

E-Log: Fine-Grained Elastic Log-Based Anomaly Detection and Diagnosis for Databases

Lingzhe Zhang¹, Tong Jia¹, Xinyu Tan, Xiangdong Huang, Mengxi Jia¹, Hongyi Liu¹, Zhonghai Wu¹, and Ying Li², Member, IEEE

Abstract—Database Management Systems (DBMS) form the backbone of modern large-scale software systems, where reliable anomaly detection and diagnosis are essential for ensuring system availability. However, existing log-based methods often impose significant performance overhead by collecting large volumes of logs, which is impractical for DBMS requiring high read/write throughput. This paper addresses a critical yet underexplored challenge: how to balance logging granularity with runtime efficiency for effective anomaly management in databases. We present E-Log, a novel fine-grained elastic log-based framework for anomaly detection and diagnosis. E-Log intelligently adjusts the amount and detail of logging based on system state—maintaining lightweight logging during normal operation for efficient anomaly detection, and triggering rich, informative logging only upon anomaly suspicion for accurate diagnosis. This adaptive strategy significantly reduces runtime overhead while preserving diagnostic precision. We implement E-Log on Apache IoTDB and evaluate it using benchmarks including TSBS, TPCx-IoT, and IoT-Bench. Experimental results show that E-Log improves anomaly detection accuracy by 3.15% and diagnosis performance by 9.32% compared to state-of-the-art methods. Moreover, it reduces log storage size by 43.53% and increases average write throughput by 26.22%. These results highlight E-Log’s potential to enable efficient, accurate, and scalable anomaly management in high-performance database systems.

Index Terms—Anomaly detection, anomaly diagnosis, database, elastic log.

I. INTRODUCTION

DATABASE management system (DBMS), such as Alibaba OceanBase [1], RocksDB [2], PingCAP TiDB [3], and Apache IoTDB [4], play a crucial role in modern software systems. They serve as foundational infrastructure to meet the demands of extremely high-volume data storage [5], [6].

Despite their widespread adoption, existing databases encounter recurring anomalies including system failures and

Received 24 May 2024; revised 6 July 2025; accepted 26 July 2025. Date of publication 1 August 2025; date of current version 9 October 2025. This work was supported by the National Key Research and Development Plan under Grant 2021YFF0704202. (*Corresponding authors:* Tong Jia; Ying Li.)

Lingzhe Zhang, Tong Jia, Mengxi Jia, Hongyi Liu, Zhonghai Wu, and Ying Li are with Peking University, Beijing 100871, China (e-mail: zhang.lingzhe@stu.pku.edu.cn; jia.tong@pku.edu.cn; mxjia@pku.edu.cn; hongyiliu@pku.edu.cn; wuzh@pku.edu.cn; li.ying@pku.edu.cn).

Xinyu Tan is with Timecho Ltd., Beijing 100192, China (e-mail: xinyu.tan@timecho.com).

Xiangdong Huang is with Tsinghua University, Beijing 100084, China (e-mail: huangxdong@tsinghua.edu.cn).

This article has supplementary downloadable material available at <https://doi.org/10.1109/TSC.2025.3594870>, provided by the authors.

Digital Object Identifier 10.1109/TSC.2025.3594870

performance degradation. These anomalies often result in substantial financial losses. For instance, Alibaba Cloud contends with Intermittent Slow Queries (iSQs) [7], causing billions of dollars in losses annually. Amazon reports that every 0.1 s of loading delay due to database anomalies results in an additional 1% financial loss. Consequently, the detection, diagnosis, and mitigation of anomalies in databases are of paramount importance.

As system logs record important events and states of running processes, they represent a valuable resource for anomaly detection and diagnosis. Consequently, log-based anomaly detection and diagnosis have emerged as effective methods for ensuring software availability, attracting substantial research attention. Anomaly detection models [8], [9], [10], [11], [12], [13], [14] involve determining whether anomalies have occurred and can be broadly categorized into supervised models and unsupervised models. Supervised models [8], [9], such as LogRobust [8], need labeled examples of both normal and abnormal cases to make predictions. In contrast, unsupervised models [10], [11], [12] detect deviations relying solely on standard data. Anomaly diagnosis models [15], [16], [17], [18], [19] categorize the specific types of anomalies and can be further classified as graph-based or deep learning-based. Graph-based models [15], [16], [17], like LogKG [16], utilize knowledge graph-based methods to classify failure types. Deep learning-based models [18], [19], such as Cloud19 [19], leverage LSTM-based methods to model the sequence of events.

Although the superiority of these log-based methods has been illustrated in specific scenarios, we find that **they never consider the negative impact of log collection on system performance**. Even recent studies such as FAMOS [20], EagerLog [21] and TVDiag [22] rely on collecting increasingly large volumes of data for anomaly diagnosis. Collecting more logs can give more insights into software behavior, and this is acceptable for applications. But for databases, which need high speed, this extra cost is too high. DBMS, serving as the contemporary foundation for data storage in software, demand high read and write throughput. In benchmark scenarios like TPC [23], even a 10% decrease in throughput can lead to a decline in the ranking of the corresponding database, resulting in a significant loss of user orders.

In our view, **solving the anomaly detection and diagnosis problem in DBMS is a balancing act between the quality of log data and system overhead**. In most cases, more log data can reflect more anomaly characteristics. For instance,

incorporating debug-level application logs into state-of-the-art models often facilitates easy diagnosis of anomalies due to the explicit content within debug logs. However, regrettably, the associated overhead of collecting debug-level logs is deemed unacceptable. In Apache IoTDB [4], enabling debug-level logs can lead to approximately a 300% decrease in write throughput compared to enabling only error-level logs.

To justify these assumptions, we begin with an empirical study focusing on three aspects: (1) the runtime overhead of log collection, (2) the impact of log quantity on anomaly detection and diagnosis models, and (3) the utility and human effort involved in log reduction within industrial scenarios. The findings confirm that log collection results in significant performance degradation, underscoring the importance of minimizing log quantity to reduce the overhead they generate during actual software runtime. Furthermore, we also find that in the log-based anomaly detection and diagnosis problem, we only need a small portion of the logs to achieve high-effectiveness detection, but a large volume of logs is required for further accurate diagnosis of the database. In summary, it is essential to strike a balance between log quantity and information content to minimize log collection overhead and maximize anomaly detection and diagnosis effectiveness.

Building upon these insights, we propose **E-Log**, a fine-grained elastic log-based anomaly detection and diagnosis method. It utilizes a small amount of logs for continuous anomaly detection when the database is in a normal state and activates a large volume of logs for fine-grained anomaly diagnosis. In detail, E-Log comprises an anomaly detection and diagnosis classification model based on LSTM and self-attention mechanisms. It includes a component called *Class LPS Adjuster*, built on Log4J, to control logging in the database. Moreover, the framework leverages an *LPS Reducer*, which dynamically adjusts the volume of logs used by the anomaly detection model at runtime. Its goal is to achieve minimal log usage while maintaining high model effectiveness. In addition, a *Cascade LPS Discriminator* is employed to activate relevant full-chain logs when needed.

According to the results of the empirical study, we also find that the number of logs that can be streamlined varies significantly depending on the database workload. However, datasets built with different workloads are not currently available. Therefore, we utilize three different benchmarks (TSBS [24], TPCx-IoT [23], and IoT-Bench [25]) to construct a comprehensive dataset on Apache IoTDB. This dataset consists of 82 million records and spans 20.59 GB. We injected 11 types of anomalies, covering all categories reported in state-of-the-art research on distributed databases [7], [26], [27]. It encompasses various anomalies, from resource-related issues to database software faults.

We evaluate E-Log by implementing it on Apache IoTDB in our dataset. The results demonstrate that E-Log significantly surpasses current state-of-the-art methods and substantially reduces runtime overhead. The model evaluation results indicate that E-Log achieves a remarkable improvement of approximately 3.15% over existing methods in anomaly detection and 9.32% over existing methods in anomaly diagnosis. Furthermore, in

terms of runtime overhead, using E-Log compared to info-level log, the space for storing logs on disk is reduced by 43.53%, and the average write throughput is increased by 26.22%. To summarize, our key contributions are as follows:

- We conduct a comprehensive study on the runtime overhead of log collection and how log quantity impacts anomaly detection and diagnosis models. Our study highlights that it is essential to strike a balance between log quantity and information content to minimize log collection overhead and maximize anomaly detection and diagnosis effectiveness.
- Inspired by the findings, we propose a fine-grained elastic log-based anomaly detection and diagnosis method for DBMS named **E-Log**. This approach utilizes a small amount of logs for continuous anomaly detection when the database is in a normal state and activates a large volume of logs for fine-grained anomaly diagnosis.
- We construct and release the first dataset for anomaly detection and diagnosis on databases with various workloads, totaling 20.59 GB in size and comprising a total of 82 million records.
- The effectiveness of E-Log is confirmed by implementing it on Apache IoTDB in our dataset. Experiments show that E-Log outperforms state-of-art methods in anomaly detection by 3.15% and in anomaly diagnosis by 9.32%. It also reduces the logs' disk storage space by 43.53% and improves average write throughput by 26.22% compared to info-level logs.

This paper is structured as follows: Section II provides the necessary background, including the formal definition of the log-based anomaly detection and diagnosis problem. Section III presents a comprehensive study on the runtime overhead of log collection and the impact of log volume on anomaly detection and diagnosis models. Section IV introduces E-Log, our proposed fine-grained elastic log-based anomaly detection and diagnosis method for DBMSs. Section V describes the construction and release of a new dataset for anomaly detection and diagnosis in databases under various workloads. Section VI evaluates the effectiveness of E-Log using this dataset. Section VII reviews related work. Finally, Section VIII concludes the paper.

II. BACKGROUND & MOTIVATION

In this section, we present the background and formal definitions of log-based anomaly detection and diagnosis, along with the motivation behind our work and a comparison with related systems.

A. Log-Based Anomaly Detection & Diagnosis

Log-based anomaly detection and diagnosis aim to ensure system reliability by analyzing runtime logs to automatically identify and explain abnormal behaviors. This process is typically divided into two stages: anomaly detection and anomaly diagnosis.

Anomaly Detection: This task identifies irregularities in system behavior from logs and is commonly framed as a binary classification problem [8], [9], [10], [11], [12]. A log sequence

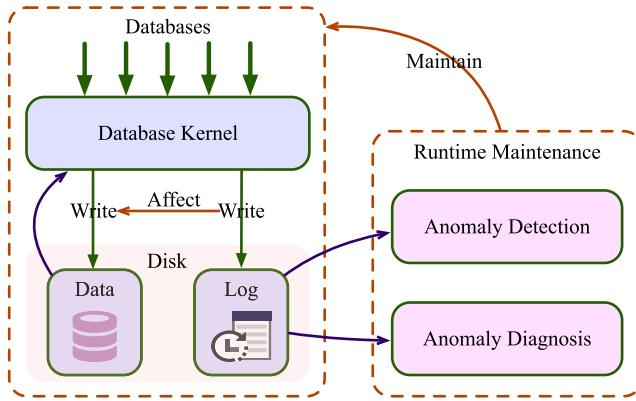


Fig. 1. Motivation and research focus.

is a time-ordered list of log entries, while an event abstracts a source-code print statement, typically generalized by invariant tokens and wildcards. The detection pipeline generally includes three steps: log parsing (e.g., using Drain3 [28], [29]), log grouping (e.g., time-window-based [30]), and model training. Methods include supervised approaches [8], [9], such as LogRobust and PLELog, which leverage labeled data or estimated labels to build predictive models.

Anomaly Diagnosis: Once an anomaly is detected, diagnosis determines its root cause or specific type (e.g., CPU/IO saturation), which is usually framed as a multi-class classification problem [15], [16], [17], [18], [19]. Existing approaches include graph-based models such as LogKG [16] and LogCluster [17], which construct log knowledge graphs or cluster logs for type inference, and deep learning models like Cloud19 [19], which use vector representations for anomaly classification.

Together, these techniques support automated runtime maintenance by enabling early detection and actionable insights for failure mitigation.

B. Motivation

The motivation and research focus are illustrated in Fig. 1. During the data write process in a database system, the kernel performs two types of write operations to disk: one is writing data, which will be later used for query processing; the other is writing logs, which themselves do not carry query value but are essential for runtime maintenance. This runtime maintenance primarily includes tasks such as anomaly detection and diagnosis, which are the key focus of this paper.

By automatically analyzing logs, the database can maintain stable operation. However, log writing competes for system resources such as CPU, memory, and I/O bandwidth, thereby impacting the performance of normal data writes. Therefore, we aim to achieve the best possible runtime maintenance with the minimal amount of logging.

This is the key distinction between our work and existing SOTA approaches: most SOTA log-based anomaly detection and diagnosis systems focus solely on achieving high maintenance accuracy by utilizing large volumes of logs, without considering the resource overhead introduced by logging. To the best of our knowledge, this work is the first to address both log effectiveness and logging overhead simultaneously.

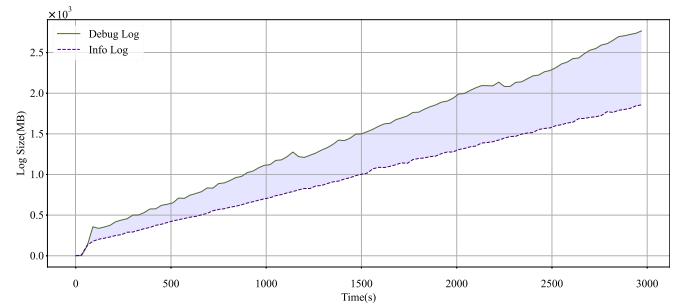


Fig. 2. Storage size.

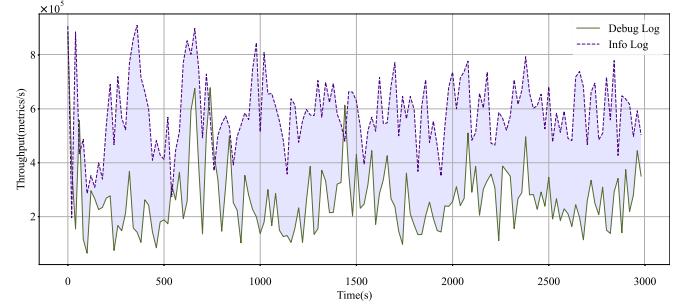


Fig. 3. Write throughput.

III. EMPIRICAL STUDY

In this section, we conduct a comprehensive study on the runtime overhead of log collection and how log quantity has impact on anomaly detection and diagnosis models. Additionally, we invite several professional developers to perform empirical experiments on log reduction with Apache IoTDB in real industrial scenarios.

A. Runtime Overhead of Log Collection

Runtime overhead is considered a major cost of logging [31], as generating log strings involves string concatenations and possible method invocations, and writing logs into log files involves expensive IO. Moreover, storing these logs also requires a significant amount of additional disk space.

To investigate the runtime overhead of log collection on real systems, we conducted corresponding experiments using TSBS [24] workloads on Apache IoTDB v1.2.2. In the experiments, we separately enabled **Debug** level log and **Info** level log, and recorded the corresponding real-time write throughput and the disk space occupied by the logs.

As shown in Fig. 2, when debug-level log is enabled, the log production per second increases by 74.07% compared to only enabling info-level log. Therefore, in the case of enabling debug-level log, the system's write throughput is significantly reduced, as shown in Fig. 3. Under debug-level log, the system's write throughput is 49.74% of that under info-level log.

Summary: Log collection incurs noticeable resource consumption (e.g., CPU, memory, IO) and performance degradation. In databases, it is essential to minimize the quantity of logs to reduce the overhead they generate.

TABLE I
LOG REDUCTION IN ANOMALY DETECTION MODELS

Model		HDFS	BGL	Thunderbird
LR	LPS	55.17%	97.53%	99.93%
	lines	84.37%	98.94%	96.45%
SVM	LPS	65.52%	95.78%	99.93%
	lines	84.58%	98.90%	96.45%
Decision Tree	LPS	75.86%	92.44%	99.57%
	lines	72.09%	91.96%	94.04%
Isolation Forest	LPS	68.97%	73.55%	93.60%
	lines	76.20%	85.44%	85.38%
LogRobust	LPS	58.62%	94.48%	99.93%
	lines	54.60%	50.81%	96.45%
PLELog	LPS	65.52%	94.62%	99.93%
	lines	68.86%	97.00%	96.45%

B. Log Quantity Impact on Anomaly Models

While minimizing logs can reduce their overhead on database performance, not all logs can be reduced. Logs serve as a crucial entry point for observing the system, especially in anomaly detection and diagnosis. If logs are too scarce, it may become challenging to achieve high-effectiveness anomaly detection and diagnosis. Therefore, in this research question, for anomaly detection, we conduct experiments based on mostly utilized datasets (HDFS, BGL, Thunderbird in LogHub [32]). For anomaly diagnosis, we utilize CMCC [16] and manually collected anomalous data in Apache IoTDB [4] and Alluxio [33].

We adopt the Las Vegas Wrapper (LVM) method from feature selection [34] and conduct experiments targeting different state-of-the-art models. In practical systems, the logs that we can reduce are Log Print Statements (*LPS*). In the generated logs, each *LPS* corresponds to multiple *lines* of actual output logs. Therefore, we base our feature selection on *LPS* and take precautions to prevent any reduction in model effectiveness caused by feature selection.

As shown in Table I, it can be found that, while maintaining consistent model performance, all anomaly detection models show a significant reduction in log events. The *LPS* reduction ranges from a minimum of 55.17% (with the LR model on the HDFS dataset) to more than 99% (in the Thunderbird dataset). This suggests that a majority of *LPS* in the HDFS, BGL, and Thunderbird datasets are, in fact, superfluous. We also explore the reduction of log *lines*. Even though the LR model in the HDFS dataset only reduced 55.17% of log events, the actual reduction in log *lines* reached 84.37%. This indicates that the eliminated *LPS* are of high frequency, constituting a large proportion of the entire dataset.

To further investigate how anomaly detection accuracy changes when specific logs are reduced, we analyzed the detection accuracy before and after log reduction. As shown in Table II, using the Thunderbird dataset—which achieves the highest log reduction ratio—as an example, we observe that as noisy and redundant logs are trimmed, the performance of all anomaly detection models not only remains stable but even improves. This demonstrates that excessive logging is

TABLE II
ANOMALY DETECTION ACCURACY WITH/OUT LOG REDUCTION IN THUNDERBIRD DATASET

Model		Precision	Recall	F1-Score
LR	w/o	0.971	0.995	0.983
	w	1.000	0.999	0.999
SVM	w/o	0.991	0.996	0.994
	w	1.000	0.998	0.999
Decision Tree	w/o	1.000	0.999	0.999
	w	1.000	0.999	1.000
Isolation Forest	w/o	0.005	0.001	0.002
	w	0.775	0.097	0.173
LogRobust	w/o	0.910	0.559	0.692
	w	1.000	1.000	1.000
PLELog	w/o	0.968	0.996	0.982
	w	1.000	0.996	0.998

TABLE III
LOG REDUCTION IN ANOMALY DIAGNOSIS MODELS

Model		CMCC	Apache IoTDB	Alluxio
LogKG	LPS	3.30%	5.82%	6.29%
	lines	0.32%	7.26%	0.58%
LogCluster	LPS	22.42%	6.35%	33.14%
	lines	1.30%	7.42%	6.08%
Cloud19	LPS	82.94%	62.43%	73.71%
	lines	28.83%	2.84%	3.22%

unnecessary for anomaly detection and may even degrade detection effectiveness.

Conversely, as depicted in Table III, it is evident that for both LogKG and LogCluster, the reduction in either event templates or log *lines* is minimal. The *LPS* reduced is mostly under 20%, and the log *lines* reduced are within 10%. For Cloud19, although there is a possibility to reduce a significant number of *LPS* (even up to 82.94% of events in the CMCC dataset), the actual number of log *lines* reduced is not substantial. This suggests that the anomaly diagnosis problem relies heavily on log information, and we cannot reduce many logs to minimize corresponding overhead.

Summary: In log-based anomaly detection and diagnosis problem, we only need a small portion of the logs to achieve high-effectiveness detection, but a large volume of logs is required for further accurate diagnosis of the system.

C. Log Reduction in Industrial Scenarios

To further investigate the feasibility of log reduction in real industrial scenarios, we invite professional developers from the Apache IoTDB team to manually reduce logs for three real industrial systems (referred to as System A, System B, and System C for privacy reasons).

The manual log reduction followed these steps: (1) Request logs from three real industrial scenarios from the test group and analyze the logs with high print frequency, categorized by

TABLE IV
LOG REDUCTION IN INDUSTRIAL SCENARIOS

Property	System A	System B	System C
Reduced Lines	37.20%	43.68%	77.25%
Reduced Repair Time	47.33%	39.25%	67.66%
Consumption Time	60.5 hours		

module. (2) Conduct review meetings to discuss and modify log levels. (3) After the review, downgrade or remove the relevant logs. (4) Evaluate the overall impact. After thorough review by professional developers, logs that contribute little to anomaly detection, diagnosis, or troubleshooting—such as performance tuning and SQL execution error logs—are removed. Meanwhile, logs that capture potential issues, such as historical behavior records, are retained.

As depicted in Table IV, the professional developers from the Apache IoTDB team consume a total of 60.5 hours on data collection, redundant log investigation, and code changes. For different industrial scenarios, there are logs that can be streamlined, but the amount of reducible logs varies significantly with changes in deployment environment and workload, ranging from 37.20% to 77.25%. Correspondingly, reducing logs also leads to a decrease in troubleshooting and repair time for software system faults during runtime, ranging from 39.25% to 67.66%.

Summary: In real industrial scenarios, there are numerous logs that are irrelevant for anomaly detection, diagnosis, and troubleshooting. However, the extent to which logs can be reduced using the manual reduction method mentioned earlier varies considerably based on the deployment environment and workload. Additionally, a substantial amount of time is required to investigate and streamline these redundant logs.

D. Threats to Validity

During our study, we have identified the following major threats to the validity.

Limited Workload: Our empirical study initially uses only the TSBS [24] workload for Apache IoTDB. Although TSBS is a widely adopted benchmark for time-series databases and demonstrates runtime overhead in log collection, it cannot confirm universal applicability. To address this, we later collect logs under two additional typical workloads—TPCx-IoT [23] and IoT-Bench [25].

Limited Datasets: Our anomaly detection experiments use three public log datasets. While widely adopted in prior work, these datasets may not reflect all characteristics of logs, especially from distributed databases. To mitigate this, we collaborate with Apache IoTDB developers to manually reduce logs from three real industrial systems. We also collect additional datasets under various workloads to better represent real-world scenarios.

Reimplementation: We primarily rely on public implementations of the studied models. For ML methods (LR, SVM,

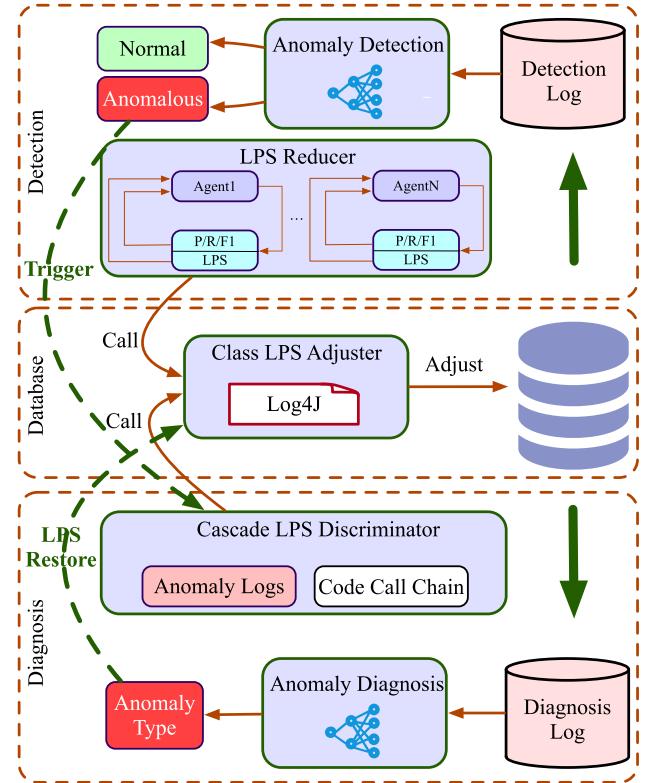


Fig. 4. Architecture of E-Log.

Decision Tree, Isolation Forest), we use Loglizer; for LogRobust, logdeep; and for PLELog, the original paper's basic implementation. LogKG [16], LogCluster [17], and Cloud19 [19] are manually reimplemented using the settings and parameters described in their respective papers. To mitigate this threat, we replicate experiments on the original datasets and achieve results consistent with those reported.

IV. E-LOG: A FINE-GRAINED ELASTIC LOG-BASED ANOMALY DETECTION AND DIAGNOSIS METHOD

Our empirical study demonstrates that minimizing logs to reduce corresponding overhead is feasible for anomaly detection but practically infeasible for the anomaly diagnosis problem. Additionally, in real industrial software applications, manually reducing logs requires a significant amount of work hours. Therefore, to address this issue as effectively as possible, we propose E-Log, which is a automatic fine-grained elastic log-based anomaly detection and diagnosis method. Its core idea is to use a small amount of logs for continuous anomaly detection when the database is in a normal state. Once the system enters an anomalous state, it activates a large volume of logs and conducts fine-grained diagnosis.

A. Architecture

As shown in Fig. 4, E-Log consists of three main components: *Detection*, *Diagnosis*, and *Database*. The *Database* component represents the *Class LPS Adjuster* exposed by the database system, implemented based on Log4J which provides a

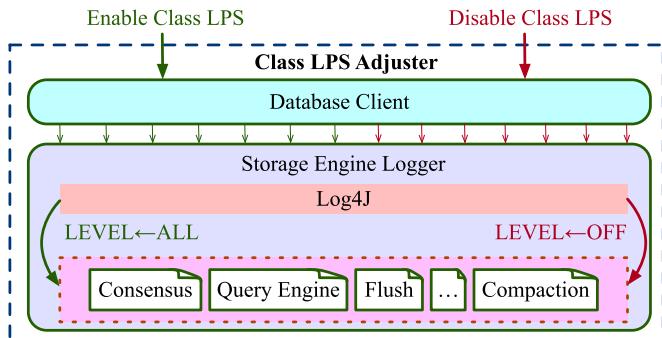


Fig. 5. Workflow of class LPS adjuster.

class-level LPS adjustment interface. The *Detection* and *Diagnosis* components dynamically adjust the runtime logs output by the Database using the *Class LPS Adjuster* and perform log-based anomaly detection and diagnosis. These three components are interconnected and interact with each other. As illustrated in the figure, the green dashed arrow from the Detection module triggers the Diagnosis module, while the green dashed arrow from the Diagnosis module triggers the LPS Restore function in the Database module. Additionally, the red solid lines represent function calls between submodules: both the Detection and Diagnosis modules need to call the *Class LPS Adjuster* in the Database module.

The *Detection* component comprises an anomaly detection binary classification model based on LSTM (Long Short-Term Memory) and self-attention mechanisms, along with a reinforcement learning-based LPS Reducer. The *LPS Reducer* dynamically adjusts the logs required by the anomaly detection model during runtime, aiming to obtain the minimum logs with the highest model effectiveness. When the anomaly detection model determines that the system has entered an anomalous state, it records the logs within the current time window (*Anomaly Logs*) and transitions to the *Diagnosis* component.

The *Diagnosis* component receives the *Anomaly Logs* and combines them with Code Call Chain information. It then activates the *Class LPS Adjuster* to enable relevant full-chain LPS, referred to as the *Cascade LPS Discriminator*, and collects the corresponding *Diagnosis Logs* generated by the database during this time window. It includes a multi-class anomaly diagnosis model based on LSTM and self-attention mechanisms to identify the current anomaly type. After the anomaly diagnosis, the system restores the LPS volume to its original state and continues with anomaly detection.

B. Class LPS Adjuster

The *Class LPS Adjuster* is embedded in the specific database system. Its workflow is depicted in Fig. 5. It is built based on Log4J and provides an external interface through the *Database Client* to enable or disable LPS for a specific class. The command is then passed to the internal *Storage Engine*, which adjusts the corresponding log level to ALL or OFF using Log4J.

In other words, the *Class LPS Adjuster* exposes a CLI interface from within the database, allowing external programs to

dynamically control the log level of a specific class via command-line instructions. This mechanism enables toggling logging behavior between full verbosity (ALL) and complete silence (OFF) for runtime maintenance purposes.

For example, during anomaly diagnosis, an external component can enable detailed logging for a specific class by executing the following command: `ENABLE_ALL_LOG org.apache.iotdb.tsfile.FileWriter`. Once the investigation is complete, it can disable logging for the same class to reduce resource overhead: `DISABLE_ALL_LOG org.apache.iotdb.tsfile.FileWriter`. This fine-grained, on-demand logging control allows E-Log to maintain a balance between log effectiveness and system performance, aligning with its overall goal of cost-efficient runtime maintenance.

C. Anomaly Detection & Diagnosis Models

E-Log follows the same process as state-of-the-art work for log detection and diagnosis [8], [9], [10], [11], [12], [15], [16], [17], [18], [19], as shown in Fig. 6: log parsing, log grouping, anomaly detection, and diagnosis.

Log parsing initially extracts log events from system logs and transforms the log sequence into an event sequence, where each type of log event can correspond to one or more LPS. Specifically, E-Log utilizes the state-of-the-art Drain [28] method for log parsing, which has been proven highly effective and efficient in existing studies [35]. After log parsing, each event can be represented as a distinct event e_i , and the entire collection of unique events can be denoted as $\omega = \{e_1, e_2, \dots, e_n\}$.

Log grouping organizes logs into sequence groups using fixed time windows [30]. After log grouping, the entirety of raw log messages $S_t = (s_1, s_2, \dots, s_{M_t})$ in a specific time window t can be represented as $E_t = (e_{(s_1)}, e_{(s_2)}, \dots, e_{(s_{M_t})})$. Here, M_t represents the length of each grouped log sequence in the time window t .

The E_t can also be referred to as sequential embedding, forming naturally sequential patterns in log data. On this basis, we can obtain quantitative patterns. To capture this aspect, we further perform the following transformation: For each group of log sequence S_t , we compute the count vector as $C_t = (c_t(e_1), c_t(e_2), \dots, c_t(e_n))$, where $c_t(e_k)$ represents the frequency of $e_k \in \omega$ in the sequential embedding E_t .

Beyond sequential and quantitative patterns, an event sequence can also encapsulate semantic patterns. To extract the semantic information from log events, E-Log treats each log event as a sentence in natural language. In detail, E-Log utilizes the Word2Vec algorithm to convert each type of event $e_k \in \omega$ as a semantic vector $v_{(e_k)}$. Then we can compute the semantic embedding of E_t as $V_t = (v_{(e_{(s_1)})}, v_{(e_{(s_2)})}, \dots, v_{(e_{(s_{M_t})})})$.

For each pattern previously analyzed, we employ an LSTM with self-attention to amplify key information. Each embedding output— E , C , and V —serves as the input for an LSTM. The LSTM's hidden state is updated at each time step t as $h_t = LSTM(h_{t-1}, [E_t, C_t, V_t])$. Here, h_t represents the hidden state of the LSTM at time t , and $[E_t, C_t, V_t]$ denotes the concatenation of the sequential embeddings E_t , quantitative

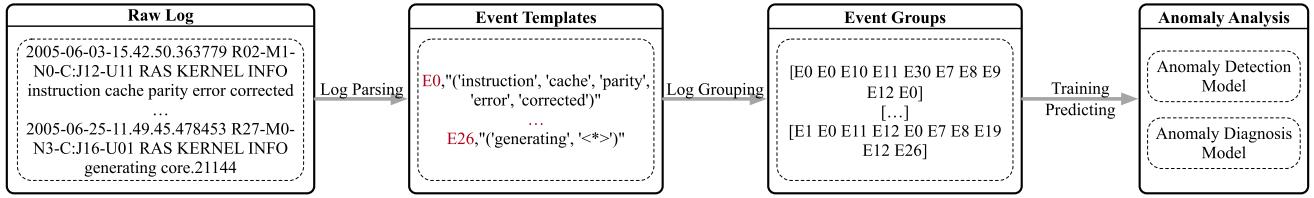


Fig. 6. Workflow of anomaly detection & diagnosis model.

embeddings C_t , and semantic embeddings V_t at time t . We then apply a fully connected layer to produce the final output. The number of output nodes in this layer depends on whether it's an anomaly detection task (2 output nodes) or an anomaly diagnosis task (N output nodes).

We employ an LSTM with self-attention in our framework, motivated by both theoretical considerations and empirical success in related work. LSTMs are well-suited for modeling the sequential nature of log data, as they capture long-term dependencies and temporal correlations—key for distinguishing normal patterns from anomalous behavior. Meanwhile, self-attention complements recurrent models by enabling the network to dynamically focus on informative time steps and features, rather than treating all parts of the sequence equally. This mitigates limitations of purely sequential models, particularly in long sequences where anomaly indicators may be sparse or subtle. By integrating LSTM with self-attention, our model jointly captures temporal dependencies and highlights salient patterns across sequential, quantitative, and semantic embeddings, resulting in improved anomaly detection and diagnosis. This design also aligns with recent best practices, where hybrid architectures combining sequence modeling and attention have proven highly effective.

D. LPS Reducer

LPS Reducer employs multi-agent reinforcement learning to elastically adjust the logs required by the anomaly detection model during runtime, aiming to obtain the minimum logs with the highest model effectiveness.

In detail, E-Log creates an agent for each code package, and the state representation of each agent at time step t , denoted as S_t , is defined in (1). In this representation, $L_{t,i}$ refers to the currently active logging point set (LPS) in the i -th code package at time t , while $R_{t,i}$ refers to the currently inactive LPS in the i -th package. The term $a_{t,i}$ denotes the candidate LPS in the i -th package that is about to be disabled (transitioned to inactive), and $b_{t,i}$ denotes the candidate LPS that is about to be enabled (transitioned to active). P_{t-1} represents the anomaly detection performance metrics (including precision, recall, and F1-score) observed at the previous time step $t - 1$.

$$S_t = \{(L_{t,1}, R_{t,1}, a_{t,1}, b_{t,1}), \dots, (L_{t,n}, R_{t,n}, a_{t,n}, b_{t,n}), P_{t-1}\} \quad (1)$$

Based on S_t , the action space at time t , denoted as A_t , is defined as $A_t \in \text{DISABLE } a_{t,i}, \text{ENABLE } b_{t,i}$, where each action corresponds to either disabling a selected LPS or enabling a new one in the i -th code package. The reward function R_t

is defined in (2), where p_t , r_t , and f_t denote the precision, recall, and F1-score of anomaly detection at time t , respectively. In practical settings, the parameter α is typically set much larger than β to prioritize maintaining detection accuracy and minimizing information loss.

$$R_t = \alpha \cdot \left(\frac{p_t - p_{t-1}}{p_{t-1}} + \frac{r_t - r_{t-1}}{r_{t-1}} + \frac{f_t - f_{t-1}}{f_{t-1}} \right) + \beta \cdot \sum_i (|L_{t-1,i}| - |L_{t,i}|) \quad (2)$$

However, if we consider each log entry to have equal weight in reinforcement learning, the state space becomes excessively large. For example, if there are 5 LPS in the system, the state space includes configurations where all 5 LPS are active, any combination of 4 LPS are active, any combination of 3 LPS are active, any combination of 2 LPS are active, any single LPS is active, and no LPS are active.

Therefore, in practice, we first compute the mutual information between each LPS and the anomaly labels. LPSs with higher mutual information are prioritized for activation when inactive, while those with lower mutual information are candidates for deactivation when already active.

Mutual information is a fundamental concept from information theory that quantifies how much information one random variable provides about another. Specifically, the mutual information $MI(s; l)$ between the activation status of an LPS s and an anomaly label l measures the reduction in uncertainty of l given knowledge of s , and vice versa. A mutual information of zero indicates that s and l are statistically independent.

The overall importance of an LPS s is calculated as the average of its mutual information with all anomaly labels, as shown in (3b). The definition of $MI(s; l)$ is given in (3a), where $s \in S$ denotes a specific LPS, $l \in L$ an anomaly label, $p(s, l)$ the joint probability distribution of S and L , and $p(s)$ and $p(l)$ their respective marginal distributions.

$$MI(s; l) = \sum_{s \in S, l \in L} p(s, l) \log \left(\frac{p(s, l)}{p(s)p(l)} \right) \quad (3a)$$

$$MI(s; L) = \frac{1}{|L|} \sum_{l \in L} MI(s; l) \quad (3b)$$

Regarding complexity, assuming there are $|S|$ LPSs and $|L|$ anomaly labels, the time complexity of computing all mutual information scores is $O(|S||L|)$. Since this computation can be performed periodically and the results cached, the runtime overhead remains low and is acceptable for online scenarios.

Algorithm 1: Cascade LPS Discriminator Algorithm.

```

Input : All LPS within the current anomaly time
          window  $L$ ; Relevant code  $C$ 
Output: All LPS to be enabled  $LF$ 

1  $LF \leftarrow \emptyset$ 
2 Function  $TraceUp(bb)$ :
3   if  $bb$  has parent then
4      $LF \leftarrow LF \cup bb.LPS();$ 
5      $TraceUp(bb.parent());$ 
6 Function  $TraceDown(bb)$ :
7    $LF \leftarrow LF \cup bb.LPS();$ 
8   for child  $\in bb.childs()$  do
9      $TraceDown(child);$ 
10 for  $l \in L$  do
11    $bb \leftarrow FindLPSInCode(C, l);$ 
12    $TraceUp(bb);$ 
13    $TraceDown(bb);$ 
14 return  $LF$ 
```

By the above method, E-Log can elastically adjust the required number of LPS during runtime to obtain the minimum logs with the highest model effectiveness. Furthermore, using this reinforcement learning approach to continuously try disabling or enabling LPS can also ensure that the associated information needed for potential future anomalies is not overlooked.

E. Cascade LPS Discriminator

Every time the anomaly detection model detects that the current database has entered an anomalous state, it triggers further anomaly diagnosis. When starting the anomaly diagnosis, it first uses all logs within the current time window and combines them with the relevant call chain in the code to enable the relevant full-chain LPS.

The operating principle of the *Cascade LPS Discriminator* is illustrated in Algorithm 1. Assuming all LPS within the current anomaly time window as L and all relevant code as C . For each $l \in L$, it first locates its corresponding position in the code, then recursively adds all parent code block LPS ($TraceUp$) and all LPS in child code blocks ($TraceDown$).

For example, suppose an anomaly is detected involving an LPS located in the method `flushChunkData()` of the class `FileWriter`. The discriminator will automatically trace upward to include LPS from the caller method `flushChunkGroup()`, and further up to `endFile()`. It will also trace downward to enable all LPS in nested methods such as `writePage()` or `compressPageData()`.

V. DATASET

As aforementioned, variations in workload may impact the results of log reduction, anomaly detection, and diagnosis. Therefore, we construct a multi-workload dataset. The dataset is

TABLE V
ROOT CAUSE OF THE FAILURES IN APACHE IOTDB

Failure type	Root cause
CPU Saturation Memory Saturation Network Bandwidth Limited	Insufficient system resource
Export operations Import operations	DBA operations
Too much background tasks Too frequency disk flushing	Misconfiguration

based on Apache IoTDB [4] v1.2.2 in a Docker environment. For the running of Apache IoTDB, we employ 4 Docker containers, with 3 serving as data nodes and 1 as a configuration node. These containers are equipped with configurations, including 8 Intel(R) Xeon(R) Platinum 8260 CPUs at 2.40 GHz, 16 GB of DIMM RAM, 1.1 TB NVMe disk, and running on OpenJDK 11.

Based on Apache IoTDB, we apply three typical benchmarks: TSBS [24], TPCx-IoT [23], and IoT-Bench [25]. Among them, TSBS is a benchmark widely used for testing time-series databases, featured in many rankings, such as benchANT [36]. TPCx-IoT is an industry-standard benchmark in the database domain, providing verifiable performance, cost-effectiveness, and availability metrics for commercial systems. IoT-Bench is a tool provided by Apache IoTDB itself for benchmarking against other databases and time series solutions.

A. Database Anomalies

We analyze the definitions and classifications of anomalies in both system and database research [7], [26], [27], [37], [38], [39], [40], and select three representative root causes along with a total of seven corresponding failure types. These root causes are chosen because they are widely adopted in many SOTA studies and collectively cover more failure types than most existing works. While real-world scenarios undoubtedly involve more diverse types of failures, the exact number is difficult to quantify. Nevertheless, the selected failure types encompass a broad range of typical cases, addressing anomalies at both the system and database levels.

As shown in Table V, for insufficient system resources, we select the three most common types (CPU, memory, network). For DBA operations, we inject anomalies involving excessive export and import operations. For system misconfiguration, we simulate two prevalent misconfigurations in various databases: too many background tasks and too frequent disk flushing. Among these, for the ‘too many background tasks’ anomaly, we used the most common compaction task in the LSM database system as an example [5], [41], [42].

B. Anomaly Injection

During the above benchmark runs, we use Chaos Mesh [43] for anomaly injection. It is an open-source cloud-native chaos engineering platform that provides a wide range of fault injection types, allowing users to simulate various types of anomalies that

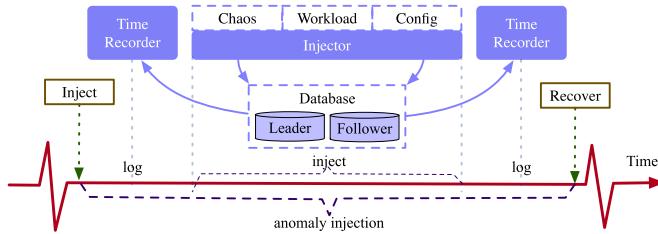


Fig. 7. Architecture of anomaly injection.

TABLE VI
DATASET DETAILS

No.	Benchmark	# Log Entries	Data Size	Labeled
1	TSBS	67,990,578	17.01 GB	✓
2	TPCx-IoT	984,567	0.28 GB	✓
3	IoT-Bench	13,088,871	3.30 GB	✓

may occur in the real world. However, it is only suitable for simulating anomalies in docker environments. For anomalies related to workload and internal bugs, we employ dynamic workload adjustment and database configuration modification approaches for injection.

The main architecture of anomaly injection is demonstrated in Fig. 7. During the experiment, the database initially operates normally for a set period. Subsequently, various types of anomalies are injected into the database. Before each injection, the current timestamp is recorded. The anomalies are then introduced and sustained for a specified duration. After the injection, the current timestamp is recorded again. The database is then allowed to resume normal operation for a certain period before the next anomaly is introduced. This process is repeated, with each cycle injecting a different type of anomaly, thus maintaining the database cluster in a cyclic state of normal - abnormal - normal - . . . - abnormal - normal.

In the aforementioned architecture, the key component is the anomaly injector, which is responsible for injecting anomalies into the database. In the implementation, insufficient system resources anomalies utilize Chaos Mesh, DBA operations anomalies are accomplished by adjusting the read / write workload, and misconfiguration anomalies are implemented through hot modifications to the database configuration.

C. Dataset Details

Based on the aforementioned process, the comprehensive statistics of the dataset are concluded in Table VI. Overall, the dataset has a size of 20.59 GB and contains a total of 82 million records. The datasets are available in the E-Log-Dataset repository, <https://github.com/AIOPS-LogDB/E-Log-Dataset>.

VI. EXPERIMENT AND EVALUATION

In this section, we present our implementation of E-Log, describe the experimental environment and settings, and evaluate the model results and the runtime overhead.

TABLE VII
EVALUATION RESULTS ON ANOMALY DETECTION

Model		TSBS	TPCx-IoT	IoT-Bench
PLELog	P	63.16%	73.68%	59.71%
	R	48.98%	95.46%	77.12%
	F1	55.17%	83.17%	67.31%
LogRobust	P	100.00%	97.83%	100.00%
	R	71.01%	91.84%	67.12%
	F1	83.05%	94.74%	80.33%
LogAnomaly	P	89.47%	100.00%	92.86%
	R	98.08%	91.84%	100.00%
	F1	93.58%	95.75%	96.29%
E-Log(Ours)	P	98.00%	100.00%	98.25%
	R	98.00%	96.00%	100.00%
	F1	98.00%	97.96%	99.12%

A. Implementation

1) *Experiment Platform:* Our anomaly detection and diagnosis experiments are conducted based on the aforementioned dataset. For the runtime collection overhead analysis, we also choose to use Apache IoTDB [4] as our experimental platform for the following reasons: (i) it is an open-source platform, (ii) it is a typical distributed database, and (iii) it has an active community. The configuration for the runtime collection overhead experiments matches the configuration used for constructing the dataset.

Our codebase has been integrated into Apache IoTDB v1.2.2.¹ IoTDB uses Apache Log4j as the logging framework, which aligns with our design. The Class LPS Adjuster is implemented in the core code of the database, while the LPS Reducer, Cascade LPS Discriminator, and Anomaly Detection and Diagnosis Models are implemented separately in an external AI Node.

2) *Configuration:* In our experiments, we utilize the previously mentioned benchmark to evaluate the effectiveness of E-Log in comparison with state-of-the-art methods. For E-Log, unless specified otherwise, we set the time window size to 5 seconds, set the hidden size to 64, set the α of LPS Reducer to 100, and the β of LPS Reducer to 1. Moreover, for state-of-the-art models, we utilize info-level log for detection and diagnosis, which is also the most common practice in current industrial scenarios.

B. Model Evaluation

We first conduct anomaly detection compared with state-of-the-art models: PLELog [9], LogRobust [8], and LogAnomaly [44]. As shown in Table VII, regardless of the benchmark, E-Log achieves the best performance, surpassing the second-best LogAnomaly model by approximately 5%. While PLELog and LogRobust models show good performance under the TPCx-IoT workload, they do not perform as well under TSBS and IoT-Bench workloads.

The reason E-Log achieves better results with reduced log volume for detection can be attributed to two main factors. Firstly, in our experiment, we observe that reducing certain LPS not only

¹<https://anonymous.4open.science/r/E-Log>

TABLE VIII
EVALUATION RESULTS ON ANOMALY DIAGNOSIS

Model		TSBS	TPCx-IoT	IoT-Bench
LogKG	P	50.75%	50.19%	50.71%
	R	53.57%	52.98%	53.01%
	F1	47.22%	48.25%	50.61%
LogCluster	P	21.79%	43.75%	25.93%
	R	21.43%	50.00%	33.33%
	F1	19.36%	46.67%	29.17%
Cloud19	P	90.67%	78.53%	89.61%
	R	64.09%	71.57%	77.35%
	F1	73.53%	68.93%	82.22%
E-Log(<i>Ours</i>)	P	81.05%	86.84%	88.07%
	R	88.70%	85.73%	81.28%
	F1	83.85%	86.14%	82.66%

TABLE IX
ABLATION STUDY OF E-LOG

Approach		TSBS	TPCx-IoT	IoT-Bench
Anomaly Detection	w/o	91.25%	93.27%	90.19%
	w	98.00%	97.96%	99.12%
Anomaly Diagnosis	w/o	75.64%	81.33%	78.79%
	w	83.85%	86.14%	82.66%

does not negatively impact the model's effectiveness but, in some cases, their presence can mislead the model. This implies that removing them can actually enhance the model's performance. In our experiments with Apache IoTDB, we find that such LPS constitute approximately 35%. Secondly, E-Log simultaneously utilizes the sequential, quantitative, and semantic information of logs, and employs self-attention mechanisms to strengthen log features.

We then further conduct anomaly diagnosis compared with state-of-the-art models: LogKG [16], LogCluster [17], and Cloud19 [19]. To evaluate the effectiveness of each model, we compare them using macro precision, macro recall, and macro f1-score. As depicted in Table VIII, E-Log also achieves the best diagnosis performance, surpassing the second-best Cloud19 by 10.32%, 17.21%, and 0.44% in TSBS, TPCx-IoT, and IoT-Bench, respectively. For graph-based models like LogKG and LogCluster, the difference is even more significant. This is because E-Log, when detecting anomalies in the system, utilizes the Cascade LPS Discriminator to automatically identify and open more relevant logs for diagnosis, providing more comprehensive log information.

To verify whether the promising results of E-Log stem from the dynamic adjustment of log volume mentioned earlier, we conduct an ablation study by disabling the LPS Reducer during the anomaly detection phase and the Cascade LPS Discriminator during the anomaly diagnosis phase—thus running experiments under a fixed logging level. The f1-score results, shown in Table IX, indicate that removing these two components leads to a significant performance drop: anomaly detection accuracy decreases by an average of 6.79%, and anomaly diagnosis accuracy drops by an average of 5.63%. This demonstrates the effectiveness of each component in E-Log.

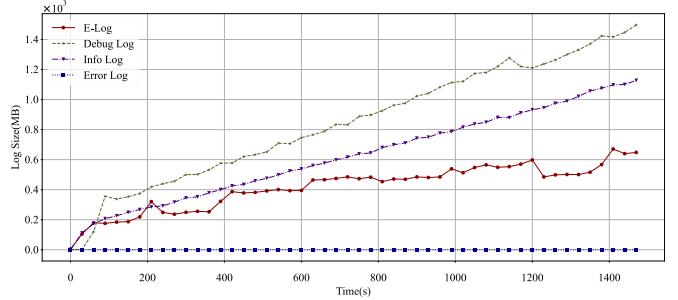


Fig. 8. Storage size.

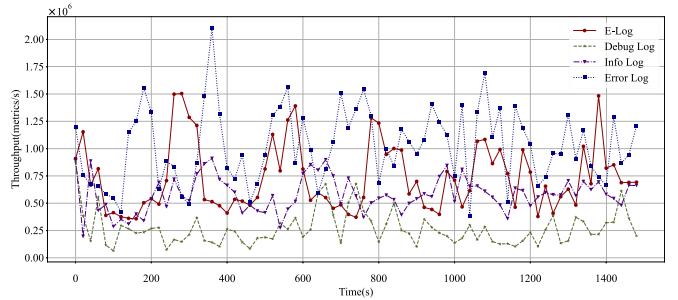


Fig. 9. Write throughput.

In summary, whether in anomaly detection or diagnosis problems, E-Log consistently achieves high effectiveness, assisting system operators in maintaining distributed database systems.

C. Runtime Overhead

Compared to static logger strategies, another advantage of E-Log is that, although it may use more logs for anomaly diagnosis, the number of logs it utilizes during normal system operation is minimal. Therefore, we further record the changes in storage size and write throughput during normal system operation and when anomalies occur under TSBS benchmark.

As illustrated in Fig. 8, we record the changes in storage space occupied by logs during runtime and compare them with scenarios where debug-level logs, info-level logs, and error-level logs are enabled. From the figure, it can be observed that enabling only error-level logs basically generates no logs, and therefore, it is not suitable for anomaly detection and diagnosis. Enabling debug-level logs and info-level logs generates a significant amount of logs, reaching sizes of 1496.67 MB and 1127.43 MB, respectively, in 30 minutes. With E-Log, the overall change is dynamic—when anomalies occur, logs are rapidly generated, and when there are no anomalies, log generation is minimal, ultimately accounting for only 57.47% of info-level logs. Moreover, in real industrial scenarios, log generation would be even less because anomaly occurrences are infrequent. Additionally, in the figure, the storage size of logs sometimes decreases. This is because Log4j automatically compresses previous logs after reaching a certain amount and starts recording new logs in a new file. This compression has no impact on the effectiveness of E-Log.

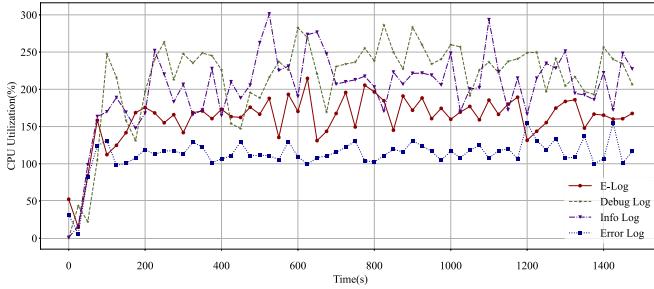


Fig. 10. CPU usage analysis.

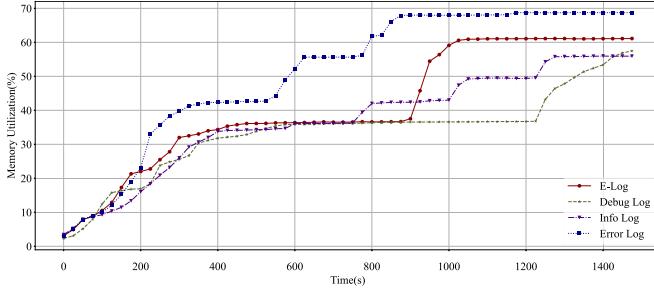


Fig. 11. Memory usage analysis.

We also monitor the real-time changes in write throughput corresponding to Fig. 8, as depicted in Fig. 9. The write throughput is highest when using error-level logs, followed by E-Log, with an average write throughput of 71.79% compared to using error-level logs. Although using E-Log still impacts the database performance to some extent, it provides sufficient system information. Moreover, compared to using info-level logs and debug-level logs, using E-Log results in 126.22% and 274.14% of the respective write throughputs. In other words, using E-Log, both in terms of the provided log information (reflecting anomaly detection and diagnosis effectiveness) and system performance, surpasses the most commonly used info-level logs in industrial scenarios.

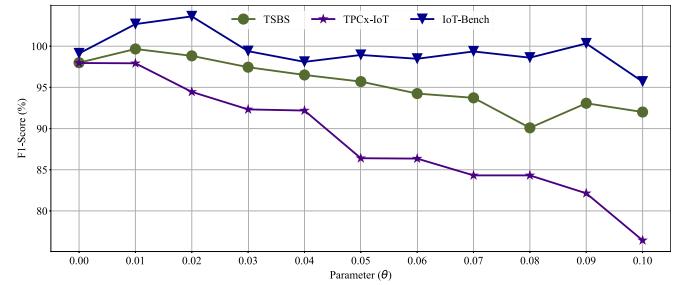
To further investigate how E-Log achieves this enhanced write throughput, we analyzed the hardware resource utilization during write operations.

As shown in Fig. 10, we examined CPU utilization under different logging levels during write operations. It can be observed that E-Log reduces average CPU usage by 25.60% compared to info-level logging and by 33.62% compared to debug-level logging, which further corroborates the write throughput results.

A similar pattern is observed in the memory usage analysis, as shown in Fig. 11. E-Log's memory consumption is on average only 10.21% lower than that of error-level logging, while being 3.99% and 7.44% higher than info-level and debug-level logging, respectively. This is because, in this context, memory is not the limiting factor, and higher resource utilization naturally leads to slightly increased memory consumption.

D. Parameters Sensitivity Analysis

As described in the Method section, in our LPS Reducer, α is typically set much larger than β in practice to minimize information loss. To empirically validate this, we conducted

Fig. 12. Anomaly detection results with varying θ .

experiments by fixing $\beta = 1$ and varying α , ultimately adjusting $\theta = \frac{\beta}{\alpha+\beta}$ from 0.00 to 0.10.

As shown in Fig. 12, as θ increases, the F1-score first improves and then consistently declines across all datasets. For example, on the IoT-Bench dataset, increasing θ from 0.00 to 0.01 yields a 3.44% improvement in F1-score, and from 0.01 to 0.02, a further gain of 0.88%. However, when θ increases beyond 0.03, the F1-score starts to drop, with a decline of 4.05% observed at $\theta = 0.03$. These results confirm that to achieve optimal performance, α must be set significantly larger than β ; otherwise, information loss occurs, leading to degraded anomaly detection effectiveness.

VII. RELATED WORK

A. Anomaly Detection and Diagnosis for Database

Several anomaly detection and diagnosis approaches have been developed for databases [7], [26], [27], [45], [46], [47], [48], [49], [50], [51]. These methodologies can be divided into three categories: system-targeting methods, SQL-targeting methods, and components-targeting methods.

System-targeting methods focus on anomaly detection and diagnosis at the overall runtime state of the database. OpenGauss [26], [27], an autonomous database system, implements an LSTM-based auto-encoder with an attention layer for system-level anomaly diagnosis, also leveraging features in metrics data. FluxInfer [45] builds a weighted undirected dependency graph to illustrate the dependency relationships of anomalous monitoring data and employs a weighted PageRank algorithm for diagnosing online database anomalies.

SQL-targeting methods address faults at the SQL and query level for detection and diagnosis. OpenGauss-AI [26] and openGauss-DBMind [27] perform SQL-level diagnosis after detecting slow SQL at the system level. Specifically, openGauss-AI identifies time-consuming operators within SQL statements without executing them, while openGauss-DBMind employs an LSTM-based approach to locate the root cause of slow SQL. iSQUAD [7] adopts the Bayesian Case Model to diagnose anomalies of intermittent slow queries (iSQs) in cloud databases, based on Key Performance Indicators (KPI) data. SQLCheck [47] proposes a holistic toolchain for automatically finding and fixing anti-patterns in databases.

Components-targeting methods focus on diagnosing faults within the database's internal components. Sentinel [49]

constructs a fine-grained model of data system behavior through debug logs to assist DBAs in diagnosing anomalies.

This work falls under the category of system-targeting methods and is the first to utilize logs for anomaly detection and diagnosis in databases. Its key contribution lies in simultaneously addressing the high performance requirements for read and write operations and the need for high availability, achieving a well-balanced trade-off between the two.

B. Log-Based Anomaly Detection and Diagnosis

Log analysis for anomaly detection and diagnosis is a well-established research area [8], [9], [10], [12], [16], [44], [52], [53], [54], [55], [56], [56], [57], [58], [59], [60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73], [74]. These methodologies typically involve extracting templates and key information from logs, followed by constructing models for anomaly detection and classification. There are mainly two types of models in this domain: graph-based and deep-learning models.

Graph-based models leverage log events parsed from log files to create a graph-based representation. They detect conflicts and anomalies by comparing event sequences against this graph. For instance, HiLog [12] performs an empirical study on four anti-patterns that challenge the assumptions underlying anomaly detection and diagnosis models, proposing a human-in-the-loop approach to integrate human expertise into log-based anomaly detection. LogKG [16] introduces a Failure-Oriented Log Representation (FOLR) method to extract failure-related patterns, using the OPTICS clustering method for anomaly diagnosis.

Deep-learning models, conversely, use various neural networks to model sequences of log events. LogRobust [8] applies Term Frequency-Inverse Document Frequency (TF-IDF) and word vectorization to convert log events into semantic vectors, thus improving the accuracy of anomaly detection. UniParser [53] employs a token encoder and a context encoder to learn patterns from log tokens and their adjacent contexts.

To the best of our knowledge, we are the first to introduce elastic log to log-based anomaly detection and diagnosis. Unlike prior work, our work pays attention to the impact of logs on database performance. Besides, our approach improves the effectiveness of anomaly detection and diagnosis from both the quantity and quality perspectives of logs, while related work focuses on building more sophisticated models to accurately capture the anomaly characteristics in the log data.

VIII. CONCLUSION

In this paper, we study the problem of log-based anomaly detection and diagnosis for DBMS. We conduct a comprehensive study on Apache IoTDB, revealing the importance of striking a balance between log quantity and information content to minimize log collection overhead and maximize anomaly detection and diagnosis effectiveness. Based on these findings, we propose E-Log, a fine-grained elastic log-based anomaly detection and diagnosis method that utilizes a small amount of logs for continuous anomaly detection when the database is in a

normal state and activates a large volume of logs for fine-grained anomaly diagnosis. With low runtime overhead, E-Log achieves superior anomaly detection and diagnosis performance.

In the future, our research will focus on developing better methods to identify the relationships between various LPS and types of anomalies, with the aim of further enhancing the effectiveness of anomaly diagnosis. Additionally, we will explore the use of a wider variety of data to minimize the data collection overhead incurred during database runtime for anomaly detection and diagnosis.

REFERENCES

- [1] Z. Yang et al., "Oceanbase: A 707 million tpmC distributed relational database system," *Proc. VLDB Endowment*, vol. 15, no. 12, pp. 3385–3397, 2022.
- [2] S. Dong, A. Kryczka, Y. Jin, and M. Stumm, "RocksDB: Evolution of development priorities in a key-value store serving large-scale applications," *ACM Trans. Storage*, vol. 17, no. 4, pp. 1–32, 2021.
- [3] D. Huang et al., "Tidb: A raft-based htpb database," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3072–3084, 2020.
- [4] C. Wang et al., "Apache IoTDB: A time series database for IoT applications," *Proc. ACM Manage. Data*, vol. 1, no. 2, pp. 1–27, 2023.
- [5] Y. Kang et al., "Separation or not: On handing out-of-order time-series data in leveled LSM-tree," in *Proc. IEEE 38th Int. Conf. Data Eng.*, 2022, pp. 3340–3352.
- [6] L.-Z. Zhang, X.-D. Huang, Y.-K. Wang, J.-L. Qiao, S.-X. Song, and J.-M. Wang, "Time-tired compaction: An elastic compaction scheme for lsm-tree based time-series database," *Adv. Eng. Inform.*, vol. 59, 2024, Art. no. 102224.
- [7] M. Ma et al., "Diagnosing root causes of intermittent slow queries in cloud databases," *Proc. VLDB Endowment*, vol. 13, no. 8, pp. 1176–1189, 2020.
- [8] X. Zhang et al., "Robust log-based anomaly detection on unstable log data," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Foundations Softw. Eng.*, 2019, pp. 807–817.
- [9] L. Yang et al., "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng.*, 2021, pp. 1448–1460.
- [10] T. Jia, Y. Wu, C. Hou, and Y. Li, "LogFlash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems," in *Proc. IEEE 32nd Int. Symp. Softw. Rel. Eng.*, 2021, pp. 80–90.
- [11] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Networks*, 2021, pp. 1–8.
- [12] T. Jia, Y. Li, Y. Yang, G. Huang, and Z. Wu, "Augmenting log-based anomaly detection models to reduce false anomalies with human feedback," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2022, pp. 3081–3089.
- [13] L. Zhang et al., "A survey of AIops for failure management in the era of large language models," 2024, *arXiv:2406.11213*.
- [14] L. Zhang, T. Jia, M. Jia, Y. Li, Y. Yang, and Z. Wu, "Multivariate log-based anomaly detection for distributed database," in *Proc. 30th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2024, pp. 4256–4267.
- [15] Y. Xie, K. Yang, and P. Luo, "LogM: Log analysis for multiple components of hadoop platform," *IEEE Access*, vol. 9, pp. 73522–73532, 2021.
- [16] Y. Sui et al., "LogKG: Log failure diagnosis through knowledge graph," *IEEE Trans. Serv. Comput.*, vol. 16, no. 5, pp. 3493–3507, Sep./Oct. 2023.
- [17] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proc. 38th Int. Conf. Softw. Eng. Companion*, 2016, pp. 102–111.
- [18] H. Ikeuchi, A. Watanabe, T. Kawata, and R. Kawahara, "Root-cause diagnosis using logs generated by user actions," in *Proc. IEEE Glob. Commun. Conf.*, 2018, pp. 1–7.
- [19] Y. Yuan, W. Shi, B. Liang, and B. Qin, "An approach to cloud execution failure diagnosis based on exception logs in openstack," in *Proc. IEEE 12th Int. Conf. Cloud Comput.*, 2019, pp. 124–131.
- [20] C. Duan et al., "Famos: Fault diagnosis for microservice systems through effective multi-modal data fusion," in *Proc. IEEE/ACM 47th Int. Conf. Softw. Eng.*, 2025, pp. 2613–2624.
- [21] C. Duan et al., "EagerLog: Active learning enhanced retrieval augmented generation for log-based anomaly detection," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2025, pp. 1–5.

- [22] S. Xie et al., "TVDiag: A task-oriented and view-invariant failure diagnosis framework for microservice-based systems with multimodal data," *ACM Trans. Softw. Eng. Methodol.*, 2025.
- [23] M. Poess, R. Nambiar, K. Kulkarni, C. Narasimhadvara, T. Rabl, and H.-A. Jacobsen, "Analysis of TPCx-IoT: The first industry standard benchmark for IoT gateway systems," in *Proc. IEEE 34th Int. Conf. Data Eng.*, 2018, pp. 1519–1530.
- [24] Timescale, "TSBS," Jan. 2024. [Online]. Available: <https://github.com/timescale/tsbs>
- [25] R. Liu and J. Yuan, "Benchmarking time series databases with iotdb-benchmark for iot scenarios," 2019, *arXiv:1901.08304*.
- [26] G. Li et al., "openGauss: An autonomous database system," *Proc. VLDB Endowment*, vol. 14, no. 12, pp. 3028–3042, 2021.
- [27] X. Zhou et al., "Dbmind: A self-driving platform in opengauss," *Proc. VLDB Endowment*, vol. 14, no. 12, pp. 2743–2746, 2021.
- [28] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. IEEE Int. Conf. Web Serv.*, 2017, pp. 33–40.
- [29] J. Zhu et al., "Tools and benchmarks for automated log parsing," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Softw. Eng. Pract.*, 2019, pp. 121–130.
- [30] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?," in *Proc. 44th Int. Conf. Softw. Eng.*, 2022, pp. 1356–1367.
- [31] H. Li, W. Shang, B. Adams, M. Sayagh, and A. E. Hassan, "A qualitative study of the benefits and costs of logging from developers' perspectives," *IEEE Trans. Softw. Eng.*, vol. 47, no. 12, pp. 2858–2873, Dec. 2021.
- [32] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," in *Proc. 34nd Int. Symp. Softw. Rel. Eng.*, 2023, pp. 355–366.
- [33] H. Li, *Alluxio: A Virtual Distributed File System*. Berkeley, CA, USA: Univ. California Press, 2018.
- [34] H. Liu and R. Setiono, "Feature selection and classification—A probabilistic wrapper approach," in *Industrial and Engineering Applications or Artificial Intelligence and Expert Systems*. Boca Raton, FL, USA: CRC Press, 2022, pp. 419–424.
- [35] T. Zhang, H. Qiu, G. Castellano, M. Rifai, C. S. Chen, and F. Pianese, "System log parsing: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 8, pp. 8596–8614, Aug. 2023.
- [36] D. Seybold and J. Domaschka, "Benchmarking-as-a-service for cloud-hosted DBMS," in *Proc. 22nd Int. Middleware Conf., Demos Posters*, 2021, pp. 12–13.
- [37] L. Wu, J. Tordsson, J. Bogatinovski, E. Elmroth, and O. Kao, "MicroDiag: Fine-grained performance diagnosis for microservice systems," in *Proc. IEEE/ACM Int. Workshop Cloud Intell.*, 2021, pp. 31–36.
- [38] V. Jeyakumar, O. Madani, A. Parandeh, A. Kulshreshtha, W. Zeng, and N. Yadav, "Explainit!—A declarative root-cause analysis engine for time series data," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 333–348.
- [39] D. Liu et al., "MicroHECL: High-efficient root cause localization in large-scale microservice systems," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng., Softw. Eng. Pract.*, 2021, pp. 338–347.
- [40] P. Liu et al., "FluxRank: A widely-deployable framework to automatically localizing root cause machines for software service failure mitigation," in *Proc. IEEE 30th Int. Symp. Softw. Rel. Eng.*, 2019, pp. 35–46.
- [41] S. Sarkar, D. Staratzis, Z. Zhu, and M. Athanassoulis, "Constructing and analyzing the LSM compaction design space," *Proc. VLDB Endowment*, vol. 14, no. 11, pp. 2216–2229, 2021.
- [42] S. Sarkar, N. Dayan, and M. Athanassoulis, "The LSM design space and its read optimizations," in *Proc. IEEE Int. Conf. Data Eng.*, 2023, 3578–3584.
- [43] C. Mesh, "A powerful chaos engineering platform for kubernetes," *Chaos Mesh*, 2021. Accessed: Aug. 18, 2025. [Online]. Available: <https://chaos-mesh.org/>
- [44] W. Meng et al., "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, vol. 19, no. 7, pp. 4739–4745.
- [45] P. Liu, S. Zhang, Y. Sun, Y. Meng, J. Yang, and D. Pei, "Fluxinfer: Automatic diagnosis of performance anomaly for online database system," in *Proc. IEEE 39th Int. Perform. Comput. Commun. Conf.*, 2020, pp. 1–8.
- [46] Y. Remil, A. Bendimerad, R. Mathonat, P. Chaleat, and M. Kaytoue, "“What makes my queries slow?": Subgroup discovery for SQL workload analysis," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2021, pp. 642–652.
- [47] P. Dintyala, A. Narechania, and J. Arulraj, "SQLCheck: Automated detection and diagnosis of SQL anti-patterns," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 2331–2345.
- [48] D. Dundjerski and M. Tomašević, "Automatic database troubleshooting of azure SQL databases," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1604–1619, Jul.–Sep. 2022.
- [49] B. Glasbergen, M. Abebe, K. Daudjee, and A. Levi, "Sentinel: Universal analysis and insight for data systems," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2720–2733, 2020.
- [50] C. Seo, Y. Chae, J. Lee, E. Seo, and B. Tak, "NoSQL database performance diagnosis through system call-level introspection," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, 2022, pp. 1–9.
- [51] P. Dexter, B. Sendir, and K. Chiu, "Detecting and reacting to anomalies in relaxed uses of raft," in *Proc. 20th IEEE/ACM Int. Symp. Cluster, Cloud Internet Comput.*, 2020, pp. 659–668.
- [52] X. Han and S. Yuan, "Unsupervised cross-system log anomaly detection via domain adaptation," in *Proc. 30th ACM Int. Conf. Inf. Knowl. Manage.*, 2021, pp. 3068–3072.
- [53] Y. Liu et al., "Uniparser: A unified log parser for heterogeneous log data," in *Proc. ACM Web Conf.*, 2022, pp. 1893–1901.
- [54] L. Zhang, T. Jia, M. Jia, Y. Wu, H. Liu, and Y. Li, "ScalaLog: Scalable log-based failure diagnosis using LLM," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2025, pp. 1–5.
- [55] G. Chu et al., "Anomaly detection on interleaved log data with semantic association mining on log-entity graph," *IEEE Trans. Softw. Eng.*, vol. 51, no. 2, pp. 581–594, Feb. 2025.
- [56] M. Landauer, F. Skopik, and M. Wurzenberger, "A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 1354–1375, 2024.
- [57] H. Guo et al., "LogFormer: A pre-train and tuning pipeline for log anomaly detection," in *Proc. AAAI Conf. Artif. Intell.*, 2024, vol. 38, no. 1, pp. 135–143.
- [58] S. Hashemi and M. Mäntylä, "OneLog: Towards end-to-end software log anomaly detection," *Automated Softw. Eng.*, vol. 31, no. 2, 2024, Art. no. 37.
- [59] L. Zhang, T. Jia, K. Wang, M. Jia, Y. Yang, and Y. Li, "Reducing events to augment log-based anomaly detection models: An empirical study," in *Proc. 18th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2024, pp. 538–548.
- [60] L. Zhang et al., "Towards close-to-zero runtime collection overhead: Raft-based anomaly diagnosis on system faults for distributed storage system," *IEEE Trans. Serv. Comput.*, vol. 18, no. 2, pp. 1054–1067, Mar./Apr. 2025.
- [61] L. Zhang, T. Jia, M. Jia, Y. Wu, H. Liu, and Y. Li, "XRAGLog: A resource-efficient and context-aware log-based anomaly detection method using retrieval-augmented generation," in *Proc. AAAI Workshop Preventing Detecting LLM Misinformation*, 2025.
- [62] N. Zhao et al., "An empirical investigation of practical log anomaly detection for online service systems," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Foundations Softw. Eng.*, 2021, pp. 1404–1415.
- [63] Y. Li et al., "Exploring the effectiveness of llms in automated logging generation: An empirical study," 2023, *arXiv:2307.05950*.
- [64] L. Zhang, Y. Zhai, T. Jia, X. Huang, C. Duan, and Y. Li, "Agentfm: Role-aware failure management for distributed databases with LLM-driven multi-agents," in *Proc. 33rd ACM Int. Conf. Found. Soft. Eng.*, 2025, pp. 525–529.
- [65] L. Zhang et al., "Thinkfl: Self-refining failure localization for microservice systems via reinforcement fine-tuning," 2025, *arXiv:2504.18776*.
- [66] L. Zhang, T. Jia, M. Jia, and Y. Li, "LogDB: Multivariate log-based failure diagnosis for distributed databases (Extended from multilog)," 2025, *arXiv:2505.01676*.
- [67] L. Zhang et al., "A survey of AIOps in the era of large language models," *ACM Comput. Surveys*, 2025.
- [68] M. He, T. Jia, C. Duan, H. Cai, Y. Li, and G. Huang, "Weakly-supervised log-based anomaly detection with inexact labels via multi-instance learning," in *Proc. IEEE/ACM 47th Int. Conf. Softw. Eng.*, 2025, pp. 2918–2930.
- [69] P. Xiao, T. Jia, C. Duan, H. Cai, Y. Li, and G. Huang, "LogCAE: An approach for log-based anomaly detection with active learning and contrastive learning," in *Proc. IEEE 35th Int. Symp. Softw. Rel. Eng.*, 2024, pp. 144–155.
- [70] M. He, T. Jia, C. Duan, H. Cai, Y. Li, and G. Huang, "LLMeLog: An approach for anomaly detection based on llm-enriched log events," in *Proc. IEEE 35th Int. Symp. Softw. Rel. Eng.*, 2024, pp. 132–143.

- [71] S. He et al., "An empirical study of log analysis at microsoft," in *Proc. 30th ACM Joint Eur. Softw. Eng. Conf. Symp. Foundations Softw. Eng.*, 2022, pp. 1465–1476.
- [72] Y. Fu, M. Yan, Z. Xu, X. Xia, X. Zhang, and D. Yang, "An empirical study of the impact of log parsers on the performance of log-based anomaly detection," *Empirical Softw. Eng.*, vol. 28, no. 1, 2023, Art. no. 6.
- [73] B. Yu et al., "Deep learning or classical machine learning? An empirical study on log-based anomaly detection," in *Proc. 46th IEEE/ACM Int. Conf. Softw. Eng.*, 2024, pp. 403–415.
- [74] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surveys*, vol. 54, no. 6, pp. 1–37, 2021.



Lingzhe Zhang received the MS degree in software engineering from Tsinghua University, China. He is currently working toward the PhD degree in software engineering with Peking University. His research interests include cloud computing, distributed storage system, and anomaly detection.



Tong Jia received the PhD degree in software engineering from Peking University, China, in 2019. He is currently an assistant research professor with the Institute for Artificial Intelligence, Peking University. He has authored or coauthored more than 30 papers in international journals and conferences. His research interests include AIOps and anomaly detection.



Xinyu Tan received the MS degree from Tsinghua University, China, in 2023. He is currently a senior development engineer with Timecho Ltd., Beijing, China. He has led the development and maintenance of the distributed module and observability module for Apache IoTDB.



Xiangdong Huang received the PhD degree from Tsinghua University, Beijing, China, in 2017. He is currently an associate professor with the School of Software, Tsinghua University. He is also the CEO with Timecho Ltd., Beijing. His research interests include Big Data storage systems and time-series data management.

Mengxi Jia Author photograph and biography not available at the time of publication.



Hongyi Liu received the MS degree in 2021 from Peking University, where he is currently working toward the PhD degree with the School of Software & Microelectronics. His research interests include could computing, software reliability, anomaly detection, and AI for software engineering.



Zhonghai Wu received the PhD from Zhejiang University, Hangzhou, China, in 1997. He was a post-doctoral and associate professor with the Institute of Computer Science and Technology, Peking University, Beijing, China. He is currently a professor and dean with the School of Software and Microelectronics, Peking University. His research interests include cloud computing, Big Data, AI, and information security.



Ying Li (Member, IEEE) received the PhD degree in computer science and engineering from Northwestern Polytechnical University, China, in 2001. In 2012, she was a STSM and senior manager with the Department of Distributed Computing, IBM China Research Center. She is currently a professor with the School of Software and Microelectronics, Peking University, China. She has authored or coauthored more than 60 papers in international journals and conferences. She filed more than 30 US/CN granted patents. Her research interests include distributed computing and dependability engineering. She was a PC member of several international conferences and reviewer of international journals. She is also a member of IEEE. She was rewarded with "IBM Global Research Accomplishment Award" twice, "CIO Leadership Award", and "IBM Master Inventor".