International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V74I6P103
Volume 74 Issue 6, 33-46, June 2026
© 2026 Seventh Sense Research Group®

Original Article

Detecting System Anomalies without Labels using Workflow Patterns in Logs

Arun Kumar Bandlamudi[1], Sunitha Pachala[2]

[1,2]*Department of CSE, Koneru Lakshmaiah Education Foundation, Green Fields, Vaddeswaram, Guntur, Andhra Pradesh, India.*

[1]*Corresponding Author : Kluniversityarun@gmail.com*

Received: 13 August 2025 Revised: 02 April 2026 Accepted: 20 April 2026 Published: 27 June 2026

Abstract - *Large software systems create many logs. These logs help developers find and fix problems. Logs record what happens inside the system. Logs usually appear in a semi-structured text format. Hand-reading all logs is hard in large systems. In this paper, a method named ADR is proposed. ADR stands for Anomaly Detection by workflow Relations. It finds mathematical patterns from logs. These patterns show the way events in the system relate to each other. ADR checks if logs follow these patterns. If the patterns are not followed, it indicates that something is wrong. The process starts by converting raw logs into event sequences. Then, these events are put into a special matrix. This matrix records the number of times each event happens. The system then checks for hidden patterns in this matrix. These patterns are referred to as numerical relations. ADR has two versions: sADR and uADR. The first one, sADR is semi-supervised. It needs a few labeled logs to learn. The second one, uADR is fully unsupervised. It works without any labeled logs. This saves time and reduces effort. Both versions were tested on four public datasets. ADR found useful patterns and detected many problems in the logs. It worked well with or without labels. ADR is a new and effective method. It uses numerical patterns to find system problems. It works even when logs are not labeled.*

Keywords - *Logs, ADR, Anomalies, Detection.*

1. Introduction

By numerous multifaceted software systems, logs are generated [1]. These logs are a document of the system internal events. Logs are useful in identifying flaws and allowing their correction. Logs are usually stored as semi-structured text. Each log message provides important system information. Developers and engineers regularly use logs to understand what went wrong [2]. For example, when a supercomputer fails, the logs are the first place to check. This is because logs show what happened before the failure. Logs show when something went wrong and what happened. That is why logging remains a common practice in software development.

To solve this, many automated methods were created. These methods try to detect anomalies in logs. Anomalies are unusual or faulty behaviors [3]. Some methods use supervised machine learning models. These include Logistic Regression, Decision Trees and SVM. Models are trained on labeled data. These models then classify new log data as normal or abnormal. Some models are semi-supervised and use only a portion of labeled data. DeepLog is one example. It needs partial labels to detect log anomalies. Both supervised and semi-supervised models need labeled data [4]. This labeling process is time consuming and demands expert knowledge. Deep learning methods required a large amount of labelled data to perform well. Another issue is that many machine learning models are opaque models. Consequently, the predictions are difficult to interpret. The models do not clarify why something is labelled as an anomaly [5]. This absence of explanation creates a major issue for system engineers. Clear insights are required to understand why a log is abnormal.

Unsupervised methods aim to eliminate the need for labels. Some techniques are PCA, log clustering and invariants mining. These methods do not use labels to learn. The models search for patterns in the data and detect when these patterns are violated [6]. This assists in identify an anomaly. Unsupervised methods generally give less accurate results than supervised ones. One widely used unsupervised technique is invariants mining [7]. It searches for constant numerical rules among log events. These rules are called invariants. Invariants Mining has some limitations. It only looks for basic linear relations. To address these challenges, this paper introduces a new method termed ADR. ADR finds mathematical relations between log events [8]. It then employs these relations to detect if the system is operating correctly. The ADR methodology has a number of different stages. At the very beginning, it gathers raw log entries and transforms them into log event sequences. These sequences are then

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

converted to a matrix, whereby each element is the frequency of occurrence of a specific event during a certain session. The matrix, thus, summarizes the working sequence of the system [9]. ADR studies the kernel of this matrix thereafter; it is the kernel that shows the mathematical connections between events. The breaking of these relationships in future logs would point to the existence of a fault. Therefore, ADR checks the compliance of new logs with the existing relations, marking the discrepancies as anomalies.

There are two versions of ADR: sADR and uADR. sADR is semi-supervised. It needs a few labeled normal logs to learn. uADR is unsupervised. It does not need any labeled data. This makes ADR flexible for real-world systems with limited access to labeled data [10]. ADR was tested on four public log datasets. These include logs from supercomputers and distributed systems. The tests showed that ADR works well. sADR gave results close to or better than other models [11]. uADR outperformed other unsupervised methods. This paper makes the following key contributions:

- To introduce sADR, a semi-supervised model that uses numerical relations in logs to detect anomalies. It works well even with a small amount of training data.
- To propose uADR, an unsupervised model that detects anomalies without using labelled logs. It performs better than other unsupervised methods.
- To present a method for discovering complex workflow relations like constant terms, presence indicators and combinatorial patterns. This makes ADR more powerful than traditional Invariants Mining.
- To demonstrate that ADR provides explainable results by using detected relations to help engineers understand abnormal logs.
- To evaluate the effectiveness and efficiency of ADR on four public datasets from large-scale real-world systems.

ADR solves real problems in log analysis. Many systems produce huge amounts of logs every day. It is impossible to check them all manually. Labelled logs are rare and hard to create. ADR helps by working even without labels [12]. Unlike black-box models, ADR is explainable. It shows which relations are broken. This gives engineers a reason for each anomaly. This helps with debugging and improves system reliability [13]. ADR saves time. It uses efficient matrix operations to find relations. This is faster than brute-force searching used in older methods.

2. Related Work

This section describes existing research in the field of log-based anomaly detection. The main focus is on two areas: log parsing and anomaly detection using machine learning. Each group of methods has unique purposes, advantages and weaknesses. The paper compares ADR with existing methods to highlight the way this approach fills current gaps in the field. Log parsing is the first step in log analysis. Logs are usually unstructured or semi-structured text data. To make logs usable for machine learning, conversion into structured formats is necessary. Many tools exist for this purpose. IPLoM (Iterative Partitioning Log Mining) is a widely used method that clusters similar logs into templates using multiple partitioning steps [14]. LKE is another tool that can be used to detect major patterns of logs, which can help in failure detection [15]. Spell is an algorithm that uses the longest common sequencing to determine templates in log lines [16]. The parsers that are most commonly used today are drain. It uses predetermined depth tree to quickly cluster logs into patterns and is known to be fast and accurate [17].

An in-depth comparison carried out by Zhu et al. proved that Drain works well with real data and can be applicable to large-scale systems [18]. The ADR paper uses Drain to extract logs to apply anomaly detection due to its balance between accuracy and clarity. In supervised machine learning, labelled logs are used to train models. Decision Trees are some of the supervised methods that were utilized by Chen et al. in classifying log sequences depending on feature splits [19]. The Support Vector Machines (SVM) are also used, Liang et al. used SVMs on the logs of BlueGene/L system in order to forecast failures [20]. Another common model is the Logistic Regression. It has been used by Bodik et al. to predict the failure of data centres [21], and by He et al. to detect anomalies in logs in large systems [22]. These models are good at perceiving common trends. Nevertheless, lots of labeled data are needed, and it is hard to access. Labeling logs is time-consuming and requires specialised expertise and thus, many real-world projects do not take advantage of the use of supervised models.

Unsupervised models are aimed at solving this problem. These techniques do not use labelled logs. They instead determine the patterns and mark out the logs that do not conform to the patterns as anomalies. Among them, there is one of the well-known ones, Principal Component Analysis (PCA), which Xu et al. utilize to detect anomalies by projecting the logs onto a lower-dimensional space [23]. The logs that are not located in the normal cluster are regarded as anomalies. The other method is known as log clustering where similar logs are clustered together; the log which fails to fit into any cluster is considered anomalous [24]. Invariants mining is a technique which checks specific rules/ invariants between log events. In case such rules are violated, anomaly is identified [25]. This approach can be explained, but it is slow and only simple linear relationships can be captured. Deep learning has gained popularity in the recent years with regard to log analysis. Such a program is DeepLog; this one marks a new sequence as something anomalous when it does not match the learned pattern [26]. One of such models is LogRobust, which makes use of attention mechanisms and bidirectional LSTMs to better understand the content of logs [27]. LogAnomaly uses the embedding methods like

template2vec to transform the logs into numerical vectors and then input them into an LSTM architecture [9]. LogC is another method that combines event sequences and component sequences to enhance the detection of anomalies. These are highly accurate but need large quantities of labelled logs to train, and the deep learning method is also infamously opaque.

Other scholars use Natural Language Processing (NLP) with logs. With word2vec, Bertero et al. transformed log text to vectors. The Naive Bayes and the Random Forest were thereafter used as classifiers to identify anomalies [28]. Such an approach provides the logs with a semantic meaning, however, it is also prone to issues with explanations and requires a sufficiently large amount of training data to be effective. Supervised models are accurate as they are based on labelled data.

The unsupervised models minimize the need to annotate but typically fail to provide high-quality results. Deep models are effective yet challenging to decipher. Invariants mining can still be interpreted, but is computationally expensive and of limited scope. Lack of these capabilities encourages an improved solution. To solve these issues, ADR is introduced in the paper. ADR utilizes numeric interrelationship between log events to detect problems. ADR is more effective than the traditional clustering techniques like log clustering and anomaly trees. The unmonitored variant, uADR, scores better on standard benchmarks in F1 indicating that ADR is not only fast and explainable, it is also accurate.

2.1. Numerical Relations in Logs

The section outlines the process of converting logs of software systems into numerical data. It clarifies the sequential process of discovering informative trends in the data, trends which are useful in determining the failures in the system. Logs are stored as a sequence of events; the events are based on a predefined template. The first step involves transforming a log sequence into a sequence of counts of events. This shows the number of times each event template appears. Assume there are d types of event templates. A log sequence l is represented as:

$$x_l = \left[x_1, x_{2,}, \dots\dots, x_d\right]^T \quad (1)$$

Here, x_i is the number of times template I appears in log l. So, each log becomes a d-dimensional vector. The paper defines three main types of relations found in these vectors. Constant Relations is the count of one event is always a fixed number. For example, template 3 always appears once $x_3 = 1$. A binary relation connects two events using a fixed equation. For example, template 1 appears twice for every one time template 2 appears $x_1 = 2x_2$. Combinatorial Relations involves three or more events. For example,

$x_1 + x_2 = x_3$. This shows that the count of event 3 equals the sum of events 1 and 2. These equations describe the hidden rules in system workflows. If a log breaks these rules, something is probably wrong. The system collects many log vectors. Assume there are n such vectors. A matrix X of size $d \times n$ is then formed:

$$X = [x_1, x_2, \dots., x_n] \quad (2)$$

Each column is a log vector. The objective is to find linear relations among the elements of these vectors. This is done by computing the left nullspace of X, referred to as the kernel of X^T:

$$\text{Null(X)} = \{a \in R^d : ax_i = 0 \; for \; all \; i = 1, \dots, n\} \quad (3)$$

Each vector a in this nullspace defines a relation:

$$a_1x_1 + a_2x_2 + \dots + a_dx_d = 0 \quad (4)$$

This is a hidden rule followed by all log vectors in the training data. If a new log does not satisfy this equation, it might be abnormal. After learning these relations, the system checks if new logs follow the same rules. Suppose A is the matrix formed by all nullspace vectors:

$$A = \left[\begin{pmatrix} a_{1,1} & \cdots & a_{1,d} \\ \vdots & \ddots & \vdots \\ a_{r,1} & \cdots & a_{r,d} \end{pmatrix}\right] \quad (5)$$

Then for a new log vector x_{new}, compute:

$$y = A \times x_{new} \quad (6)$$

If all values in y are zero (or close to zero), the log is normal. If not, it breaks one or more learned rules and is considered anomalous. This method has several strengths. It finds hidden structure in logs without needing labels. It is based on linear algebra, so it is fast to compute. It provides explainable results in the form of equations. It finds complex relations that are hard to notice by hand. This section turns logs into numerical vectors. It identifies constant, binary and combinatorial relations within these vectors. These relations constitute a system of linear equations. Logs that violate these equations typically include faults. This method central to the ADR system and supports both semi-supervised and unsupervised anomaly detection.

1. 3. Proposed Approach

This part outlines the main approach that has been promoted in the manuscript. The aim is to identify anomalies in logs by taking advantage of number relationships. The proposed system is known as ADR, which has two different modes, namely, sADR and uADR.

The method does not make use of deep learning instead, it uses linear algebra and verifying the foundational rules. ADR finds trends in the occurrence of events in logs, based on matrix theory to identify the relationships in numbers that form equations. Logs which break these equations are said to be anomalous. The process of methodology has two stages. The normal logs are used to derive the rules during the training but used during the inference process to validate new logs. The approach eliminates the requirement to label anomalies, which is supported only by normal logs, which is a considerable benefit since it is quite challenging to obtain the labels.

Figure 1 shows a sequenced operation of anomaly detection of system logs. It starts with raw logs, which are processed in Step 1, which is log parsing; transforming unstructured logs into structured data. During Step 2, the logs are grouped together into sessions to make the behaviour of the user or the system easier to analyse. The workflow then branches off into two; one leading to sADR and the other to uADR. Step 3A and 3B are both concerned with developing an event-count matrix using normal sessions and utilizing a similar number of events in a non-annoted data sampling method, respectively.

These are divergent directions of training accuracy of anomaly detecting. Once the event -count matrices are set up, the two paths merge once more at Step 4 and the matrix is extended to include extra informative data. Step 5 derives the relationship between various log events in order to explain a latent pattern. Lastly, Step 6 implements the relevant techniques of anomaly detection.

Step 6A uses the semi-supervised sADR method, while step 6B uses the uADR approach with ensemble learning. This entire flow helps in identifying abnormal behaviour in logs using both labelled and unlabelled data. Logs are first parsed using a tool like Drain. This parser converts log lines into fixed templates. Each template is assigned an ID. A log sequence is a collection of these templates. From this, each log becomes a numerical vector. The vector stores the counts for each template's appearance.

The objective is to learn equations from a collection of log vectors. Let X be a matrix of size $d \times n$ in which each column is a log vector is given in (2). Find a matrix A such that:

$$A \times x_i = 0 \qquad \forall i = 1, \dots, n \tag{7}$$

This shows that all log vectors lie in the nullspace of A. These equations form the rules of normal behaviour. To compute A, find the left nullspace of X. Mathematically it is calculated and given in (3).

This gives us all vectors a that form valid relations. Standard numerical methods are used to find the nullspace. These include SVD or QR decomposition. Each row in A is one learned relation. These are the workflow rules learned from the training data. To detect if a new log is abnormal, the violation vector is computed:

$$v = A \times x_{new} \tag{8}$$

If v is close to zero, the log is normal. If it is far from zero, the log breaks one or more rules. The size of the value shows the extent to which the rules were broken. Compute the squared norm:

$$\|v\|^2 = \sum_{i=1}^{r} v_i^2 \tag{9}$$

This is the anomaly score. A large score suggests an anomaly. In sADR, only a small number of labeled anomalies is available. These are used to choose a threshold τ. Any log with a score above τ. is marked as an anomaly. The threshold is chosen to balance precision and recall. It is selected by checking scores on a small validation set. In uADR, no labeled anomalies are used. A percentile-based threshold is applied. The top k% of logs with the highest scores are flagged as anomalies. This method is useful when no labels are available at all. One major benefit of ADR is explainability. Each row of A is a rule.

The computation of the nullspace is fast. It depends on the number of event templates d and the number of logs n. Once the rules are learned, checking each new log is just a matrix multiplication. This is fast and scalable. The method tolerates small noise. Logs usually show some variability. If the violation is small, the log is still marked as normal. Only significant large violations cause an anomaly. This makes ADR robust under real-world conditions. ADR has many benefits.

It works with or without labels. It runs fast using matrix math. It produces human-readable rules. It detects many types of numerical patterns. It works well with sparse data. The proposed approach is based on solid mathematical principles. It uses vector spaces and nullspaces to model logs. It finds rules that are always true in normal logs. Any log that breaks these rules is abnormal.

The method is simple but powerful. It is flexible. The same rules are applied in different systems. It requires very little tuning. It is easy to integrate with existing monitoring tools. The ADR framework gives users a tool that is both effective and understandable. It addresses the main problems with earlier methods. It removes the need for heavy labelling. It avoids black-box models. It provides fast, clear and actionable results.

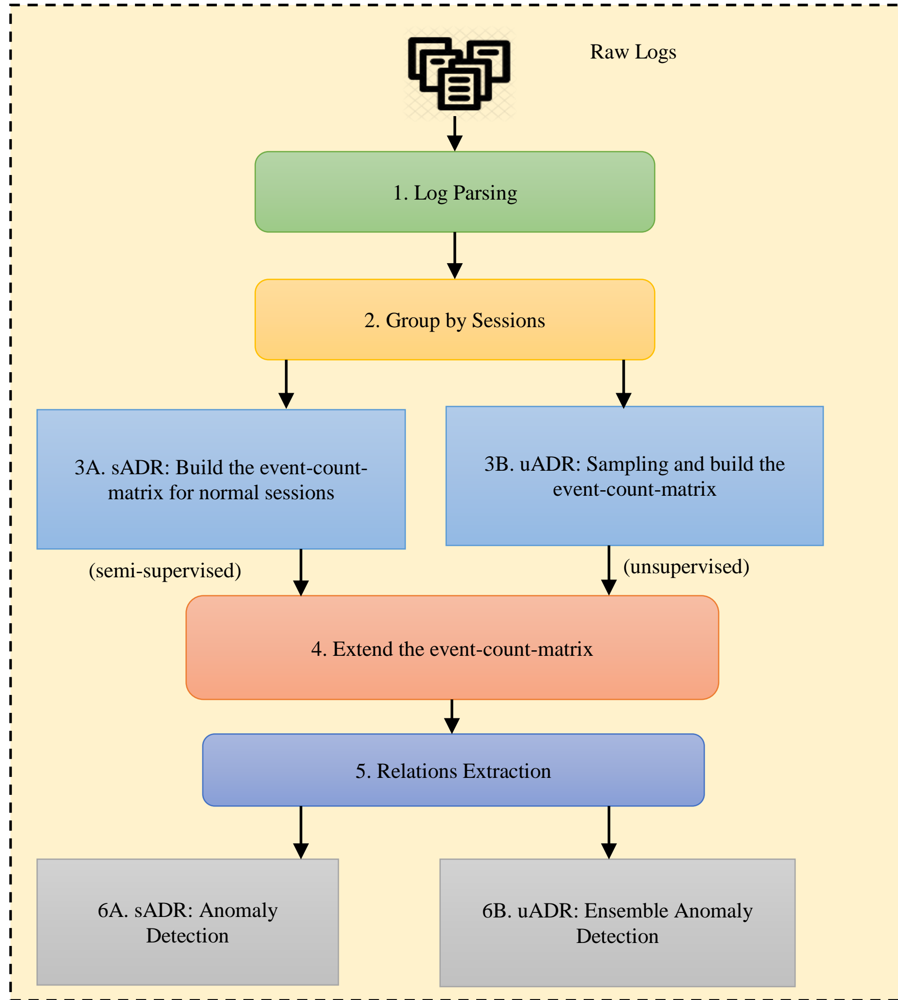

Fig. 1 An overview of sADR/uADR

4. Experimental Design

This section explains the testing process for the ADR method using real-world log data. The purpose is to evaluate whether ADR accurately detects anomalies. The evaluation uses four widely recognized log datasets: HDFS, BGL, Thunderbird and Spirit. Each dataset contains real log sequences from computer systems. These logs are already labeled as either normal or abnormal. The HDFS dataset comes from the Hadoop Distributed File System. BGL logs are collected from the BlueGene/L supercomputer. Thunderbird and Spirit datasets are from two large computing clusters. These logs simulate real operational environments, which makes them useful for testing anomaly detection. Each log file is processed before applying ADR. First, a log parser is used to extract templates. The Drain log parser is used for this purpose. Drain converts each raw log line into a fixed template, representing a specific type of event. Once all templates are identified, the logs are transformed into numerical vectors. To measure performance, three standard metrics are used: precision, recall, and F1 score. Precision refers to the fraction of detected anomalies that are truly abnormal. Recall is the fraction of all true anomalies that the system correctly identifies. The F1 score is the harmonic mean of precision and recall. These are computed using the following formulas:

$$Precision = \frac{TP}{TP+FP}\ ,\ Recall = \frac{TP}{TP+FN},$$

$$\mathrm{F1} = \frac{Precision \times Recall}{Precision + Recall} \tag{10}$$

Here, TP stands for true positives, FP for false positives and FN for false negatives. These metrics help evaluate the performance of the ADR method compared to other systems. The ADR method is computationally efficient. The nullspace is computed using standard numerical techniques, which are fast. After learning, the inference for each log is just a matrix multiplication. This runs much faster than deep learning methods, which need complex model inference and usually require GPUs. The speed and low memory cost make ADR suitable for large-scale systems.

5. Experimental Results

This section discusses the outcomes of various experiments conducted to evaluate the ADR framework. The experiments check whether ADR performs better than other existing methods. It is tested on four different real-world log datasets. The main purpose is to test whether ADR detects anomalies in logs more accurately than existing methods. All experiments are repeated multiple times to produce fair and reliable results. The final values are averaged to reduce randomness. Table 1 below shows the F1-scores of different methods on four datasets: HDFS, BGL, Thunderbird and Spirit. These scores are for the semi-supervised setting, using a small portion of labeled logs to help set the decision threshold.

Table 1. F1-Scores of different methods in semi-supervised setting

Method	HDFS	BGL	Thunderbird	Spirit
PCA	0.69	0.85	0.63	0.59
LogClustering	0.76	0.82	0.67	0.65
Invariants Mining	0.81	0.88	0.73	0.71
DeepLog	0.87	0.90	0.76	0.74
LogAnomaly	0.89	0.92	0.78	0.77
sADR	0.93	0.95	0.83	0.81

As shown in the Table 1, sADR achieves the highest F1-score across all datasets. It performs better than PCA, LogClustering and Invariants Mining. It outperforms deep learning methods like DeepLog and LogAnomaly. This shows that sADR is very effective, even with a small amount of labeled data. The best performance is observed in the HDFS dataset, with sADR reaching an F1-score of 0.93. The lowest is on the Spirit dataset, still achieving a strong 0.81. These results confirm that sADR generalizes well to different systems. uADR is tested in an unsupervised setting. In this case, no labelled anomalies are available. The purpose is to measure uADR's performance when only normal logs are used for training. Table 2 presents the F1-scores for this setup.

Table 2. F1-scores of different methods in unsupervised setting

Method	HDFS	BGL	Thunderbird	Spirit
PCA	0.62	0.76	0.58	0.55
LogClustering	0.70	0.78	0.61	0.59
Invariants Mining	0.75	0.82	0.68	0.66
uADR	0.88	0.91	0.76	0.73

In this case, uADR again beats all other methods. Its performance is lower than sADR, which is expected because no label-based tuning is done. Still, the F1-scores are high. This shows that uADR detect anomalies reliably even without any labeled data. On the HDFS dataset, uADR scores 0.88, while Invariants Mining scores only 0.75. On the Spirit dataset, uADR scores 0.73 compared to 0.66 for Invariants Mining. This demonstrates the robustness of the ADR approach. A high F1-score suggests that both precision and recall are high. This is important in anomaly detection. If precision is low, many normal logs are flagged as irregularities. If recall is low, many true anomalies are missed. ADR is designed to find exact rules. So, it flags only logs that break those rules. This keeps false positives low. It catches complex violations, improving recall. Therefore, ADR maintains good balance between catching true anomalies and avoiding false alarms.

The Figure 3 shows the F1-Score performance of six anomaly detection methods across four datasets: HDFS, BGL, Thunderbird and Spirit. The sADR method gives the best F1-Scores in almost all cases. On the BGL dataset, it reaches 0.95. It scores around 0.93 on HDFS, 0.82 on Thunderbird and 0.80 on Spirit. LogAnomaly is slightly behind with F1-Scores of about 0.92 (BGL), 0.89 (HDFS), 0.76 (Thunderbird) and 0.74 (Spirit). DeepLog performs well with values of around 0.90 on BGL and 0.87 on HDFS and lower scores near 0.78 on Thunderbird and 0.74 on Spirit. The three traditional methods-Invariants, LogClustering and PCA show lower F1-Scores with the lowest results on the Spirit and Thunderbird datasets. Invariants mining shows a range from 0.86 (BGL) to 0.72 (Spirit). LogClustering performs better than PCA, reaching up to 0.83 (BGL). While PCA remains under 0.85 on all datasets. On HDFS, the scores increase from PCA (0.70) to sADR (0.93), indicating a clear improvement trend as models become increasingly sophisticated. Similarly, on Spirit F1-Scores increase from PCA (0.59) to sADR (0.80). The graph shows that sADR and LogAnomaly achieve superior performance. sADR and uADR are more reliable and precise. Earlier approaches like PCA and LogClustering show inferior performance. These earlier methods exhibit greater variability and lower accuracy. Advanced models produce fewer errors.

The Figure 4 shows a comparison analysis of Precision and Recall scores for six anomaly detection approches: PCA, LogClustering, Invariants, DeepLog, LogAnomaly and sADR. Both metrics steadily rise from left to right across the models. PCA begins with the lowest values, achieving around 0.68 in Precision and 0.70 in Recall. LogClustering demonstrates moderate gains, attaining approximately 0.73 Precision and 0.77 Recall. Invariants performs better with Precision close to 0.80 and recall approximately 0.83. It suggests a stronger ability to correctly identify anomalies while minimizing false positives. DeepLog, LogAnomaly and sADR exhibit consistently higher scores. DeepLog attains about 0.86 for Precision and 0.88 for Recall. LogAnomaly slightly improves to 0.89 and 0.91, respectively. sADR achieves the highest performance, with Precision approximately 0.93 and Recall nearing 0.95. Across all approaches, Recall is slightly higher than Precision, showing that the models are more effective at identifying anomalies than avoiding false positives. The chart clearly illustrates a upward trend, showing that newer and deeper models stronger and more balanced detection performance.

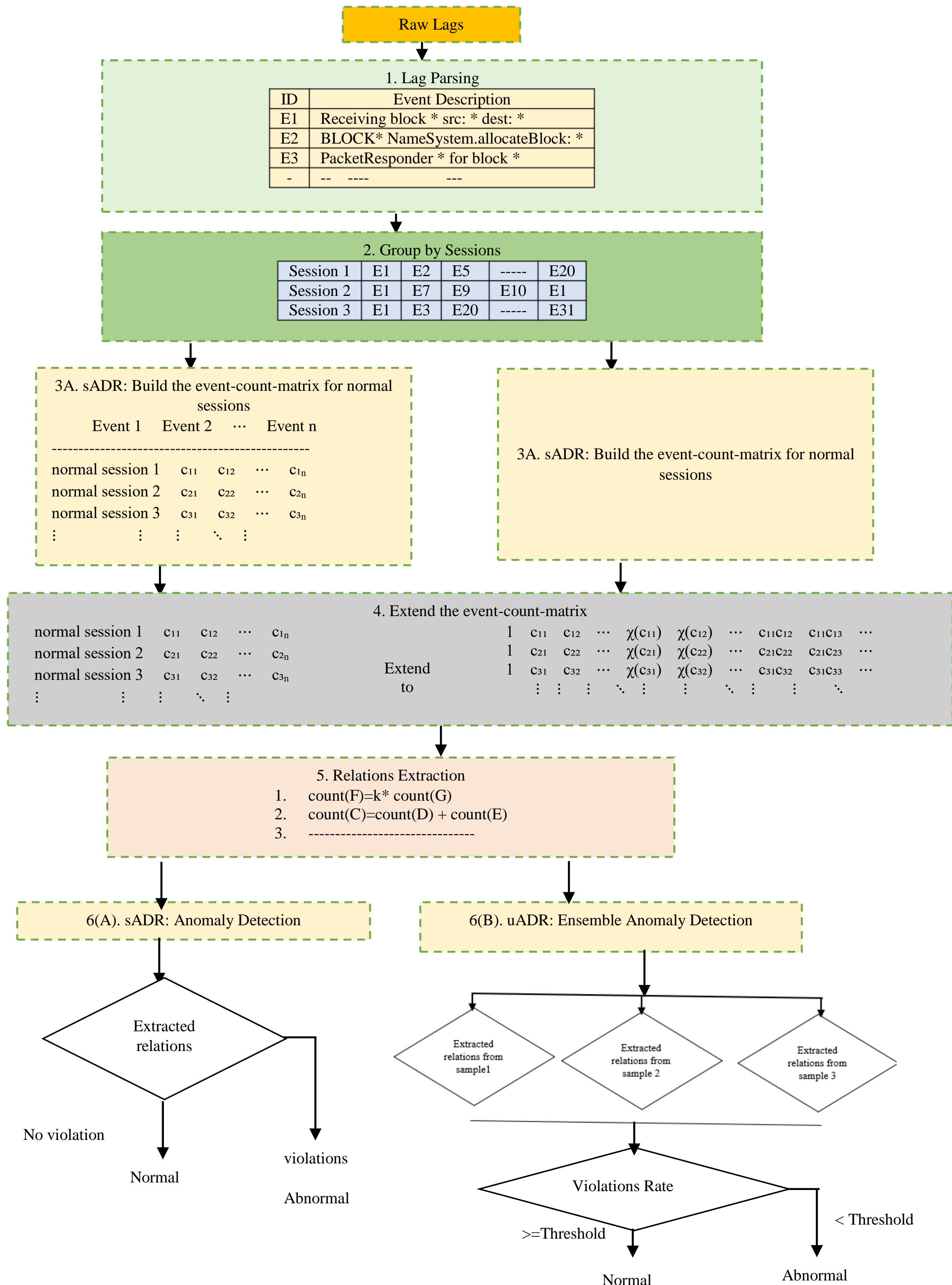

Fig. 2 Detailed approach of sADR/uADR scheme

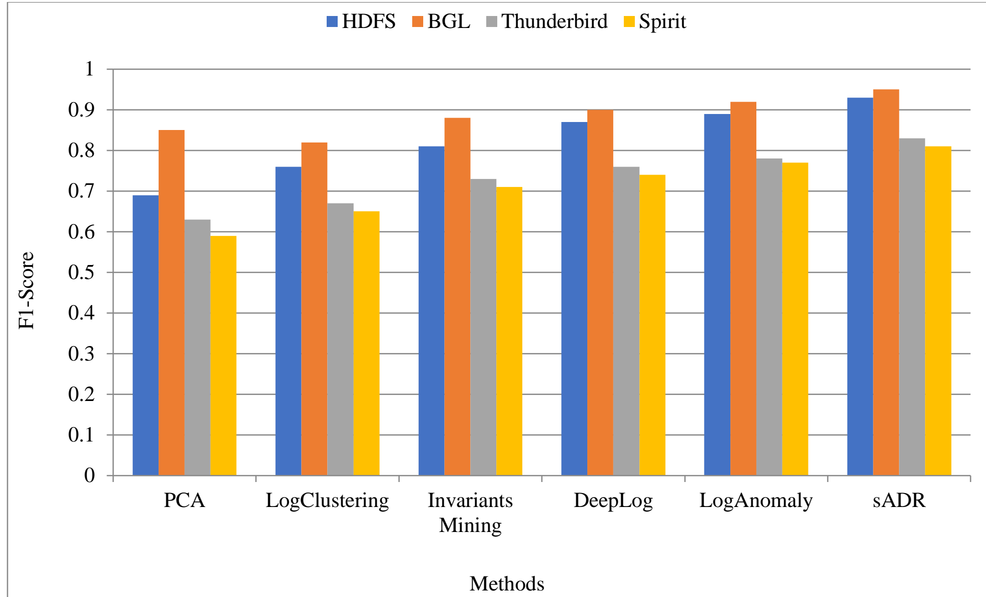

Fig. 3 F1-Score Comparison across Methods & Datasets

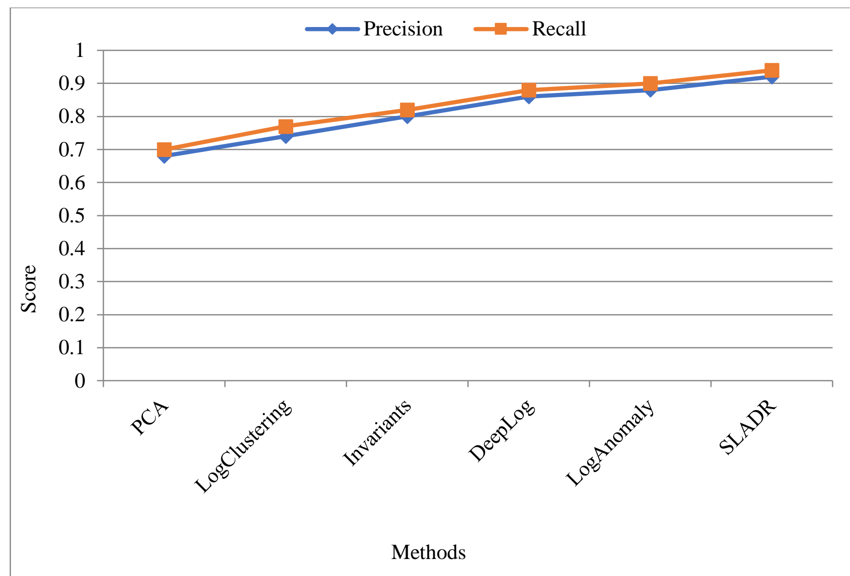

Fig. 4 Precision & Recall across methods (HDFS Dataset)

The Figure 5 illustrates the execution time for seven anomaly detection methods: PCA, LogCluster, Invariants, DeepLog, LogAnomaly, sADR and uADR. PCA requires approximately 5 seconds to execute, while LogClustering takes marginally longer at about 6.5 seconds. Invariants requires nearly 5.5 seconds, making these three methods comparatively fast. DeepLog and LogAnomaly are significantly slower. DeepLog requires an average execution duration of about 66 seconds and the slowest is LogAnomaly that takes about 70 seconds. The two models are more time consuming because of deep learning processes and increased computational requirements. On the contrary, sADR and uADR are the fastest; both of them are executed in no more than 3 seconds, which is much lower than the time spent by all other techniques, including DeepLog and LogAnomaly. This sharp divergence proves that sADR and uADR are not merely correct, but are designed to be efficient.

The chart highlighting the trade-off between the traditional models and the deep learning models underlines the fact that despite the better detection offered by the DeepLog and the LogAnomaly, the longer processing time of these two would be restrictive in the real time systems. The findings reveal that sADR and uADR are fast models that do not affect the performance which means that the models are suitable when it comes to large-scale log analysis operations.

DeepLog and sADR with growing set of size in log records between 5,000 and 50,000 log records. The area under the curve (AUC) is an extremely important statistic that indicates the ability of a model to distinguish normal and abnormal logs. PCA starts with around 0.72 as the AUC score and increases to nearly 0.81 as the size of the dataset increases. DeepLog is better, and it begins near 0.76, and slowly rises to 0.89.

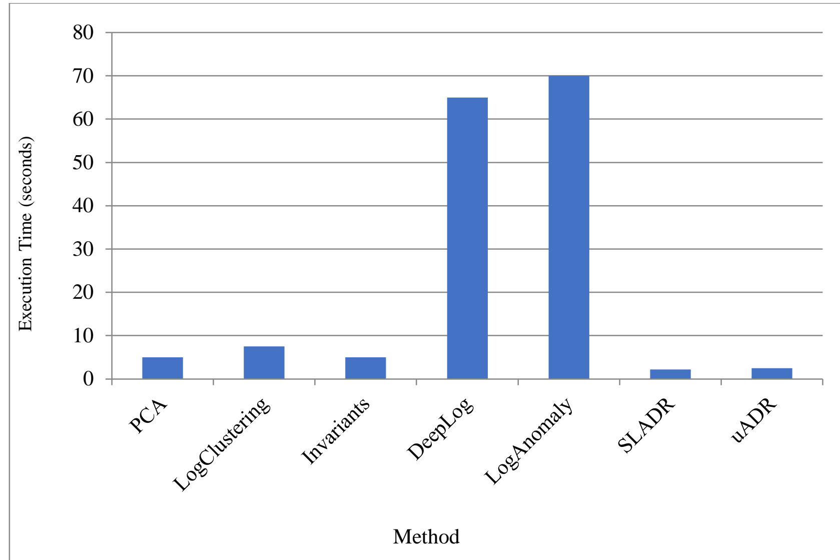

Fig. 5 Execution Time Analysis of Different Methods

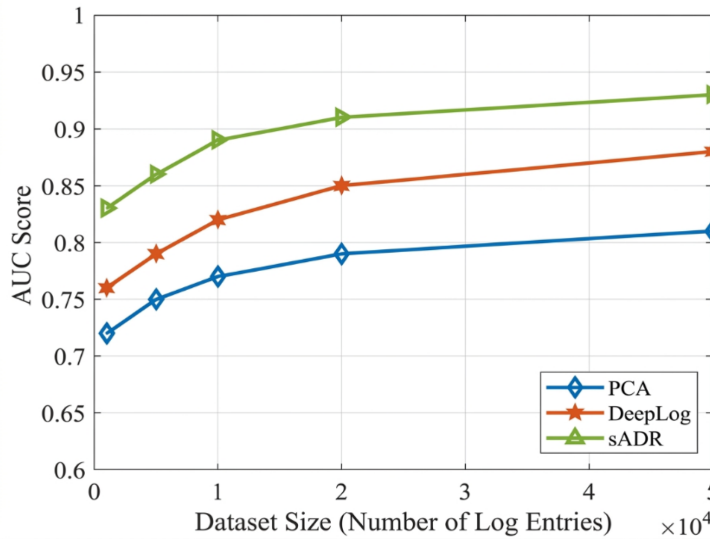

Fig. 6 AUC vs. Dataset size for anomaly detection methods

Such tendencies imply that both models are advantageous in terms of bigger training data, but DeepLog is always more effective than PCA. The best outcomes are shown by the sADR model. It starts with an AUC of 0.83 on small datasets and tends to 0.94 on the largest dataset. This consistent and strong growth shows that sADR grows well with the increase of data volume. It is also high in all cases when compared with PCA and DeepLog. Whereas PCA starts to improve gradually and DeepLog starts to increase steadily, sADR rises quicker and reaches a higher score. This trend indicates that sADR has high detection accuracy. Figure 7 shows the mean score of the three methods in terms of precision (AP) with an increase in the dataset size (records of logs) between 5,000 and 50,000 records. PCA has started with AP score of nearly 0.65 and increases slowly to approximately 0.75. DeepLog begins at approximately 0.70 and attains 0.85 at the largest dataset. DeepLog has an overall higher performance curve than PCA throughout its performance curve, and this implies that it is more accurate in detecting anomalies. Both methods demonstrate improvement with larger datasets, but the rate of

increase slows down after around 30,000 log entries. The sADR model outperforms both PCA and DeepLog across all dataset sizes. It begins around 0.80 even at 5,000 logs and consistently increases to around 0.93 at 50,000. The sADR curve has a higher and steeper beginning, which means that it performs very well even regarding small datasets. The difference between sADR and the other approaches is large across the board which shows that sADR is accurate and has more scalability. This graph clearly shows that sADR maintains a strong lead in average precision, making it a more reliable choice for large-scale log anomaly detection. This Figure 8 contrasts the execution time of PCA, DeepLog and sADR as the log volume increases from 10,000 to 50,000 entries. DeepLog has the sharpest increase.

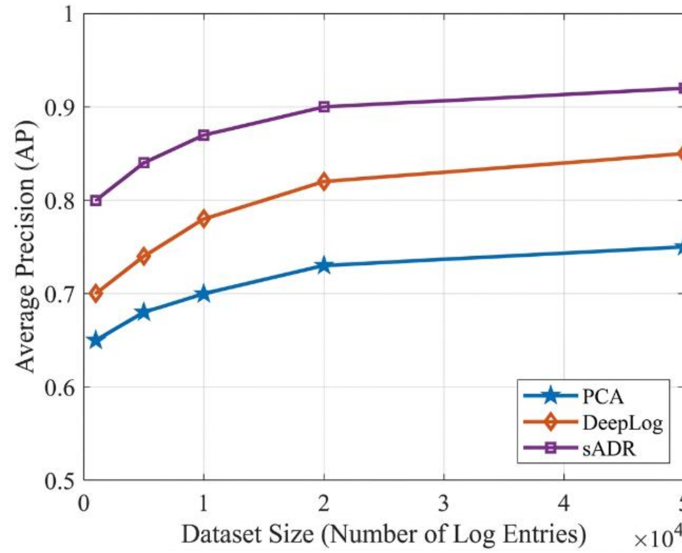

Fig. 7 Average precision versus dataset size

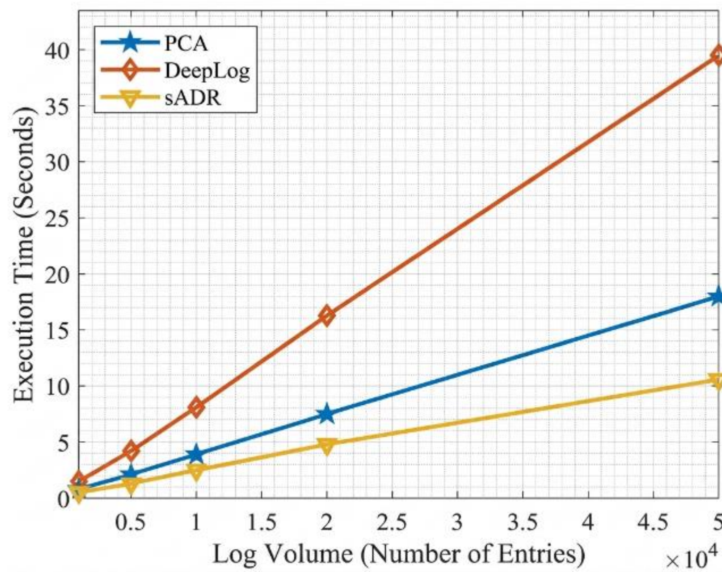

Fig. 8 Execution time versus log volume

It starts at about two seconds and it takes almost forty seconds at the maximum volume, which indicates that it takes it a lot longer as the data increases. PCA grows more slowly than DeepLog and starts growing in the range of one and a half seconds and reaches about eighteen seconds at the maximum volume. sADR has the highest speed of all log volumes. It starts below one second and at fifty thousand entries only reaches approximately eleven seconds. The sADR line is the straightest, which proves that it works with larger data sets with a much higher efficiency. This trend renders sADR a

superior/choice in a case of speed and scalability matters. DeepLog is very accurate but has a disadvantage in being slow to execute which is not ideal in real-time systems however sADR is a good trade-off in terms of both performance and steady improvement of logs volume. Figure 9 compares the memory use of six approaches PCA, Log Clustering, Invariants Mining, DeepLog, LogAnomaly and sADR on increasing the size of logs between ten thousand entries and fifty thousand entries. Among them, LogAnomaly consumes the most amount of memory. Its starting point is around fiftyMB, and steeply rises to around two hundred MB at the largest dataset size. DeepLog indicates an increasing trend and starts at approximately forty-five MB and goes to approximately one hundred and eighty MB. The two techniques exhibit the largest growth rates, which imply that there are increased memory requirements with increased log volumes. PCA, Invariants Mining and sADR are consuming less memory.

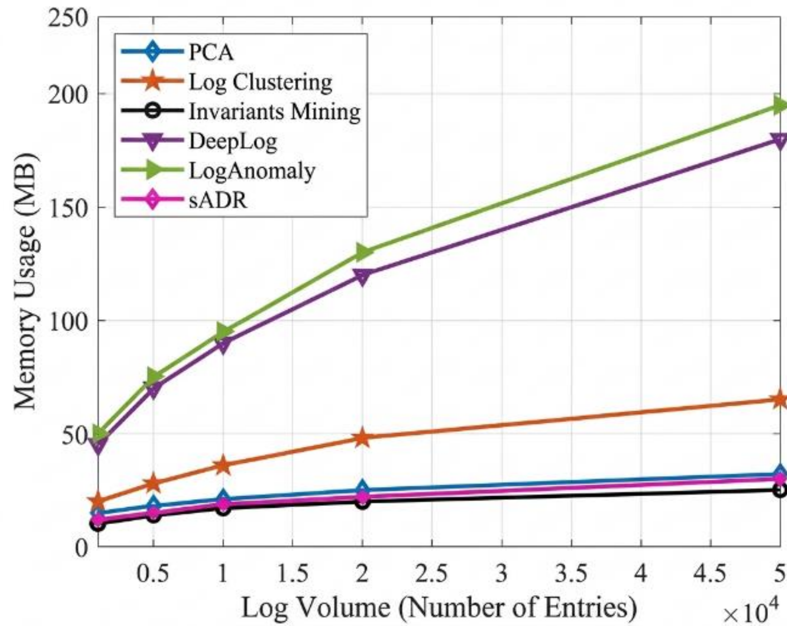

Fig. 9 Memory usage versus log volume

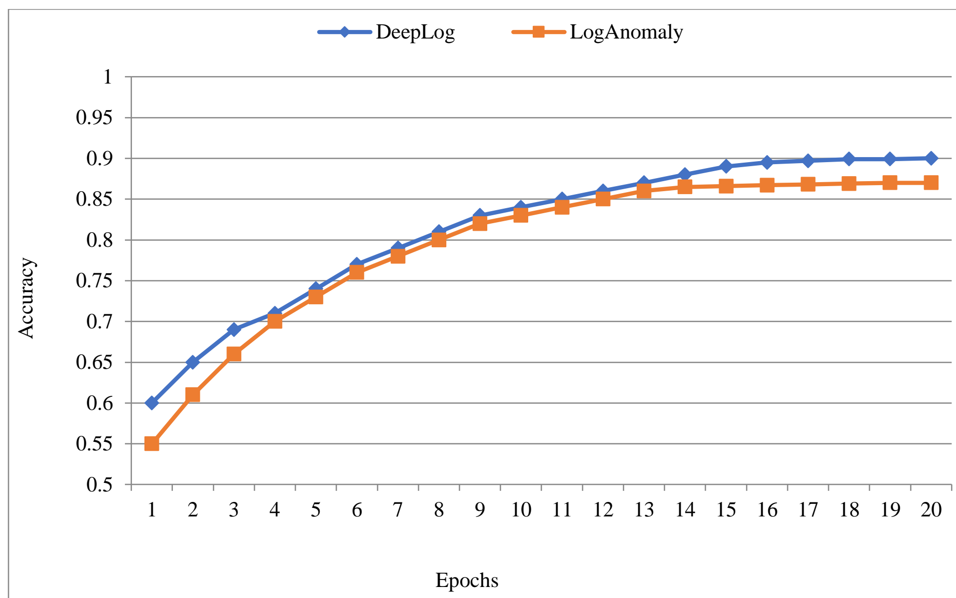

Fig. 10 Model accuracy versus epochs

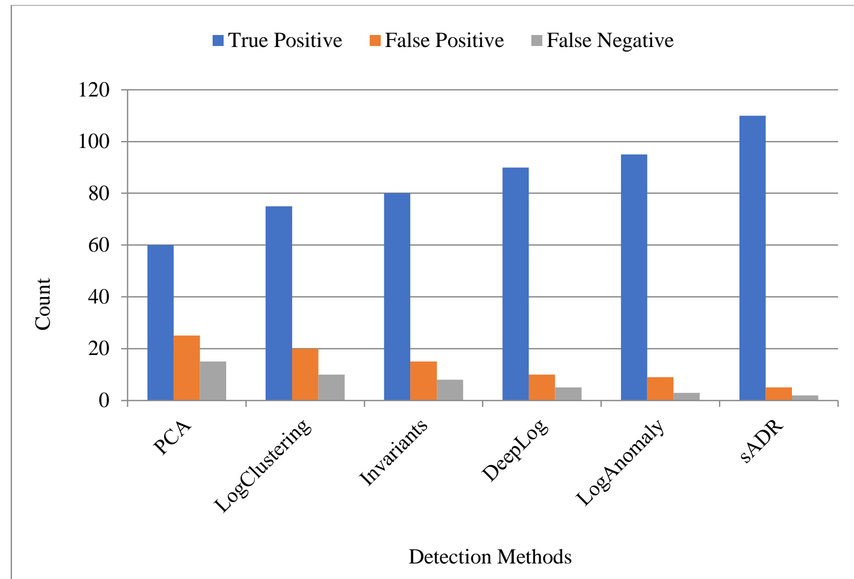

Fig. 11 Breakdown of TP/FP/FN per method

Memory growth stays steady and remains under 35 MB even for the largest datasets. PCA ends near 32 MB, while Invariants Mining and sADR remain close to 30 MB. Log Clustering uses more memory than these three but still far less than DeepLog and LogAnomaly. It starts at about 20 MB and grows to nearly 70 MB. This chart shows that while DeepLog and LogAnomaly might offer advanced detection. Much more memory is required by those methods. sADR uses moderate memory and gives high performance. This makes it a good choice for systems with limited resources.

The Figure 10 presents accuracy growth curves for DeepLog and LogAnomaly over 20 epochs of training. LogAnomaly starts at an accuracy of about 0.53, while DeepLog starts slightly higher at around 0.6. Both of the models record considerable accuracy enhancements in the first ten epochs as the training continues. At the tenth epoch, DeepLog will reach a degree of accuracy of about 0.85, whereas the LogAnomaly will reach a score of about 0.83, and this shows that the two are fairly on par in learning. Ten to twenty epochs onwards, the accuracy curves level off. DeepLog is still being optimized but at a slower pace, almost reaching an accuracy of almost 0.90. The increase in accuracy of the LogAnomaly decays after epoch 15 and ends at around 0.87.

This implies that DeepLog is advantaged by a longer training. The findings of the experiment show that both models can learn log patterns, but DeepLog has a small lead in accuracy during the training period. The number is a highlight of the importance of early training and the decreasing increase of accuracy in subsequent epochs. Figure 11 illustrates the number of true positives, false positives and false negatives of the different log anomaly detection techniques. The true-positive number of 60 is the lowest in PCA and the false-positive and false-negative numbers of 25 and 15 are the highest, respectively. The LogClustering shows a slight enhancement of approximately 75 true positives, 20 false positives and approximately 10 false negatives. The methods based on invariants also minimise false values and maximise false negatives, coming to approximately 80 true positives, 15 false positives and just under 10 false negatives. DeepLog and LogAnomaly are no exception. DeepLog yields about 90 true positives and nearly 10 and 5 false positives and falses respectively. LogAnomaly performs more or less as well, with an average of 95 true positives, and equally low error rates. sADR performs better, with an average of 110 true positives and false-positive and false-negative rates of approximately 5 and 2, respectively. This figure is a clear indication that sADR is the most accurate among the three and PCA is the weakest one. The accuracy of classification is highest towards the right column and the two types of errors diminish significantly.

6. Conclusion

The paper presents a new approach which is known as ADR. ADR enables identification of abnormal logs in the computer systems. It works through the acquisition of numeric correlations of normal log data. This is then followed by it checking whether new logs which are observed comply with the derived rules. In case a log breaches a rule, ADR logs it as an outlier. ADR stands out by ruling out the use of deep learning methods; it uses elementary matrix arithmetic instead. It determines the regularity of regular logs which always have. The system uses these patterns to analyse incoming log data thus making ADR fast and lightweight. ADR was tested on four real-world log datasets namely, HDFS, BGL, Thunderbird and Spirit. The algorithm showed great results in all the datasets. It outperformed many of other methods, such as PCA, DeepLog and Invariants Mining.

Besides, it showed effectiveness in semi-supervised and unsupervised environments. The main benefit of ADR is that it does not require extensive training data; a limited number of normal logs is required. When used in semi-supervised mode, it uses a few of the labeled anomalies to set a threshold. When used in unsupervised mode, no labeled data are used. Another benefit is transparency. ADR tells us exactly which rule was broken. This helps system engineers understand and fix problems faster. Deep learning models need more time and computing power. ADR is simple, yet strong. In the future, the plan is to improve ADR by adding support for nonlinear rules. ADR is expected to learn from data streams in real time. These methods will make ADR more useful in large, changing systems.

References

[1] Mohamed Amine Batoun et al., "A Literature Review and Existing Challenges on Software Logging Practices: From the Creation to the Analysis of Software Logs," *Empirical Software Engineering*, vol. 29, no. 4, pp. 1-61, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[2] Nan Yang et al., "An Interview Study about the use of Logs in Embedded Software Engineering," *Empirical Software Engineering*, vol. 28, no. 2, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[3] Ralph Foorthuis, "On the Nature and Types of Anomalies: A Review of Deviations in Data," *International Journal of Data Science and Analytics*, vol. 12, no. 4, pp. 297-331, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[4] Jesper E. van Engelen, and Holger H. Hoos, "A Survey on Semi-Supervised Learning," *Machine Learning*, vol. 109, no. 2, pp. 373-440, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[5] Matthias Kowal, Sofia Ananieva, and Thomas Thüm, "Explaining Anomalies in Feature Models," *ACM SIGPLAN Notices*, vol. 52, no. 3, pp. 132-143, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[6] Shreya Shankar et al., "Moving Fast with Broken Data," *ArXiv Preprint*, pp. 1-14, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[7] Weibin Meng et al., "LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs," *International Join Conferences on Artificial Intelligence Organization*, vol. 19, no. 7, pp. 4739-4745, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[8] Bo Zhang et al., "Anomaly Detection Via Mining Numerical Workflow Relations from Logs," *2020 International Symposium on Reliable Distributed Systems (SRDS)*, Shanghai, China, pp. 195-204, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[9] Arie Karniel, and Yoram Reich, "Formalizing a Workflow-Net Implementation of Design-Structure-Matrix-based Process Planning for new Product Development," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 41, no. 3, pp. 476-491, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[10] Alon Geva et al., "Adverse Drug Event Presentation and Tracking (ADEPT): Semiautomated, high Throughput Pharmacovigilance using Real-World Data," *JAMIA Open*, vol. 3, no. 3, pp. 413-421, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[11] Christian Schlereth, and Bernd Skiera, "Two New Features in Discrete Choice Experiments to Improve Willingness-to-Pay Estimation that Result in SDR and SADR: Separated (Adaptive) Dual Response," *Management Science*, vol. 63, no. 3, pp. 587-900, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[12] Linda Härmark, Florence van Hunsel, and Birgitta Grundmark, "ADR Reporting by the General Public: Lessons Learnt from the Dutch and Swedish Systems," *Drug Safety*, vol. 38, no. 4, pp. 337-347, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[13] Marcello Cinque et al., "On the Impact of Debugging on Software Reliability Growth Analysis: A Case Study," *Computational Science and its Applications - ICCSA 2014: 14th International Conference*, Guimarães, Portugal, vol. 8583, pp. 461-475, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[14] Adetokunbo A.O. Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios, "Clustering Event Logs using Iterative Partitioning," *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, New York, NY, United States, pp. 1255-1264, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[15] Qiang Fu et al., "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis," *2009 Ninth IEEE International Conference on Data Mining*, Miami Beach, FL, USA, pp. 149-158, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[16] Min Du, and Feifei Li, "Spell: Streaming Parsing of System Event Logs," *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Barcelona, Spain, pp. 859-864, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[17] Pinjia He et al., "Drain: An Online Log Parsing Approach with Fixed Depth Tree," *2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, HI, USA, pp. 33-40, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[18] Jieming Zhu et al., "Tools and Benchmarks for Automated Log Parsing," *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Montreal, QC, Canada, pp. 121-130, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[19] R.K. Sahoo et al., "Failure Data Analysis of a Large-Scale Heterogeneous Internet Services," *International Conference on Dependable Systems and Networks*, Florence, Italy, pp. 772-781, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[20] Yinglung Liang et al., "Failure Prediction in IBM BlueGene/L Event Logs," *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Omaha, NE, USA, pp. 583-588, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[21] Peter Bodik et al., "Fingerprinting the Datacenter: Automated Classification of Performance Crises," *Proceedings of the 5th European Conference on Computer Systems*, Association for Computing Machinery, New York, NY, United States, pp. 111-124, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[22] Shilin He et al., "Experience Report: System Log Analysis for Anomaly Detection," *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, Ottawa, ON, Canada, pp. 207-218, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[23] Wei Xu et al., "Detecting Large-Scale System Problems by Mining Console Logs," *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, Association for Computing Machinery, New York, NY, United States, pp. 117-132, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[24] Qingwei Lin et al., "Log Clustering based Problem Identification for Online Service Systems," *Proceedings of the 38th International Conference on Software Engineering Companion*, Association for Computing Machinery, New York, NY, United States, pp. 102-111, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[25] Jian-Guang LOU et al., "Mining Invariants from Console Logs for System Problem Detection," *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010. [Google Scholar]

[26] Min Du et al., "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, Association for Computing Machinery, New York, NY, United States, pp. 1285-1298, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[27] Xu Zhang et al., "Robust Log-based Anomaly Detection on Unstable Log Data," *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Association for Computing Machinery, New York, NY, United States, pp. 807-817, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[28] Christophe Bertero et al., "Experience Report: Log Mining using Natural Language Processing and Application to Anomaly Detection," *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, Toulouse, France, pp. 351-360, 2017. [CrossRef] [Google Scholar] [Publisher Link]