EasyChair Preprint

№ 14748

Reducing Events to Augment Log-Based Anomaly Detection Models: an Empirical Study

Lingzhe Zhang, Tong Jia, Kangjin Wang, Mengxi Jia, Yong Yang and Ying Li

September 7, 2024

Reducing Events to Augment Log-based Anomaly Detection Models: An Empirical Study

Lingzhe Zhang
Peking University
Beijing, China
zhang.lingzhe@stu.pku.edu.cn

Tong Jia*
Peking University
Beijing, China
jia.tong@pku.edu.cn

Kangjin Wang
Alibaba Group
Beijing, China
kangjin.wkj@alibaba-inc.com

Mengxi Jia
Peking University
Beijing, China
mxjia@pku.edu.cn

Yong Yang
Peking University
Beijing, China
yang.yong@pku.edu.cn

Ying Li*
Peking University
Beijing, China
li.ying@pku.edu.cn

Abstract

As software systems grow increasingly intricate, the precise detection of anomalies have become both essential and challenging. Current log-based anomaly detection methods depend heavily on vast amounts of log data leading to inefficient inference and potential misguidance by noise logs. However, the quantitative effects of log reduction on the effectiveness of anomaly detection remain unexplored. Therefore, we first conduct a comprehensive study on six distinct models spanning three datasets. Through the study, the impact of log quantity and their effectiveness in representing anomalies is qualifies, uncovering three distinctive log event types that differently influence model performance. Drawing from these insights, we propose LogCleaner: an efficient methodology for the automatic reduction of log events in the context of anomaly detection. Serving as middleware between software systems and models, LogCleaner continuously updates and filters *anti-events* and *duplicative-events* in the raw generated logs. Experimental outcomes highlight LogCleaner's capability to reduce over 70% of log events in anomaly detection, accelerating the model's inference speed by approximately 300%, and universally improving the performance of models for anomaly detection.

CCS Concepts

• **Software and its engineering → Maintaining software**.

Keywords

Anomaly Detection, Log Reduction, Log Analysis

ACM Reference Format:
Lingzhe Zhang, Tong Jia*, Kangjin Wang, Mengxi Jia, Yong Yang, and Ying Li*. 2024. Reducing Events to Augment Log-based Anomaly Detection Models: An Empirical Study. In *Proceedings of the 18th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24), October 24–25, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3674805.3695403

*Corresponding author.

ESEM '24, October 24–25, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-1047-6/24/10
https://doi.org/10.1145/3674805.3695403

1 Lay Abstract

As software systems become more complex, detecting problems or unusual behaviors (called anomalies) in these systems is both critical and difficult. Most current methods for finding these anomalies rely on processing huge amounts of log data, which can slow down the process and may lead to inaccurate results due to unnecessary or noisy logs. Despite this, the impact of reducing log data on anomaly detection hasn't been studied much.

To address this gap, we conduct a detailed study using six different models and three datasets to see how the quantity and quality of log data affect the ability to detect anomalies. We discover that not all log events are equally important—some help models perform better, while others have little or even negative effects. Based on these findings, we develop a tool called LogCleaner.

LogCleaner is designed to automatically reduce the number of log events while still maintaining the important information needed for detecting anomalies. It works as a middle layer between the software system and the detection models, continuously removing unnecessary events (which we call "anti-events" and "duplicative-events") from the raw logs.

Our experiments show that LogCleaner can remove more than 70% of log events without hurting the ability to detect anomalies. In fact, by reducing the noise, it speeds up the models by about 300% and improves their overall performance. LogCleaner offers a practical solution for developers and engineers looking to make anomaly detection faster and more accurate.

2 Introduction

Modern software systems are becoming increasingly complex, leading to more frequent failures that can cause considerable losses even during short periods of unavailability[9, 48]. Detecting anomalies accurately has therefore become critical for ensuring reliable and continuously available services. System logs provide valuable runtime information about software states and events, making them an indispensable resource for log-based anomaly detection approaches. With the ability to pinpoint failures and prevent further

deterioration, log-based anomaly detection have garnered significant attention as important ways to maintain highly secure and resilient software systems in the face of rising complexity.

In recent years, anomaly detection based on system logs has gained significant research attention. These log-based anomaly detection models can be broadly classified into two categories: supervised models[2, 3, 10, 26, 30, 40, 47, 49] and unsupervised models[1, 7, 11, 19, 20, 22, 23, 33, 42]. Supervised models, such as RobustLog[49], necessitate labeled data comprising both normal and abnormal instances to construct their predictive frameworks. In contrast, unsupervised models detect deviations relying solely on standard data. They are primarily split into deep neural network-based[7, 22, 33, 42] and graph-based models[1, 11, 19, 20, 22].

Despite the promising results demonstrated by these anomaly detection methods, they directly leverage extensive log data generated by software systems, leading to the following practical challenges:

- **Inefficient Inference:** With an increasing number of logs, the model's inference speed tends to slow down. If a substantial portion of these original logs consists of irrelevant entries, it can result in unnecessary degradation of inference speed and resource wastage. [43].
- **Misleading by Noise Logs:** It is acknowledged that having more logs provides a wealth of information, but in reality, many logs are of low quality, and some even contain noise. This can mislead the model [25, 49].

In fact, not all logs generated by software systems are essential. However, the quantitative effects of log reduction on the effectiveness of anomaly detection remain unexplored. The significance of various log types and the subsequent performance trade-offs post their elimination remain uncertain.

To fill this significant gap, we conduct an empirical study to quantify the impact of log reduction on anomaly detection The investigation spans six anomaly detection models (LR[2], SVM[26], Decision Tree[3], Isolation Forest[14], RobustLog[49], PLELog[40]) applied to three datasets[18] (HDFS, BGL, Thunderbird). We design two approaches: a retry-based method and a clustering-based approach, to validate the extent of log event reduction possible under constrained model performance degradation thresholds. The results reveal that for anomaly detection models, log events can be significantly reduced, and the reduction of logs can even enhance model effectiveness. In extreme cases, such as the Thunderbird dataset, a single log event (originally 1406 log events) can identify most anomalies.

Furthermore, this work conducts an in-depth analysis of reducible log events for anomaly detection. The events are categorized into *anti-events* and *duplicative-events* based on whether their removal improves or does not affect model performance. Additionally, whose removal degrades model effectiveness are identifies as *key-events*.

Building on the findings of the empirical study, we introduce **LogCleaner**, a comprehensive methodology designed for the automatic reduction and reporting of *anti-events* and *duplicative-events* in log events, specifically tailored for anomaly detection. LogCleaner is divided into an profiling and an online component. In the profiling part, it utilizes historical logs, applying TF-IDF to eliminate sporadic log events, then using mutual information to reduce *anti-events*. Finally, it employs a graph-based clustering approach to eliminate *duplicative-events*, resulting in a reduced event set. In the online part, it functions as middleware between software systems and models, streamlining raw generated logs to reduced logs using the reduced event set. The reduced logs are then employed for anomaly detection. Additionally, whenever there is a variation in the code of the software system, the system's logs and existing labels are re-extracted for re-profiling. This re-profiling process enables LogCleaner to adapt effectively to potential future anomalies.

We evaluate LogCleaner's effectiveness across the aforementioned models and datasets. Results show that LogCleaner can reduce over 70% of log events in anomaly detection, accelerate the model's inference speed by approximately 300%, while universally improve the performance of models for both anomaly detection. In summary, the contributions are as follows:

- We conduct a comprehensive study to quantify the impact of log event reduction on anomaly detection model effectiveness. Our findings reveal the remarkable extent to which the number of log events can be reduced without compromising model performance. Furthermore, our empirical study categorizes log events into *key-events*, *anti-events*, and *duplicative-events* based on the impact of their removal on model performance.
- Inspired by the findings, we introduce **LogCleaner**, an efficient methodology for the automatic reduction of log events in the context of anomaly detection. Serving as middleware between software systems and models, LogCleaner continuously updates and filters *anti-events* and *duplicative-events* in the raw generated logs.
- We validate LogCleaner's effectiveness across 6 anomaly detection models on 3 datasets. Experiments demonstrate LogCleaner universally improves detection model performance while reducing over 70% of log events and accelerating the model's inference speed by approximately 300%.

3 Background

This section provides background on log-based anomaly detection, introduces the models that will be used in the empirical study, and outlines the common overall framework for log-based anomaly detection.

3.1 Anomaly Detection

Anomaly detection[1–3, 7, 10, 11, 19, 20, 26, 30, 40, 42, 49] aims to identify irregularities in system behavior. Detection methods are broadly categorized into supervised and unsupervised models. Supervised models[2, 3, 10, 26, 30, 40, 49], like **RobustLog**[49], require labeled data that includes both normal and abnormal examples to form predictive frameworks. **PLELog**[40] addresses the issue of insufficient labels via probabilistic label estimation and designs an attention-based GRU neural network to detect anomalies. Loglizer[17] offers a comprehensive toolkit featuring several machine-learning-based log analysis models designed for automated anomaly detection, including linear regression (**LR**)[2], **SVM**[26], **Decision Tree**[3], **Isolation Forest**[14]. In contrast, unsupervised models[1, 7, 11, 19, 20, 42] identify deviations based only on standard data. In this paper, we focus on supervised models

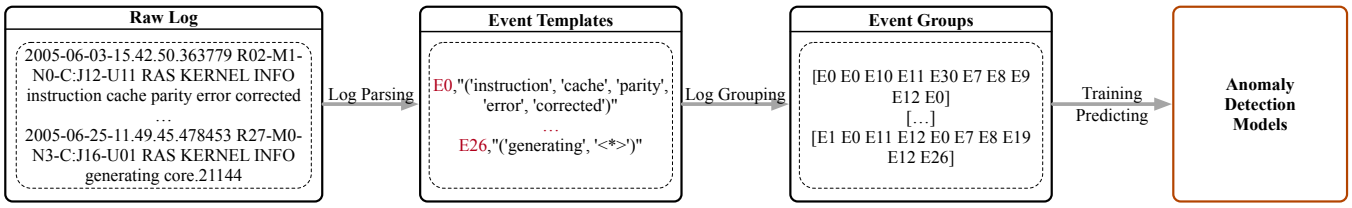


Figure 1: Log-based Anomaly Detection: The Common Workflow

in analysis, given that labeled data allows a systematic assessment of log event reduction's impact on model performance. Unsupervised approaches cannot conclusively link performance changes to specific log event removals due to the absence of labels.

3.2 The Common Workflow

Despite that the target and approaches of anomaly detection are quite different, they share share common workflow[19, 24]. As shown in figure 1, the framework consists of three steps: (1) log parsing, (2) log grouping, (3) anomaly detection.

3.2.1 Log parsing. Raw logs consist of semi-structured text encompassing various fields like timestamps and severity levels. For the benefit of downstream tasks, log parsing is employed to transform each log message into a distinct event template, which includes a constant part paired with variable parameters. For example, the log template "E0,('instruction', 'cache', 'parity', 'error', 'corrected')" can be extracted from the log message "2005-06-03-15.42.50.363779 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected" in figure 1. There are many log parsing methods, based on frequent pattern mining[5, 34, 37, 44], clustering[12, 35, 38, 39], and heuristics[16, 21, 31]. This paper utilizes the **Brain**[44] implemented by Logparser[15, 51].

3.2.2 Log grouping. After being parsed into event templates, log data can be organized into sequence groups using session, sliding, or fixed windows. Determining an optimal window size is challenging. For instance, a small window size might impede the models' ability to recognize anomalies that stretch across multiple sequences. Conversely, a large window size could lead to log sequences encompassing multiple anomalies, thereby complicating the detection process[24]. This study adopts both session-based and fixed windows of 100 logs, aligning parameters with those presented in the survey[24].

3.2.3 Anomaly detection. After converting log events into sequences, they are processed by the previously mentioned anomaly detection models. These models undergo profiling training and then facilitate online prediction.

4 Study Design

This section outlines the datasets and models under evaluation and provides an overview of the methodology adopted for the empirical study.

4.1 Datasets

In our assessment of models for log-based anomaly detection, three datasets are employed[18]: HDFS, BGL and Thunderbird. The details of each dataset are as follows:

HDFS dataset originates from over 200 Amazon EC2 nodes. It encompasses a total of 11,175,629 log messages. These messages are grouped into distinct log windows based on their block_id, representing individual program executions within the HDFS system. Notably, 16,838 log blocks (amounting to 2.93%) within this dataset signify system anomalies.

BGL dataset is derived from a supercomputing system and was gathered by Lawrence Livermore National Labs (LLNL). It comprises a total of 4,747,963 log messages. Every message within the BGL dataset has been manually categorized as either normal or anomalous. Notably, of these, 348,460 log messages (representing 7.34%) are marked as anomalous.

Thunderbird dataset is an open collection of logs sourced from the Thunderbird supercomputer at Sandia National Labs (SNL). This dataset encompasses both regular and anomalous messages, each of which has been manually classified. While the Thunderbird dataset encompasses a massive collection of over 200 million log messages, this paper opts to use an initial continuous subset of 10 million log lines for the sake of computational efficiency. Notably, this subset includes 353,794 anomalous log messages, constituting 3.53% of the total.

4.2 Evaluated Models

In this study, we evaluate the six representative models described in section 3.1. The source code for all anomaly detection models[6, 17, 40] are public. In terms of log parsing, we employ the Brain[44] method as implemented by Logparser[15, 51]. For log grouping, we adopt different strategies based on the dataset: session-based windows are applied to the HDFS dataset, while for the BGL and Thunderbird dataset, we utilize fixed windows comprising 100 logs.

4.3 Approach

As previously mentioned, the aim of study is to quantify the effect of log event reduction on the effectiveness of anomaly detection models. To achieve this, we introduce two empirical study methodologies: the Retry-based approach and the Cluster-based approach.

4.3.1 Retry-based approach. The core concept behind the Retry-based approach is to iteratively remove one log event at a time and then retrain the model to assess its effectiveness. If the model's effectiveness decreases after the removal, that particular log event is retained; otherwise, it's deemed useless.

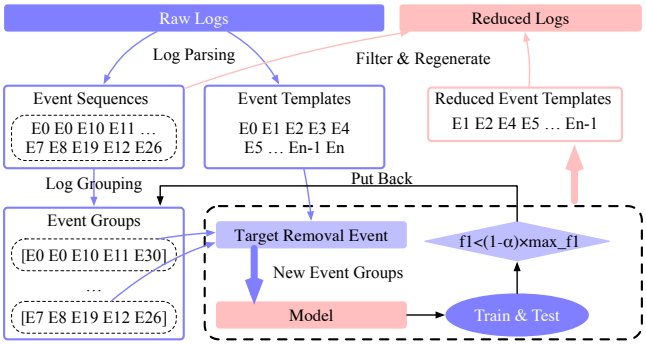


Figure 2: Process of Retry-based Approach

As illustrated in figure 2, this approach initially employs the previously mentioned log parsing tool to extract event sequences and templates. Following this, the respective log grouping algorithm is utilized to organize these events into groups. Sequentially, it attempts to eliminate one log event from the event templates (designated as the target removal event), and simultaneously remove the corresponding event from each event group. It's crucial to note that after removing an event, the log grouping algorithm isn't re-executed. This ensures the preservation of labels for each event group. Thus, even if an event group is devoid of any events, its associated label is still retained.

$$f1 < (1 - \alpha) \times f1_{max} \tag{1}$$

The regenerated event groups are subsequently fed into the respective model for retraining and testing. From this, it obtains metrics such as precision, recall, and the F1-score. If the model's performance meets the criteria defined in equation 1, it indicates that the removal of that particular event impacts the model's effectiveness. As a result, this event is reintegrated into both the event templates and event groups. On the other hand, if the event doesn't significantly influence the performance, it is deemed redundant and removed, with the F1-score at that point recorded as $f1_{max}$. Here, α represents the permissible threshold for performance degradation. It's important to highlight that minor temporary performance dips during the model's training and testing phase don't necessarily signify a permanent degradation in model efficacy. Such deviations might merely be due to natural random fluctuations. Thus, even though it establishes a threshold with α, the overall model effectiveness might not necessarily decline upon the completion of experiments.

4.3.2 Clustering-based approach. The Retry-based approach can produce near-optimal results. However, its necessity to retrain the model every time an event is removed becomes prohibitively time-consuming when dealing with datasets that have numerous events and a large volume of log events. This is because most model training durations are directly proportional to the volume of log events. For instance, considering the Thunderbird dataset, which consists of 1,406 event templates, if the SVM model initially takes around 10 minutes for each train-test iteration, completing the entire experiment will demand almost 10 days of computational time. Moreover, it's essential to highlight that many experiments require multiple runs to ensure consistent and reliable results.

Thus, to expedite the categorization and filtering of irrelevant log events, we introduce the Clustering-based approach. As illustrated in figure 3, it begins by extracting all log templates. Sequentially, it identifies each log event within the event templates as the Target Test Event. For each event group, only the corresponding Target Test Event is retained. These single event groups are then subjected to the specific model for retraining and testing. The resulting precision, recall, and F1-score are documented for every iteration. Ultimately, it obtains the precision, recall, and F1-score associated with each individual log event. Using a clustering algorithm (KMeans in this paper), based on the precision, recall, and F1-score, it classifies log events into two categories: irrelevant and relevant events. Finally, within the scope of relevant events, the Retry-based approach is executed.

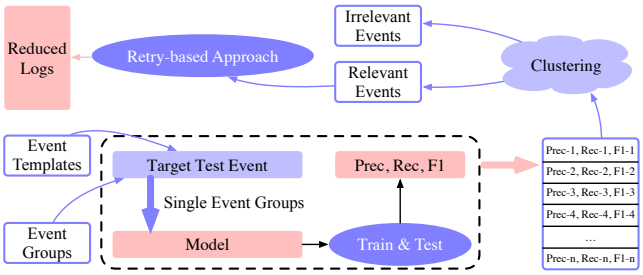


Figure 3: Process of Clustering-based Approach

In summary, although the Retry-based approach can yield near-optimal results, it runs slowly when there are many event templates (as the model needs to be retrained every time an event is removed). Consequently, in our subsequent experiments, we employ the Retry-based approach for the HDFS dataset. For the BGL and Thunderbird datasets, we opt for the Clustering-based approach.

4.4 Research Questions

The objective of this research is to quantify the impact of log event reduction on the performance of anomaly detection models. Several research questions are formulated to guide the investigation, leveraging the experimental approaches previously described.

RQ1: To what extent can each anomaly detection method reduce log events? We aim to assess the extent to which each method can reduce log events while maintaining the performance of existing anomaly detection approaches. For this purpose, useless otherwise mentioned, we conduct experiments on the studied models across various datasets with $\alpha = 0.02$. This parameter choice ensures that the natural random fluctuations do not lead to false identification of an event as relevant, while also ensuring that the model's performance does not degrade significantly.

RQ2: How does log event reduction impact the performance of existing anomaly detection approaches? In RQ1, we conduct experiments using $\alpha = 0.02$, indicating a slight decline in the model's performance. However, as previously analyzed, this isn't necessarily the case. Hence, in this research question, we will further delve into a quantitative analysis of the impact of event reduction on the performance of anomaly detection approaches.

RQ3: What types of log events can be reduced without degrading anomaly detection performance? In RQ1 and RQ2, we validate the quantity of log events that can be reduced and provide the performance of the model post-reduction. However, we also wish to analyze how log events should be distinguished in scenarios without access to the source code. Therefore, in this research question, we conduct several case studies on anomaly detection models to verify the different types of log events.

5 Empirical Results

This section presents and addresses the research questions proposed in section 4.4.

5.1 RQ1: Reduction Extent of Log Events for Anomaly Detection Methods

For RQ1, we conduct experiments on the studied models across various datasets with $\alpha = 0.02$, to investigate the potential reduction in the volume of log events and lines. The experimental results are shown in table 1.

Table 1: Data Volume Reduction in Anomaly Detection Across Various Models and Datasets ($\alpha = 0.02$)

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

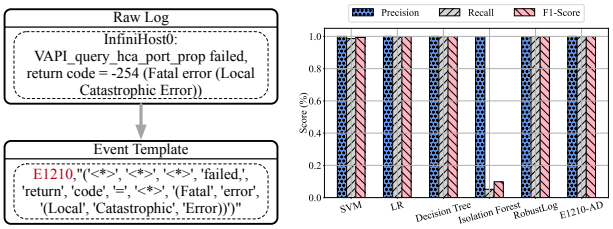
It can be find that, while maintaining consistent model performance, all anomaly detection models show a significant reduction in log events. The reduction ranges from a minimum of 55.17% (with the LR model on the HDFS dataset) to more than 99% (in the Thunderbird dataset). This suggests that a majority of log events in the HDFS, BGL, and Thunderbird datasets are, in fact, superfluous.

We also explore the reduction of log lines (where one log event corresponds to multiple lines of actual printed logs, as each log event essentially corresponds to each line of code where developers write print statements). Even though the model LR 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 log events are of high frequency, constituting a large proportion of the entire dataset. Globally, all anomaly detection models show a significant reduction in log lines. To further underscore the significance of these results, we take the SVM model on the BGL dataset as an example. The original BGL log file is 743.19MB in size. After reduction, it is whittled down to just 7.87MB. This not only dramatically accelerates the model's training speed but can also provide feedback to system developers, thereby reducing the overhead associated with log collection.

Table 2: Remaining Log Events After Reduction in Anomaly Detection Across Various Models and Datasets ($\alpha = 0.02$)

Model	HDFS (29 events)	BGL (688 events)	Thunderbird (1406 events)
LR	13	17	1
SVM	10	29	1
Decision Tree	7	52	6
Isolation Forest	9	182	90
RobustLog	12	38	1
PLELog	10	37	1

We also observe a pervasive and startling reduction in some datasets. Therefore, as depicted in table 2, we further examine the remaining log events after reduction. It's evident that the reason why various models have a relatively low reduction ratio on the HDFS dataset is due to the dataset itself containing only 29 events. For the BGL dataset, apart from the Isolation Forest model, all other models require fewer than 50 out of the 688 events. As for the Thunderbird dataset, an even more remarkable result emerges: for models like LR, SVM, and RobustLog, they only require one out of the 1,406 events to achieve exceptionally high accuracy.



(a) Log Event Template (b) Performance

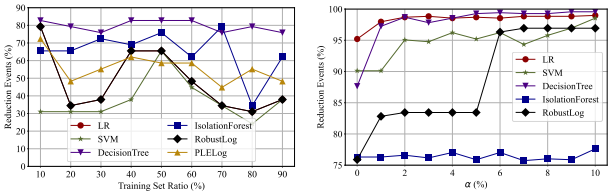
Figure 4: Analysis of the Only Remaining Event (E1210) and Its Impact on Model Performance

For the Thunderbird dataset, a notable observation is that the sole remaining event for these models is E1210, as illustrated in figure 4(a). We hypothesize that the occurrence of the E1210 log event could signify the presence of an anomaly within this dataset. To validate this, we specifically devise a heuristic method for detection, dubbed E1210-AD. The results confirm the speculation, as depicted in figure 4(b). Apart from the Isolation Forest model, the accuracy of other models essentially reaches above 98.5%, aligning closely with the performance of E1210-AD.

Subsequently, we conduct further experiments to investigate how the quantity of reduced logs varies with changing values of α. As depicted in figure 5, for the HDFS dataset, the amount of

Table 3: Comparison of Anomaly Detection Model Performance with/out event reduction ($\alpha = 0.02$)

Model		HDFS			BGL			Thunderbird		
		Precision	***Recall***	***F1-Score***	***Precision***	***Recall***	***F1-Score***	***Precision***	***Recall***	***F1-Score***
LR	**w/o**	0.952	0.711	0.814	0.193	0.829	0.313	0.971	0.995	0.983
	w	0.948	0.970	0.959	0.982	0.456	0.623	1.000	0.999	0.999
SVM	**w/o**	0.959	0.889	0.923	0.877	0.378	0.529	0.991	0.996	0.994
	w	0.959	0.889	0.923	0.979	0.433	0.601	1.000	0.998	0.999
Decision Tree	**w/o**	0.998	0.998	0.998	0.992	0.406	0.576	1.000	0.999	0.999
	w	0.998	0.998	0.998	0.989	0.450	0.618	1.000	0.999	1.000
Isolation Forest	**w/o**	0.822	0.742	0.780	0.613	0.166	0.261	0.005	0.001	0.002
	w	0.936	0.911	0.923	0.917	0.240	0.381	0.775	0.097	0.173
RobustLog	**w/o**	0.985	0.888	0.934	1.000	0.991	0.996	0.910	0.559	0.692
	w	0.994	0.995	0.994	1.000	1.000	1.000	1.000	1.000	1.000
PLELog	**w/o**	0.983	0.843	0.908	0.943	0.986	0.969	0.968	0.996	0.982
	w	0.998	0.971	0.984	0.999	0.967	0.983	1.000	0.996	0.998



(a) HDFS (b) BGL

Figure 5: Extent of Log Event Reduction in Anomaly Detection Methods Depending on the Variation of α

reduction doesn't significantly change when $\alpha \in \{0, 0.01\}$. Beyond this range, it remains relatively stable. This is attributed to the fact that when α is too low, the results might be swayed by the model's inherent random fluctuations, making it challenging to accurately determine the relevance of each log event.

For the BGL dataset, a similar pattern emerges, with exceptions observed for RobustLog and Isolation Forest. In the case of RobustLog, a noticeable change occurs when $\alpha = 0.05$. This suggests that there's a set of events that can be eliminated for RobustLog, but doing so might genuinely impact its performance. As for the Isolation Forest, its behavior remains consistent throughout, likely because it struggles to effectively identify anomalies in the BGL dataset.

Summary. A large number of log events can be reduced in anomaly detection. In some extreme cases, it can even be reduced to a single event.

5.2 RQ2: Performance Following Event Reduction

For RQ2, we conduct a detailed analysis of the performance of anomaly detection models with and without event reduction, setting $\alpha = 0.02$.

As demonstrated in table 3, the performance of nearly all models improved with event reduction. Some even experienced significant enhancements. For instance, RobustLog on the Thunderbird dataset initially has an F1-Score of 69.2%. However, after event reduction, it soares to 100%. For the Isolation Forest model, its original performance on the Thunderbird dataset is nearly negligible with an F1-Score of 0.02%. However, after event reduction, this score improved to 17.3%. Even more impressively, for the LR model on the BGL dataset, by balancing Precision and Recall (Precision shifted from 19.3% to 98.2%, and Recall shifted from 82.9% to 45.6%), the overall F1-Score nearly doubles compared to the initial performance.

Summary. For anomaly detection, after performing log reduction, the model's effectiveness can also be significantly improved.

5.3 RQ3: Types of Log Events That Can Be Reduced

In RQ1 and RQ2, we discover that, for anomaly detection, the model performance can improve to varying degrees with a significant reduction in log events. Thus, in RQ3, we delve deeper into this phenomenon by case study.

Initially, we examine the reasons for the enhancement in model performance after event reduction. In a particular instance with the LR model in the HDFS dataset, upon removing the entry "*E3,[*]Served block[*]to[*]*", there's a notable improvement: Precision by 0.63%, Recall by 22.88%, and F1-Score by 13.49%. The reason for such a phenomenon is that this work discovers that this event appears in the normal label with a ratio of 23.97% and in the abnormal label with a ratio of 21.54%. This suggests that this

particular log event acts as a distractor, potentially misleading the model's classification efforts. Furthermore, a similar pattern can be observed across all analyzed models.

Finding 1. In the dataset, there exists a type of event called **anti-event**. Its presence has no bearing on whether the system has generated an anomaly. Instead, it can mislead the model's classification.

However, during experiments, it is observed that the number of anti-events is relatively small. In fact, the most frequently eliminated events belong to another category termed as duplicative-events. Taking the Decision Tree experiment on the HDFS dataset as an example: when using only *E9*, it obtains a Precision of 100%, Recall of 37.56%, and F1-Score of 54.61%. With only *E11*, the Precision is 100%, Recall is 37.55 %, and F1-Score is 54.59%. When both *E9* and *E11* are used simultaneously, the metrics are Precision at 100%, Recall at 37.56%, and F1-Score at 54.61%. These results suggest that in anomaly detection, certain log events can effectively substitute for others. In such cases, it can safely remove the redundant logs without any loss of information.

Finding 2. In anomaly detection, certain events can encompass the information of others. These overshadowed events can be safely removed without compromising model effectiveness. These events are termed as **duplicative-event**.

Beyond the anti-events and duplicative-events that can be reduced, there remains a category of log events in the dataset that play a pivotal role in model effectiveness, termed as **key-event**. Taking the experiment with the Decision Tree on the HDFS dataset as an example, it can be found that when using *E20* alone, the Precision is 95.85%, Recall is 29.92%, and F1-Score is 45.61%. When using *E26* alone, the Precision is 97.26%, Recall is 59.46%, and F1-Score is 73.80%. However, when both *E20* and *E26* are used together, the Precision is 96.74%, Recall is 88.24%, and F1-Score soar to 92.30%. This suggests that the system information reflected by E20 and E26 is complementary to each other. Such events are the ones that truly need to retain.

Finding 3. There exists a category of log events in the dataset that are crucial for model performance, with their information complementing each other. These are termed as **key-events**. It is these events that truly need to retain.

6 LogCleaner

Our empirical study identifies three types of log events that have different effects on the models. However, the experiments presented earlier required continuous model execution to determine the log events that can be reduced. Moreover, these methods do not provide an opportunity to reintroduce eliminated log events, even though they may represent potential future anomalies. In this section, we introduce **LogCleaner**, an automated approach to reduce log events without relying on model execution. Additionally, it allows for the reintroduction of some reduced log events when the system encounters false negatives, providing the model with the minimum log event set for current-state detection.

As demonstrated in figure 6, LogCleaner is divided into an profiling and an online component. In the profiling phase, it aims to automatically generate a reduced event set. To achieve this, raw logs are initially parsed into event templates and grouped into event groups with corresponding labels. Subsequently, TF-IDF is applied to filter out infrequently occurring events. The remaining events (Frequency Events) undergo processing by the **Anti-Event Optimizer**, which utilizes both event groups and associated labels, employing mutual information to eliminate anti-events. The events surviving this stage (Relevant Events) then go through the **Duplicative-Event Separator**, where the OPTICS algorithm clusters similar events, retaining only one event within each cluster. Finally, the reduced event set are generated based on the retained events.

In the online phase, LogCleaner serves as middleware between software systems and models, streamlining raw generated logs into reduced logs using the reduced event set generated in profiling phase. These reduced logs are then utilized for anomaly detection. Additionally, whenever there is a variation in the code of the software system, the system's logs and existing labels are re-extracted for re-profiling. This re-profiling process enables LogCleaner to adapt effectively to potential future anomalies.

6.1 Anti-Event Optimizer

In RQ3, it can be discovered that certain anti-events have no correlation with the occurrence of system anomalies or the specific anomalies that are triggered. Consequently, these anti-events negatively impact the model's classification performance.

The analysis suggests that the presence or absence of such log events bears no relation to labels. Therefore, mutual information, a method from the feature selection domain, can be employed to estimate the relationship between each log event and its corresponding label.

$$\mathrm{MI}(e;l) = \sum_{e,l} p(e,l) \log\left(\frac{p(e,l)}{p(e)p(l)}\right) \tag{2}$$

As illustrated in equation 2, all events are denoted as $e \in E$, all labels as $l \in L$, $p(e,l)$ represents the joint probability distribution of E and L, while $p(e)$ and $p(l)$ are the marginal probability distributions of E and L, respectively.

Ultimately, as depicted in equation 3, the mutual information for each event e is represented as the average of its mutual information with all labels l. Among them, events with $\mathrm{MI}(e;L) \leq \theta_{anti}$ are deemed as anti-events.

$$\mathrm{MI}(e;L) = \frac{1}{|L|} \sum_{l \in L} \mathrm{MI}(e;l) \tag{3}$$

6.2 Duplicative-Event Separator

For duplicative-events, LogCleaner initially constructs an appear graph based on the co-occurrence patterns of log events. This entails representing each log event as a vector (e_i). When two log events (e_i, e_j) co-occur within a single event group, the weight of the edge ($w(e_i, e_j)$) between them in the graph is incremented.

Next, LogCleaner employs the OPTICS algorithm, a density-based clustering method, to cluster the adjacency matrix of the

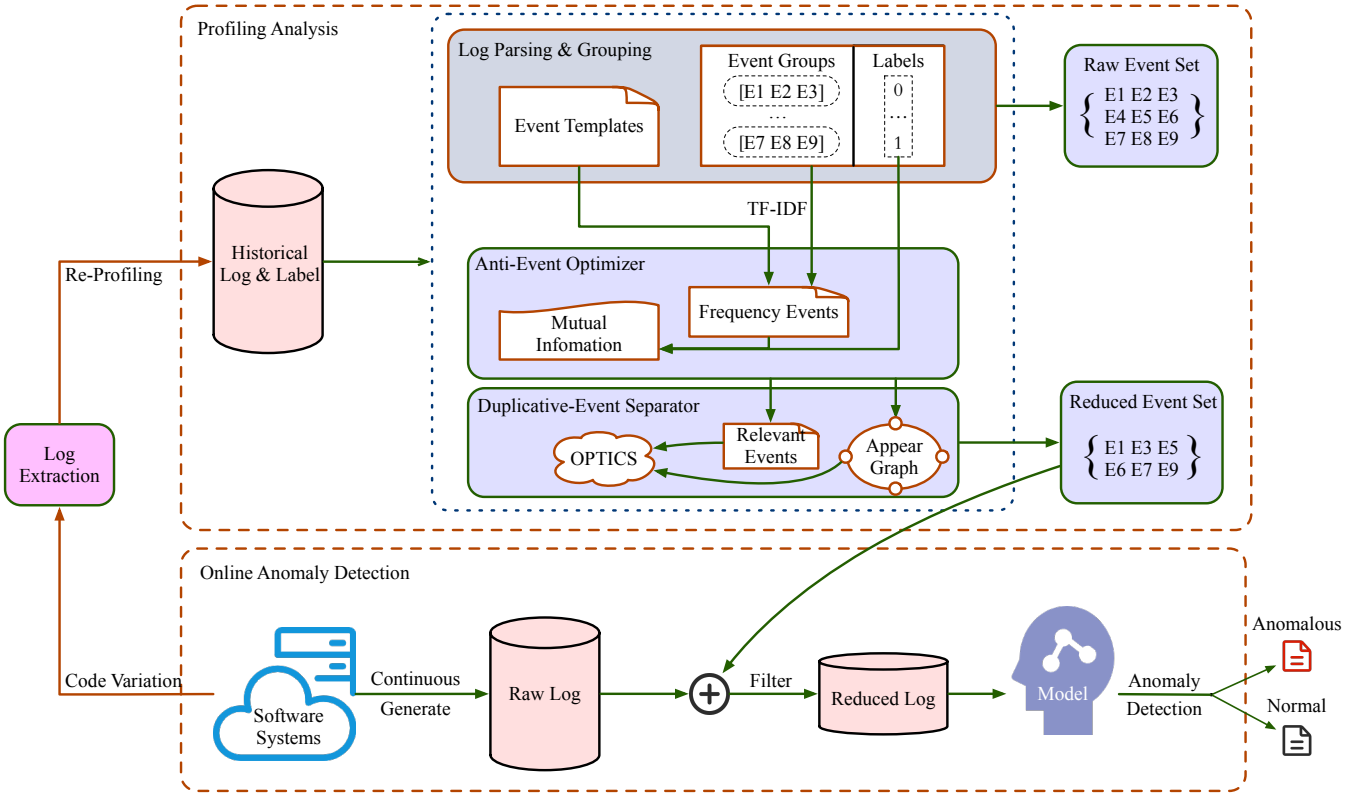


Figure 6: Workflow of LogCleaner

aforementioned appear graph. Within OPTICS, it is believed that for a point to be considered as a core point, the number of log events in its neighborhood should satisfy $min\_samples \leq \theta_{dup}$. For each cluster, LogCleaner retains the event with the highest $\mathrm{MI}(e; l)$ value as the representative event and remove the other events in the cluster. Furthermore, all outlier events are preserved as E_r.

7 Experiment and Evaluation

This section evaluates the overall results, conduct an ablation study of LogCleaner, and assess the influence of hyperparameters.

We perform experiments on the 6 models and 3 datasets previously discussed. Unless otherwise specified, LogCleaner utilizes TF-IDF to filter out events with a frequency below 0.1. The Anti-Event Optimizer's threshold, θ_{anti}, is set to 0, while the threshold θ_{dup} for the Duplicative-Event Separator is set to 2.

7.1 Overall Evaluation Results

For anomaly detection, as illustrated in table 4, the number of reducible events is significant across all datasets. For the Thunderbird and BGL datasets, the events are reduced by approximately 70%. While this doesn't quite match the results from the previous empirical study, LogCleaner operates quickly and the reduced events are applicable across all models.

Furthermore, the performance of each model after applying LogCleaner is analyzed. As illustrated in Table 5, where LC_RobustLog represents the RobustLog model enhanced with LogCleaner, and

Table 4: Data Volume Reduced by LogCleaner

Type \ Dataset	HDFS	BGL	Thunderbird
events	48.28%	73.13%	69.91%
lines	52.62%	24.66%	51.32%

LC_PLELog represents the PLELog model enhanced with LogCleaner. After applying LogCleaner, we observe significantly improved results compared to the original models. LC_RobustLog achieves an increased F1-score of 5.17%, 0.07%, and 30.16% on the HDFS, BGL, and Thunderbird datasets, respectively, in comparison to the RobustLog. Similarly, LC_PLELog achieves an increased F1-score of 7.62%, 1.41%, and 1.61% on the HDFS, BGL, and Thunderbird datasets, respectively, compared to the PLELog. The observed enhancements align with the findings from the previous empirical study.

7.2 Inference Time

The substantial reduction in log events greatly enhances the model's inference speed. Therefore, we conduct experiments to validate the average inference time of the models before and after applying LogCleaner in the context of anomaly detection.

As shown in Table 6, we record the time each anomaly detection model takes to infer the entire test set across various datasets,

Table 5: Evaluation Results on Anomaly Detection Effectiveness

Model		HDFS	BGL	Thunderbird
Isolation Forest	P	82.20%	61.30%	0.49%
	R	74.20%	16.58%	0.10%
	F1	77.99%	26.10%	0.17%
RobustLog	P	98.50%	100.00%	90.96%
	R	88.80%	99.14%	55.90%
	F1	93.40%	99.57%	69.25%
PLELog	P	98.34%	94.30%	96.84%
	R	84.39%	99.82%	99.64%
	F1	90.83%	96.92%	98.22%
LC_RobustLog	P	99.23%	100.00%	100.00%
	R	97.91%	99.27%	98.63%
	F1	98.57%	99.64%	99.31%
LC_PLELog	P	99.81%	99.98%	100.00%
	R	97.13%	96.73%	99.62%
	F1	98.45%	98.33%	99.83%

Table 6: Inference Time Comparison with(out) LogCleaner in Anomaly Detection

Model		HDFS	BGL	Thunderbird
LR	w/o	3371.02	1052.37	11870.16
	w	1809.62	279.39	5313.95
SVM	w/o	3150.26	1051.88	11869.10
	w	1906.75	280.33	5214.05
Decision Tree	w/o	3666.74	1078.35	11870.16
	w	1443.89	264.69	5313.95
Isolation Forest	w/o	12557.61	2343.57	32426.30
	w	10327.30	651.90	12161.74
RobustLog	w/o	37571.02	10282.24	5859.79
	w	14428.25	2881.30	4131.11
PLELog	w/o	10802.95	25306.99	15899.39
	w	7466.62	19303.02	9512.93

measured in milliseconds. Clearly, the inference speed of all models significantly improved after applying LogCleaner, ranging from 21.59% to 307.41%.

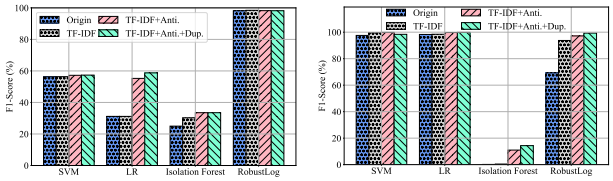
7.3 Effectiveness of Event Reduction Components

To evaluate the effectiveness of each component, we conduct an ablation study for anomaly detection. We assess under various configurations: utilizing TF-IDF alone, combining TF-IDF with the Anti-Events Optimizer (Anti.), and employing both TF-IDF and Anti-Events Optimizer (Anti.) alongside the Duplicative-Events Separator (Dup.). As presented in table 7, each component plays a role in reducing the number of events, with the Duplicative-Events Separatorr (Dup.) having the most pronounced effect.

Table 7: LogCleaner Ablation Experiment on Templates Reduction

TF-IDF	Anti.	Dup.	HDFS	BGL	Thunderbird
			0.0%	0.0%	0.0%
✓			0.0%	21.65%	5.19%
✓	✓		6.90%	44.33%	35.34%
✓	✓	✓	48.28%	73.13%	69.91%

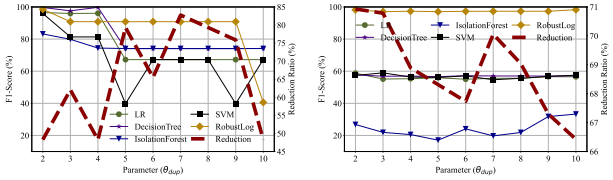
We also validate the performance changes of models on the BGL and Thunderbird datasets. As illustrated in figure 7, in some instances, the model's performance remains unaffected with the addition of more components. However, for models such as LR and Isolation Forest in the BGL dataset and RobustLog in the Thunderbird dataset, the effectiveness of the models increases with the addition of components. Notably, the Anti-Event Optimizer (Anti.) plays the most significant role in this improvement.



(a) BGL (b) Thunderbird

Figure 7: Ablation Experiment of F1-score

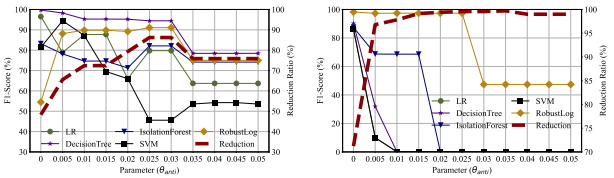
7.4 Influence of Hyperparameters



(a) HDFS (b) BGL

Figure 8: Evaluation Results by Varying θ_{dep}

To verify whether LogCleaner have chosen the optimal hyperparameters, experiments by varying the hyperparameters are carried out. As depicted in figure 8, it's evident that while in HDFS, as θ_{dep} increases, the number of event reductions rises, the model's performance diminishes. Conversely, in BGL, as θ_{dep} increases, the model's performance remains consistent, but the number of event reductions drastically decreases. Thus, a setting of $\theta_{dep} = 2$ is a relatively optimal parameter.



(a) HDFS (b) BGL

Figure 9: Evaluation Results by Varying θ_{anti}

We also conduct experiments related to θ_{anti}. As illustrated in figure 9, for both HDFS and BGL datasets, as θ_{anti} increases, event reduction grows. However, the performance of various models also deteriorates. Thus, to ensure optimal model effectiveness, it is prudent to set $\theta_{anti} = 0$.

8 Related Work

8.1 Log-based Anomaly Detection

Log analysis for anomaly detection is a well-established research area[4, 7, 8, 13, 19, 20, 29, 33, 40, 49]. 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, LogFlash[20] utilizes a real-time streaming process for log transitions, enhancing the speed of anomaly detection. HiLog[19] performs an empirical study on four anti-patterns that challenge the assumptions underlying anomaly detection models, proposing a human-in-the-loop approach to integrate human expertise into log-based anomaly detection.

Deep-learning models, conversely, use various neural networks to model sequences of log events. LogRobust[49] 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[29] employs a token encoder and a context encoder to learn patterns from log tokens and their adjacent contexts.

8.2 Log Compression & Placement

Given the substantial volume of logs generated by modern systems, assisting developers in adding appropriate logging statements is a promising research area, as highlighted in prior studies[43, 45, 46, 50]. Errlog[45], LogEnhancer[46], and Log20[50] enhance debugging capabilities by strategically inserting supplementary logging statements into the source code. Concurrently, LogReducer[43] leverages eBPF to manage logging overhead in performance-critical areas, ensuring that logging remains effective.

However, the process of archiving massive volumes of logs over extended periods can introduce substantial storage overhead. To address this challenge, several studies have focused on log compression techniques to reduce storage requirements. Approaches such as Nanolog[41], CLP[36], and Cowic[27] construct dictionaries for fields in logs and replace strings by referencing these dictionaries. Additionally, LogZip[28] and RoughLogs[32] employ sophisticated statistical models to identify and reduce redundancy in logs.

9 Discussion

9.1 Application of LogCleaner

LogCleaner has been implemented for a subset of users in Apache IoTDB, yielding positive feedback. Beyond its application as described in this paper, some users have employed LogCleaner to identify key events and alert developers about unnecessary print statements in the logs that can be removed. Developers can selectively delete these prints to enhance system performance. It has aided developers in discovering that over 50% of the log print statements in the system are unnecessary. As a result, the performance of Apache IoTDB has improved by approximately 8%.

9.2 Threats to Validity

The major threats to the validity can be identified as following.

Limited models. In the empirical study, we mainly evaluate six representative models that have publicly available source code. In the future, we plan to re-implement more log-based detection models that have not released their source code, based on the descriptions provided in their papers. Subsequently, a larger-scale evaluation will be conduct.

Implementation. We primarily utilize publicly available implementations of the studied models. The implementation of LogCleaner is also based on popular libraries, and three authors have thoroughly reviewed the source code to ensure accuracy and reliability.

Limited datasets. The experiments are conducted on three log datasets. While they are widely used in existing studies on log-based anomaly detection, they may not fully represent all characteristics of log data. In future research, we plan to conduct experiments on additional datasets to cover a broader range of real-world scenarios.

10 Conclusion and Future Work

In this paper, we examine event reduction's effect on log-based anomaly detection models. Through empirical study on six models across three datasets, we identify three distinctive log event types that impact model performance differently. Based on these findings, we propose LogCleaner: an efficient methodology for the automatic reduction of log events in the context of anomaly detection. Serving as middleware between software systems and models, LogCleaner continuously updates and filters *anti-events* and *duplicative-events* in the raw generated logs. This approach not only accelerates the model's inference speed but also enhances the effectiveness of model classification.

In future research, we intend to leverage reinforcement learning to enhance the efficacy of log reduction. Furthermore, we also aspire to integrate LLM to pinpoint key events.

Acknowledgment

This work was supported by the PKU-ZTE Cooperation Research Project.

References

[1] Anton Babenko, Leonardo Mariani, and Fabrizio Pastore. 2009. AVA: automated interpretation of dynamically detected anomalies. In *Proceedings of the eighteenth international symposium on Software testing and analysis*. 237–248.

[2] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B Woodard, and Hans Andersen. 2010. Fingerprinting the datacenter: automated classification of performance crises. In *Proceedings of the 5th European conference on Computer systems*. 111–124.

[3] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE, 36–43.

[4] Rui Chen, Shenglin Zhang, Dongwen Li, Yuzhe Zhang, Fangrui Guo, Weibin Meng, Dan Pei, Yuzhi Zhang, Xu Chen, and Yuqing Liu. 2020. Logtransfer: Cross-system log anomaly detection for software systems with transfer learning. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 37–47.

[5] Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient Log Parsing Using n n-Gram Dictionaries. *IEEE Transactions on Software Engineering* 48, 3 (2020), 879–892.

[6] donglee afar. 2023. logdeep. https://github.com/donglee-afar/logdeep

[7] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[8] Chiming Duan, Tong Jia, Ying Li, and Gang Huang. 2023. AcLog: An Approach to Detecting Anomalies from System Logs with Active Learning. In *Proceedings of the 27th IEEE International Conference on Web Services*. 1021–1030.

[9] Stephen Elliot. 2014. DevOps and the cost of downtime: Fortune 1000 best practice metrics quantified. *International Data Corporation (IDC)* (2014).

[10] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Mikko Terho, and Jelena Vlasenko. 2013. Failure prediction based on log files using random indexing and support vector machines. *Journal of Systems and Software* 86, 1 (2013), 2–11.

[11] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks (IJCNN)*. IEEE, 1–8.

[12] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1573–1582.

[13] Xiao Han and Shuhan Yuan. 2021. Unsupervised cross-system log anomaly detection via domain adaptation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3068–3072.

[14] Sahand Hariri, Matias Carrasco Kind, and Robert J Brunner. 2019. Extended isolation forest. *IEEE transactions on knowledge and data engineering* 33, 4 (2019), 1479–1489.

[15] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. 2016. An evaluation study on log parsing and its use in log mining. In *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 654–661.

[16] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.

[17] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 207–218.

[18] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2023. Loghub: A large collection of system log datasets towards automated log analytics. (2023).

[19] Tong Jia, Ying Li, Yong Yang, Gang Huang, and Zhonghai Wu. 2022. Augmenting Log-based Anomaly Detection Models to Reduce False Anomalies with Human Feedback. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3081–3089.

[20] Tong Jia, Yifan Wu, Chuanjia Hou, and Ying Li. 2021. LogFlash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 80–90.

[21] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. 2008. Abstracting execution logs to execution events for enterprise applications (short paper). In *2008 The Eighth International Conference on Quality Software*. IEEE, 181–186.

[22] Jinhan Kim, Valeriy Savchenko, Kihyuck Shin, Konstantin Sorokin, Hyunseok Jeon, Georgiy Pankratenko, Sergey Markov, and Chul-Joo Kim. 2020. Automatic abnormal log detection by analyzing log history for providing debugging insight. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. 71–80.

[23] Max Landauer, Sebastian Onder, Florian Skopik, and Markus Wurzenberger. 2023. Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications* 12 (2023), 100470.

[24] Van-Hoang Le and Hongyu Zhang. 2022. Log-based anomaly detection with deep learning: How far are we?. In *Proceedings of the 44th international conference on software engineering*. 1356–1367.

[25] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2022. SwissLog: Robust anomaly detection and localization for interleaved unstructured logs. *IEEE Transactions on Dependable and Secure Computing* (2022).

[26] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure prediction in ibm bluegene/l event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 583–588.

[27] Hao Lin, Jingyu Zhou, Bin Yao, Minyi Guo, and Jie Li. 2015. Cowic: A column-wise independent compression for log stream analysis. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 21–30.

[28] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R Lyu. 2019. Logzip: Extracting hidden structures via iterative clustering for log compression. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 863–873.

[29] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, et al. 2022. Uniparser: A unified log parser for heterogeneous log data. In *Proceedings of the ACM Web Conference 2022*. 1893–1901.

[30] Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. 2018. Detecting anomaly in big data system logs using convolutional neural network. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 151–158.

[31] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. 2009. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1255–1264.

[32] Michael Meinig, Peter Tröger, and Christoph Meinel. 2019. Rough Logs: A Data Reduction Approach for Log Files.. In *ICEIS (2)*. 295–302.

[33] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.. In *IJCAI*, Vol. 19. 4739–4745.

[34] Meiyappan Nagappan and Mladen A Vouk. 2010. Abstracting log lines to log event types for mining software system logs. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 114–117.

[35] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2021. Self-supervised log parsing. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part IV*. Springer, 122–138.

[36] Kirk Rodrigues, Yu Luo, and Ding Yuan. 2021. {CLP}: Efficient and Scalable Search on Compressed Text Logs. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 183–198.

[37] Issam Sedki, Abdelwahab Hamou-Lhadj, Otmane Ait-Mohamed, and Mohammed A Shehab. 2022. An Effective Approach for Parsing Large Log Files. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 1–12.

[38] Keiichi Shima. 2016. Length matters: Clustering system log messages using length of words. *arXiv preprint arXiv:1611.03213* (2016).

[39] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 785–794.

[40] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1448–1460.

[41] Stephen Yang, Seo Jin Park, and John Ousterhout. 2018. {NanoLog}: A Nanosecond Scale Logging System. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 335–350.

[42] Kun Yin, Meng Yan, Ling Xu, Zhou Xu, Zhao Li, Dan Yang, and Xiaohong Zhang. 2020. Improving log-based anomaly detection with component-aware analysis. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 667–671.

[43] Guangba Yu, Pengfei Chen, Pairui Li, Tianjun Weng, Haibing Zheng, Yuetang Deng, and Zibin Zheng. 2023. LogReducer: Identify and Reduce Log Hotspots in Kernel on the Fly. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1763–1775.

[44] Siyu Yu, Pinjia He, Ningjiang Chen, and Yifan Wu. 2023. Brain: Log Parsing with Bidirectional Parallel Tree. *IEEE Transactions on Services Computing* (2023).

[45] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. 2012. Be conservative: Enhancing failure diagnosis with proactive logging. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 293–306.

[46] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. 2012. Improving software diagnosability via log enhancement. *ACM Transactions on*

Computer Systems (TOCS) 30, 1 (2012), 1–28.

[47] Lingzhe Zhang, Tong Jia, Mengxi Jia, Ying Li, Yong Yang, and Zhonghai Wu. 2024. Multivariate Log-based Anomaly Detection for Distributed Database. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4256–4267.

[48] Lingzhe Zhang, Tong Jia, Mengxi Jia, Yong Yang, Zhonghai Wu, and Ying Li. 2024. A Survey of AIOps for Failure Management in the Era of Large Language Models. *arXiv preprint arXiv:2406.11213* (2024).

[49] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[50] Xu Zhao, Kirk Rodrigues, Yu Luo, Michael Stumm, Ding Yuan, and Yuanyuan Zhou. 2017. Log20: Fully automated optimal placement of log printing statements under specified overhead threshold. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 565–581.

[51] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.