which dynamically selects the most relevant features.

1) Temporal Multimodal Data Alignment: Log data is naturally event-driven; each entry is generated only when an SQL request is executed, whereas numerical data is periodically sampled, making the two modalities inherently asynchronous. To address this temporal misalignment, we adopt a "reference-and-align" strategy: for log-centric detection, we use the log timeline and labels as the reference and align the numerical time series to the corresponding log windows; for time-series-centric detection, we use the time-series timeline as the reference and align log events to the corresponding time windows. We set the default alignment window between logs and numerical data to the sampling rate, i.e, to 2 seconds. Using a smaller window would redundantly associate the same numerical values with multiple log windows, whereas a larger window would merge multiple sampling intervals and potentially obscure fine-grained numerical variations. More specifically, log sequences are first parsed into templates and converted into event vectors within fixed time windows. Each vector is then aligned with the corresponding timestamps, ensuring consistent granularity across different modalities.

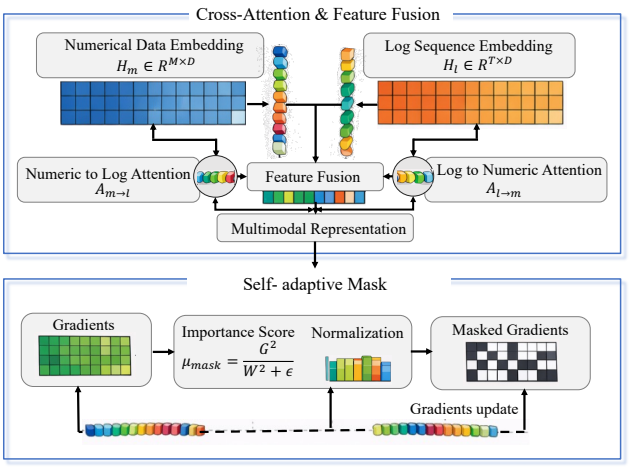

Fig. 2. Multimodal Anomaly Detection Architecture.

Log templates construction. Each raw log entry is mapped to a log template with a unified signature $\mathcal{T} = \langle C, O, A, \Phi, S, R\rangle$. Here, C is the semantic category, O specifies the operation type, A represents the extracted attributes, Φ defines constraints on A for template applicability, S indicates the SQL execution status, and R denotes response time. Please note that not all templates instantiate all fields; for example, connection-level templates lack response time, while query-related templates include both structural attributes and response time.

Template instantiations. Based on the unified template signature, we instantiate six categories of database log templates by specifying the concrete semantics of each field.

(1) Database Connection Templates. They capture database connection events: $\mathcal{T}\text{conn} = \langle \texttt{Connection}, \text{CONNECT}, \text{Sconn}\rangle$, where $\text{S}_{\text{conn}} \in \{\text{success}, \text{failed}, \text{uninitialized}\}$.

(2) Select Templates. These templates represent read-only SQL queries and are defined as: $\mathcal{T}\text{select} = \langle \texttt{SELECT}, \text{select}, A\text{select}, S_s, \Phi_{\text{select}}, R\rangle$. A_{select} includes attributes such as tables (accessed tables), joins (number of joins), aggregation (e.g., AVG, SUM), and nested (boolean for subqueries like EXISTS or IN (SELECT)). Φ_{select} defines structural constraints for template applicability, while S_{select} indicates execution success, and R represents execution time.

(3) Update, Insert and Delete Templates. Update templates describe data modification operations and are formalized as $\mathcal{T}_{\text{update}} = \langle \texttt{UPDATE}, \text{update}, A_{update}, \Phi_{\text{update}}, S_{\text{update}}, R\rangle$. The attribute set A_{update} includes affected tables and modification types, while Φ_{update} specifies applicability conditions related to write operations. The S_{update} represent its execution result and R_{update} reflects the execution duration of the update. The insert and delete templates follow the same formalization.

(4) External Request Templates. These templates capture non-SQL service calls or API invocations: $\mathcal{T}\text{external} = \langle \texttt{External}, \text{request}, A\text{external}, S_{\text{delete}}\rangle$. A_{external} includes service identifiers, and S_{delete} is execution status.

2) Multimodal Data Fusion: Our multimodal data fusion operates on top of existing embedding backbones, including both log-based and time-series-based anomaly detection models, to combine their representations. Figure 2 depicts the data fusion procedure and its key modules.

Cross-attention and feature fusion. This module performs bidirectional cross-attention between the numerical data embeddings and the log sequence embeddings to achieve fine-grained multimodal interaction [24]. To enable mutual interaction between numerical data and logs, we utilize a bidirectional cross-attention mechanism. Given numerical data embeddings $H_m \in \mathbb{R}^{M\times D}$ and log embeddings ($H_l \in \mathbb{R}^{T\times D}$), the attention from numerical data to logs is formulated as:

$$A_{m\to l} = \text{softmax}\left(\frac{(H_m W_Q)(H_l W_K)^\top}{\sqrt{D}}\right)(H_l W_V) \tag{3}$$

where W_Q, W_K, and W_V are learnable weight matrices. Similarly, the reverse attention $A_{l\to m}$ from logs to numerical data is obtained by exchanging the query and key-value sources, enabling each modality to attend to the other. Subsequently, the two directional attention representations are fused and integrated with the source features to construct the final multimodal representation.

Self-adaptive mask. Multimodal data often contains many variables (27 in our case), while only a subset of them is truly discriminative for a specific anomaly scenario. For example, in SELECT-related anomaly scenarios, indicators including qps, select_scan, select_full_join, and slow_queries are highly correlated with the anomaly pattern, while numerical data such as innodb_buffer_pool_write_requests and table_locks_waited contribute minimally.

Training with all numerical data indiscriminately may allow noisy or irrelevant features to dominate the gradient updates, thereby affecting model convergence and the effectiveness of multimodal fusion. To address this issue, we design an adaptive feature masking mechanism inspired by the concept of modality selection [25], which suppresses gradients from less informative features during training (see bottom of Figure 2). The mask is applied to the last fully connected layer, with gradient and weight matrices denoted as $G \in \mathbb{R}^{O\times I}$ and the weight matrix $W \in \mathbb{R}^{O\times I}$, where I and O denote the input and output dimensions, respectively. To quantify the contribution of each dimension, we adopt a gradient-to-weight ratio normalization to compute the importance score matrix:

$$\mu_{\mathbf{mask}} = \frac{G^2}{W^2 + \epsilon} \tag{4}$$

where ϵ is a small constant (e.g., 10^{-8}) to avoid division by zero. The larger the value of $\mu_{\mathbf{mask}}$, the more important the corresponding feature is in the current batch, and thus the more of its gradient update is retained. After constructing $\mu_{\mathbf{mask}}$, we normalize it on each dimension to form a probability distribution:

$$P_{o,i} = \frac{\mu_{o,i}}{\sum_{j=1}^{I} \mu_{o,j}} \tag{5}$$

where $\mu_{o,i}$ is the (o,i)-th entry of the relevance matrix $\mu_{\mathbf{mask}}$ and I is the dimensions of numerical features. Given a retention ratio, we perform multinomial sampling without replacement over each row to select a subset of high-importance features $\mathcal{K}$. A binary mask matrix is then constructed as:

$$M_{o,i} = \begin{cases} 1, & i \in \mathcal{K} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Finally, the mask is applied to the gradient update, which dynamically sparsifies feature gradients and selectively preserves the most informative features during backpropagation.

B. Mixed-type Anomaly Metric

As mentioned earlier, existing metrics are tailored to single-type anomalies. Instead, we introduce the Mixed-type Anomaly metric, MA-F1, which incorporates proximity-aware penalty weights for false positives (FP) and false negatives (FN), determined by their distance from the true anomaly labels. The metric builds on the concept of True Anomaly (TA) segments, defined as follows:

$$\mathcal{S}_{TA} = \{(a_1, b_1, M_1), (a_2, b_2, M_2), \ldots, (a_k, b_k, M_k)\}, \tag{7}$$

where a_k and b_k represent the starting and ending indices of the k-th TA segment, respectively. $M_k = b_k - a_k + 1$ denotes the length of the TA segment, indicating the number of anomalous points within the segment.

We adjust the contribution of FP and FN samples based on their proximity to TA segments. For each FP or FN sample, we compute its distance to the nearest TP segment as:

$$d_i = \min_{(a,b,M)\in\mathcal{S}_{TA}} \begin{cases} a - i & \text{if } i < a \\ i - b & \text{if } i > b \\ 0 & \text{if } a \le i \le b \end{cases} \tag{8}$$

where d_i denotes the minimum distance between the misclassified i-th sample and the nearest TA segment. Accordingly, the weight of an FP/FN sample is determined based on the length of the nearest TA segment and its distance d_i as:

$$w_i = \frac{M}{M + \beta d_i}, \beta > 0 \tag{9}$$

where M is the length of the nearest TA segment, β is a decay factor that controls the rate of weight reduction with increasing distance. The weights of correctly classified samples, including True Positives (TP) and True Negatives (TN), remain static at 1. This ensures that correctly classified samples do not introduce additional bias into the evaluation process.

Using all the previous definitions, MA-F1 is defined as:

$$\text{MA-F1} = 2 \times \frac{|I_{TP}|}{2|I_{TP}| + \sum_{i\in I_{FP}} w_i + \sum_{i\in I_{FN}} w_i} \tag{10}$$

where $|I_{TP}|$, $|I_{FP}|$, and $|I_{FN}|$ represent the number of true positive, false positive, and false negative samples, respectively. This evaluation metric aims to mitigate the impact of imbalanced FP/FN distributions on evaluation, making the F1 calculation more robust and suitable for fair assessment in anomaly detection tasks.

V. R-SQLs Discovery Method

In this section, we present our R-SQL discovery method, which includes three key components: Timeout Chains Discovery, R-SQLs & Pod-aware SQL Optimization.

A. Timeout Chains Discovery

More often than not, anomalies arise from table-level dependencies, which often manifest as increased query latency. A slowdown in an upstream table can propagate through the dependency chain, causing blocking, queuing, and ultimately timeouts. Moreover, SQL operations are usually executed concurrently, and each has a very short execution time. As a result, during anomalous periods, many SQL queries may run simultaneously.

Identifying the longest timeout chain is crucial for understanding how delays propagate across dependent queries and for revealing how localized slowdowns accumulate into system-wide anomalies. However, determining this chain becomes challenging when execution overlaps are complex. We propose to solve this problem using a time-dependency graph.

Time-dependency graph. Given a set of SQL templates $\{Q_1, Q_2, \ldots, Q_i, \ldots, Q_j\}$, the time-dependency graph is a directed, weighted graph where nodes represent SQL queries, and edges capture the temporal dependencies between them. The direction of each edge is determined by the execution times of the queries:

(1) $Q_i \to Q_j$: If the end time of Q_i is less than or equal to the start time of Q_j, i.e., $Q_i.t_i + Q_i.D_i \le Q_j.t_i$, a directed edge $Q_i \to Q_j$ is established, indicating that Q_i executes before Q_j.

(2) $Q_i \leftrightarrow Q_j$: If the time intervals of Q_i and Q_j overlap, i.e., $Q_i.t_i \le Q_j.t_j + Q_j.D_j$ and $Q_j.t_j < Q_i.t_i + Q_i.D_i$,, a bidirectional edge is created, indicating that Q_i and Q_j execute in parallel.

The weight of each edge represents the execution time (duration) of the target node. For example, if Q_i runs for 20 ms and Q_j runs for 50 ms, then the edge $Q_i \to Q_j$ has weight 50 ms, while $Q_j \to Q_i$ has weight 20 ms.

After constructing the temporal dependency graph, we use dynamic programming to compute the longest execution time for each timeout node by traversing the nodes and updating their execution times, ensuring that each node's time is derived from the longest path. We then identify nodes with execution times exceeding the timeout threshold, trace their paths, calculate the total execution time, and update the critical path and maximum duration. Finally, the longest timeout path and its maximum duration are returned.

During the construction and evaluation of the time-dependency graph, we compute, for each query node, the longest accumulated execution time from its valid predecessors in $O(n)$ time, where n denotes the number of queries. In the Timeout Chains Discovery stage, we select the timeout node with the maximum dynamic programming value and perform a single backtracking using the stored predecessor pointers, whose cost is linear in the length of the resulting

chain. Consequently, the overall computational cost remains linear in the graph size.

B. R-SQLs & Pod-aware SQL Optimization

After identifying the longest timeout chain, the next step is to analyze its root cause. Thus, we need to further construct a dependency graph of the SQL queries in the chain. Specifically, given a timeout chain $critical\_path$, we construct an SQL dependency chain based on execution order, focusing primarily on table-level dependencies. In this graph, nodes represent individual SQL templates, while directed edges capture dependency relationships between queries.

A table dependency chain represents the relationships and dependencies between multiple SQL queries or templates in a database system. Given a timeout SQL templates $\{Q_1, Q_2, \ldots, Q_i, \ldots, Q_j\}$, we construct a directed graph where nodes represent SQL queries, and edges denote table dependencies between them, specifically:

(1) $Q_i \rightarrow Q_j$ (Write Dependency): If Q_i and Q_j are both write operations (INSERT, UPDATE, DELETE) and Q_i completes after Q_j starts, i.e., $Q_i.t_i + Q_i.D_i > Q_j.t_i$, then a directed edge $Q_i \rightarrow Q_j$ is established, indicating that Q_j depends on the execution of Q_i.

(2) $Q_s \rightarrow Q_w$ (Read-After-Write Dependency): If a SELECT query Q_s accesses a table modified by the most recent UPDATE or DELETE operation Q_w, and Q_w completes after Q_s starts, i.e., $Q_w.t_i + Q_w.D_i > Q_s.t_i \,\&\, Q_w.T \cap Q_s.T \neq \emptyset$, then a directed edge $Q_s \rightarrow Q_w$ is established, ensuring that Q_s reads the latest updates from Q_w. Write-After-Read dependencies are omitted since reads in MySQL do not block subsequent writes.

The dependency chain helps understand, for example, how a write operation (e.g., an INSERT or UPDATE query) affects subsequent read operations (e.g., SELECT queries). Consider the following scenario, the execution times are ordered chronologically, with an interval of 100 ms:

- Q_1:t_1, INSERT INTO orders (order_id, customer_id) VALUES (?, ?), correspond time = 300
- Q_2: t_1+100, UPDATE customer_id FROM orders WHERE order_id = ?, correspond time = 102
- Q_3: $t_1 + 200$, SELECT order_id FROM orders WHERE customer_id IN (SELECT customer_id FROM customers WHERE customer_name = "Jack"), correspond time = 120

Based on the execution and response times of the SQL queries, we establish the dependencies: $Q_2 \rightarrow Q_1$, $Q_3 \rightarrow Q_2$, and $Q_3 \rightarrow Q_1$. Q_3 must wait for both Q_1 and Q_2 to complete before execution, while Q_2 depends on the outcome of Q_1.

We further optimize the execution order of distributed SQL queries to improve concurrency efficiency. The SQLs are grouped based on the pods they are executed on, and sorted by the dependency strength (i.e., in-degree in a graph) of each Pod. Queries without dependencies are prioritized for optimization to ensure efficient scheduling. As illustrated in our example, we first address the Q_8 node in Pod0 with the fewest dependencies, allowing subsequent dependency relationships to be resolved step by step.

VI. Experimental Evaluation

We conduct extensive experiments on the TS3D dataset to evaluate its effectiveness in supporting temporal multimodal anomaly detection and distributed SQL-level root cause localization tasks.

The code of our proposed methods, including anomaly detection, MA-F1 metric, and root cause localization, is publicly available [2].

A. Experimental Setup

Anomaly detection. We select three representative baselines for log anomaly detection: DeepLog [26], LogAnomaly [27], RobustLog [28]. In addition, we evaluated two popular anomaly detection baselines used in a popular time series benchmark [29], namely AutoEncoder [30] and TimesNet [31].

To validate the applicability of our proposed metric, MA-F1, with both point and segment anomalies, we compare it against 9 metrics: F1, PA-F1 [16], AUC-PR [32], VUS-PR [33], AUC-ROC [34], Event-based-F1 [35], R-based-F1 [36], and Affiliation-F1 [37].

Root discovery. No existing method currently identifies the root cause SQL chain based on table-level dependencies. To compare with our approach, we designed three possible strategies to solve our task: (1) *ET-SQL*: uses only execution time to identify anomalous SQL queries, (2) *ALL-SQL*: models the dependencies between all SQL queries within a specified time range, and (3) *POD-SQL*: constructs dependencies based on SQL queries within each pod and filters out normal dependencies using execution time.

To evaluate the root discovery algorithm's performance, we use four metrics. *ACC-sorted* measures accuracy while considering the order of predicted links and is defined as ACC-sorted $= \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(\hat{l}_i = l_i)$, where $\mathbb{1}(\cdot)$ is an indicator function that returns 1 if the condition holds and 0 otherwise. ACC-unsorted $= \frac{|L \cap \hat{L}|}{|L|}$ evaluates accuracy based on set overlap, ignoring order. Error-Rate $= \frac{1}{\hat{n}}\sum_{i=1}^{\hat{n}} \mathbb{1}(\hat{l}_i \notin L)$ quantifies the proportion of incorrectly predicted links relative to the total predicted length. Lastly, APLR (Anomaly Prediction Length Ratio) $= \frac{\hat{n}}{n}$ captures the ratio of the predicted anomaly length to the ground truth length.

B. Anomaly Detection Performance

1) Efficacy Comparison: We compare multimodal, log, and numerical data settings across three log-based anomaly detectors and two time-series-based models.

Comparison with log data. The results in Table IV show that multimodal data consistently outperforms log-only data, especially in Recall, F1, and MA-F1, with improvements of 26%, 11%, and 11%, respectively. This indicates that combining numerical data with log semantics improves anomaly coverage. The improvement mainly stems from the complementary nature of the two modalities: logs capture operational semantics,

[2]https://github.com/Yian-Wei/TS3D

TABLE IV
ANOMALY DETECTION PERFORMANCE COMPARISON: MULTIMODAL VS. SINGLE-MODAL DATA.

Data			Multimodal data				Single-modal data			
Metrics			Precision	Recall	F1	MA-F1	Precision	Recall	F1	MA-F1
Single2Single	Log	DeepLog	0.966	**0.927**	**0.946**	**0.959**	**0.988**	0.830	0.902	0.906
		LogAnomaly	0.986	**0.866**	**0.922**	**0.927**	**0.987**	0.827	0.900	0.904
		RobustLog	**0.802**	**0.965**	**0.876**	**0.892**	0.777	0.958	0.858	0.875
	Numerical	AutoEncoder	**0.338**	**0.352**	**0.345**	0.439	0.335	0.335	0.328	**0.452**
		TimesNet	**0.230**	**0.240**	**0.235**	**0.324**	0.188	0.196	0.192	0.274
Single2Multi	Log	DeepLog	0.972	**0.885**	**0.926**	**0.936**	**0.985**	0.795	0.880	0.884
		LogAnomaly	**0.989**	**0.831**	**0.903**	**0.906**	0.985	0.806	0.886	0.891
		RobustLog	**0.760**	**0.784**	**0.772**	**0.790**	0.654	0.664	0.659	0.687
	Numerical	AutoEncoder	**0.251**	**0.668**	**0.365**	**0.681**	0.151	0.401	0.219	0.480
		TimesNet	**0.087**	**0.231**	**0.126**	**0.290**	0.083	0.222	0.121	0.278
Multi2Single	Log	DeepLog	0.965	**0.944**	**0.954**	**0.968**	**0.983**	0.897	0.938	0.944
		LogAnomaly	**0.984**	**0.910**	**0.945**	**0.951**	0.980	0.902	0.940	0.947
		RobustLog	0.824	**0.958**	**0.886**	**0.905**	**0.883**	0.701	0.782	0.791
	Numerical	AutoEncoder	**0.294**	**0.571**	**0.388**	**0.631**	0.287	0.558	0.379	0.617
		TimesNet	**0.142**	**0.275**	**0.187**	**0.334**	0.141	0.275	0.187	0.333
Multi2Multi	Log	DeepLog	0.768	**0.938**	**0.845**	**0.962**	**0.786**	0.913	0.844	0.950
		LogAnomaly	**0.801**	0.904	**0.849**	0.945	0.770	**0.931**	0.843	**0.959**
		RobustLog	0.588	**0.927**	**0.719**	**0.773**	**0.633**	0.786	0.701	0.742
	Numerical	AutoEncoder	**0.696**	**0.370**	**0.484**	**0.508**	0.688	0.366	0.478	0.504
		TimesNet	**0.472**	**0.251**	**0.328**	**0.356**	0.465	0.248	0.323	0.354

while numerical data reflect system states, providing richer contextual information for anomaly perception.

Precision gains are limited and slightly lower because sometimes the multimodal fusion mechanism amplifies anomalous features, leading to false positives. These false positives mainly arise because (i) labels are log-triggered, while failures often begin with earlier degradations, so early detections may fall outside labeled intervals; (ii) sampled/aggregated numerical data introduce granularity mismatch and alignment noise that can shift or widen predicted windows; and (iii) in high-concurrency settings (e.g., multi2multi), indicator jitters may be flagged by a more sensitive detector.

Among the four interaction modes, Multi2Multi shows the lowest overall performance, especially in F1. This is mainly because multi-pod data is highly asynchronous and heterogeneous, which amplifies misalignment and noise. Overall, the multimodal fusion significantly improves recall and robustness but also exposes challenges in temporal alignment and noise suppression in complex multi-pod scenarios.

Comparison with numerical data. Adding logs to metrics yields higher F1 and MA-F1 in most settings than using time series alone. The gain is more pronounced in cross-domain scenarios with stronger distribution shifts, especially in Single2Multi. For AutoEncoder, in the Single2Multi setting, recall increases from 0.401 to 0.668 and MA-F1 rises from 0.480 to 0.681; in the Multi2Single setting, multimodal data also brings a stable improvement (Precision, Recall, F1, and MA-F1 are higher than the numerical counterpart).

These results indicate that the discrete event semantics provided by logs complement the numeric time series patterns, and that multimodal signals improve cross-scenario robustness and anomaly coverage, particularly when the metric distribution drifts with increasing workload complexity. Also, due to inherent differences in sampling granularity and temporal alignment between logs and metrics, multimodal fusion may slightly widen or shift the predicted anomaly boundaries; since MA-F1 is more sensitive to boundary tightness, this also explains the small fluctuations observed in a few in-domain settings (e.g., Single2Single).

TABLE V
ABLATION STUDY ON MULTIMODAL ANOMALY DETECTION.

Method	Components	Precision	Recall	F1	MA-F1
DeepLog	w/o Cross Attention	0.833	0.849	0.841	0.916
	w/o Self-adaptive mask	**0.834**	0.847	0.840	0.915
	Multimodal AD	0.768	**0.938**	**0.845**	**0.962**
Log Anomaly	w/o Cross Attention	0.823	0.863	0.842	0.924
	w/o Self-adaptive mask	**0.831**	0.850	0.840	0.917
	Multimodal AD	0.801	**0.904**	**0.849**	**0.945**
Robost Log	w/o Cross Attention	0.520	0.910	0.662	0.737
	w/o Self-adaptive mask	**0.596**	0.889	0.714	0.764
	Multimodal AD	0.588	**0.927**	**0.720**	**0.773**

2) Ablation Study: We perform an ablation study to evaluate the key components of our multimodal anomaly detection

TABLE VI
TRAINING COST OF EACH EPOCH (S)

Dataset		Single2Single		Single2Multi		Multi2Single		Multi2Multi	
Data Type		Multimodal	Log-only	Multimodal	Log-only	Multimodal	Log-only	Multimodal	Log-only
Method	DeepLog	1.32	0.68	6.71	3.80	2.13	1.47	22.26	10.10
	LogAnomaly	1.62	1.03	8.03	4.91	3.25	2.11	11.92	10.33
	RobustLog	2.76	2.10	3.23	2.80	2.87	2.32	3.93	3.19

framework: cross attention and the self-adaptive mask. We apply the three log-based backbones on the 1-day Multi2Multi dataset. Table V depicts the results.

We observe that the multimodal framework consistently achieves the highest F1 and MA-F1 across all backbones, with recall improving by up to 8% and MA-F1 by up to 4%. This result confirms that cross attention effectively models cross-modal interactions and alignment, improving anomaly coverage and segment-level boundary quality. The self-adaptive mask selectively re-weights cross-modal features, enhancing anomaly detection by suppressing noise and misaligned segments. Without this module, the model becomes more conservative, improving precision but reducing recall and MA-F1, as it misses weaker or boundary anomalies.

3) Runtime Comparison: We measure the per-epoch training cost of different models under four dataset settings, comparing multimodal with log inputs. As shown in Table VI, multimodal training incurs a higher per-epoch cost than log training, with the gap becoming more pronounced in larger data volumes and richer modalities. For example, in the Multi2Multi setting, DeepLog requires 22.26 s per epoch with multimodal inputs compared to 10.10 s with log inputs.

This increase in training cost arises from two main factors. First, multimodal learning introduces additional computational overhead during both forward and backward propagation, as the model must encode extra modalities, perform cross-modal fusion (e.g., higher-dimensional feature concatenation), and execute the adaptive mask-encoding module, all of which increase computation as well as memory and I/O traffic. Second, the absolute training time is influenced by the effective sample size. The increase in execution time observed in the Multi2Multi and Single2Multi suggests larger training sets or denser window sampling, leading to more iterations per epoch.

In addition, we evaluate the computational cost of temporal multimodal alignment on the Multi2Multi dataset using 24 hours of data (1.2M samples). The total preprocessing time for multimodal data is 80.22 sec, which includes both log template parsing and temporal alignment between log and metric modalities. In contrast, preprocessing logs alone, i.e., log template conversion without multimodal alignment, requires 67.67 sec. Thus, the addition of temporal alignment introduces an overhead of only 12.55 sec, corresponding to an average alignment cost of approximately 10.46 microseconds per data point (about 15% of the total preprocessing time). These results show that the overhead of temporal multimodal alignment is minimal relative to the overall processing pipeline.

C. Mixed-type Anomaly Metric Analysis

1) Metric Evaluation: We evaluate the sensitivity of our metric by varying recall from 100% to 0% and increasing the false positive rate from 5% to 10%. This allows us to assess whether each metric consistently captures the known performance differences. We do so by constructing six synthetic detection outputs (Rank1–Rank6) with progressively degraded performance. Specifically, Rank1 achieves 100% recall for all anomaly types with only 5% false positives; Rank2 covers all types but introduces boundary errors near anomalies and occasional false alarms in distant regions; Rank3 detects only one anomaly type; Rank4 randomly misses 50% of anomalies; Rank5 misses 50% of anomalies while increasing false positives to 10%; and Rank6 misses 90% of anomalies with 10% false positives.

TABLE VII
METRIC PERFORMANCE ON 20% ANOMALY INJECTION RATE.

Metric	Rank1	Rank2	Rank3	Rank4	Rank5	Rank6
F1	0.92	0.85	0.90	0.80	0.54	0.20
PA-F1	0.92	0.85	1.00	0.90	0.76	0.20
Event-based-F1	0.92	0.85	1.00	0.85	0.66	0.25
R-based-F1	0.88	0.85	0.96	0.81	0.68	0.28
Affiliation-F	0.93	0.93	0.99	0.85	0.78	0.71
AUC-PR	0.85	0.75	0.86	0.72	0.39	0.19
AUC-ROC	0.98	0.96	0.91	0.83	0.71	0.52
VUS-PR	0.96	0.97	0.98	0.97	0.96	0.95
VUS-ROC	0.99	0.99	0.99	0.98	0.97	0.96
MA-F1	0.99	0.93	0.90	0.80	0.57	0.22

Table VII provides the results of Rank1–Rank6 models. As observed, MA-F1 exhibits a strictly monotonic decrease from Rank1 (0.9980) to Rank6 (0.2358), perfectly aligning with the model performance degradation, indicating its high sensitivity to declining detection capability. In contrast, although other metrics show an overall downward trend, they suffer from non-monotonicity or insufficient discriminative power. For example, Standard-F1 and PA-F1 perform better in Rank3 than in Rank2, indicating inconsistencies, while AUC-ROC drops only 0.18 from Rank5 to Rank6, compared to the 0.34 decrease of MA-F1, demonstrating its superior discriminative power. This can be explained by the fact that TSD-F1 dynamically assigns weights to false positives (FP) and false negatives (FN) based on their distance from true anomaly (TA) segments. This approach effectively minimizes the impact of isolated misclassifications on the overall evaluation while preventing

the overestimation of segment anomalies, resulting in a more stable and accurate performance for anomaly detection.

2) Parameter Study: To better understand the factors that influence the MA-F1 metric, we ran a parameter sensitivity analysis on the decay factor β. We use the 24-hour Multi2Multi dataset, comprising approximately 1.2 million training samples. The results in Figure 3 show that MA-F1 exhibits an overall increasing trend as β becomes larger, indicating that β plays a critical role in regulating the penalty assigned to misclassified samples during evaluation.

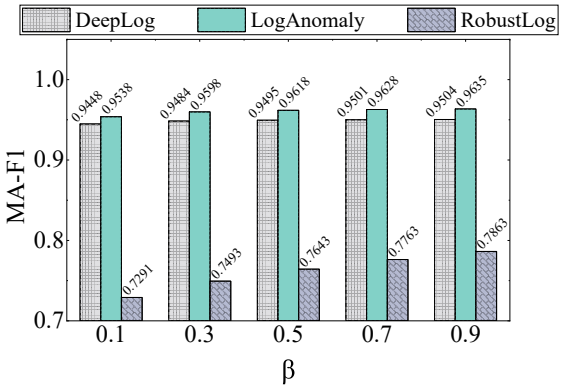

Fig. 3. Impact of β on Performance.

More specifically, a larger β reduces the influence of false positives (FP) and false negatives (FN) that are far from true anomaly segments, thereby allowing the metric to focus more on the detection quality of anomaly segments and their boundaries. This behavior is consistent with practical anomaly detection scenarios, where isolated misclassifications far from anomalous regions typically have limited impact on system diagnosis, while errors occurring near or within anomaly segments are more critical. By down-weighting distant misclassifications, larger values of β more accurately reflect a model's overall capability in detecting anomaly segments. While larger β often improves MA-F1, an excessively large β may overly downweight isolated mispredictions, whereas an overly small β can over-penalize errors far from anomaly segments and mask true segment-level performance. For stability across models and window settings, we use $\beta = 0.5$ by default, balancing point-wise and segment-level anomalies.

We conducted an additional experiment to determine the optimal window size (full details are available in our GitHub repository). Our results show that increasing the window size consistently improves Precision, Recall, F1, and MA-F1, with the largest gains between sizes 2 and 6. Larger windows capture longer event contexts, helping models learn temporal dependencies and better distinguish normal behavior from anomalies. Performance gains plateau at sizes 8–10, suggesting limited benefit from longer contexts. In terms of efficiency, the results show a slight runtime decrease with larger windows, since bigger windows generate fewer training sequences within the same time span.

D. R-SQLs Discovery Performance

1) Efficacy Comparison: Table VIII shows the quality performance of R-SQLs discovery. We observe that our method (denoted as TS3D-SQL) achieves the highest scores in both ACC-sorted (denoted as ACC) and ACC-unsorted (abbreviated as ACCS), with both values being identical. In contrast, ALL-SQL exhibits slight accuracy drops in all scenarios, while POD-SQL exhibits drops in only two scenarios.This is because our method accurately identifies table dependency chains while preserving their correct order, ensuring consistency in both order-sensitive and order-agnostic accuracy measurements. Unlike TS3D-SQL, ALL-SQL constructs dependencies within a fixed time window, introducing redundant relationships that reduce accuracy, while POD-SQL limits dependency construction to individual pods, potentially missing critical cross-pod relationships. These structural limitations lead to the observed accuracy drops in different scenarios.

TABLE VIII
COMPARISON OF R-SQLS DISCOVERY PERFORMANCE.

Type	Methods	ACC ↑	ACCS ↑	ER↓	APLR↑
Multi2Multi	ET-SQL	0.42	0.42	0.54	0.54
	ALL-SQL	0.55	0.53	0.31	0.65
	POD-SQL	0.55	0.55	0.53	0.23
	TS3D-SQL	**0.65**	**0.65**	**0.19**	**0.89**
Single2Multi	ET-SQL	0.53	0.53	0.54	0.54
	ALL-SQL	**0.57**	0.55	0.52	0.65
	POD-SQL	0.37	0.31	0.51	0.54
	TS3D-SQL	0.55	**0.55**	**0.36**	**0.89**
Multi2Single	ET-SQL	0.53	0.53	0.54	0.54
	ALL-SQL	0.55	0.49	0.51	0.60
	POD-SQL	0.55	0.55	0.49	0.32
	TS3D-SQL	**0.72**	**0.72**	**0.33**	**0.71**
Single2Single	ET-SQL	0.53	0.53	0.54	0.54
	ALL-SQL	0.57	0.53	0.48	0.56
	POD-SQL	0.57	0.53	0.37	0.23
	TS3D-SQL	**0.67**	**0.67**	**0.26**	**0.91**

Additionally, our method achieves the lowest Error Rate (abbreviated as ER), indicating that most of the identified R-SQL dependency chains are correct, with minimal false identifications. This high performance is attributed to the fact that our approach combines time-dependency and table-dependency, significantly reducing irrelevant SQL entries and enhancing the precision of dependency parsing. This pre-filtering strategy reduces incorrect matches, ensuring more accurate R-SQL dependency chains.

Finally, our method achieves the highest APLR (up to 35% improvement), indicating that it identifies the most complete R-SQL dependency chains. This is because our approach models dependencies based on temporal relationships, effectively filtering out irrelevant connections while accurately capturing critical SQL associations, thereby enhancing the completeness of R-SQL identification. In contrast, POD-SQL performs the worst, as it constructs dependencies only within individual pods, making it incapable of identifying essential cross-pod

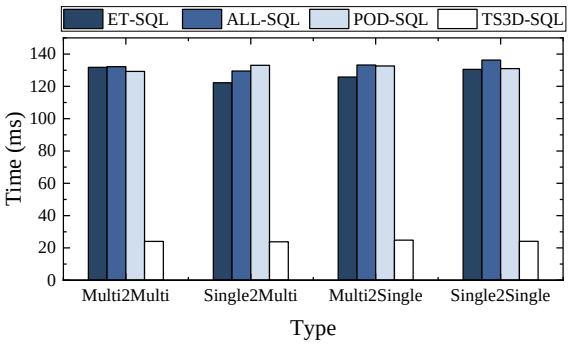

Fig. 4. R-SQLs Discovery Time per 100 SQLs.

SQL associations. This limitation reduces its detection coverage and leads to the omission of important dependencies.

2) Runtime Comparison: Figure 4 provides the efficiency evaluation results. As observed, our method achieves the fastest execution speed, requiring only 1/5 of the time compared to other methods, while ALL-SQL exhibits the slowest performance due to its high computational overhead. The 5x runtime improvement yielded by our method stems from early filtering and selective dependency construction. Instead of building dependencies for all SQL queries within a time window, we first filter out queries by execution time and focus only on potentially abnormal timeout-related SQL chains, which substantially reduces the number of queries and dependencies to be analyzed. In contrast, ALL-SQL constructs dependencies for all SQLs, leading to excessive and redundant computations. ET-SQL still incurs high latency due to parsing all SQL statements, while POD-SQL, although restricting dependency construction to individual pods, continues to process a large volume of SQL queries.

VII. Related Work

A. Anomaly Detection for Databases

A wide range of anomaly detection and diagnosis methods have been proposed for database systems, spanning both metrics-based and log-based approaches. Metrics-based methods include iSQUAD [38], which leverages a Bayesian Case Model over Key Performance Indicators (KPIs) to detect anomalies in intermittent slow queries (iSQs); OpenGauss [39], which integrates an LSTM-based autoencoder with attention for system-level diagnosis; and FluxInfer [40], which models correlations among anomalous metrics using a weighted undirected dependency graph and applies a weighted PageRank algorithm for online anomaly identification.

On the other hand, log-based approaches, such as Sentinel [41], PinSQL [13], and MultiLog [12], rely on textual analysis of database logs. Sentinel constructs fine-grained behavior models from debug logs to assist DBAs, PinSQL identifies SQL statements highly correlated with performance anomalies to improve DBMS robustness, and MultiLog employs an LSTM with self-attention for node-level encoding together with a cluster-level autoencoder and meta-classifier to detect global anomalies in distributed databases.

To the best of our knowledge, our work is the first to construct a multimodal dataset specifically tailored for database management and analysis. Unlike prior work that relies on a single modality, our dataset integrates database metrics, logs, and SQL statements to provide a holistic view of system behavior. Furthermore, our approach jointly leverages logs and metrics from distributed nodes to build a multimodal anomaly detection model and utilizes executed SQL queries to identify anomalous SQL chains across the database cluster.

B. Database Diagnostics Tools

Many cloud service providers offer diagnostic tools to help optimize distributed database performance [42]–[45]. AWS Performance Insights [46] captures performance metrics and SQL query information to quickly identify bottlenecks. Azure Intelligent Insights uses machine learning to automatically monitor database performance and provide optimization recommendations. Tencent DB Brain [47] leverages AI to analyze SQL statements and execution plans, recommending optimization strategies to handle large-scale databases. Huawei DBAS [48] provides multi-dimensional monitoring and diagnostics, predicting performance bottlenecks and offering optimization solutions. AliCloud DAS [49] diagnoses issues such as load and latency through automation and machine learning, offering tailored optimization strategies for complex use cases. Google Cloud SQL Insights [50] predicts query performance, analyzes execution plans, and optimizes query efficiency. Oracle Autonomous Database [51] optimizes database configuration through self-management and machine learning, improving stability and performance.

In complex environments with numerous SQL queries, database diagnostics tools often struggle to accurately pinpoint the exact cause of performance issues. While they can identify slow or resource-heavy queries, they face difficulties determining which specific queries are contributing to the root cause and execution chain of performance degradation. Our approach addresses this problem by dynamically analyzing the dependency graph of SQL executions to identify the anomalous SQL chain.

VIII. Conclusion

This paper presents TS3D, a temporal multimodal dataset for distributed databases. We show the effectiveness and utility of the dataset through two representative tasks: temporal multimodal anomaly detection and distributed SQL-level root cause localization. Extensive experiments validate the effectiveness and generalizability of the TS3D dataset. In future work, we plan to further expand the scale and SQL diversity of the TS3D dataset to better capture the complexity of real-world distributed database scenarios.

Acknowledgement

This work was supported in part by the NSFC under Grant No. 62472377. Lu Chen is the corresponding author.

AI-Generated Content Acknowledgement

The authors declare that all scientific ideas, analyses, and results were conceived, implemented, and validated by the authors. AI tools were used solely for text polishing.

References

[1] "Pingcap's tidb," https://www.pingcap.com/, 2025, 2025.03.09.

[2] "Apache iotdb," https://iotdb.apache.org/, 2025, 2025.03.09.

[3] "Google's spanner," https://cloud.google.com/spanner, 2025, 2025.03.09.

[4] "Huawei's gaussdb," https://www.huaweicloud.com/intl/en-us/product/gaussdb.html, 2025, 2025.03.09.

[5] "Alibaba's database autonomy service (das)," https://www.alibabacloud.com/en/product/das?_p_lc=1, 2025, 2025.03.09.

[6] "Aiops dataset," https://github.com/alibaba/clusterdata, 2025, 2025.03.09.

[7] Z. Li, N. Zhao, S. Zhang, Y. Sun, P. Chen, X. Wen, M. Ma, and D. Pei, "Constructing large-scale real-world benchmark datasets for aiops," *CoRR*, vol. abs/2208.03938, 2022.

[8] "Aiops-challenge-2020-data," https://github.com/NetManAIOps/AIOps-Challenge-2020-Data, 2025, 2025.03.09.

[9] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *FSE*, 2019, pp. 683–694.

[10] M. Ma, S. Zhang, J. Chen, J. Xu, H. Li, Y. Lin, X. Nie, B. Zhou, Y. Wang, and D. Pei, "Jump-starting multivariate time series anomaly detection for online service systems," in *USENIX*, pp. 413–426.

[11] L. Wang, C. Zhang, R. Ding, Y. Xu, Q. Chen, W. Zou, Q. Chen, M. Zhang, X. Gao, H. Fan, S. Rajmohan, Q. Lin, and D. Zhang, "Root cause analysis for microservice systems via hierarchical reinforcement learning from human feedback," in *ACM SIGKDD*, A. K. Singh, Y. Sun, L. Akoglu, D. Gunopulos, X. Yan, R. Kumar, F. Ozcan, and J. Ye, Eds., 2023, pp. 5116–5125.

[12] L. Zhang, T. Jia, M. Jia, Y. Li, Y. Yang, and Z. Wu, "Multivariate log-based anomaly detection for distributed database," in *ACM SIGKDD*, 2024, pp. 4256–4267.

[13] X. Liu, Z. Yin, C. Zhao, C. Ge, L. Chen, Y. Gao, D. Li, Z. Wang, G. Liang, J. Tan *et al.*, "Pinsql: Pinpoint root cause sqls to resolve performance issues in cloud databases," in *ICDE*, 2022, pp. 2549–2561.

[14] M. Mondal, M. Khayati, H. Sandlin, and P. Cudré-Mauroux, "A survey of multimodal event detection based on data fusion," *VLDB J.*, vol. 34, no. 1, p. 9, 2025.

[15] A. Khelifati, M. Khayati, P. Cudré-Mauroux, A. Hänni, Q. Liu, and M. Hauswirth, "VADETIS: an explainable evaluator for anomaly detection techniques," in *ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 2661–2664. [Online]. Available: https://doi.org/10.1109/ICDE51399.2021.00298

[16] S. Kim, K. Choi, H. Choi, B. Lee, and S. Yoon, "Towards a rigorous evaluation of time-series anomaly detection," in *AAAI*, 2022, pp. 7194–7201.

[17] C. Bell, M. Kindahl, and L. Thalmann, *MySQL high availability: tools for building robust data centers*. " O'Reilly Media, Inc.", 2010.

[18] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao, "Amazon aurora: Design considerations for high throughput cloud-native relational databases," in *ACM SIGMOD*, 2017, pp. 1041–1052.

[19] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss *et al.*, "Cockroachdb: The resilient geo-distributed sql database," in *ACM SIGMOD*, 2020, pp. 1493–1509.

[20] "Kubernetes clusters," https://kubernetes.io/, 2025, 2025.03.09.

[21] "Prometheus," https://prometheus.io/, 2025, 2025.03.09.

[22] "locust," https://locust.io/, 2025, 2025.03.09.

[23] F.-L. Chu, "Forecasting tourism demand: a cubic polynomial approach," *Tourism management*, vol. 25, no. 2, pp. 209–218, 2004.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, vol. 30, pp. 5998–6008, 2017.

[25] Y. Yang, H. Pan, Q. Jiang, Y. Xu, and J. Tang, "Learning to rebalance multi-modal optimization by adaptively masking subnetworks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 47, no. 6, pp. 4553–4566, 2025.

[26] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *ACM SIGSAC*, 2017, pp. 1285–1298.

[27] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *IJCAI*, 2017, pp. 4739–4745.

[28] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, S. Furao, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proc. of ESEC/FSE*, 2019, pp. 807–817.

[29] Q. Liu and J. Paparrizos, "The elephant in the room: Towards a reliable time-series anomaly detection benchmark," in *NeurIPS*, vol. 37, 2024, pp. 108 231–108 261.

[30] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proc. of the 2nd MLSDA Workshop*, 2014, pp. 4–11.

[31] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long, "Timesnet: Temporal 2d-variation modeling for general time series analysis," *arXiv preprint arXiv:2210.02186*, 2022.

[32] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *ICML*, 2006, pp. 233–240.

[33] J. Paparrizos, P. Boniol, T. Palpanas, R. S. Tsay, A. Elmore, and M. J. Franklin, "Volume under the surface: a new accuracy evaluation measure for time-series anomaly detection," *Proc. VLDB Endow.*, vol. 15, no. 11, pp. 2774–2787, 2022.

[34] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[35] A. Garg, W. Zhang, J. Samaran, R. Savitha, and C.-S. Foo, "An evaluation of anomaly detection and diagnosis in multivariate time series," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 6, pp. 2508–2517, 2021.

[36] N. Tatbul, T. J. Lee, S. Zdonik, M. Alam, and J. Gottschlich, "Precision and recall for time series," *NeurIPS*, vol. 31, 2018.

[37] A. Huet, J. M. Navarro, and D. Rossi, "Local evaluation of time series anomaly detection algorithms," in *ACMSIGKDD*, A. Zhang and H. Rangwala, Eds., 2022, pp. 635–645.

[38] M. Ma, Z. Yin, S. Zhang, S. Wang, C. Zheng, X. Jiang, H. Hu, C. Luo, Y. Li, N. Qiu *et al.*, "Diagnosing root causes of intermittent slow queries in cloud databases," *Proc. VLDB Endow.*, vol. 13, no. 8, pp. 1176–1189, 2020.

[39] G. Li, X. Zhou, J. Sun, X. Yu, Y. Han, L. Jin, W. Li, T. Wang, and S. Li, "opengauss: An autonomous database system," *Proc. VLDB Endow.*, vol. 14, no. 12, pp. 3028–3042, 2021.

[40] P. Liu, S. Zhang, Y. Sun, Y. Meng, J. Yang, and D. Pei, "Fluxinfer: Automatic diagnosis of performance anomaly for online database system," in *IPCCC*. IEEE, 2020, pp. 1–8.

[41] U. Kanjir, N. Duric, and T. Veljanovski, "Sentinel-2 based temporal detection of agricultural land use anomalies in support of common agricultural policy monitoring," *ISPRS*, vol. 7, no. 10, p. 405, 2018.

[42] Y. Yao, L. Chen, Z. Fang, Y. Gao, C. S. Jensen, and T. Li, "Camel: Efficient compression of floating-point time series," *ACM SIGMOD*, vol. 2, no. 6, pp. V2mod227:1–V2mod014:26, 2025.

[43] Y. Yao, D. Li, H. Jie, L. Chen, T. Li, J. Chen, J. Wang, F. Li, and Y. Gao, "Simplets: An efficient and universal model selection framework for time series forecasting," *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 3741–3753, 2023.

[44] Y. Yao, H. Jie, L. Chen, T. Li, Y. Gao, and S. Wen, "Tsec: An efficient and effective framework for time series classification," in *ICDE*. IEEE, 2024, pp. 1394–1406.

[45] Y. Yao, S. Dai, Y. Li, L. Chen, D. Li, Y. Gao, and T. Li, "A demonstration of TENDS: time series management system based on model selection," *Proc. VLDB Endow.*, vol. 17, no. 12, pp. 4357–4360, 2024.

[46] "Amazon web services," https://aws.amazon.com/es/ec2/, 2025, 2025.03.20.

[47] "Tencentdb for dbbrain," https://www.tencentcloud.com/products/dbbrain, 2025, 2025.03.20.

[48] "Huawei dbas," https://www.huaweicloud.com/intl/en-us/product/das.html, 2025, 2025.03.20.

[49] "Alicloud das," https://www.alibabacloud.com/help/en/das/, 2025, 2025.03.20.

[50] "Google cloud sql insights," https://www.datadoghq.com/, 2025, 2025.03.20.

[51] "Oracle autonomous database," https://www.oracle.com/sg/autonomous-database/, 2025, 2025.03.20.