NovaAD: Semi-Supervised Anomaly Detection in Log Series

Nayef Roqaya
Philipps University of Marburg
Marburg, Germany
roqaya@staff.uni-marburg.de

Hajira Jabeen
University of Cologne
Cologne, Germany
hajira.jabeen@uk-koeln.de

Thorsten Papenbrock
Philipps University of Marburg
Marburg, Germany
papenbrock@informatik.uni-marburg.de

ABSTRACT

Many modern IT systems communicate their states, activities, and interactions through textual event logs. In recent years, these event streams have become increasingly important for knowledge extraction, data-driven applications, system health monitoring, error detection, system optimization, and other IT operations tasks. Automatic and reliable detection of anomalous events is crucial for these tasks. However, existing anomaly detection approaches typically require extensive parameter tuning and feature engineering, rely on large amounts of representative training data, or achieve limited detection accuracy. Furthermore, some approaches suffer from poor run-time efficiency, making them impractical for real-time applications where rapid anomaly detection is essential for maintaining system reliability and operational continuity. In this paper, we propose NovaAD, an end-to-end semi-supervised anomaly detection algorithm for textual event series that requires only a small set of normal data to guide the learning process. Moreover, it eliminates the need for manual configuration through a universal feature set and automatic self-configuration techniques. NovaAD demonstrates robustness in handling unstable log data by incorporating semantic embedding features as part of its feature vector and effectively detects anomalies using an ensemble strategy that combines voting and stacking classifiers. Compared to previous approaches, NovaAD achieves the best runtime performance on both CPU and GPU platforms while maintaining high accuracy and efficiency in anomaly detection. Through a detailed experimental evaluation on publicly available real-world log datasets, including BGL, HDFS, Thunderbird, and Spirit, our analysis shows that NovaAD achieves the best anomaly detection performance in terms of F1-score, precision, and recall across all evaluated datasets. These results are consistently observed across multiple runs and statistically validated against state-of-the-art baseline methods, while also providing the fastest training and inference runtimes among all compared approaches.

ACM Reference Format:
Nayef Roqaya, Hajira Jabeen, and Thorsten Papenbrock. 2026. NovaAD: Semi-Supervised Anomaly Detection in Log Series. In *Proceedings of International Conference on Extending Database Technology (EDBT '26).* ACM, New York, NY, USA, 16 pages. https://doi.org/XXXXXXX.XXXXXXX

EDBT '26, March 24–27, 2026, Tampere, Finland

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

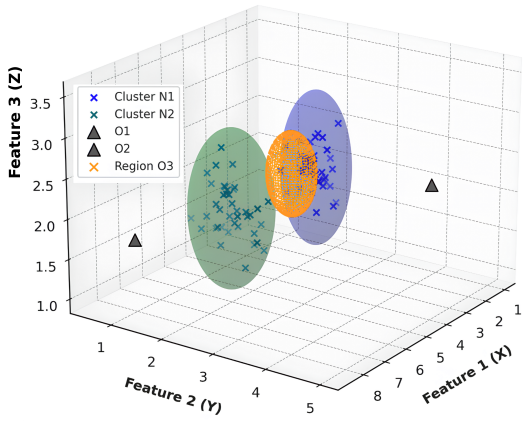

Figure 1: A simple example of anomalies in a three-dimensional dataset.

1 ANOMALY DETECTION IN LOG SERIES

Anomaly detection (AD) involves detecting certain data elements that differ from all other data elements in a set or sequence of elements. Figure 1 illustrates an example of anomaly detection in a three-dimensional feature space defined by three artificial features. Two main clusters ($N1$ and $N2$) represent normal data distributions, while $O1$ and $O2$ are isolated anomalies located far from the clusters. Furthermore, the region $O3$ illustrates a collective anomaly, where a group of points forms a dense but unexpected region between the clusters. This example highlights different manifestations of anomalies in multidimensional data [34]. Anomaly detection, which is the process of automatically identifying such conspicuous data points, has become essential for building trust worthy computer systems [47]. System logs, which record important events and runtime states, are particularly valuable for this purpose, as they capture notable system behaviors and provide rich data for anomaly detection

Existing log-based anomaly detection approaches share three high-level steps: (1) extraction of log events and sequences, (2) training of an anomaly detection model with a specific machine learning or data mining algorithm, and (3) classifying new log events with the trained model. These algorithms typically learn hidden patterns directly from the data or derive them from large datasets. Log analysis identifies the anomalous events and often provides additional descriptive information, such as timestamps, anomaly scores, or severity levels. Due to the effectiveness of such anomaly detection approaches, log-based anomaly detection is widely adopted [73]. Although current anomaly detection methods have been proven

to be effective in various studies, they still face several practical challenges.

- **Feature selection:** Data logs are often high-dimensional and contain a mix of textual and numerical data. Therefore, extracting relevant features for the effective detection of anomalies continues to be a major challenge, often requiring manual intervention. Due to the diversity of information and heterogeneity of schemata, automatically extracting and instrumenting effective features in an anomaly detection system is difficult but desirable.
- **Auto parameterization:** Detecting anomalies in event series is a complex task, as it usually covers considerations with respect to the similarity, time, and semantics of events. To accurately address these aspects, most anomaly detection algorithms rely on training data and the careful tuning of multiple hyperparameters. In our study, these include search ranges, model architectures, the number of components used in dimensionality reduction, the number of topics in topic modeling, and other model-specific parameters. As a consequence, choosing these parameters for each dataset poses a challenge and is usually solved via manual configuration, tuning, and/or training at various stages of the detection systems. Consequently, pursuing an end-to-end approach that can independently configure and fine-tune itself remains a formidable challenge.
- **Robustness in handling unstable logs:** Unstable logs are log entries characterized by inconsistencies, missing fields, or irregular structures, which may arise from variations in logging practices, dynamic system behavior, or incomplete data capture. These irregularities hinder the ability to distinguish genuine system faults from benign fluctuations, thereby increasing the complexity of anomaly detection and root cause analysis. Consequently, they can lead to elevated false positive rates and delays in problem resolution.
- **Efficiency of Anomaly Detection:** Efficient anomaly detection is essential to prevent issues such as security breaches, system failures, and delayed responses in fields such as cybersecurity, healthcare and industrial monitoring. In particular, slow detection can lead to increased costs and missed opportunities. In large-scale data environments, rapid detection ensures system efficiency and quick issue resolution, thus enhancing overall resilience and decision-making.

To address these four challenges, we present NovaAD, a semi-supervised anomaly detection algorithm for log-based, i.e., textual event series. With NovaAD, we make the following contributions:

(1) An *effective feature set* for semi-supervised based anomaly detection in textual event data.
(2) A *self-configuration technique* that automates the parameter tuning and makes the algorithm fully automated in the context of a semi-supervised approach.
(3) *Robustness in handling unstable logs* is achieved by incorporating semantic embedding features.
(4) An *efficient anomaly detection* contributes by preventing response delays and minimizing risks. Enhances system reliability and operational efficiency by ensuring quick detection and resolution of anomalies.

In this paper, we first review related studies on anomaly detection in textual event logs in Section 2 and introduce the terminology used for log data processing in Section 3. Section 4 then details the proposed anomaly detection approach NovaAD, which consists of feature extraction, novelty detection, label estimation, and ensemble detection components. In Section 5 and Section 6, we provide the results of our experimental evaluation demonstrating the efficiency and effectiveness of NovaAD. Finally, Section 8 summarizes our findings and outlines potential directions for future research.

2 RELATED WORKS

System logs play a crucial role in anomaly detection in numerous practical applications. Researchers have explored various learning techniques to enhance the accuracy of supervised, semi-supervised, or unsupervised log-based anomaly detection. In what follows, we discuss the most recent and, in their respective evaluations, best-performing state-of-the-art anomaly detection algorithms for log event series.

Log Data Representation for Anomaly Detection. The authors in [65] employed ChatGPT for log parsing and proposed a novel log grouping strategy. Their approach requires only a small set of labeled examples, which ChatGPT uses to generate pseudo-labels for the remaining log data, thereby effectively enlarging the training set. Furthermore, SemiSMAC incorporates a Sequential Model-based Algorithm Configuration (SMAC) to automatically optimize the hyperparameters of the embedded models. This integration consistently improves performance, particularly in resource-constrained environments. The *LogADSBERT* anomaly detection method [30] is built on *Sentence-BERT*. It uses the Sentence-BERT model to capture the semantic characteristics of log events and uses a bidirectional long- and short-term memory (Bi-LSTM) network to perform anomaly detection. In CNN [46], the method integrates logkey2vec embeddings with multiple one-dimensional convolutional layers, followed by dropout and max-pooling operations, to automatically learn event relationships within log sequences. NeuralLog [35] avoids explicit log parsing by extracting semantic information directly from raw log messages and representing them as semantic vectors. The *LogBD* anomaly detection method [42] uses pre-trained models and domain adaptation. It utilizes the *BERT* pre-trained model to capture the semantic information of log data, effectively addressing challenges arising from the polysemy of log statements and vocabulary evolution. Anomaly detection in LogBD is based on a distance metric constructed through domain adaptation, where Temporal Convolutional Networks (TCNs) are employed to extract representative features from diverse system logs and map them onto a unified hypersphere space. Although sentiment analysis is not typically applied to log messages, the *ClusterLog* algorithm [**10024030**] clusters log keys based on their semantic similarity over time and reduces the complexity of log sequences by minimizing the number of unique log keys needed for representation. This approach improves the ability of sequence-based models to learn log patterns more effectively by grouping logs that are *semantically* and *sentimentally* similar. The study in [64] reports the findings of a preliminary experiment on anomaly detection in network performance. This experiment applied topic modeling to runtime variable data collected from base stations within live Radio

Access Networks (RANs). The results indicate that topic modeling effectively organizes semantically related data in a manner comparable to that of human experts. Furthermore, anomalies in test cases were successfully identified using latent Dirichlet allocation (LDA) topic models. Previous studies [78], [32] noted that log data often exhibit instability, mainly because log statements in source code are frequently modified during software updates. As a result, some newly generated log events or sequences may not exist in the training dataset. This instability can substantially reduce the performance of existing unsupervised and semi-supervised models when they encounter unfamiliar log patterns in real-world settings. To address this issue, PLELOG [76] is able to remain immune to unstable log data by employing word embedding and TF-IDF-based aggregation techniques that preserve the contextual meaning of log messages.

Supervised Anomaly Detection in Log Data. Supervised approaches to anomaly detection leverage labeled datasets to effectively identify irregularities in system logs [6, 7, 10, 16, 37]. For example, Liang et al. [37] examined the performance of four classifiers: RIPPER (a rule-based classifier), Support Vector Machines (SVM), a conventional Nearest-Neighbor (NN) method, and a specialized NN model to predict system failures based on log data. Likewise, Bodik et al. [6] utilized logistic regression to recognize recurring performance issues within data centers. To improve the precision of anomaly detection in log analysis, recent advances have incorporated deep learning methodologies into the field[14, 70, 77, 78]. Beyond advanced supervised approaches such as *LogRobust*, Vinayakumar et al. [70] proposed a stacked LSTM model that exhibits high precision in detecting anomalies within log sequences. In contrast to previous methodologies, our research introduces a semi-supervised log-based anomaly detection approach NovaAD, which relies solely on easily accessible normal log sequences. This approach eliminates the need for extensive manual labeling, a common limitation of supervised techniques. In particular, our findings indicate that NovaAD achieves performance comparable to the state-of-the-art supervised method *LogRobust* [78], underscoring its practical advantages and strong robustness.

Unsupervised and Semi-supervised Anomaly Detection in Log Data. Unsupervised and semi-supervised methods are often more practical for anomaly detection, as they reduce reliance on labeled data for model training [4, 20, 22, 26, 31, 36, 69]. Beyond established techniques such as *Deeplog* [14], *LogAnomaly* [48], *LogCluster* [39], *PLELog* [76], and *PCA* [15], several novel approaches have been introduced. For example, Lou et al. [44] proposed *Invariant Mining (IM)*, which identifies anomalies by detecting deviations from execution flow invariants derived from historical log sequences. Likewise, He et al. [26] developed *Log3C*, which employs *Hierarchical Agglomerative Clustering* [9] to group similar log sequences and analyze their relationships with system *Key Performance Indicators* (KPIs). Unlike traditional approaches, NovaAD enhances the effectiveness of unsupervised and semisupervised methods by integrating key advantages of supervised, semi-supervised and unsupervised learning techniques. It uses novelty detection and label estimation to learn only from limited normal historical log data, eliminating the need for manual labeling. The method relies on unsupervised metrics to evaluate novelty detection and label estimation without requiring ground truth at this stage. Ultimately, it significantly improves the performance of anomaly detection.

3 KEY TERMINOLOGY IN LOG DATA PROCESSING

A ***log message*** is an unstructured textual record produced during the operation of a computer system, software application, or network service that captures the state of a system at a specific point in time. The log parsing process converts raw log messages into many structured components, such as *event timestamps*, *log levels*, *log event (templates)*, and *log parameters*. In our study, the log level component was excluded because it was inconsistently recorded between sources or did not provide a significant value for anomaly detection. A ***log event*** represents the textual component of the message and serves as a predefined *template* created by the system or application developers responsible for the implementation of the log functionality. In contrast, ***log parameter*** represents the variable part, containing system-specific attributes such as IP addresses. A ***log sequence*** is an ordered series of log events, which can be identified using task IDs or techniques such as sliding windows. If a log sequence contains system anomalies, it is classified as an anomalous log sequence; otherwise, it is considered a normal log sequence. For example, a sample raw log message can be represented as: **`(2025-08-05 12:34:56 INFO user 192.168.1.10 logged into the system)`**. Table 1 shows the decomposition of a single log message into four key components, while table 2 presents a time-ordered sequence of log events, where each entry is represented by a template that contains placeholders for system-specific values. This structured representation facilitates the analysis of system behavior over time.

Table 1: Example of a parsed single log message.

Component	Example Value
log event timestamp	2025-08-05 12:34:56
log level	INFO
log event (template)	User <*> logged into the system
log parameters	192.168.1.10

Table 2: Example of a list log events (Log Sequence).

Timestamp	Log Event (Template)
12:00:01	User <*> logged into system
12:00:05	User <*> accessed <file>
12:00:10	Disk <*> reached <percent> capacity

Unstable logs are log entries that originate from rare or infrequently occurring templates, while ***Stable logs*** are log entries that follow frequently recurring and consistent log templates over time.

4 ANOMALY DETECTION WITH NOVAAD

In this section, we present our novel approach **S**emi-**S**upervised **A**nomaly **D**etection in Log Series (**NovaAD**) aimed at improving the accuracy of anomaly detection in logs. In line with *PLELog* [76],

NovaAD offers semi-supervised anomaly detection capabilities for textual event series and, therefore, requires only a subset of normal labeled data. The system architecture for this approach is depicted in **Figure 2**, which outlines the following six high-level phases:

In subsequent sections, we further elaborate on these phases by discussing all five steps in more detail.

4.1 Log Parsing

The first phase processes the input logs, either in batches or as a continuous stream, extracting textual events from unstructured log messages and transforms them into structured log events. The raw log messages are usually unstructured and contain many parameters, making automated analysis challenging [24, 25]. To address this, *PLELog* [76] extracts structured log events from raw messages using the Drain parser. Since structured events are easier to analyze, using this parsing approach also in our analysis pipeline improves its effectiveness. Specifically, PLELog uses the state-of-the-art drain method [25], which has been shown to be accurate and efficient in prior studies [23]. Similarly, NovaAD incorporates Drain [25, 79] to convert raw logs into structured templates, enabling more reliable downstream analysis.

4.2 Feature Engineering

In this phase, a set of distinguishing features is extracted, configured, and fine-tuned to facilitate novelty detection and label estimation for unlabeled data within the training set. To ensure compatibility with subsequent processing steps, the extracted feature vectors are standardized and scaled. Our feature extraction process incorporates various characteristics that provide a comprehensive description of the data.

This diverse set of characteristics paints a rich and multidimensional picture of the data, facilitating more nuanced and insightful analyses. The first group of features, i.e., Dominant Topic Index and Entropy features are based on topic modeling. Similarly to previous works that use topic-based features for spammer detection in social media posts [33, 41] or anomaly detection in log data [71], our aim is to capture semantic event categories in the form of topics from event texts. Despite the brevity of the texts, they still provide sufficient context for topic modeling. For topic-based features (Dominant Topic Index and Entropy), we use the topic modeling technique *Latent Dirichlet Allocation (LDA)* [5] to automatically extract topics from the textual data in each log-event. We treat each textual log event E as an individual document d. The fundamental concept underlying *LDA* is that a document can be viewed as a composite of various topics, and each topic, in turn, is composed of a specific set of words. These topics exhibit a higher probability of sharing common words.

Each document d_l is represented as a multinomial distribution θ_l on a set of latent topics $TP = [tp_1, tp_2, ..., tp_k]$ with k being the number of topics. Let $\theta_l = [p(tp_1|d_l), p(tp_2|d_l), ..., p(tp_k|d_l)]$ be the topic distribution of document d_l with $p(tp_j|d_l) = x_{lj}$ representing the probability of topic tp_j in document d_l. Each topic tp_j is also a multinomial distribution $\phi_j = [p(w_1|tp_j), p(w_2|tp_j), ..., p(w_m|tp_j)]$ on a set of words $W = \{w_1, w_2, ..., w_m\}$ with m being the number of words. By definition, $\sum_{a=1}^{m} p(w_a|tp_j) = 1$. Latent Dirichlet Allocation (LDA) is a generative probabilistic model commonly used for topic modeling. It is typically trained on a specific corpus of text data to discover topics within those data. We provide our text documents as input and then fit the model to our data to discover topics. For this work, we take an LDA model and consider the textual content of our log events as documents $D = \{d_1, d_2, ..., d_n\}$. Based on each document's topic distribution, we can then calculate for each log event d_l an entropy and dominant topic as topic-based features.

(1) Dominant Topic Index Feature (DTIF). DTIF is a feature derived from topic modeling [5, 64], representing the main theme of an event based on its Latent Dirichlet Allocation (LDA) topic distribution. In this process, LDA treats the textual content of the event as a document and assigns a topic distribution, from which the dominant topic is identified as the one with the highest probability and its corresponding index is used as the feature value. This feature is motivated by the observation that log events often share underlying semantic patterns that are not directly visible in their surface text. Capturing the dominant topic enables the grouping of semantically [63] related events despite lexical variations, thereby reducing noise and enhancing anomaly detection by revealing thematic deviations that may indicate abnormal system behavior.

LDA provides probability distributions for topics throughout the vocabulary, manifested as a collection of words from the vocabulary, each accompanied by an associated probability. Based on these distributions of words and topics throughout the corpus [5, 64], the feature of the topic captures the topic assigned to the textual content of an event through the index of its dominant topic. Each document, i.e., event is considered a mixture of topics, and the feature of the topic is the probabilities of the topics in that mixture. For log event d_l, a set of latent topics TP, and topic distributions $[p(tp_1|d_l), p(tp_2|d_l), ..., p(tp_k|d_l)]$ for document d_l, where $p(tp_j|d_l)$ to denote the probability of assigning the topic tp_j to the log event d_l, we calculate the *topic feature* as follows:

$$Topic(d_l) = [p(tp_1|d_l), p(tp_2|d_l), ..., p(tp_k|d_l)] \quad (1)$$

The dominant topic $Topic_{\text{dominant}}$ is selected as the topic with the highest probability [66] because it best represents the primary semantic content of the log message, allowing clearer interpretation and more effective anomaly detection by filtering out noisy topics with low probability. This approach follows standard practices in topic modeling, where the highest-probability topic serves as the most reliable indicator of the main theme of a document.

$$TopicIndex(d_l)_{\text{dominant}} = \arg\max_i p(tp_i|d_l) \quad (2)$$

Here, $\arg\max_i p(tp_i|d_l)$ returns the index of the topic tp_i with the maximum probability of the distribution of the topic.

(2) Entropy Feature (EF). EF in topic modeling measures the uncertainty in the topic distribution of a log event. For each event, the probability distribution over topics generated by Latent Dirichlet Allocation (LDA) is used to calculate its entropy [33, 41, 71]. Higher entropy values indicate that the event's textual content spans multiple topics, reflecting greater complexity or uncertainty. This is particularly useful for anomaly detection, as anomalous events often display unusual or mixed semantic patterns compared to normal events, leading to elevated entropy. By capturing areas

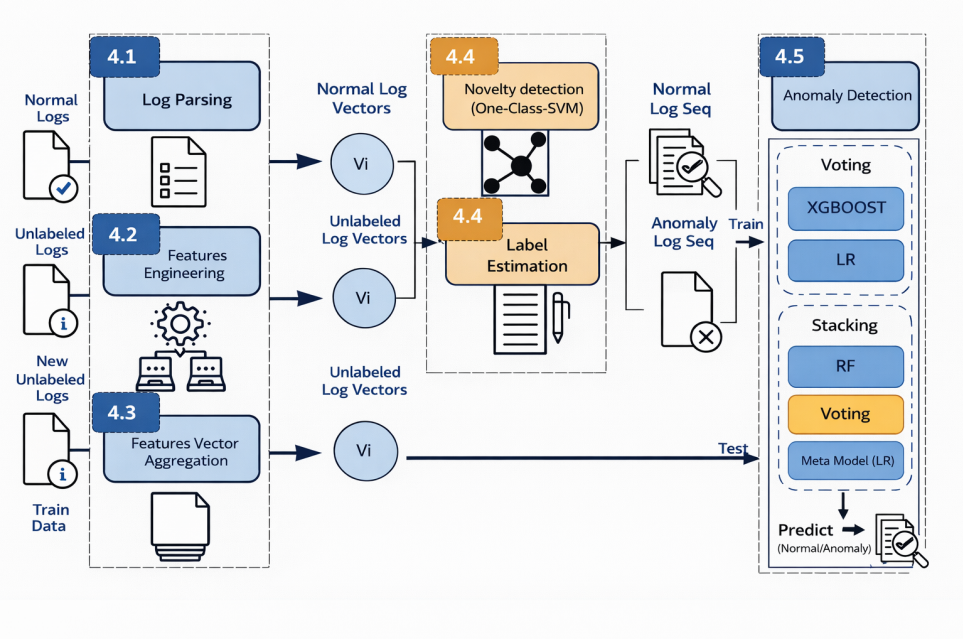

Figure 2: Overview of NovaAD.

of increased uncertainty, EF highlights potential anomalies that may not be apparent from dominant topics alone. Integrating this feature with others, such as the Dominant Topic, improves the identification of events with ambiguous content and improves overall anomaly detection performance [33]. For each log-event, the topic distribution θ_l indicates how strongly the event relates to different topics. High entropy implies that the event is spread across multiple topics, suggesting ambiguity or complexity, while low entropy indicates a focused interest in specific topics. For log event d_l on a set of latent topics TP, we compute *entropy feature* with the equation:

$$Entropy(d_l) = [Entropy(d_l, tp_1), ..., Entropy(d_l, tp_k)] \quad (3)$$
$$\text{with } Entropy(d_l, tp_k) = -\sum_{j=1}^{K} x_{lj} \log_2 x_{lj}$$

where $x_{lj} = p(tp_j \mid d_l)$ denotes the probability that the log-event d_l is associated with the topic j, and k is the total number of latent topics.

(3) SBERT Feature (SBTF). SBTF captures the semantic essence of a textual log event using a pre-trained BERT-based Sentence Transformer [42] and generates sentence-level embeddings that represent the underlying meaning of each log, stabilizing the representation of logs with surface-level variations. By encoding core concepts or objects within the text, the SBERT feature enables differentiation between events and effectively represents the semantics of log content [21].

Pre-trained language models, including BERT [13] and GPT [68], can be used to extract semantic features as embeddings. For example, PLELog [76] leverages word embeddings and TF-IDF aggregation to handle unstable log data. In our method, we employ the SBERT transformer [56], which is optimized for generating sentence embeddings and is computationally lighter than GPT. By capturing rich semantic information, contextual nuances, and word relationships, SBERT enhances the ability to measure semantic similarity across log events, facilitating reliable comparison even when textual expressions differ [76]. We use a *Sentence Transformer* based on a pre-trained *BERT (bert-base-nli-mean-tokens)* model, which is $transform_{BERT}(d_l)$, to generate fixed-sized embeddings EM_{d_l} for entire textual log events d_l.

The output dimensionality of the embedding at the sentence level EM_{d_l} is $n = |EM_{d_l}| = 768$, which is the inherited dimensionality of the underlying model *SBERT*. Using high-dimensional sentence embeddings generated by the Sentence Transformer model introduces heightened complexity and computational costs. To address this challenge, we integrate Principal Component Analysis (PCA) [57] to reduce the dimensionality of the embeddings. By carefully tuning the number of components, we aim to select an optimal value that maximizes the explained variance [19]. This step helps to retain as much information as possible from the original high-dimensional data while reducing its dimensionality. In Principal Component Analysis (PCA), the components are ordered by the amount of variance they explain in the data. We can balance reducing dimensionality and retaining meaningful information by selecting the appropriate number of components. Then, we use PCA components as the final feature *SBERT*.

$$SBERT(d_l) = PCA(EM_{d_l}) \quad (4)$$
$$\text{with } EM_{d_l} = transform_{BERT}(d_l)$$

(4) Sentiment Feature (SEF). SEF captures the emotional tone, attitude, or sense expressed in log event texts. This is particularly useful for handling unstable logs, where messages from users, systems, or error descriptions may vary, as sentiment provides contextual cues that can correlate with anomalies or system malfunctions [**10024030**, 29].

By integrating sentiment with semantic features such as SBERT, we gain a complementary perspective on log events, capturing both meaning and emotional tone. This combination improves the robustness of anomaly detection, especially for logs with variable expressions or implicit indicators of system issues [**10024030**]. To construct this feature, we apply sentiment analysis using the pre-trained Sentiment-RoBERTa model [29], enabling reliable binary sentiment classification across diverse English-language texts. For each event d_l, the model predicts a positive (value 1) or negative (value 0) sentiment, which we can use as a *sentiment feature* for the detection of anomalies.

$$Sentiment(d_l) = SentimentRoberta(d_l) \tag{5}$$

(5) Word Count Feature (WCF). The WCF measures the total number of words in a log message [50]. Logs with unusually high or low word counts compared to typical entries may indicate anomalies, as they deviate from expected patterns. This feature captures the length and structural characteristics of messages, providing a simple yet effective signal to detect atypical log events. For a log-event d_l, the Word-Count feature is defined as:

$$\text{Word Count}(d_l) = \sum_{i=1}^{n} w_i \tag{6}$$

where w_i represents each word in the log event and n is the total number of words in the log message. Logs with unusually high or low word counts compared to typical entries may reflect anomalous system behavior or errors.

(6) Character Count Feature (CCF). The Character Count feature measures the total number of characters in a log message [51]. Logs that deviate significantly from typical length distributions—either unusually short or excessively long—may indicate anomalies. Such deviations can signal irregularities, including truncated messages, excessive verbosity, or unexpected data insertions, making this feature valuable for anomaly detection. For a log event d_l, the Character Count feature is defined as:

$$\text{Character Count}(d_l) = \sum_{i=1}^{n} c_i \tag{7}$$

where c_i represents each character in the log message and n is the total number of characters in the log message. Logs with significantly higher or lower character counts compared to typical entries may reflect irregularities such as truncated messages, excessive verbosity, or unexpected data injections, which are critical for anomaly detection.

(7) Temporal Feature (TF). The Temporal feature captures time-related attributes of log events, including year, month, day, hour, minute, and second [43]. These features provide insight into the temporal distribution of events and are critical for identifying anomalies associated with irregular timings. Unusual patterns, such as spikes in activity during non-peak hours, missing log entries during expected periods, or other timing irregularities, may indicate system malfunctions, security breaches, or performance issues. Combining multiple temporal aspects—year, month, day, and hour—can offer particularly meaningful insights for datasets that span shorter time periods. For a log event d_l, the Temporal Features are represented as:

$$\text{Temporal Feature}(d_l) = (\text{Year, Month, Day, Hour, Minute, Second}) \tag{8}$$

where (Year) is the year of the log event, (Month) represents the month of the log event, (Day) corresponds to the day of the log event, and (Hour), (Minute), and (Second) indicate the specific time of day when the event was logged.

Feature Configuration and Tuning. NovaAD configures and tunes the features by customizing their extraction methods based on specific parameters. This approach enables the automation of our model and allows it to choose the best parameters independently, thus ensuring the highest possible accuracy.

SBERT feature. The *BERT* model produces a 768-dimensional embedding vector. To improve computational efficiency while preserving important patterns, we apply Principal Component Analysis (*PCA*) to the BERT embeddings. To determine the optimal number of components (r) for *PCA*, we explore a range of values and select the one that maximizes *explained variance*. The explained variance is calculated as the cumulative sum of the explained variance ratios for the selected components [18]. At each iteration, we compare the current explained variance with the highest observed value. After completion of the iterations, the best r and its associated explained variance are identified. The explained variance ratio for the r-th principal component is given by:

$$\text{explained\_variance\_ratio}_r = \frac{\lambda_r}{\sum_{i=1}^{z} \lambda_i} \tag{9}$$

where λ_r is the eigenvalue corresponding to the r-th principal component. λ_i is the eigenvalue corresponding to the i-th principal component, and i ranges from 1 to z, where z is the total number of dimensions (features) in the dataset. $\sum_{i=1}^{z} \lambda_i$ is the sum of all eigenvalues, representing the total variance of the data.

Topic modeling based feature. We propose an automatic approach to determine the optimal number of topics for each dataset, with the goal of optimizing the topic modeling process. Given the diversity of datasets containing various textual log events, we define a range of potential values for the number of topics. Subsequently, we apply *Grid Search* to the *LDA* model to identify the best parameters. This method enables NovaAD to automatically select the most appropriate number of topics for each dataset, ensuring optimal topic modeling results.

Table 3: Optimal Settings for SBERT and Topic Features.

Hyperparams	HDFS	BGL	TH (1G)	TH (2G)	SP (100MB)	SP (150MB)
Number of Topics	3	4	5	6	7	8
PCA Components	20	30	35	40	40	40

Log Event Feature Vector. For each log event d_l, we finally consolidate the seven introduced features into a single feature vector $Features(d_l)$. In the following Sections 4.2 and 4.2, we describe how the feature vector is further configured and transformed:

$$Feaures(d_l) = [TopicIndex(d_l)_{\text{dominant}}, Entropy(d_l), SBERT(d_l), Sentiment(d_l), WordCounts(d_l), CharacterCounts(d_l), Temporal(d_l)]$$

Feature Transformation. Feature transformation is a fundamental component of feature engineering and encompasses widely used techniques such as standardization and encoding [53, 54]. In this work, we apply the *Standard Scaler* to normalize the numerical features, transforming them to have a mean of 0 and a standard deviation of 1. Furthermore, we use a *Label Encoder* for categorical features to transform feature categories into integers ranging from 1 to N, where N is a feature cardinality. These steps are necessary because they prepare and optimize the input features of the learning algorithm; they address scale, distribution, and nonlinearity issues to improve the accuracy, interpretability, and robustness of the model.

4.3 Feature Vector Aggregation

This phase constructs log-sequence feature vectors by computing the mean of log-event vectors within each sequence, thereby summarizing the characteristics of the entire log-sequence. To aggregate the features of the log sequence, we calculate the mean of the vectors of the characteristic of the individual log event within each sequence [3]. The choice of mean aggregation is motivated by its ability to normalize feature values across sequences of varying lengths, ensuring stability and robustness in anomaly detection tasks. The mean provides a balanced representation of the sequence while preserving essential structural information. Furthermore, averaging reduces the influence of noisy or transient fluctuations while retaining the central operational pattern of the sequence, which has been shown to improve robustness in sequence-based anomaly detection frameworks [14, 75].

4.4 Novelty Detection and Label Estimation

This phase aims to identify data points that significantly deviate from the patterns learned during training. The model is trained in a semi-supervised manner using only normal instances, with no anomalous data included in the training set. The novelty detection results are then evaluated using an unsupervised score, without requiring ground-truth labels. Following the feature aggregation step, we adopt a novelty detection approach using a One-Class Support Vector Machine (OC-SVM) [60] to assign pseudo-labels to previously unlabeled log sequences in the training set. The OC-SVM is trained exclusively on subsets of normal log sequences, learning a decision boundary that characterizes the distribution of normal system behavior. Once trained, the model is applied to unlabeled sequences, classifying them as normal (if they fall within the learned boundary) or anomalous (if they deviate significantly). The advantage of this approach lies in its ability to generalize normal behavior from known examples while identifying outliers without requiring explicit anomaly labels. The main stages of novelty detection are described below.

Stage 1. Hyperparameter Optimization for One-Class SVM. In terms of the One-Class Support Vector Machine (One-Class SVM), two important hyperparameters are `gamma` and `nu`, which control the behavior of the algorithm. Gamma (γ): This parameter defines the influence of each individual training example. A small value of γ implies a smoother decision boundary, where the model considers points farther away from the hyperplane. A large value of γ leads to a more complex decision boundary, which may result in overfitting. Nu (ν): The parameter ν controls the trade-off between maximizing the margin and minimizing classification errors. It also defines the upper bound of the fraction of margin errors and the lower bound on the fraction of support vectors. To optimize the performance of the One-Class Support Vector Machine (OC-SVM) to label unlabeled log sequences in the training set, we performed a grid search over a range of parameters γ and ν. Specifically, the search explored the values of γ between 0.2 and 0.5 and ν between 0.01 and 0.08, using five evenly spaced intervals for each. For each (γ, ν) combination, the OC-SVM model was trained solely on normal log sequences. The predictions were then made on the unlabeled set, classifying instances as normal (+1) or anomalous (−1). To assess the effectiveness of each parameter configuration in the absence of ground truth labels in this stage, we use a hybrid scoring function that integrates both *local* and *global* anomaly detection scores. The Local Outlier Factor (LOF) score [2, 8] measures the local density deviation of data points classified as normal, while the Silhouette score [58, 61] assesses the global quality of the clustering based on the predicted labels of OC-SVM. The score is defined as follows:

$$\text{Hybrid\_Score} = \alpha \cdot \frac{1}{n} \sum_{i=1}^{n} \text{LOF}_i + (1 - \alpha) \cdot \text{Silhouette} \tag{10}$$

where LOF_i is the Local Outlier Factor score for the i-th sample, n is the total number of samples, Silhouette is the silhouette score that evaluates the separation of the cluster based on the predicted OC-SVM labels, and $\alpha \in [0, 1]$ is a weighting factor (e.g. $\alpha = 0.5$ for equal weighting between LOF and Silhouette). Using only one of these scores has limitations. LOF focuses on local neighborhoods and may not capture global structure, while the Silhouette score may overlook subtle anomalies in dense regions. Therefore, combining both allows us to balance the sensitivity of local outlier detection with the broader structure revealed by clustering based on the predicted labels of OC-SVM. The configuration with the highest Hybrid_Score is selected as the best performing model and used in the final evaluation phase.

Stage 2. Training a one-class support vector machine. In this stage, the optimal (γ, ν) parameters identified in Stage 1 are used to train a One-Class Support Vector Machine (OC-SVM) model on the subset of training data labeled as normal log sequences. The model learns a decision boundary that characterizes normal behavior, enabling it to detect deviations from this boundary during inference - such deviations are likely indicative of anomalies.

Stage 3. Inferring pseudo-labels. In this stage, the trained One-Class Support Vector Machine (OC-SVM) model is used to assign pseudo-labels to the unlabeled log sequences within the training set. The model assesses each sequence based on the previously learned decision boundary that characterizes normal system behavior.

Based on the results of the novelty detection phase, log sequences classified with a label of −1 are designated as anomalies, while all remaining sequences are considered normal. Consequently, we used this result to assign labels to unlabeled instances within the training set. The estimation of the labels 1 or 0 for the unlabeled log sequences in the training set is derived from the pseudo-labels generated by the One-Class Support Vector Machine (OC-SVM) described in Section 4.4 (Stage 3). Sequences classified as negative are

regarded as potential anomalies (novelties), whereas sequences classified as positive are considered consistent with normal behavior. The labeling process follows the following criterion:

$$\hat{y} = \begin{cases} 1, & \text{if OC-SVM decision} = -1 \quad \text{(novelty detected)} \\ 0, & \text{otherwise (classified as normal)} \end{cases} \tag{11}$$

where $\hat{y}$ represents the assigned label, with 1 denoting an anomaly and 0 indicating a normal log sequence. Through this label estimation strategy, all previously unlabeled training data are assigned labels. This process enables the application of supervised learning techniques for subsequent anomaly classification, effectively bridging the gap between unsupervised and supervised approaches by converting ambiguous log sequences into structured, labeled training instances.

4.5 Anomaly Detection

In the final phase, the labeled data obtained from phases 4 and 5, together with a set of normal data, are used within an ensemble learning framework. This approach harnesses the strengths of supervised learning to improve the accuracy of anomaly classification, allowing a more effective distinction between normal and anomalous instances. For the anomaly detection stage, we construct a labeled training dataset by combining the outputs from the Novelty Detection and Label Estimation processes (Sections 4.4 and **??**) with the subset of normal data. In this study, we propose a hierarchical ensemble learning framework to improve predictive performance on the target task. The approach combines *Voting Classifier* [59] and *Stacking Classifier* [1] to take advantage of the strengths of multiple machine learning models. First, a *Voting Classifier* is used to aggregate predictions from two base models: XGBoost [11] and Logistic Regression [80]. Soft voting is used, which means that the predicted probabilities of the base models are averaged to make the final prediction. This combination allows the ensemble to benefit from XGBoost's ability to capture complex nonlinear patterns while incorporating Logistic Regression's robustness and interpretability. Next, a *Stacking classifier* is used to further improve the generalization. The *Voting Classifier* described above is treated as a base learner, and a Random Forest model is added as a second base learner. The output of these base models is then fed into a Logistic Regression *meta-model*, which produces the final prediction. This hierarchical stack captures complementary information from diverse algorithms, allowing the model to generalize better across different types of pattern in the data. The selection of models is intentional and grounded in their complementary strengths. XGBoost was chosen for its high precision and ability to capture complex nonlinear relationships. Logistic Regression served as a base model in the voting ensemble, providing a stable linear baseline, while Random Forest added diversity in the stacking framework to detect patterns the voting ensemble might miss. Finally, Logistic Regression was used as the metamodel to combine the strengths of the base learners, ensuring interpretability and reducing overfitting. In general, this hierarchical ensemble approach is designed to maximize predictive accuracy while leveraging the complementary strengths of different machine learning algorithms. The main stages of anomaly detection are outlined below:

Stage 1. Hyperparameter Optimization for Classification. To enhance the predictive performance of the XGB Classifier [11] and the RandomForest Classifier [45], we performed hyperparameter tuning using *RandomizedSearchCV* [38]. For XGBoost, the search space comprised *max_depth*, *n_estimators*, *learning_rate*, *subsample*, and *colsample_bytree*; for Random Forest, it included *max_depth*, *n_estimators*, *max_features*, and *min_samples_split*. These parameters were chosen because of their strong influence on model complexity, regularization, and generalization. The search was carried out with ten iterations and evaluated using five-fold stratified cross-validation, using the F_1 score to balance precision and recall in the presence of class imbalance. This design provided a practical trade-off between parameter space exploration and computational efficiency. For logistic regression, the default hyperparameters were retained, as preliminary experiments indicated negligible gains from the tuning [62, 74]. Given its limited parameter set and linear decision boundary, tuning efforts were concentrated on the more complex, nonlinear classifiers.

Stage 2. Model Training. Following hyperparameter optimization, the best parameter configurations identified in Stage 1 are applied to the base models. The XGB Classifier is initialized with its optimal values for `max_depth`, `n_estimators`, `learning_rate`, `subsample`, and `colsample_bytree`, while the RandomForest Classifier is trained using the tuned settings of `max_depth`, `n_estimators`, `max_features`, and `min_samples_split`. All models are fixed with a random seed to ensure reproducibility. These optimized classifiers are then integrated into an ensemble framework. First, the XGB Classifier and a Logistic Regression model are combined in a Voting Classifier with a soft voting strategy, where the predicted probabilities from the models are averaged to produce the final output. This takes advantage of the complementary strengths of linear and nonlinear learners to improve predictive robustness. Building on this, a Stacking Classifier is constructed by incorporating the Voting classifier and the optimized randomForest classifier as base learners. Their output probabilities serve as meta-features for a Logistic Regression meta-model, which learns to make the final classification decision. By combining models with diverse learning biases, the stacking strategy captures richer patterns and often outperforms individual classifiers.

Stage 3. Prediction. Once the stacking ensemble is trained, it can be applied to unseen data to generate predictions. The ensemble consists of multiple *base models* and a *meta-model*. In this setup, the base models include a `VotingClassifier`, which itself combines multiple classifiers using *soft voting*, and additional individual classifiers, such as a random forest. For each test sample, the prediction process proceeds in two stages: (1) Base model predictions: Each base model produces a prediction or class probability for the sample. In the case of the `VotingClassifier`, each internal classifier outputs class probabilities, which are averaged to produce a single prediction from the set. (2) Meta-model prediction: The predictions from all base models are collected and used as input features for the *meta-model*, a logistic regression in this case. The meta-model learns how to optimally combine the base model output to produce the final class prediction. This hierarchical approach enables the stacking ensemble to take advantage of the strengths of individual classifiers while mitigating their weaknesses. The final predictions

are then evaluated using metrics such as *precision*, *recall*, and *F1-score*, providing a comprehensive assessment of the model's ability to balance false positives and false negatives.

5 EVALUATION

We deploy our approach on datasets from the log hub [28] for evaluation. We test the performance in terms of accuracy and run-time of our anomaly detection approach in multiple steps and answer the following questions.

(1) **Q1**: How effectively does NovaAD perform in terms of accuracy in anomaly detection?
(2) **Q2**: How efficient is NovaAD in terms of utilization of computational resources and execution time?
(3) **Q3**: What is the impact of different features of NovaAD on its overall effectiveness?
(4) **Q4**: How robust is NovaAD when handling unstable logs with evolving patterns and noise?

5.1 Implementations and Environments

All experiments were implemented in Python 3.11. We reproduced and evaluated the baseline methods—LogRobust, *CNN, NeuralLog, LogAnomaly, DeepLog, and PLELog—under* a unified experimental setup on a dedicated server running Ubuntu Linux. The server is equipped with an AMD EPYC 7513 CPU (32 cores, 64 threads, up to 2.6 GHz), 256 GB of RAM, and an NVIDIA A10 GPU with 24 GB VRAM.

To ensure fair, controlled, and reproducible comparisons, both the proposed method and all baseline models were evaluated using identical log sequences (blocks) across all experimental settings. Specifically, the same block-level sequences were used to construct the training (60%), validation (10%), and testing (30%) splits for every method. Experiments were conducted under two hardware configurations—CPU-only and GPU-accelerated—while keeping all data partitions and configurations fixed. To assess the stability and variability of the results, each experiment was repeated *three times* under identical settings, and the observed performance deviations were analyzed accordingly.

5.2 Datasets

This study evaluates NovaAD on four widely adopted public datasets for log-based anomaly detection: the HDFS dataset (Hadoop Distributed File System), the BGL dataset (Blue Gene/L supercomputer), selected subsets of the Thunderbird dataset (TH_1G and TH_2G), and the Spirit dataset (SP_100MB and SP_150MB). These datasets have been extensively used in prior studies on log-based anomaly detection [12, 14, 26, 27, 49, 52, 65, 76, 78]. Since BGL, Thunderbird, and Spirit do not include explicit tags such as block IDs for sequence extraction, we follow previous studies [14, 49, 76] and employ a fixed sliding window strategy with a window size of 120. For HDFS dataset, log sequences can be directly extracted using the block IDs included in the messages.

BGL dataset consists of 4, 747, 963 log messages generated by the Blue Gene/L supercomputer, which features 128, 000 processors and was deployed at the Lawrence Livermore National Laboratory over seven months. Each log message was manually labeled as normal or anomalous by domain experts, with 348,458 messages identified as anomalous. Log messages are first ordered by generation timestamp, and the associated node is considered when forming sequences. A log sequence is classified as anomalous if it contains at least one anomalous log message. Using this approach, our method extracts 48, 652 normal sequences and 36, 274 anomalous sequences from the dataset.

HDFS dataset comprises 11, 175, 629 log messages generated by executing Hadoop-based MapReduce jobs on more than 2,000 Amazon *EC2* nodes over 38.7 hours. In total, there are 558, 223 normal log sequences and 16, 838 anomalous log sequences in HDFS.

Thunderbird dataset is an open dataset of logs collected from a Thunderbird supercomputer system at Sandia National Laboratories (SNL) in Albuquerque, consisting of 9, 024 processors and 27,072 GB of memory. The logs contain both alert and non-alert messages, identified using alert category tags. In the first column, a "–" indicates a non-alert message, while all other entries correspond to alert messages. For our experiments, we consider two subsets: TH_1G, which contains 6, 013, 029 log messages, including 44, 732 normal log sequences and 7, 601 anomalous log sequences; and TH_2G, which contains 13, 451, 157 log messages, including 105, 532 normal log sequences and 9, 095 anomalous log sequences.

Spirit dataset was collected from the Spirit supercomputer at Sandia National Laboratories (SNL) and contains over 172 million log messages, each labeled normal ("–") or abnormal (any other entry). For this study, we used two subsets: SP_100MB, which includes 811, 723 log messages, comprising 4, 060 normal log sequences and 2, 927 anomalous log sequences; and SP_150MB, which includes 1, 225, 059 log messages, comprising 5, 712 normal log sequences and 4, 792 anomalous log sequences.

Consistent with previous studies [14, 27, 40, 49, 70, 76, 78], we divided each data set into a training set, a validation set, and a test set, following a 6: 1: 3 ratio. This approach was used to evaluate the performance of the log-based anomaly detection method. Existing studies [14, 49, 78], typically shuffle all log sequences before splitting the dataset, a method that helps to avoid the occurrence of unstable log data. Following the methodology of PLELog [76], we partition the dataset chronologically to ensure that all log sequences in the training set temporally precede those in the test set. This approach more accurately reflects real-world scenarios. Furthermore, research in the field of software development practice prediction [67] has emphasized the importance of chronological splitting to prevent data leakage. To evaluate our semi-supervised method, NovaAD, and following the approach of PLELog [76], we simulated a semi-supervised setting by selecting 50% of the normal training logs as labeled normal sequences, while treating the remaining sequences (both normal and anomalous) as unlabeled.

5.3 Evaluation Metrics

F_1*-score* is a standard evaluation metric for anomaly detection and is defined as the harmonic mean of precision and recall [55]. It is particularly suitable for imbalanced datasets, as it balances false positives and false negatives:

$$F1\text{-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{12}$$

Precision and *Recall* are computed as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{13}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{14}$$

where TP, FP, and FN denote true positives, false positives, and false negatives, respectively.

5.4 Baselines for Performance Evaluation

As NovaAD adopts a semi-supervised learning approach, we compared it with various state-of-the-art supervised, semi-supervised and unsupervised log-based anomaly detection techniques:

Supervised Anomaly Detection Approach.

CNN [46] propose a convolutional neural network (CNN)-based approach for anomaly detection in large-scale system logs. Their method leverages logkey2vec embeddings combined with multiple one-dimensional convolutional layers, followed by dropout and max-pooling, to automatically capture event relationships within log sequences.

NeuralLog [35] is a log-based anomaly detection approach that eliminates the need for explicit log parsing. NeuralLog captures the semantic information of the raw log messages and encodes them into semantic vector representations. Subsequently, these vectors are utilized by a Transformer-based classification model to detect anomalies, enabling effective modeling of contextual dependencies within log sequences.

LogRobust [78] transforms log events into semantic vector representations by extracting their underlying meaning. To detect anomalies, it uses an attention-based Bi-LSTM model, which effectively captures contextual relationships within log sequences. The attention mechanism enables the model to automatically weigh the significance of different log events. This design allows LogRobust to effectively recognize and manage variability and instability in both individual log events and entire sequences.

Unsupervised Anomaly Detection Approach.

LogAnomaly [48] shares similarities with Deeplog in that it also learns normal log patterns from standard log sequences. However, the key difference is that LogAnomaly represents log sequences using semantic information, as opposed to the one-hot encoding used by Deeplog. This semantic representation improves prediction effectiveness. Additionally, LogAnomaly tracks the frequency of log events during representation, enabling it to detect anomalies based on irregularities in the count of log events.

Deeplog [14] treats log sequences as natural language sequences and employs a deep neural network, specifically Long Short-Term Memory (LSTM) networks, to learn normal log patterns from typical log sequences. During the anomaly detection process, Deeplog predicts the next log event. If the actual log-event does not appear in the top prediction results, it is considered an anomaly.

Semisupervised Anomaly Detection Approach.

PLELog [76] a novel and practical log-based anomaly detection approach that takes advantage of a semi-supervised learning framework to eliminate the need for manual, time-consuming labeling. PLELog incorporates historical anomaly knowledge through probabilistic label estimation, thereby enhancing the effectiveness of supervised approaches. Additionally, PLELog is robust against unstable log data by utilizing semantic embedding and detects anomalies efficiently and accurately through the design of an attention-based GRU neural network.

6 RESULTS AND ANALYSIS

6.1 Log Parser Consistency and Extracted Sequences

To evaluate the consistency and reliability of the drain [25] log parser, we executed it three times for each dataset using the default configuration for each dataset. For each run, we recorded the total number of log messages, the number of normal log sequences, and the number of anomalous log sequences. Across all runs, the parser produced identical results, indicating that the extraction process is stable and robust, and is not affected by repeated executions or variations in data handling. Table 4 presents a summary of the datasets and the corresponding extracted sequences. As observed, the parser consistently identifies the same number of normal and anomalous sequences for each dataset, demonstrating reproducibility of the parsing process. This consistency ensures that downstream analyses, including anomaly detection, receive reliable and reproducible input data, eliminating potential variability.

Table 4: Summary of log datasets and extracted sequences.

Dataset	# Logs	Normal Seq.	Anom. Seq.	Seq
BGL	4,747,963	48,652	36,274	Sliding
HDFS	11,175,629	558,223	16,838	BlockID
Thunderbird 1G	6,013,029	44,732	7,601	Sliding
Thunderbird 2G	13,451,157	105,532	9,095	Sliding
Spirit 100MB	811,723	4,060	2,927	Sliding
Spirit 150MB	1,225,059	5,712	4,792	Sliding

6.2 Effectiveness of NovaAD

This section evaluates the effectiveness and robustness of NovaAD in detecting anomalous log sequences under controlled and reproducible experimental conditions. We compare NovaAD against established baseline methods using fixed block-level log sequences and standardized evaluation metrics. To account for potential variability, all experiments were conducted using identical data partitions and repeated three times, with the F1-score deviation across runs analyzed to assess result stability. The results presented below provide a quantitative comparison of detection performance and highlight the empirical behavior of NovaAD relative to competing approaches. Table 5 compares the proposed NovaAD with representative supervised (LogRobust, CNN, NeuralLog), unsupervised (LogAnomaly, DeepLog), and semi supervised (PLELog) log anomaly detection approaches across six benchmark datasets: HDFS, BGL, Thunderbird (1G and 2G), and Spirit (100MB and 150MB). Performance is evaluated using precision, recall, F1 score, and F1 standard deviation (F1 std), with results averaged over three independent runs.

Among supervised methods, CNN and LogRobust achieve strong F1-scores across most datasets, benefiting from full label supervision. CNN attains the highest F1-scores on several datasets, including BGL and Spirit, while NeuralLog exhibits unstable performance on some datasets, particularly HDFS and Spirit, as reflected by lower F1-scores and higher variance.

Unsupervised methods, namely LogAnomaly and DeepLog, show limited robustness on complex datasets. Although they occasionally achieve high precision, their recall drops significantly on the Thunderbird datasets, leading to very low F1 scores and indicating difficulty in capturing diverse anomaly patterns without supervision.

The semi supervised method PLELog improves recall compared to unsupervised approaches; however, it still suffers from noticeable precision–recall imbalance and inconsistent performance across datasets, especially on large-scale logs.

In contrast, NovaAD achieves consistently high F1 scores across all datasets and exhibits low F1 std values, indicating stable performance across repeated runs. While NovaAD does not always achieve the absolute highest F1 score compared to fully supervised models such as CNN, it remains highly competitive and significantly outperforms unsupervised and semi-supervised baselines, including PLELog. These results demonstrate that NovaAD effectively balances detection accuracy and robustness without relying on extensive labeled data, making it suitable for practical log anomaly detection scenarios.

2) Q2-Efficiency of NovaAD. Beyond detection performance, the practical adoption of a log anomaly detection method critically depends on its computational efficiency. Tables 6 and 7 report the training and testing runtime in minutes of NovaAD and several baseline methods on both CPU and GPU platforms across six benchmark datasets, including HDFS, BGL, Thunderbird (TH), and Spirit (SP). Each reported value represents the mean and standard deviation over three independent runs.

CPU Runtime Analysis. On CPU, fully supervised methods—LogRobust, CNN, and NeuralLog—exhibit **substantial training overhead**, particularly on large-scale datasets such as HDFS and Thunderbird. NeuralLog demonstrates the highest computational cost, with training exceeding **1,000 minutes** on HDFS, reflecting the complexity of its deep sequential modeling. While these methods achieve high detection accuracy, their extensive training requirements may limit applicability in large or frequently evolving systems.

Unsupervised methods show heterogeneous behavior. DeepLog achieves extremely low CPU runtime (a few minutes), but detection performance suffers on complex datasets. LogAnomaly incurs substantial training and testing cost on large logs, particularly for Thunderbird (2G), where training exceeds **696 minutes** and testing exceeds **4 minutes**, indicating reduced scalability.

Semi-supervised PLELog reduces training cost relative to fully supervised approaches but still requires non-negligible computation during both training and inference, potentially affecting responsiveness in online settings.

NovaAD demonstrates **consistently low computational overhead** across all datasets. CPU training time remains small (<25 minutes), and inference time is nearly constant (~0.01–0.06 minutes, i.e., ~0.6–3.6 seconds). Low standard deviation across multiple runs indicates **stable and predictable runtime**, highlighting suitability for practical deployment scenarios requiring fast model preparation and low-latency inference.

GPU Runtime Analysis. GPU acceleration significantly reduces training times for computationally intensive methods. For example, NeuralLog training on HDFS decreases from 1,163 minutes (CPU) to 174 minutes (GPU), an order-of-magnitude speedup. Supervised methods generally still require more time than other paradigms, but GPU acceleration makes them more feasible for large-scale logs.

Unsupervised methods achieve extremely fast training and inference on GPU. DeepLog training on HDFS drops below 1 minute, and inference remains <0.02 minutes (~1.2 seconds) across datasets. LogAnomaly also benefits from GPU acceleration, though training on Thunderbird (2G) remains ~46 minutes.

Semi-supervised PLELog gains moderate speedups (~2×), whereas NovaAD shows only slight differences between CPU and GPU runtimes. Training times remain <25 minutes, and inference is near 0.01 minutes (~0.6 seconds). This demonstrates **hardware-agnostic efficiency**, suitable for both high-performance servers and resource-constrained systems.

CPU vs GPU Comparison.

- **Training Efficiency:** NovaAD is consistently fast across CPU and GPU. CPU training is often 2–5× faster than PLELog and 5–50× faster than supervised methods on large datasets. GPU acceleration provides minor additional improvements, reflecting efficient algorithm design.
- **Inference Latency:** Inference times are near-constant (~0.01–0.06 minutes), outperforming all baseline methods in low-latency scenarios.
- **Stability:** Low standard deviation indicates consistent performance regardless of dataset size or hardware.
- **Scalability:** CPU runtime scales gracefully with dataset size, whereas LogAnomaly and supervised methods show dramatic increases for large logs (TH 2G).

Summary. NovaAD combines computational efficiency, low-latency inference, and scalability across CPU and GPU platforms, making it highly suitable for **large-scale, real-time, or continuous log anomaly detection** deployments.

Beyond detection performance, the practical adoption of a log anomaly detection method critically depends on its computational efficiency. This section evaluates the runtime and scalability of NovaAD on both CPU and GPU platforms, with all experiments including baseline methods—executed on the same server to ensure fair and consistent comparisons. By analyzing execution time and resource utilization under these hardware configurations, we provide insights into the feasibility of deploying NovaAD in large-scale real-world log analysis scenarios, highlighting its suitability for both high-performance and resource-constrained environments. Table 6 summarizes the average CPU training and testing runtime of NovaAD and baseline methods on six benchmark datasets. All measurements are reported as the mean and standard deviation over three independent executions.

Fully supervised approaches, including LogRobust, CNN, and NeuralLog, exhibit substantial training overhead, particularly on large-scale datasets such as HDFS and Thunderbird. Among them,

Table 5: Anomaly detection results on HDFS, BGL, Thunderbird (TH), and Spirit (SP) datasets. Type abbreviations: Sup. = Supervised, Unsup. = Unsupervised, Semi-sup. = Semi-supervised. Each method was run three times per dataset, and the reported values are the averages. The F1-std column shows the standard deviation of F1 scores across the three runs.

	HDFS				BGL				TH (1G)				TH (2G)				SP (100MB)				SP (150MB)			
	Pre	**Rec**	**F1**	**F1-std**	**Pre**	**Rec**	**F1**	**F1-std**	**Pre**	**Rec**	**F1**	**F1-std**	**Pre**	**Rec**	**F1**	**F1-std**	**Pre**	**Rec**	**F1**	**F1-std**	**Pre**	**Rec**	**F1**	**F1-std**
Supervised																								
LogRobust	0.933	0.949	0.942	±0.0008	0.984	0.903	0.941	±0.021	0.998	0.994	0.996	±0.001	0.998	0.989	0.993	±0.001	0.953	0.966	0.960	±0.008	0.987	0.960	0.973	±0.008
CNN	0.960	0.987	0.974	±0.001	0.997	0.994	0.996	±0.00	0.999	0.996	0.998	±0.00	0.998	0.990	0.994	±0.001	1.0	0969	0982	±0.002	1.0	0.967	0.983	±0.008
NeuralLog	0.394	0.315	0.354	±0.002	0.926	0.827	0.874	±0.002	0.997	0.992	0.995	±0.001	0.998	0.978	0.988	±0.001	0.706	0.696	0.608	±0.379	0.902	0.857	0.879	±0.026
Unsupervised																								
LogAnomaly	0.985	0.233	0.382	±0.0008	0.904	0.865	0.821	±0.154	0.247	0.0029	0.005	±0.0005	0.527	0.011	0.022	±0.003	0.976	1.0	0.987	±0.002	0.978	1.0	0.988	±0.002
DeepLog	0.984	0.234	0.3804	±0.0005	0.996	0.731	0.830	±0.146	0.216	0.003	0.006	±0.0006	0.140	0.013	0.156	±0.111	0.955	1.0	0.976	±0.003	0.956	1.0	0.978	±0.001
Semi-supervised																								
PLELog	0.967	0.723	0.827	±0.008	0.701	0.986	0.842	±0.03	0.286	0.985	0.441	±0.02	0.198	0.992	0.331	±0.01	0.894	0.824	0.854	±0.05	0.869	0.857	0.860	±0.04
NovaAD	**0.967**	**0.994**	**0.981**	±0.006	**0.956**	**0.986**	**0.971**	±0.006	**1.0**	**0.996**	**0.997**	±0.002	**1.0**	**0.987**	**0.993**	±0.002	**0.841**	**0.958**	**0.893**	±0.05	**0.918**	**0.950**	**0.932**	±0.001

NeuralLog shows the highest computational cost, reflecting the complexity of its deep sequential modeling. While these methods achieve strong detection accuracy, their high training cost may limit applicability in large or frequently evolving systems.

The unsupervised methods display heterogeneous behavior. DeepLog achieves very low runtime but, as shown in the accuracy evaluation, suffers from limited detection capability on complex datasets. LogAnomaly, on the other hand, incurs considerable training and testing cost on larger logs, especially for Thunderbird (2G), indicating reduced scalability.

The semi-supervised method PLELog lowers the training cost relative to fully supervised models but still requires non-negligible computation during both training and inference, which can affect responsiveness in online settings.

NovaAD demonstrates consistently low computational overhead across all datasets. Its training time remains small even as data volume increases, and its inference time is close to constant across datasets. In addition, the low standard deviation observed for both training and testing indicates stable runtime behavior across multiple runs.

These findings show that NovaAD maintains efficient execution while scaling to large log volumes, providing fast model preparation and low-latency inference suitable for practical deployment scenarios such as real-time monitoring and continuous anomaly detection.

6.3 Feature Ablation Study of NovaAD

To assess the contribution of each feature, we use Mutual Information (MI) [17, 72], a non-linear measure that quantifies the dependency between a feature and the target variable. MI operates on probability distributions, evaluating how much the joint distribution of a feature and the target deviates from the assumption of independence. The mutual information between a feature X and the target variable Y is formally defined as:

$$MI(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{15}$$

where $P(x,y)$ denotes the joint probability distribution of X and Y, and $P(x)$ and $P(y)$ represent their respective marginal probability distributions. **??** and **??** show the MI-based feature importance analysis for the BGL and HDFS dataset, respectively, revealing a hierarchical ranking of features according to their contribution to the MI score. Mutual Information (MI) scores were scaled by multiplying with a constant factor to make them more interpretable and easier to visualize. Although the absolute values were adjusted, the relative ranking among features remains unchanged, and this ranking guides the interpretation.

Figure 3 illustrates the Mutual Information (MI) based importance of all extracted features across six log datasets (BGL, HDFS, TH_1G, TH_2G, SP_100MB, and SP_150MB). The evaluated features include semantic features (Dominant Topic and SBERT embeddings), statistical and lexical features (entropy, word count, and character count), sentiment polarity, and temporal features (year, month, day, hour, minute, and second).

Overall, entropy, SBERT embeddings, word count, and character count consistently achieve high MI scores across datasets, indicating strong discriminative capability in identifying anomalous log messages. The Dominant Topic feature also contributes non-trivially, capturing coarse-grained semantic structure in log streams. In contrast, sentiment exhibits moderate MI values, suggesting that while emotional polarity alone is not a dominant indicator, it provides complementary information when combined with other semantic and lexical cues.

Temporal features demonstrate dataset-dependent behavior. In smaller or less temporally complex datasets (BGL, HDFS, and TH_1G), features such as year, month, day, hour, minute, and second exhibit comparatively lower MI scores, implying limited standalone predictive power. However, in larger and more dynamic workloads (TH_2G, SP_100MB, and SP_150MB), all temporal components—including fine-grained features such as minute and second—show substantially increased MI values. This indicates that temporal execution patterns and periodic behaviors become increasingly informative as system scale and workload variability grow.

Notably, SBERT embeddings maintain consistently high MI scores across all datasets, highlighting their robustness and cross-dataset generalizability. Similarly, the stable contribution of lexical features (word and character counts) across datasets suggests that message length and structural complexity are reliable indicators of anomalous behavior. In the Spark datasets (SP_100MB and SP_150MB), the uniformly high MI scores across semantic, lexical, sentiment, and temporal features indicate that anomaly signals are distributed across multiple feature dimensions.

Overall, this analysis demonstrates that no single feature family is sufficient across all operational settings, motivating the proposed hybrid feature representation that jointly models semantic content,

Table 6: Training and testing runtime on CPU for HDFS, BGL, Thunderbird (TH), and Spirit (SP) datasets. Each method was run three times per dataset, and the reported values are the averages of the runs.The std values shows the standard deviation of runtime across the three runs.

	HDFS		BGL		TH (1G)		TH (2G)		SP (100MB)		SP (150MB)	
	Train/std	Test/std	Train/std	Test/std	Train/std	Test/std	Train/std	Test/std	Train/std	Test/std	Train/std	Test/std
Supervised												
LogRobust	266.16±1.05	0.49±0.02	14.58±0.3	0.37±0.0	17.35±1.5	0.41±0.006	39.62±2.07	1.01±0.0	2.580±0.14	0.07±0.0	3.89±0.20	0.10±0.0
CNN	313.96±9.15	0.35±0.03	15.54±0.3	0.27±0.0	16.30±1.5	0.29±0.03	36.83±2.24	0.74±0.01	2.29±0.05	0.05±0.0	3.91±0.29	0.08±0.00
NeuralLog	1163.21±2.65	0.58±0.02	72.94±1.5	0.52±0.06	89.05±4.56	0.62±0.02	210.37±0.19	1.52±0.06	13.57±0.16	0.11±0.00	20.31±0.11	0.17±0.02
Unsupervised												
LogAnomaly	116.90±0.07	0.04±0.00	7.49±0.3	0.11±0.01	179.18±0.4	1.2±0.02	696.51±8.44	4.420±0.20	3.30±0.09	0.03±0.00	5.26±0.38	0.06±0.00
DeepLog	7.26±0.07	0.48±0.01	4.12±0.3	0.04±0.00	9.02±0.08	0.04±0.00	9.02±0.0	0.04±0.00	1.22±0.08	0.01±0.00	1.61±0.04	0.01±0.00
Semi-supervised												
PLELog	64.99±0.45	0.72/±0.03	29.10/±0.36	0.26/±0.00	23.74±0.2	0.27±0.006	51.19±0.06	0.59/±0.00	3.17 ±0.02	0.04±0.00	4.76±0.02	0.05/±0.00
NovaAD	**22.91**±0.6	**0.06**±0.00	**4.22**±0.08	**0.01**±0.00	**3.10**±0.04	**0.01**±0.00	**5.49**±0.1	**0.02**±0.00	**0.80**±0.05	**0.01**±0.00	**1.90**±0.03	**0.01**±0.00

Table 7: Training and testing runtime on GPU for HDFS, BGL, Thunderbird (TH), and Spirit (SP) datasets. Each method was run three times per dataset, and the reported values are the averages of the runs.

	HDFS		BGL		TH (1G)		TH (2G)		SP (100MB)		SP (150MB)	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Supervised												
LogRobust	164.88±13.48	0.37±0.00	7.54±0.02	0.27±0.00	9.71±0.05	0.31±0.01	21.8±0.63	0.71±0.00	1.59±0.25	0.05±0.00	1.95±0.00	0.07±0.00
CNN	270.74±1.16	0.29±0.015	13.87±0.26	0.24±0.02	18.42±0.31	0.29±0.005	40.03±0.04	0.64±0.07	2.50±0.06	0.05±0.00	3.68±0.005	0.07±0.005
NeuralLog	174.08±1.99	0.363±0.038	8.83±0.07	0.28±0.00	11.12±0.44	0.29±0.03	24.89±0.03	0.01±0.03	1.57±0.00	0.01±0.00	2.34±0.02	0.08±0.00
Unsupervised												
LogAnomaly	7.733±0.58	0.01±0.00	0.51±0.03	0.01±0.00	13.14±0.02	0.11±0.00	46.45±1.9	0.39±0.01	0.30±0.005	0.01±0.00	0.44±0.00	0.01±0.00
DeepLog	0.86±0.03	0.01±0.00	0.53±0.00	0.01±0.00	1.65±0.03	0.01±0.00	4.92±0.1	0.02±0.005	0.17±0.00	0.01±0.00	0.23±0.01	0.01±0.00
Semi-supervised												
PLELog	25.09±0.47	0.45±0.01	8.67±0.19	0.18±0.005	10.69±0.13	0.22±0.01	23.43±0.03	0.48±0.00	1.45±0.01	0.03±0.00	2.15±0.03	0.04±0.00
NovaAD	21.12±1.75	0.06±0.005	4.14±0.01	0.01±0.00	3.20±0.06	0.01±0.00	5.45±0.01	0.01±0.00	0.98±0.00	0.01±0.00	2.03±0.01	0.01±0.00

statistical structure, sentiment cues, and temporal dynamics. These findings directly justify the feature selection and ablation strategy adopted in subsequent experiments.

For the *HDFS* dataset, the most informative features are *character count*, *dominant topic*, *SBERT*, *word count*, and *entropy*, indicating that both the structural properties of the logs (e.g., message length and variability) and their semantic or thematic content play a central role in distinguishing anomalies. *Sentiment* contributes moderately, suggesting that polarity signals still provide an additional predictive value. Temporal features such as *hour*, *second*, *minute*, *day*, and *month* rank lower, showing that although anomalies may exhibit certain time-based patterns, these signals are weaker compared to linguistic and structural characteristics. In general, these results emphasize that anomaly detection in HDFS logs relies primarily on semantic and structural cues, with temporal features serving a complementary role.

Although the MI scores obtained for all features are relatively small, this behavior is expected in high-dimensional data, such as system logs. In such settings, individual features rarely explain a large part of the variability in the anomaly labels on their own. Instead, the predictive signal is distributed across multiple features, each contributing a modest amount of information. Moreover, in practice, a set of moderately informative features often works together to achieve strong predictive performance when used in machine learning models. Thus, the relatively small MI scores observed here do not indicate a lack of usefulness. Instead, they highlight that anomaly detection in logs relies on the joint contribution of many features, rather than the dominance of a single highly informative variable.

7 ERROR ANALYTICS

1)Parsing of log error analytics.

8 CONCLUSION

In this paper, we present NovaAD, a novel semi-supervised approach to detect anomalies in textual event log data. Our main contributions are: (1) an effective feature set that integrates both content-based and temporal characteristics to enhance anomaly detection , (2) a self-configuration mechanism that automates parameter tuning for fully autonomous operation in semi-supervised settings, (3) a technique for handling unstable logs, and (4) an efficient detection process that minimizes response delays and operational risks. Experimental results demonstrate that NovaAD outperforms existing methods in both detection accuracy and runtime efficiency, which makes it suitable for large-scale real-time deployments. The combination of a robust feature set, automated configuration, and handling of unstable-log data establishes NovaAD as a scalable, practical and high-performance solution for anomaly detection in operational systems.

In future work, our goal is to further enhance the speed and scalability of all components, including feature engineering, novelty detection, label estimation, training in anomaly detection models, and decision making. This will also involve the integration of a distributed computing framework, such as Apache Spark, to enable high-throughput, low-latency processing of massive log datasets. We will, furthermore, optimize core modules for parallel execution, optimize feature extraction pipelines for distributed environments, and develop a scalable pseudo-label generation mechanism for large-scale semi-supervised learning. These improvements will allow NovaAD to operate efficiently in production-scale streaming

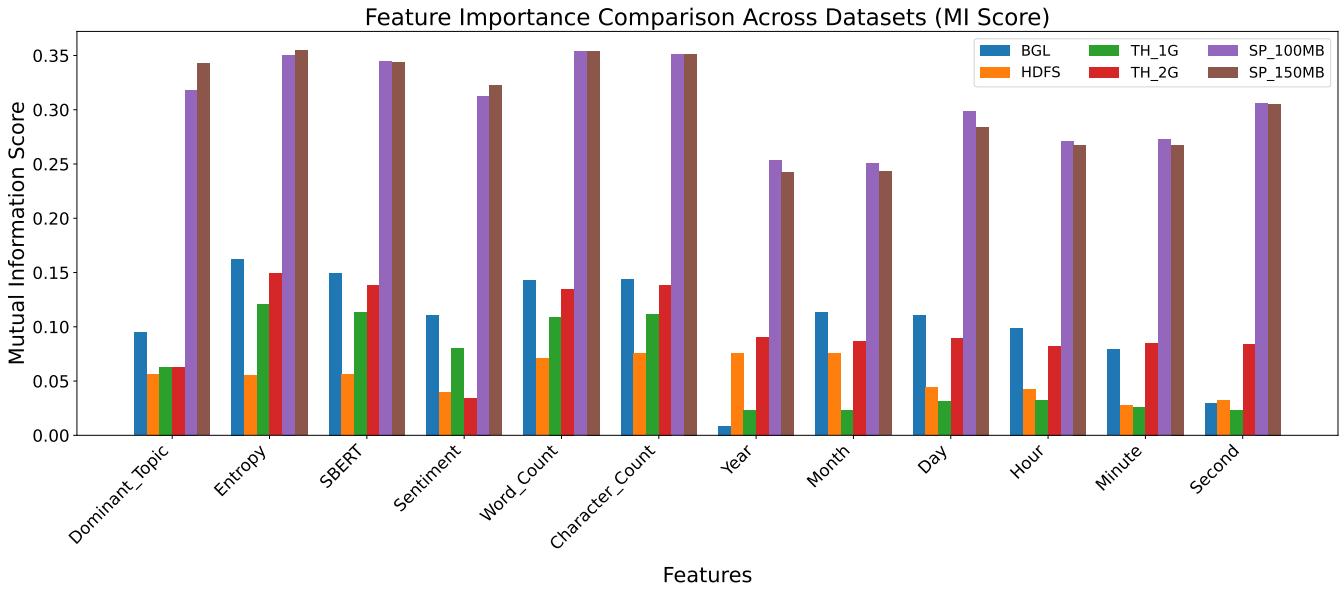

Figure 3: Mutual Information–based feature importance across datasets.

environments while maintaining high accuracy and responsiveness.

ACKNOWLEDGMENTS

The authors appreciate the support and guidance of Lin Yang at Tianjin University, whose input has been instrumental in the development of this work.

ARTIFACTS

The source code, data, and documentation have been made available at https://github.com/nayefroqaya/LogSSAD.

REFERENCES

[1] 2019. *Stacking strong ensembles of classifiers*, 545–556. ISBN: 978-3-030-19822-0. DOI: 10.1007/978-3-030-19823-7_46.

[2] Omar Alghushairy, Raed Alsini, Terence Soule, and Xiaogang Ma. 2020. A review of local outlier factor algorithms for outlier detection in big data streams. *Big Data and Cognitive Computing*, 5, 1, 1.

[3] Shan Ali, Chaima Boufaied, Domenico Bianculli, Paula Branco, and Lionel Briand. 2025. A comprehensive study of machine learning techniques for log-based anomaly detection. *Empirical Software Engineering*, 30, 5. DOI: 10.1007/s10664-025-10669-3.

[4] Anunay Amar and Peter C. Rigby. 2019. Mining historical test logs to predict bugs and localize faults in the test logs. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 140–151. DOI: 10.1109/ICSE.2019.00031.

[5] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3, null, 993–1022.

[6] 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. DOI: 10.1145/1755913.1755926.

[7] Jakub Breier and Jana Branisová. 2015. A dynamic rule creation based anomaly detection method for identifying security breaches in log records. *Wireless Personal Communications*, 94, 497–511.

[8] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*.

[9] Danny Z. Chen and Bin Xu. 2003. Geometric algorithms for agglomerative hierarchical clustering. In *Proceedings of the 9th Annual International Conference on Computing and Combinatorics*, 30–39.

[10] M. Chen, A.X. Zheng, J. Lloyd, M.I. Jordan, and E. Brewer. 2004. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.* 36–43. DOI: 10.1109/ICAC.2004.1301345.

[11] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: a scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. DOI: 10.1145/2939672.2939785.

[12] Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. 2013. Event logs for the analysis of software failures: a rule-based approach. *IEEE Transactions on Software Engineering*, 39, 6, 806–821. DOI: 10.1109/TSE.2012.67.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805. arXiv: 1810.04805.

[14] 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. DOI: 10.1145/3133956.3134015.

[15] Ricardo Dunia and S Joe Qin. 1997. Multi-dimensional fault diagnosis using a subspace approach. In *American Control Conference*. Citeseer.

[16] Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, and John Grundy. 2015. Experience report: anomaly detection of cloud application operations using log and cloud metric correlation analysis. In *2015 IEEE 26th international symposium on software reliability engineering (ISSRE)*. IEEE, 24–34.

[17] D. Gencaga, N. K. Malakar, and D. J. Lary. 2014. Survey on the estimation of mutual information methods as a measure of dependency versus correlation analysis. In *AIP Conference Proceedings*, 80–87. DOI: 10.1063/1.4903714.

[18] Michael Greenacre, Patrick Groenen, Trevor Hastie, Alfonso Iodice D'Enza, Angelos Markos, and Elena Tuzhilina. 2022. Principal component analysis. *Nature Reviews Methods Primers*, 2, 100. DOI: 10.1038/s43586-022-00184-w.

[19] Ajay Gupta and Adrian Barbu. 2018. Parameterized principal component analysis. *Pattern Recognition*, 78, 215–227. DOI: https://doi.org/10.1016/j.patcog.2018.01.018.

[20] 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. DOI: 10.1145/2983323.2983358.

[21] Hannes Hansen and Martin N. Hebart. 2022. Semantic features of object concepts generated with gpt-3. (2022). arXiv: 2202.03753 [cs.CL].

[22] Mehran Hassani, Weiyi Shang, Emad Shihab, and Nikolaos Tsantalis. 2018. Studying and detecting log-related issues. *Empirical Softw. Engg.*, 23, 6, 3248–3280. DOI: 10.1007/s10664-018-9603-z.

[23] 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)*, 654–661. DOI: 10.1109/DSN.2016.66.

[24] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. 2018. Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing*, 15, 6, 931–944. DOI: 10.1109/TDSC.2017.2762673.

[25] 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)*, 33–40. DOI: 10.1109/ICWS.2017.13.

[26] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 60–70. DOI: 10.1145/3236024.3236083.

[27] 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)*, 207–218. DOI: 10.1109/ISSRE.2016.21.

[28] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2020. Loghub: a large collection of system log datasets towards automated log analytics. *ArXiv*, abs/2008.06448.

[29] Mark Heitmann, Christian Siebert, Jochen Hartmann, and Christina Schamp. 2020. More than a feeling: benchmarks for sentiment analysis accuracy. *Available at SSRN 3489963*.

[30] Caiping Hu, Xuekui Sun, Hua Dai, Hangchuan Zhang, and Haiqiang Liu. 2023. Research on log anomaly detection based on sentence-bert. *Electronics*, 12, 17.

[31] He Jiang, Xiaochen Li, Zijiang Yang, and Jifeng Xuan. 2017. What causes my test alarm? automatic cause analysis for test alarms in system and integration testing. *CoRR*, abs/1703.00768. arXiv: 1703.00768.

[32] Suhas Kabinna, Cor-Paul Bezemer, Weiyi Shang, Mark D. Syer, and Ahmed E. Hassan. 2018. Examining the stability of logging statements. *Empirical Softw. Engg.*, 23, 1, 290–333. DOI: 10.1007/s10664-017-9518-0.

[33] Darshika Koggalahewa, Yue Xu, and Ernest Foo. 2022. An unsupervised method for social network spammer detection based on user information interests. *Journal of Big Data*, 9, 1, 1–35.

[34] Sujatha Arun Kokatnoor and Balachandran Krishnan. 2020. Self-supervised learning based anomaly detection in online social media. *International Journal of Intelligent Engineering and Systems*, 13, 446–456.

[35] Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 492–504. DOI: 10.1109/ASE51524.2021.9678773.

[36] Tao Li et al. 2017. Flap: an end-to-end event log analysis platform for system management. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1547–1556. DOI: 10.1145/3097983.3098022.

[37] 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)*, 583–588. DOI: 10.1109/ICDM.2007.46.

[38] Petro Liashchynskyi and Pavlo Liashchynskyi. 2019. Grid search, random search, genetic algorithm: A big comparison for NAS. *CoRR*, abs/1912.06059. arXiv: 1912.06059.

[39] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 102–111.

[40] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 102–111.

[41] Linqing Liu, Yao Lu, Ye Luo, Renxian Zhang, Laurent Itti, and Jianwei Lu. 2016. Detecting" smart" spammers on social network: a topic model approach. *arXiv preprint arXiv:1604.08504*.

[42] Shuxian Liu, Le Deng, Huan Xu, and Wei Wang. 2023. Logbd: a log anomaly detection method based on pretrained models and domain adaptation. *Applied Sciences*, 13, 13.

[43] Yang Liu, Shaochen Ren, Xuran Wang, and Mengjie Zhou. 2024. Temporal logical attention network for log-based anomaly detection in distributed systems. *Sensors*, 24, 24. DOI: 10.3390/s24247949.

[44] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining invariants from console logs for system problem detection. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, 24.

[45] Gilles Louppe. 2015. Understanding random forests: from theory to practice. (2015). arXiv: 1407.7502 [stat.ML].

[46] 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)*, 151–158. DOI: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037.

[47] Dan Lv, Nurbol Luktarhan, and Yiyong Chen. 2021. Conanomaly: content-based anomaly detection for system logs. *Sensors*, 21, 18. DOI: 10.3390/s21186125.

[48] Weibin Meng et al. 2019. Loganomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 4739–4745. DOI: 10.24963/ijcai.2019/658.

[49] Weibin Meng et al. 2019. Loganomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 4739–4745.

[50] Amit Kumar Mishra, Piyush Bagla, Ravi Sharma, Neeraj Kumar Pandey, and Neha Tripathi. 2023. Anomaly detection from web log data using machine learning model. In *2023 7th International Conference on Computer Applications in Electrical Engineering-Recent Advances (CERA)*, 1–6. DOI: 10.1109/CERA59325.2023.10455153.

[51] Mahsa Mohaghegh and Amantay Abdurakhmanov. 2021. Anomaly detection in text data sets using character-level representation. *Journal of Physics: Conference Series*, 1880, 012028. DOI: 10.1088/1742-6596/1880/1/012028.

[52] Karthik Nagaraj, Jennifer Neville, and Charles Killian. 2012. Structured comparative analysis of systems logs to diagnose performance problems.

[53] F. Pedregosa et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

[54] Ekaterina Poslavskaya and Alexey Korolev. 2023. Encoding categorical data: is there yet anything 'hotter' than one-hot encoding? (2023). arXiv: 2312.16930 [cs.LG].

[55] David M. W. Powers. 2020. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *ArXiv*, abs/2010.16061.

[56] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.

[57] Robert A Reris and J. Paul Brooks. 2015. Principal component analysis and optimization: a tutorial. In.

[58] Peter J. Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20, 53–65.

[59] Dymitr Ruta and Bogdan Gabrys. 2005. Classifier selection for majority voting. *Information fusion*, 6, 1, 63–81.

[60] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13, 7, 1443–1471. DOI: 10.1162/089976601750264965.

[61] Ketan Rajshekhar Shahapure and Charles Nicholas. 2020. Cluster quality analysis using silhouette score. In *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, 747–748. DOI: 10.1109/DSAA49011.2020.00096.

[62] Moshe Sipper. 2022. High per parameter: a large-scale study of hyperparameter tuning for machine learning algorithms. (2022). arXiv: 2207.06028 [cs.LG].

[63] H Joe Steinhauer, Tove Helldin, Gunnar Mathiason, and Alexander Karlsson. 2023. Topic modeling for anomaly detection in telecommunication networks. *Journal of Ambient Intelligence and Humanized Computing*, 14, 11, 15085–15096.

[64] H.Joe Steinhauer, Tove Helldin, Gunnar Mathiason, and Alexander Karlsson. 2019. Topic modeling for anomaly detection in telecommunication networks. *Journal of Ambient Intelligence and Humanized Computing*, 14. DOI: 10.1007/s12652-019-01372-5.

[65] Yicheng Sun, Jacky Wai Keung, Zhen Yang, Shuo Liu, and Yihan Liao. 2025. Semismac: a semi-supervised framework for log anomaly detection with automated hyperparameter tuning. *Information and Software Technology*, 187, 107869. DOI: https://doi.org/10.1016/j.infsof.2025.107869.

[66] Arnatchai Techaviseschai, Sansiri Tarnpradab, Vasco Chibante Barroso, and Phond Phunchongharn. 2025. A real-time semi-supervised log anomaly detection framework for alice o2 facilities. *Applied Sciences*, 15, 11. DOI: 10.3390/app15115901.

[67] Feifei Tu, Jiaxin Zhu, Qimu Zheng, and Minghui Zhou. 2018. Be careful of when: an empirical study on time-related misuse of issue tracking data. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 307–318. DOI: 10.1145/3236024.3236054.

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need. (2023). arXiv: 1706.03762 [cs.CL].

[69] Giovanni Vigna, Fredrik Valeur, Davide Balzarotti, William Robertson, Christopher Kruegel, and Engin Kirda. 2009. Reducing errors in the anomaly-based detection of web-based attacks through the combined analysis of web requests and sql queries. *J. Comput. Secur.*, 17, 3, 305–329.

[70] R. Vinayakumar, K. P. Soman, and Prabaharan Poornachandran. 2017. Long short-term memory based operation log anomaly detection. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 236–242. DOI: 10.1109/ICACCI.2017.8125846.

[71] Laura Viola, Elisabetta Ronchieri, and Claudia Cavallaro. 2022. Combining log files and monitoring data to detect anomaly patterns in a data center. *Computers*, 11, 8. DOI: 10.3390/computers11080117.

[72] Janett Walters-Williams and Yan Li. 2009. Estimation of mutual information: a survey. In *Rough Sets and Knowledge Technology*. Peng Wen, Yuefeng Li, Lech Polkowski, Yiyu Yao, Shusaku Tsumoto, and Guoyin Wang, (Eds.), 389–396.

[73] Jin Wang, Changqing Zhao, Shiming He, Yu Gu, Osama Alfarraj, and Ahed Abugabah. 2022. Loguad: log unsupervised anomaly detection based on word2vec. *Comput. Syst. Sci. Eng.*, 41, 3, 1207–1222. DOI: 10.32604/csse.2022.022365.

[74] Hilde JP Weerts, Andreas C Mueller, and Joaquin Vanschoren. 2020. Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv:2007.07588*.

[75] Xingfang Wu, Heng Li, and Foutse Khomh. 2023. On the effectiveness of log representation for log-based anomaly detection. *Empirical Software Engineering*, 28, 6. DOI: 10.1007/s10664-023-10364-1.

[76] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Plelog: semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 230–231. DOI: 10.1109/ICSE-Companion52605.2021.00106.

[77] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. 2016. Automated it system failure prediction: a deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*, 1291–1300. DOI: 10.1109/BigData.2016.7840733.

[78] Xu Zhang 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. DOI: 10.1145/3338906.3338931.

[79] 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 *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, 121–130. DOI: 10.1109/ICSE-SEIP.2019.00021.

[80] Xiaonan Zou, Yong Hu, Zhewen Tian, and Kaiyuan Shen. 2019. Logistic regression model optimization and case analysis. In *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*, 135–139. DOI: 10.1109/ICCSNT47585.2019.8962457.