

Article

ADAL-NN: Anomaly Detection and Localization Using Deep Relational Learning in Distributed Systems

Kashan Ahmed ¹, Ayesha Altaf ^{1,*} , Nor Shahida Mohd Jamail ², Faiza Iqbal ^{1,*}  and Rabia Latif ² ¹ Department of Computer Science, University of Engineering & Technology (UET), Lahore 54890, Pakistan; kashanahmed867@gmail.com² Artificial Intelligence and Data Analytics Laboratory, College of Computer and Information Sciences (CCIS), Prince Sultan University, Riyadh 11586, Saudi Arabia; njamail@psu.edu.sa (N.S.M.J.); rlatif@psu.edu.sa (R.L.)

* Correspondence: ayesha.altaf@uet.edu.pk (A.A.); faiza.iqbal@uet.edu.pk (F.I.)

Abstract: Modern distributed systems that operate concurrently generate interleaved logs. Identifiers (ID) are always associated with active instances or entities in order to track them in logs. Consequently, log messages with similar IDs can be categorized to aid in the localization and detection of anomalies. Current methods for achieving this are insufficient for overcoming the following obstacles: (1) Log processing is performed in a separate component apart from log mining. (2) In modern software systems, log format evolution is ongoing. It is hard to detect latent technical issues using simple monitoring techniques in a non-intrusive manner. Within the scope of this paper, we present a reliable and consistent method for the detection and localization of anomalies in interleaved unstructured logs in order to address the aforementioned drawbacks. This research examines Log Sequential Anomalies (LSA) for potential performance issues. In this study, IDs are used to group log messages, and ID relation graphs are constructed between distributed components. In addition to that, we offer a data-driven online log parser that does not require any parameters. By utilizing a novel log parser, the bundled log messages undergo a transformation process involving both semantic and temporal embedding. In order to identify instance–granularity anomalies, this study makes use of a heuristic searching technique and an attention-based Bi-LSTM model. The effectiveness, efficiency, and robustness of the paper are supported by the research that was performed on real-world datasets as well as on synthetic datasets. The neural network improves the F1 score by five percent, which is greater than other cutting-edge models.

Keywords: log parsing; deep relational learning; natural language processing; anomaly detection

Citation: Ahmed, K.; Altaf, A.; Jamail, N.S.M.; Iqbal, F.; Latif, R. ADAL-NN: Anomaly Detection and Localization Using Deep Relational Learning in Distributed Systems. *Appl. Sci.* **2023**, *13*, 7297. <https://doi.org/10.3390/app13127297>

Academic Editor: Valentino Santucci

Received: 3 April 2023

Revised: 26 May 2023

Accepted: 14 June 2023

Published: 19 June 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

High internal software faults (an infinite loop, wrong configuration, resource hogs, and node disconnection) and external errors regularly threaten the availability and dependability of cloud services (resource hogs). When a system performs as predicted with adequate resources, we refer to it as “normal”, and when it behaves otherwise, we call it an “anomaly”. The issues highlighted (anomalies in logs, resource utilization spikes, and request failure increases) may all be indicators of anomalies in a distributed system. These issues may be brought on by several circumstances, including hardware or software failures, resource contention, or changes in workload or consumption [1]. It is feasible to identify and localize anomalies in a distributed system by monitoring for and thoroughly studying these types of problems. This may involve gathering and analyzing data from various sources, such as log files, performance indicators, and business KPIs, and utilizing techniques such as machine learning or statistical analysis to find anomalous patterns or deviations from typical behavior. After identifying and localizing abnormalities, corrective action can be performed to resolve underlying issues and restore normal system operation.

Software developers rely heavily on log data to determine the system’s status and to identify anomalies. Modern computer systems frequently generate large volumes of

interleaved logs as they simultaneously process or execute operations [2]. This can make it difficult to manually analyze the logs and to identify anomalies, particularly as the system grows in size and complexity. Frequently, automated tools and techniques for log analysis, such as machine learning algorithms or statistical analysis methods, are required to address this issue. These tools can help identify patterns and correlations in the log data that may indicate the presence of anomalies, as well as provide insight into the components or subsystems responsible for the anomalies.

Maintaining the stability and performance of these complex systems requires the detection of anomalies utilizing neural networks and their localization utilizing relation graphs. Localization refers to the process of identifying the specific components or subsystems responsible for the anomalies. Anomaly detection involves identifying unusual or abnormal system behavior.

Recent research has focused extensively on data-driven and log-based anomaly detection, with or without localization [3–11]. Current methodologies are based on assumptions that are difficult to confirm in a real-world production setting. When employing the aforementioned techniques in a production setting, the following three issues must be considered:

- I Dependencies in logs. In distributed systems, logs are utilized to monitor the statuses and significant events of active instances or entities. Insufficiently, the current methodologies only correlate them with one component. Using log data, developers can identify dependencies between various components or subsystems in a distributed system and use this information to locate anomalies. By correlation of logs with a similar identifier on distributed components, developers can gain a deeper understanding of the relationships and interactions between various system components and use this knowledge to identify and localize anomalies.
- II Modifying the logs. In reality, log formats in software systems change frequently, by as much as 45 percent over the lifetime of a software system [9,12]
- III Hidden performance problems. Latency issues are a main indicator of biased failure [13], which refers to partially compromised functionalities [1,14–17]. Frequently, it is necessary to analyze a variety of system observations, including log data, to determine whether or not there is a performance issue, and discovering hidden issues in log data can help to improve system reliability.

For the detection and localization of anomalies in distributed systems, there are a variety of methods available. Here are some common methods that are frequently employed:

- a Statistical analysis: This technique entails analyzing performance metrics and other system data to identify anomalous patterns or deviations from normal behavior. Statistical methods such as hypothesis testing and regression analysis can be used to identify and comprehend anomalies.
- b Machine Learning: Machine learning algorithms can be used to first learn the typical behavior of a system and then to detect and label as anomalies any behavior that deviates from the learned normal behavior. This method can be used in conjunction with statistical analysis techniques to manage large amounts of data.
- c Rule-based systems: A set of rules or thresholds are defined to identify anomalies based on specific conditions or events. For instance, a rule could be configured to trigger an alarm if a server's CPU utilization exceeds a certain threshold.
- d Heuristic techniques: They employ human expertise and intuition to identify anomalies based on patterns or trends in the data. This may involve manual inspection of log data or other types of data or the use of automated tools that employ heuristics to detect anomalies.
- e Graph-based systems: Graph-based systems use graph theory and network analysis techniques to determine dependencies and correlations between a system's various components. This is useful for identifying anomalies that may not be immediately apparent from individual metrics or events.

One approach to detect and locate anomalies in distributed systems is by leveraging neural networks and relation graphs. Neural networks are models of machine learning that can be trained to recognize patterns and correlations in data, making them well suited for anomaly detection in distributed systems. Relation graphs, on the other hand, are graphical representations of the relationships between various distributed system components or subsystems. By combining these two methods, it is possible to not only detect anomalies but also to identify the specific components or subsystems that are causing them.

We provide a log anomaly detection and localization tool based on techniques of deep learning. The application excels at managing unstructured interleaved logs, which may contain both natural language text and IDs within individual log messages. This is one of the areas in which the application shines. Changes to log time intervals and log sequence orders are identified as two typical forms of log modification based on real-world log data. This research focuses on log-based data anomalies, including sequential-based log anomalies and latency issues, taking into account the aforementioned two categories as the primary considerations.

This research is conducted in stages, the first of which is relation construction, followed by log parser, anomaly identifier, and then anomaly localizing. A step that takes place offline is distinct from a phase that takes place online. Instead of assigning log templates an index, this work uses BERT, which is an encoder module of modern transformers [18], to encode the semantic information in the templates. BERT makes templates resistant to updated arriving templates and takes temporal data into account through projection onto large-dimensional embeddings. During the online phase, encoded templates are used to detect and localize anomalies in log data.

The following are the four major contributions to this research:

1. We propose a novel method for processing online logs that are dictionary based and generalized without the need for any parameter tuning. This method offers a significant increase in efficiency, as well as robustness and generalization.
2. Rarely has log analysis been used to investigate hidden performance issues. Our method for detecting anomalies involves adding semantic and temporal information to log sequences. This allows us to identify a variety of anomalies, such as sequential log anomalies and latent performance issues.
3. To identify anomalies at the instance level, a heuristic technique is developed. This heuristic technique can be utilized to rapidly detect and localize anomalies in a distributed system.
4. We collect a dataset of real-world Papertrail logs and use this research to identify and locate anomalies. Papertrail collects log data from a variety of sources, such as servers, applications, and devices, and provides a central location for viewing and analyzing these data. The results prove the validity and dependability of this investigation. We evaluate this study using both synthetic and actual datasets.

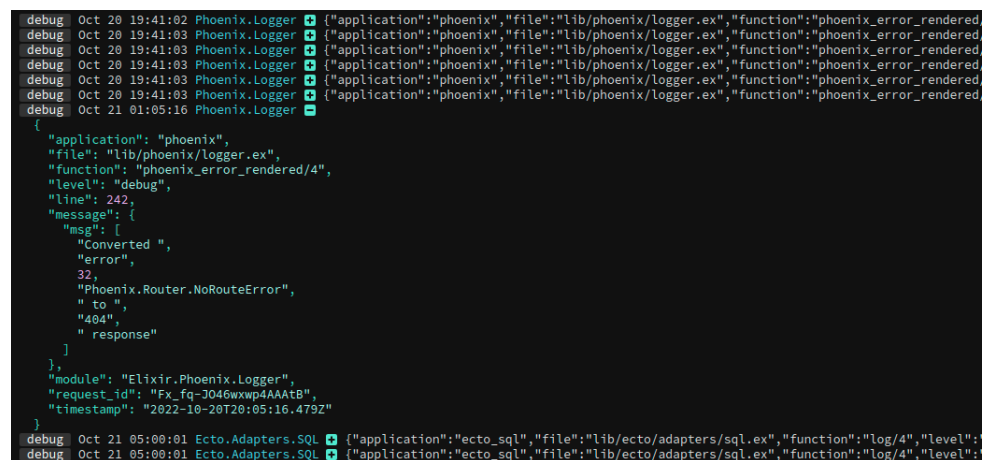
Using neural networks and relation graphs, we propose a method for anomaly detection and localization in distributed systems and name it ADAL-NN, which stands for “Anomaly Detection and Localization using Neural Networks”. We describe the specifics of our approach and evaluate its efficacy through a series of simulation experiments on a distributed system simulator. Our results indicate that our method for detecting and localizing anomalies in distributed systems is effective.

Several different areas are explored in this study, such as log correlation, heuristic-based anomaly localization, and sophisticated time embedding. Figure 1 also depicts the real-time logs we will be using.

Anomaly detection and localization in a distributed system has many advantages.

1. Improved system stability: System stability is improved when anomalies are found and corrected before they have a chance to grow into major issues that would threaten the system’s integrity.

2. Increased performance: Improved responsiveness and productivity can result from the use of anomaly detection and localization to pinpoint the source of performance degradation and to fix it.
3. Improved resource utilization: Resource optimization and waste reduction can be achieved through the use of anomaly detection and localization by highlighting and resolving potential resource contention issues.
4. Increased reliability: In order to improve the system's reliability, anomaly detection and localization can be used to pinpoint and repair the underlying causes of any unexpected behavior.
5. Better decision making: Better decision making and improvement opportunity identification can be achieved through anomaly detection and localization because of the enhanced understanding of the system's behavior and performance that is made possible.



```

debug Oct 20 19:41:02 Phoenix.Logger {"application":"phoenix","file":"lib/phoenix/logger.ex","function":"phoenix_error_rendered
debug Oct 20 19:41:03 Phoenix.Logger {"application":"phoenix","file":"lib/phoenix/logger.ex","function":"phoenix_error_rendered
debug Oct 20 19:41:03 Phoenix.Logger {"application":"phoenix","file":"lib/phoenix/logger.ex","function":"phoenix_error_rendered
debug Oct 20 19:41:03 Phoenix.Logger {"application":"phoenix","file":"lib/phoenix/logger.ex","function":"phoenix_error_rendered
debug Oct 20 19:41:03 Phoenix.Logger {"application":"phoenix","file":"lib/phoenix/logger.ex","function":"phoenix_error_rendered
debug Oct 21 01:05:16 Phoenix.Logger {"application":"phoenix","file":"lib/phoenix/logger.ex","function":"phoenix_error_rendered
{
  "application": "phoenix",
  "file": "lib/phoenix/logger.ex",
  "function": "phoenix_error_rendered/4",
  "level": "debug",
  "line": 242,
  "message": {
    "msg": {
      "Converted ",
      "error",
      32,
      "Phoenix.Router.NoRouteError",
      " to ",
      "404",
      " response"
    }
  },
  "module": "Elixir.Phoenix.Logger",
  "request_id": "Fx_fq-J046wxwp4AAAt8",
  "timestamp": "2022-10-20T20:05:16.479Z"
}
debug Oct 21 05:00:01 Ecto.Adapters.SQL {"application":"ecto_sql","file":"lib/ecto/adapters/sql.ex","function":"log/4","level":
debug Oct 21 05:00:01 Ecto.Adapters.SQL {"application":"ecto_sql","file":"lib/ecto/adapters/sql.ex","function":"log/4","level":

```

Figure 1. Papertrail's real-time logs. The logs were created by the multi-tenant system.

A total of five parts make up this study. Section 1 discusses the historical context and theoretical justifications for the research. There is a discussion of the research process and its results in this section. The second part of this paper is a literature review on log processing, anomaly detection, and anomaly localization. In addition, a study table is provided in this section for the purpose of filling in potential knowledge gaps. In Section 3, we lay out in great detail the ADAL-NN methodology that will be used throughout the various procedures. To better visualize the ADAL-NN proposed technique, an architecture diagram for anomaly detection and localization is provided here as well. In Section 4, we go over the implementation details. Extensive explanations of the dataset and the results of numerous log parsing, anomaly detection, and anomaly localization experiments are provided. Section 5 wraps up the research and discusses what comes next.

2. Literature Review

Workflow Creation. CloudSeer, IntelLog and Stitch are some of the most popular tools for reconstructing a targeted system's workflow. CloudSeer creates workflows and monitors them using interleaved logs [5], while Stitch builds the graph for dependencies as an S^3 using identifiers found in log-based data [19]. IntelLog employs natural language-processing techniques to create the workflow graphs that depict the relationships [7].

Log Parsing. Log parsing is a fundamental step in many log analysis works, such as those that have been thoroughly researched in [3,20,21]. These works parse logs by generating regular expressions from source codes in order to perform log parsing. In reality, all codes are already available online, and log parser techniques are classified into a few groups.

- (1) Threshold-based grouping: LKEs [22], LogSigs [23], Log-Mines [24], SHISOs [25], and LenMas [26] find variation between two logs before grouping them via the threshold method. For finding the template associated with a single group, threshold-based

grouping techniques typically need a predefined value. Parameter tuning is time-consuming, labor-intensive, and non-universal.

- (2) Frequency-based grouping: Finding the counts of an entity in the logs is a simple method for automatically parsing the logs. Because a group of static entities is repeated in the logs, finding the counts makes it easier to understand how the logs are created. SLCT [27], LFA [28], and LogCluster [29] first record the frequency of items before categorizing them into multiple groups.
- (3) Heuristics based on tree searching: Drain divides log messages into nodes using a fixed-depth prefix tree [30]. Spell utilizes a tree structure to parse the logs into numerous templates [31]. Each layer of the prefix tree reflects a distinct log characteristic, such as the length or first token features.
- (4) Dictionary-based clustering: Both Logram [32] and Logparse [33] demonstrate the ability to parse logs using a dictionary. Logram builds dictionaries from pre-existing templates, and Logparse uses a classification method for the classification of the logs. Logparse uses the characteristics of variable words before the construction of each word.
- (5) Based on deep learning: These techniques, on the other hand, require significantly more resources to run than other log parsers. NuLog [34] suggested a neural-network-based log parser model, and the parsing job was formulated as masked language modeling (MLM). In contrast, a number of studies have identified trust-based connection in distributed and centralized Internet of things [35–38]. A survey [39] presented on Blockchain technologies also highlights open issues and future direction.

Anomaly Detection. Numerous existing anomaly-detection methods concentrate on identifying sequential log abnormalities, which can be difficult to detect due to the large volumes of data involved. To effectively detect these anomalies, it is frequently necessary to analyze data from multiple sources and to employ techniques that can manage large amounts of data.

Data-mining-based techniques encompass both supervised and unsupervised learning methods.

Supervised: Supervised approaches (e.g., decision tree [40], support vector machines [41], regression-based methodology [42]) may learn the fixed pattern of distinct labeled logs by training on labeled log data. As a result, they consistently outperform unsupervised approaches. However, labeling a huge volume of historical data for training takes time. Furthermore, they are unable of detecting a black swan, which may be present in previous data.

Unsupervised: Unsupervised algorithms train on unlabeled historical data, rather than label it, in the same way that supervised approaches train on log data. This approach often creates an anomalous space and a non-anomalous space for the anomalous and non-anomalous sequences, respectively [3,4].

With the popularity of deep learning, deep-learning-based anomaly-detection models are frequently investigated [6,8,9,43]. DL approaches include log parser, training, and prediction of the model.

- (1) Key value method: A study demonstrates that LSTM is more accurate than a log-key-based model at predicting the structure of anomalous and non-anomalous sessions [6,43]. This is because log statements are processed into templates prior to being assigned log keys. When the source codes for a new version of the model are updated, they are treated as new templates, resulting in performance degradation.
- (2) Context-based method: Log-based data provide a plethora of semantic information about system states. Meng et al. [8] used Long Short-Term Memory via word vectors to analyze synonyms and antonyms. However, they utilize the counting vectors, which are not resistant to varying the log-based data. To detect abnormalities, Zhang et al. [9] used Att-Bi-LSTM. In contrast, Word2Vec and TF-IDF disregard contextual information in sentences. In this study, BERT is utilized to determine the contextual semantic meaning of utterances.

Anomaly Localization. A prevalent area of investigation is the utilization of statistics and machine learning for the purpose of anomaly detection. If you only rely on methods for detecting anomalies, you will end up with a significant number of false positives because anomalies tend to spread quickly through distributed and interdependent systems. The authors in [8] provide a taxonomy of anomaly-detection methods based on categorization, statistics, information theory, and clustering. Using system traces and supervised model training, the authors of [9] present a method for defect localization. In contrast to this investigation, their methodology requires the manual selection of characteristics. This procedure's disadvantage is its complexity, and its authors advocate combining a route condition time series method with an adapted random walk algorithm [10]. This enables not only the recording of sequential links in time series data, but also the incorporation of information regarding causal links, temporal order, and monitoring data priority. We learn the characteristics of anomalies, whereas their method relies on root cause metrics criteria, which frequently require specialized knowledge to establish properly. This research simplifies feature extraction by only considering temporal correlations within relevant time series slices. The authors of [5] conduct an individual analysis for every system component in addition to an ensemble analysis that only considers global predictions and disregards the underlying network structure of system relationships.

MonitorRank [11] is a real-time metric collection system intended for use in service-oriented architectures to conduct root cause analysis (RCA) when abnormalities are reported. It uses an unsupervised and heuristic (clustering) technique to generate a sorted list of potential root causes that monitoring teams can use as a guide. MonitorRank focuses on diagnosis after an abnormality has been reported, as opposed to the detection of abnormalities themselves. This means that it is not intended to identify abnormalities in real time, but rather to assist teams in determining the root cause of previously identified abnormalities. Table 1 also displays research gaps for the related study.

Table 1. Documents concerning anomaly detection and localization.

Ref.	Model	Dataset	Online Mode	Log-Parser	Anomaly Identifier	Anomaly Localizer	Explainable AI
[3]	PCA	Logs Stream	✗	✓	✓	✗	✗
[5]	Custom	Distributed DB	✗	✗	✗	✗	✗
[8]	LSTM	HDFS, BGL	✗	✓	✓	✗	✗
[9]	BiLSTM	HDFS	✗	✓	✗	✗	✗
[10]	Att. GRU	HDFS, BGL	✗	✓	✗	✗	✗
[11]	NB, HMM	-	✗	✗	✓	✗	✗
[12]	RF, Cox	Custom	✗	✓	✗	✗	✗
[13]	Fail-Stutter	-	✗	✗	✓	✓	✗
ADAL-NN	Att. BiLSTM	HDFS, Custom	✓	✓	✓	✓	✓

In distributed systems, several systems use causality graphs to perform end-to-end anomaly identification and root cause investigation. LOUD is an example of a system that creates a model based on accurate executions and employs graph centrality methods to locate faulty cloud system resources. However, LOUD [3] relies on positive training and may not be able to recognize specific abnormalities. eBay's GRANO [12] is another example of an enterprise-level software framework that includes a detection layer, anomaly graph layer, and application layer to aid fault-resolution teams. GRANO is customized for cloud-native distributed data platforms. By providing a comprehensive and integrated approach to these tasks, these types of systems can enhance the efficiency and effectiveness of anomaly detection and root cause analysis. Additionally, GRANO employs numerous techniques for identifying and localizing aberrant behavior. On the other hand, RCA advocates [7] deducing the underlying causes of performance issues by comparing application

performance issues with related system resource use. BIRCH is used to identify anomalies, and root cause analysis (RCA) is performed by employing a graph that represents anomaly transmission across systems. Additionally, there is recent research in this field as discussed in [44–53].

3. Methodology

There are two primary techniques that constitute the ADAL-NN proposed methodology as described in Figure 2. Logs are parsed, converted into sentence-level embedding, and then a model is trained on the embedded logs using an Att-BiLSTM neural network in the offline mode. In online mode, we take a new log and tweak the relation graph, parse the log using a previously trained model, detect anomalies in real time, and pinpoint their location using the relation graph.

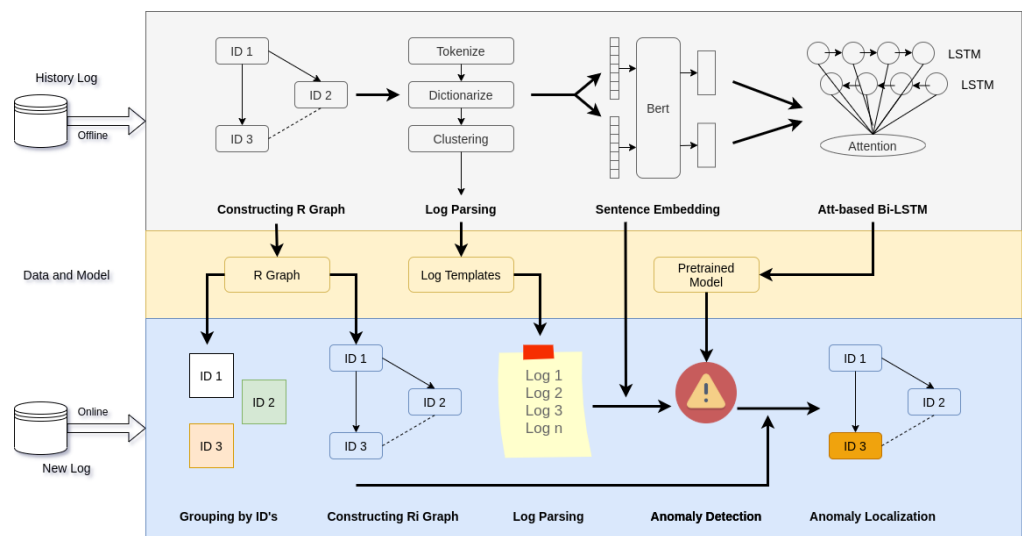


Figure 2. Anomaly detection and localization architecture diagram. The offline phase involves four steps: constructing R graph, log parsing, sentence embedding, and Att-based Bi-LSTM. The online phase involves five steps: grouping by IDs, constructing Ri graph, log parsing, anomaly detection, and anomaly localization.

We begin with an overview of this research, which is shown in Figure 2. This research is separated into offline processing and online processing sections. A description of the steps involved in detecting and localizing anomalies in log data in greater detail is as follows:

1. **Constructing R graph (offline):** During the offline phase, we construct an ID relation graph (R graph) using historical log data. This involves analyzing the log data to identify relationships between various log messages and system components. By mapping the dependencies and correlations between log messages and components, the R graph provides a representation of the system's behavior and structure.
2. **Log parsing (offline):** In this step, we tokenize and dictionary the historical log data. The log data are divided into individual words or tokens and are then grouped based on common word sets present in the dictionary data. This process allows us to extract numerous log templates that capture the natural utterances within the log data, as opposed to relying solely on event IDs.
3. **Sentence embedding (offline):** The sequence of bundled logs obtained from the log parsing step is further processed to encode semantic and temporal information. We employ the BERT (bidirectional encoder representations from transformers) encoder to capture the semantic information present in the log data. This transforms the log information into a high-dimensional embedding called $E_{context}$. Additionally,

the temporal information, which represents the order and timing of log events, is projected onto a separate high-dimensional embedding referred to as E_{time} .

4. Att-based Bi-LSTM (offline): The semantic embedding ($E_{context}$) and the temporal embedding (E_{time}) obtained from the previous steps are concatenated and input into an Attention-based Bidirectional Long Short-Term Memory (Att-based Bi-LSTM) network. This network is trained to learn the characteristics of normal, abnormal, and performance-anomalous log sequences. By capturing temporal dependencies and semantic patterns, the Att-based Bi-LSTM model can effectively identify anomalies within log sequences.
5. Grouping by IDs (online): During the online phase, when a new log message is received in real time, it is correlated with existing log messages that share the same IDs. This step aims to understand the relationships and dependencies between the incoming log message and the previous log messages associated with the same IDs. By considering the contextual information within the log data, we can gain insights into the behavior of system components.
6. Constructing R_i graph (online): Based on the correlated log messages, a new instance-specific R graph, denoted as R_i graph, is instantiated. The R_i graph represents the current state of the system and captures any changes or deviations from the normal behavior observed during runtime. This graph can be leveraged to identify and analyze anomalies that occur in real time.
7. Log parsing (online): The R_i graph obtained from the previous step is passed through the log parsing stage, similar to the offline phase. This allows us to extract and encode the relevant semantic and temporal information specific to the current instance of the system.
8. Anomaly detection (online): In this step, the encoded log information is fed into the trained model to detect anomalies. By comparing the log sequences to the learned patterns and characteristics from the offline training, the model can identify deviations that indicate the presence of anomalies. If an anomaly is detected, an alarm is generated to alert the monitoring team about the issue.
9. Anomaly localization (online): Upon detecting an anomaly, the research team proceeds to localize the anomaly by examining the malfunctioning instance. The team focuses on investigating the specific instance associated with the anomaly and reports the instance ID related to it. This information is crucial for identifying the root cause of the anomaly and for taking appropriate corrective actions.

The steps involved in the entire procedure are listed above. As our proposed method employs a deep learning architecture that leverages the power of BERT and attention-based BiLSTM networks, it is important to discuss the network structure, which can be described as follows:

1. Input layer: The input layer receives the log data as input, which are typically represented as a sequence of log events. Each log event is encoded using sentence embeddings.
2. Embedding layer: The embedding layer transforms the sentence-embedded log events into continuous vector representations, capturing the semantic information of the log events. We utilize pre-trained BERT embeddings to capture the contextual information and semantic relationships between log events.
3. LSTM layer: The LSTM layer processes the log event sequence in a recurrent manner, capturing the temporal dependencies and patterns within the logs. It consists of multiple LSTM units that maintain and update the hidden state based on the current input and the previous hidden state.
4. Attention layer: The attention layer enhances the model's ability to focus on relevant log events by assigning different weights to different parts of the log sequence. It dynamically learns the importance of each log event based on its relevance to anomaly detection.
5. Output layer: The output layer is responsible for predicting whether a given log event or log sequence is anomalous or normal. It produces the anomaly score for each log

event, indicating its deviation from normal behavior. We employ suitable functions such as Softmax as an activation function and Categorical Cross-Entropy Loss as a loss function to train the model effectively.

This research project's design includes multiple phases for detecting and localizing anomalies in log data. The procedure begins with relation creation, which entails constructing a graph that depicts the relationships and dependencies among the system's various components. The next step is log parsing, which entails extracting the pertinent information from the log data and encoding it in a format that can be used for analysis. The encoded log data are then fed into a deep learning model that has been trained to recognize anomalies using techniques such as self-attention or masked language modeling. If an anomaly is detected, the system can utilize the relation graph and other contextual data to localize the anomaly and to determine its root cause. This may entail examining the malfunctioning instance and its dependencies in the relation graph, as well as employing heuristics or other techniques to determine the source of the anomaly.

3.1. Log Parsing

We preprocess a stream of log data by removing any unnecessary characters or formatting. By separating the preprocessed log data into individual words, we tokenize it. We dictionaryize the tokens by creating a dictionary with words as the keys and the frequency of each word as the values. The word sets in the dictionaryized data are used to cluster the log data into groups. Using the Longest Common Subsequence (LCS) algorithm, we hide any variable data in the clusters. Using a prefix tree, we combine the masked clusters into a single list of dictionaries. As output, we return the merged logs as a list of dictionaries representing parsed and processed log entries. The algorithm used to parse logs for anomaly detection is detailed in Algorithm 1. The worst log parser complexity is $O(n * \log d + \log c)$, where n is the number of log entries, d is the size of the dictionary, and c is the total number of groups.

Algorithm 1: Log parsing algorithm for anomaly detection

Data: $\logData = []$
Result: $templates = []$

- 1 $filteredData \leftarrow preprocess(\logData)$
- 2 $tokenizedData, dictionary \leftarrow tokenize(filteredData)$
- 3 $clusteredData \leftarrow groupByWordset(tokenizedData, dictionary)$
- 4 $maskedData \leftarrow maskVariables(clusteredData)$
- 5 $mergedData \leftarrow mergeLogs(maskedData)$
- 6 $templates \leftarrow extractTemplates(mergedData)$
- 7 **return** $templates$

3.2. Anomaly Detection

We create an already trained Att-Bi-LSTM method for offline anomaly identification via historical log-based data as our data source. Log parsing and sentence embedding are processes that are carried out whenever a new batch of log messages is received. After that, the vectors that were generated are used to train a model that has already been constructed. In conclusion, the Att-Bi-LSTM has the capability of identifying anomalies. Take this into consideration whenever this method is going to make assumptions by using the session of log-based data that are connected by a similar identifier, such as a block identifier. Because of this, an unusual occurrence can be reliably reported until the end of the session. In other words, this approach, similar to LogRobot [9], operates in a fast mode. Algorithm 2 is used for anomaly detection using BERT embedding and LSTM-based neural network. Our log parser has the worst complexity, $O(n^2)$, where n is the length of the input sequence.

Algorithm 2: The BERT and LSTM approaches to anomaly detection

Data: $inputSequence = []$
Result: $anomalies = []$
8 $embeddings \leftarrow BERT(inputSequence, model_{BERT})$
9 $outputSequence \leftarrow LSTM(embeddings, model_{LSTM})$
10 $anomalyScores \leftarrow computeAnomalyScores(inputSequence, outputSequence)$
11 $anomalyThreshold \leftarrow computeAnomalyThreshold(anomalyScores)$
12 **for** $i \leftarrow 1$ **to** $length(inputSequence)$ **do**
13 **if** $anomalyScores_i > anomalyThreshold$ **then**
14 $anomalies \leftarrow anomalies \cup \{i\}$
15 **return** $anomalies$

3.3. Anomaly Localization

Identified instance IDs are labeled as anomalous during detection. Using an instantiated directed acyclic R graph (DAG) representation of the system, we now employ a heuristic technique for localizing anomalies in a distributed system. Beginning with a set of anomalous instances, the ADAL-NN heuristic technique follows the edges of the DAG R graph to determine the root cause of the anomalies.

The procedure begins with the collection of anomalous instances, denoted by the letter “A”. The algorithm then examines each of these IDs to determine if they are leaf nodes (i.e., nodes with zero out-degree). If an ID is a leaf node, the search is terminated and the ID is returned as the anomaly’s cause. If an ID is not a leaf node, the algorithm examines the ID’s child nodes to determine if any are anomalous. If any of the child nodes are atypical, the corresponding ID is removed from the set “A”. If none of the child nodes are anomalous, the search proceeds to the children of the children. This procedure is repeated until the anomaly’s root cause is identified. This method yields a collection of anomalous instances, completing the instance-level localization of anomalies as discussed in Algorithm 3.

Algorithm 3: Anomaly localization through the use of relation graphs

Data: $graph = [], anomalousInstances = []$
Result: $rootCauses = []$
16 $rootCauses \leftarrow \emptyset$
17 **for** $instance \in anomalousInstances$ **do**
18 **if** $degree_{out}(instance) = 0$ **then**
19 $rootCauses \leftarrow rootCauses \cup \{instance\}$
20 **else**
21 $children \leftarrow getChildren(graph, instance)$
22 $anomalousChildren \leftarrow anomalyLocalization(graph, children)$
23 **if** $anomalousChildren = \emptyset$ **then**
24 $rootCauses \leftarrow rootCauses \cup \{instance\}$
25 **return** $rootCauses$

The anomaly localization algorithm has a time complexity of $O(n)$, where n is the number of nodes in the relation graph. This is due to the fact that the algorithm loops through all of the anomalous instances and checks their graph children but does not traverse the entire graph. The algorithm’s time complexity could potentially be reduced to $O(k)$, where k is the number of root causes, by adding a break statement after the root cause is identified; however, this would increase the algorithm’s complexity.

4. Implementation

4.1. Datasets

Public Datasets. Logpai [54] is a log parser benchmark that uses 16 real-world log datasets from distributed systems, supercomputers, operating systems, mobile systems, and server applications, as well as standalone software such as HDFS, Hadoop, Spark, Zookeeper, BGL, HPC, Thunderbird, Windows, Linux, Android, HealthApp, Apache, Proxifier, OpenSSH, OpenStack, and Mac. LogHub [55] provides the above log datasets. Each dataset comprises 2000 log samples, each of which has been tagged by a rule-based log parser. In addition to sampling datasets, we chose datasets from two exemplary systems to assess the proposed ADAL-NN technique. Table 2 displays the specifics.

Table 2. Log datasets.

Log Type	No of Messages	No of Templates	Seq	Perf
HDFS	11,176,738	31	✓	✗
BGL	4,747,963	377	✓	✗
Papertrail	2,949,569	109	✓	✓

The HDFS log collection was obtained from a cluster of 203 nodes on the Amazon EC2 platform [3], and it contains 11,176,738 raw log messages. The aberrant HDFS actions were categorized manually by researching HDFS code and by talking with Hadoop professionals, including sequential-order anomalies such as “Replica quickly removed” and certain exception logs such as “Receive block error”. More information may be found in the original publication [3].

Lawrence Livermore National Labs (LLNL) gathered the BGL dataset, which is a supercomputing system log dataset [56]. BGL’s abnormalities are manually identified by its system administrators. The log messages in these anomalies most likely contain exception descriptions. “ciod: Error generating node map from file [...]” is an example of a log message. More information may be found in the original publication [56].

We also gathered real Papertrail log data, which we used for anomaly detection and localization following log parsing. We utilized Papertrail’s search and filtering capabilities to identify the instances or components that may be causing the anomaly. We used the information obtained from the process of anomaly detection and localization to identify and fix the problem’s root cause.

4.2. Experiments

We performed the following steps for the experiments of anomaly detection and localization:

1. **Datasets:** We utilized three different datasets for our experiments: Hadoop Distributed File System (HDFS), Big-Log (BGL), and Papertrail. These datasets are widely used in anomaly-detection research and provide a diverse range of log data for evaluation purposes.
2. **Preprocessing:** Before conducting the experiments, we preprocessed the datasets to prepare them for anomaly detection. This involved cleaning the data, removing irrelevant characters, and transforming the log messages into a structured format for analysis.
3. **Feature extraction:** In order to extract meaningful features from the log data, we applied various techniques such as tokenization, dictionary-based grouping, and embedding. These techniques allowed us to represent the log data in a structured and informative manner.
4. **Experimental steps:**
 - (a) **Splitting the data:** We partitioned the preprocessed datasets into training, validation, and testing sets via 8:1:1. The training set was used to train the

models, the validation set was used for hyperparameter tuning, while the testing set was employed to evaluate the performance of the trained models in detecting anomalies.

- (b) **Model selection:** We selected a range of anomaly-detection models to compare with our proposed NN model. These models included Support Vector Machines (SVM), Isolation Forest (IM), Principal Component Analysis (PCA), Neural Networks (NN), Autoencoders (AE), Variational Autoencoders (VAE), and Recurrent Neural Networks (RNN). Each model was chosen based on its popularity and effectiveness in anomaly-detection tasks.
 - (c) **Model training:** For each selected model, we performed training using the training set of the respective dataset. The models were trained to learn the normal behavior of the log data, capturing patterns and structures that characterize normal system operations.
 - (d) **Parameter tuning:** We conducted parameter tuning for each model to optimize its performance. This involved adjusting the model-specific parameters, such as the number of hidden layers, learning rate, and regularization parameters to achieve the best possible anomaly-detection results.
 - (e) **Model evaluation:** After training the models, we evaluated their performance using the testing set. We calculated various evaluation metrics, such as precision, recall, and F1 score, to assess the effectiveness of each model in detecting anomalies. We considered the weighted average F1 score as the main metric for the evaluation part.
5. **Experimental environment:**
- (a) **Hardware:** The experiments were conducted on a high-performance computing cluster comprising multiple computing nodes. Each node was equipped with sufficient computational resources, including multi-core processors and a substantial amount of memory, to handle the processing requirements of the experiments.
 - (b) **Software:** We used open-source machine learning libraries and frameworks, such as scikit-learn for machine learning, and Pytorch for deep learning, to implement and train the different models. These libraries provided the necessary tools and functions for data preprocessing, model training, and evaluation.
 - (c) **Programming environment:** The experiments were conducted using Python programming language. Python offers a wide range of libraries for machine learning and data analysis, making it suitable for implementing and evaluating anomaly-detection models.

4.3. Results

It is essential to evaluate the performance of a log parser and anomaly-detection system in order to determine how effectively they achieve their objectives. Several different metrics, including parsing accuracy, precision, recall, and F1 score, can be used to evaluate the performance of a log parsing algorithm. Accuracy, the receiver operating characteristic (ROC) curve, and the area under the curve are additional metrics that may be useful for evaluating the performance of a log parsing algorithm (AUC). If the objective was to identify anomalies as precisely as possible, precision may be the most crucial metric to consider. Alternatively, if the objective was to identify as many anomalous instances as possible, recall could be the most important metric. However, we evaluated the log parser based on parsing precision because our current objective is to evaluate the parsing results of the log parser.

The results of log parsing are depicted in Figure 3. The results obtained clearly demonstrate the effectiveness of our log parser. It is now time to implement sentence embedding and train the neural network. These results are superior to the logram parser, which achieves an accuracy of 80.9% on HDFS and 58.7% on BGL. It is remarkable that our log parser achieved a parsing accuracy of 98%. This indicates that it can correctly

parse the vast majority of log messages, which could be advantageous for extracting useful information and insights from log data.

```

=== Overall evaluation results ===
  dataset  F1_measure  Accuracy
0    HDFS    1.000000    1.0000
1     BGL    0.999831    0.9695
average accuracy is 0.98475

```

Figure 3. Result of log parsing indicating parsing accuracy for various datasets, such as HDFS and BGL.

There are a few factors that contributed to the high parsing accuracy of our log parser:

1. Robust log parsing algorithm: The log parsing algorithm used by our log parser is robust and able to accurately parse a wide variety of log messages, even if they contain errors or deviations from the expected format.
2. High-quality training data: The log parser was trained on a large and diverse dataset of log messages, which improved its ability to accurately parse new log messages.
3. Efficient error handling: The log parser has effective mechanisms for handling and correcting errors in the log messages, which contribute to its high parsing accuracy.

For model training, an attention-based BiLSTM neural network was utilized. In total, 30 epochs were used to train the neural network. Figure 4 depicts the remarkable results we achieved after training the neural network. Common performance metrics for anomaly-detection systems include detection accuracy, false positive rate, false negative rate, precision, recall, and F1 score. By carefully examining these metrics, the weighted F1 score is the best metric for identifying data anomalies. Table 3 displays the classification report for the anomaly-detection section.

```

Epoch [1/30], Train_loss: 25.2815
Epoch [2/30], Train_loss: 20.9446
Epoch [3/30], Train_loss: 21.1028
Epoch [4/30], Train_loss: 21.0351
Epoch [5/30], Train_loss: 21.3984
Epoch [6/30], Train_loss: 21.2862
Epoch [7/30], Train_loss: 21.1046
Epoch [8/30], Train_loss: 20.9793
Epoch [9/30], Train_loss: 21.1885
Epoch [10/30], Train_loss: 21.1239
Epoch [11/30], Train_loss: 20.8648
Epoch [12/30], Train_loss: 21.4129
Epoch [13/30], Train_loss: 20.8594
Epoch [14/30], Train_loss: 20.9556
Epoch [15/30], Train_loss: 20.8822
Epoch [16/30], Train_loss: 21.1633
Epoch [17/30], Train_loss: 21.1454
Epoch [18/30], Train_loss: 21.2821
Epoch [19/30], Train_loss: 21.0565
Epoch [20/30], Train_loss: 21.2376
Epoch [21/30], Train_loss: 20.9225
Epoch [22/30], Train_loss: 21.2361
Epoch [23/30], Train_loss: 20.8837
Epoch [24/30], Train_loss: 21.1691
Epoch [25/30], Train_loss: 21.0175
Epoch [26/30], Train_loss: 20.8646
Epoch [27/30], Train_loss: 21.1472
Epoch [28/30], Train_loss: 21.5601
Epoch [29/30], Train_loss: 21.0768
Epoch [30/30], Train_loss: 21.0570
Finished Training

```

Figure 4. Training in focus-based BiLSTM neural network with 30 epochs of sentence embedding of parsed logs.

The classification report is displayed in Table 3 after the model was updated with the new data. In this research, 88% was the weighted average F1 score attained by the anomaly-detection algorithm. We utilized a weighted average F1 score instead of other commonly used metrics such as precision, recall, and accuracy because there are few samples with anomalous characteristics. The weighted average F1 score accounts for the imbalance between classes in the dataset. In this instance, the F1 score is the average of the F1 scores for each class, with the number of instances in each class serving as the weights.

Table 3. Classification report of anomaly detection.

Classes	Precision	Recall	F1 Score
0	0.92	1.00	0.96
1	0.00	0.00	0.00
2	0.00	0.00	0.00
Accuracy	-	-	0.92
Macro Average	0.31	0.33	0.32
Weighted Average	0.85	0.92	0.88

Anomaly detection is an inherently imbalanced problem, where the number of normal instances significantly outweighs the number of anomalies. In such cases, accuracy can be misleading, as it can be driven by the high accuracy achieved in normal instances while failing to capture the true performance on anomalies. Similarly, precision and recall focus on correctly classifying anomalies, but they may not adequately capture the trade-off between false positives and false negatives, which is crucial in anomaly detection. On the other hand, the F1 score provides a balanced measure that combines precision and recall, making it well suited for imbalanced classification problems. Therefore, in the context of anomaly detection, the weighted average F1 score emerges as a more informative and suitable evaluation metric than accuracy, precision, or recall alone.

We also compared our results with the other methods in the literature. Table 4 shows the effectiveness of our proposed method.

Table 4. Comparison of F1 scores, precision, and recall for selected studies.

Reference	Precision	Recall	F1 Score
[3]	82	71	77
[8]	86	77	82
[13]	76	66	71
ADAL-NN	85	92	88

In the context of anomaly detection and localization, XAI systems can provide explanations for why an anomaly was detected and for its system location. Using XAI for anomaly detection and localization offers a number of advantages:

1. Improved trust: By providing explanations for their decisions and actions, XAI systems can help to build trust with users and stakeholders who may be skeptical of the capabilities of AI.
2. Enhanced accountability: XAI systems can help to improve accountability by providing a clear record of the steps that were taken to detect and locate an anomaly, which can be useful for audit and compliance purposes.
3. Better understandability: XAI systems can help to make the results of anomaly detection and localization more understandable to humans, which can help to improve the effectiveness of the process.

There are several approaches to developing XAI systems for anomaly detection and localization, including:

1. Rule-based systems: Rule-based systems provide explanations by presenting the specific rules or heuristics that were used to make a decision.
2. Model-based systems: Model-based systems provide explanations by presenting the underlying model or algorithms that were used to make a decision.
3. Example-based systems: Example-based systems provide explanations by presenting examples of similar cases that were used to make a decision.

Here, we examine two significant limitations of this study:

First, this study is only capable of identifying anomalies in log data. On the one hand, this study cannot detect abnormalities (such as excessive CPU utilization) revealed by KPI data as opposed to log data. This study employs log time interval change to identify as many potential performance issues as possible, but not all, because not all performance problems are linked to logs.

Second, the scope of this study is limited to log statements whose core meanings are stable despite their constant flux. If the updated logging statement has a different meaning than the previous ones, or if the incoming log message is significantly different from the previous ones, the pre-trained model may not capture the meaning, leading to incorrect results.

4.4. Discussion

The comparative results of different models are given in Figure 5. These include deep-learning-based models i.e., Principle Component Analysis (PCA), Initial Margin (IM), Support Vector Machine (SVM), and Neural Network (NN) on HDFS illustrated in Figure 5a, BGL illustrated in Figure 5b, and Papertrail illustrated in Figure 5c data. These illustrated data were actually weighted average F1 scores of different models as we considered them for evaluation purposes.

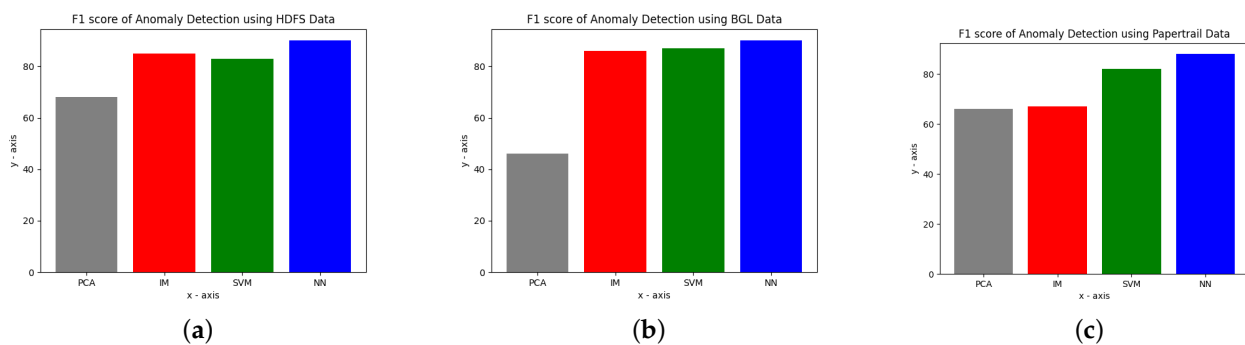


Figure 5. Evaluation of a model based on a comparison of multiple models using distinct datasets. (a) HDFS anomaly-detection evaluation. PCA, IM, SVM, and NN F1 scores are 0.68, 0.85, 0.83, 0.90, respectively. (b) BGL anomaly-detection evaluation. PCA, IM, SVM, and NN F1 scores are 0.46, 0.86, 0.87, 0.90, respectively. (c) PT anomaly-detection evaluation. PCA, IM, SVM, and NN F1 scores are 0.66, 0.67, 0.82, 0.88, respectively.

We also examine the offline training time and online inference time of the anomaly-detection model on the HDFS dataset. This investigation's execution time is measured in milliseconds per sequence (ms/seq).

In Table 5, we compare the runtime of the anomaly-detection component of deep-learning-based models and models that do not use deep learning. Training time corresponds to offline training cost, while real or inference time corresponds to online detection time. These times were carried out by using a Python library "timeit", which is used to calculate the execution time of the algorithm. Due to the fact that a deep-learning-based model must be trained over multiple epochs, only the runtime of one epoch is presented here. Due to the immense complexity of deep-learning-based models and the limited resources of this research's experimental contexts, it is evident that deep-learning-based models are significantly slower than models that do not utilize deep learning. The ADAL-NN requires four times as much training as alternatives. This is acceptable, as the overhead incurred during offline training has no bearing on the online runtime of anomaly detection. The online inference time of the ADAL-NN is roughly double that of LogRobust. Given that this model is a three-classification problem and LogRobust is a two-classification task, it may take longer to learn and predict an additional class.

Table 5. Offline and online time on HDFS data utilizing distinct models for offline and online time.

Model	Training Time	Real Time
NN	41.58	10.75
SVM	6.77	2.07
IM	14.76	2.0
PCA	8.55	8.1

A large-scale system may generate over 60 GB of logs per hour, or 120 million log lines [2] (i.e., 6 million log sequences assuming each sequence comprises 20 log lines). According to Table 3, due to the limited processing power and device memory of a single Z-book system, it takes approximately one thousand minutes. We have also run the ADAL-NN in parallel on an eight-core CPU, which has a significant impact on the online detection time. The results indicate that it takes approximately 50 min to process such a large volume of log data.

The proposed method presents a notable advancement in terms of efficiency, robustness, and generalization compared to existing approaches. Firstly, in terms of efficiency, the method incorporates various optimizations and techniques that enable faster processing and analysis of log data. By constructing the ID relation graph during the offline phase and leveraging log parsing and sentence embedding techniques, the method efficiently extracts and encodes the relevant semantic and temporal information from log sequences. This streamlined process reduces computational overhead and improves the overall efficiency of anomaly detection and localization.

Secondly, the method exhibits enhanced robustness compared to previous approaches. By employing an attention-based Bi-LSTM, the model effectively captures the intricate patterns and dependencies within log sequences, enabling accurate detection and localization of anomalies. The use of deep learning techniques enhances the model's ability to handle complex and diverse log data, making it more robust in detecting anomalies across different scenarios and system environments. Additionally, the method's integration of anomaly identification and localization components further strengthens its robustness by providing a comprehensive understanding of the detected anomalies.

Lastly, the proposed method demonstrates superior generalization capabilities. Through the utilization of large-scale datasets such as HDFS, BGL, and Custom, the model is trained on diverse log data, enabling it to learn and generalize patterns across different domains and applications. This generalization capability allows the method to effectively adapt to new and unseen log data, making it highly applicable in real-world scenarios where log data patterns may vary. The incorporation of explainable AI techniques further enhances the model's generalization by providing transparent and interpretable explanations for anomaly detection and localization, ensuring its applicability across various domains and facilitating decision-making processes.

In summary, the proposed method offers significant advancements in terms of efficiency, robustness, and generalization. By leveraging optimized techniques, deep learning models, and explainable AI components, the method achieves faster processing, accurate anomaly detection, comprehensive localization, and adaptability to diverse log data, making it a highly promising approach in the field of anomaly detection and localization.

This research provides a case study of a real-world system, Hadoop, to further illustrate the identification and localization of anomalies. The R graph of a Hadoop cluster is illustrated in Figure 6. Hadoop consists of six distinct types. The relationship between types is represented by arrow lines, with the 1:1 relationship indicated by a red dashed line and the 1:n relationship indicated by a black line. For instance, if an application creates multiple blocks, their relationship is 1:n. In our research, the 1:1 correlation is eliminated.

Following that, we provide a real-world anomalous instance. To begin, a series of incoming logs progressively launches the R graph as the R_i graph. In reality, application 01

generates a total of 14 blocks and 16 containers. In Figure 7, we only display the simplified R_i graph with three blocks, three containers, one app attempt, one attempt r, and one attempt m due to space constraints.

This research first identifies anomalies in log data organized by block ID and labels blk 01 as an anomalous block. Following that, an exception is detected in block 03; consequently, the ADAL-NN identifies block 03 as aberrant as well. Ultimately, the ADAL-NN method detects a killing signal in container 02, which is labeled as flawed. In Figure 7, all of the incorrect instances in A are colored red. The graph R_i is then subjected to heuristic searching. The ADAL-NN reports the anomalous instances blk 01, blk 03, and container 02 because they are leaf nodes in the R_i graph. Consequently, we can identify anomalies in Hadoop at the instance level (blk and container in this example). Such information can greatly aid developers in identifying the source of the issue. We can see, for instance, that not all blocks and containers are affected by these anomalies. In slave clusters, we can assume that a node failure is possible. The associated logs of blk 01, blk 03, and container 02 indicate that slave 2 is experiencing a network issue. Therefore, when we encounter anomalies, this research can aid us in swiftly identifying and analyzing root causes.

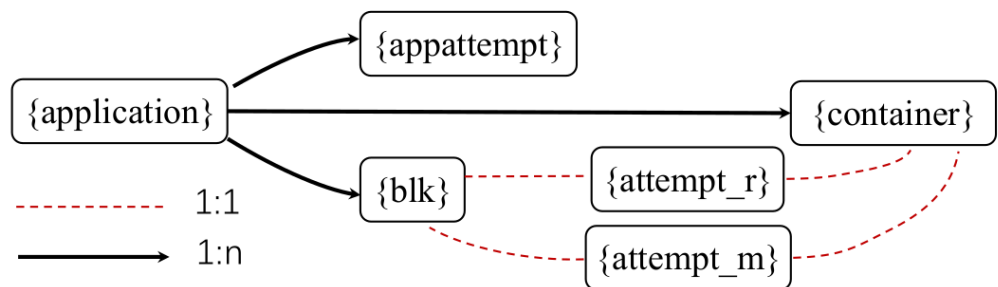


Figure 6. Original relation graph of Hadoop.

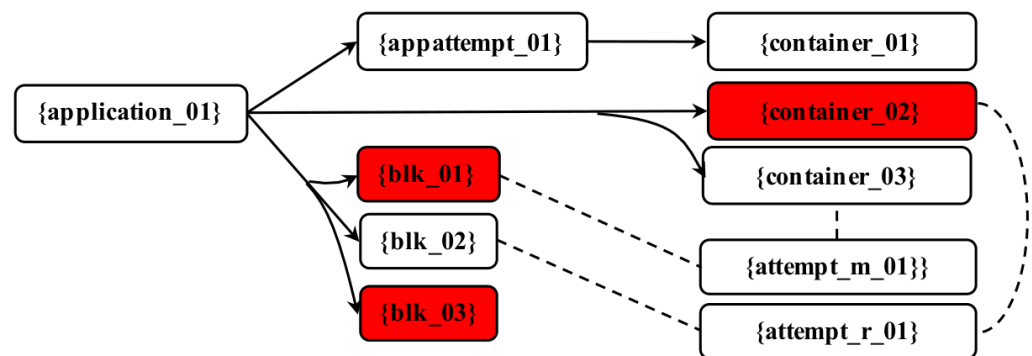


Figure 7. Simple relation graph of Hadoop.

4.5. Comparison with Deep Learning Models

To further demonstrate the effectiveness of our proposed method, we conducted additional experiments comparing it with state-of-the-art deep learning models commonly used for anomaly detection. By including these comparisons, we aim to showcase the superiority of our approach over both traditional and modern anomaly-detection techniques.

We selected the three widely used deep learning models in the task of anomaly detection, which are Autoencoders (AE), Variational Autoencoders (VAE), and Recurrent Neural Networks (RNN). These models were chosen based on their popularity and proven performance in anomaly-detection tasks. For each deep learning model, we implemented the necessary architecture and trained it using the same dataset and evaluation metrics as our proposed method. We ensured that the hyperparameters and training settings were appropriately tuned for each model. We then compared the performance of our proposed method with these deep learning models using the F1 score metrics.

In the HDFS anomaly-detection evaluation, the F1 scores for the NN, AE, VAE, and RNN models were 0.90, 0.87, 0.88, and 0.88, respectively. Similarly, in the BGL anomaly-detection evaluation, the F1 scores for the NN, AE, VAE, and RNN models were 0.90, 0.89, 0.90, and 0.89, respectively. Furthermore, in the PT anomaly-detection evaluation, the F1 scores for the NN, AE, VAE, and RNN models were 0.88, 0.86, 0.87, and 0.87, respectively. For better demonstration, we added these results into Table 6. These scores highlight the capability of the models to identify anomalies in the HDFS, BGL, and PT datasets.

Table 6. F1-scores of anomaly detection via deep learning models.

Dataset	NN	AE	VAE	RNN
HDFS	0.90	0.87	0.88	0.88
BGL	0.90	0.89	0.90	0.89
PT	0.88	0.86	0.87	0.87

The results were obtained through rigorous cross-validation and statistical analysis to ensure reliable comparisons. By including comparisons with deep learning models, we demonstrate the effectiveness and superiority of our proposed method in detecting and localizing anomalies in unstructured logs.

4.6. Future Work

Future research and development in the field of anomaly detection and localization could be conducted in the following areas.

Researchers can continue to improve the accuracy and robustness of existing anomaly-detection algorithms by developing more effective feature extraction techniques, incorporating additional context and background knowledge, and addressing class imbalance issues.

Developing new approaches for anomaly detection: There is a need for the development of new anomaly-detection methods that can handle a variety of data types, such as high-dimensional data or streaming data, and that can be applied to a wide variety of use cases.

Including domain-specific expertise: Anomalies are frequently easier to detect and localize when domain-specific knowledge is considered. For instance, in a manufacturing environment, understanding the normal operation of a specific machine can help to more precisely identify and locate anomalies.

Exploiting temporal dependencies: numerous anomalies exhibit temporal dependencies, which means they tend to occur in a particular pattern over time. The accuracy and effectiveness of anomaly-detection algorithms can be improved by incorporating this information.

Creating robust and effective localization techniques: Anomaly localization is frequently a difficult task, especially when the data are of high dimension or when the anomalies are small or subtle. The development of robust and efficient localization methods that can precisely identify the location and extent of anomalies can be a significant area of future research.

In many applications, the characteristics of the data and the types of anomalies that can occur can change over time. Important research topics include the development of anomaly-detection algorithms that can adapt to these changes and continue to detect and localize anomalies effectively.

5. Conclusions

In this study, a robust and efficient tool for identifying and localizing anomalies in interleaved, unstructured logs was presented. We sought to address the practical challenges posed by complex log dependencies, changing events, and log time interval anomalies. Our method can detect a wide variety of anomalies, including log sequential and log

time interval anomalies, and localize them at the instance level. To assess the efficacy of our methodology (ADAL-NN), we conducted extensive experiments on both real-world and simulated datasets. The outcomes of these experiments indicate that our strategy outperforms existing methods in terms of efficacy, efficiency, and longevity. In the future, we intend to apply our ADAL-NN methodology to a wider variety of systems and to develop a flexible incremental updating method to accommodate new log anomaly patterns. Our work has the potential to significantly improve the reliability and performance of distributed systems. In the future, we also intend to delve deeply into anomaly localization, real-time anomaly detection, and model explanations.

Author Contributions: The authors K.A., A.A., R.L., F.I. and N.S.M.J. made significant contributions to the research. K.A. played a role in conceptualization, methodology development, validation, and data analysis. A.A. contributed to methodology design, implementation, and data analysis. R.L. developed the software and participated in validation. F.I. contributed to data analysis and investigation. N.S.M.J. contributed to conceptualization, methodology design, and data analysis. All authors have reviewed and approved the final manuscript, taking responsibility for their contributions.

Funding: This work was supported in part by the Artificial Intelligence and Data Analytics Laboratory, College of Computer and Information Sciences, Prince Sultan University, Riyadh, Saudi Arabia, and in part by the Department of Computer Science, University of Engineering and Technology (UET), Lahore.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to acknowledge the support of Prince Sultan University for paying the article processing charges (APC) of this publication.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alibaba Cloud Reports IO Hang Error in North China. 2019. Available online: <https://equalocean.com/technology/20190303-alibaba-cloud-reports-io-hang-error-in-north-china> (accessed on 8 January 2023).
2. Mi, H.; Wang, H.; Zhou, Y.; Lyu, M.R.-T.; Cai, H. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Trans. Parallel And Distrib. Syst.* **2013**, *24*, 1245–1255. [CrossRef]
3. Xu, W.; Huang, L.; Fox, A.; Patterson, D.; Jordan, M.I. Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09), Big Sky, MT, USA, 11–14 October 2009; Association for Computing Machinery: New York, NY, USA, 2009; pp. 117–132. [CrossRef]
4. Lou, J.-G.; Fu, Q.; Yang, S.; Li, J.; Wu, B. Mining program workflow from interleaved traces. In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'10), Washington, DC, USA, 25–28 July 2010; Association for Computing Machinery: New York, NY, USA, 2010; pp. 613–622. [CrossRef]
5. Yu, X.; Joshi, P.; Xu, J.; Jin, G.; Zhang, H.; Jiang, G. Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs. *ACM SIGARCH Comput. Archit. News.* **2016**, *44*, 489–502. [CrossRef]
6. Du, M.; Li, F.; Zheng, G.; Srikumar, V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (SIGSAC'17), Dallas, TX, USA, 30 October–3 November 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 1285–1298. [CrossRef]
7. Pi, A.; Chen, W.; Wang, S.; Zhou, X. Semantic-aware Workflow Construction and Analysis for Distributed Data Analytics Systems. In Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '19), Phoenix, AZ, USA, 22–29 June 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 255–266. [CrossRef]
8. Meng, W.; Liu, Y.; Zhu, Y.; Zhang, S.; Pei, D.; Liu, Y.; Chen, Y.; Zhang, R.; Tao, S.; Sun, P.; et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19), Macao, China, 10–16 August 2019; pp. 4739–4745.
9. Zhang, X.; Xu, Y.; Lin, Q.; Qiao, B.; Zhang, H.; Dang, Y.; Xie, C.; Yang, X.; Cheng, Q.; Li, Z.; et al. Robust log-based anomaly detection on unstable log data. In Proceedings of the ESEC/FSE'19: 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Tallinn, Estonia, 26–30 August 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 807–817. [CrossRef]

10. Yang, L.; Chen, J.; Wang, Z.; Wang, W.; Jiang, J.; Dong, X.; Zhang, W. Plelog: Semi-supervised log-based anomaly detection via probabilistic label estimation. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Madrid, Spain, 25–28 May 2021; pp. 1448–1460. [\[CrossRef\]](#)
11. Zou, D.-Q.; Qin, H.; Jin, H. Uilog: Improving log-based fault diagnosis by log analysis. *J. Comput. Sci. Technol.* **2016**, *31*, 1038–1052. [\[CrossRef\]](#)
12. Kabinna, S.; Bezemer, C.-P.; Shang, W.; Syer, M.D.; Hassan, A.E. Examining the stability of logging statements. *Empir. Softw. Eng.* **2018**, *23*, 290–333. [\[CrossRef\]](#)
13. Lou, C.; Huang, P.; Smith, S. Understanding, detecting and localizing partial failures in large system software. In Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20), Santa Clara, CA, USA, 25–27 February 2020; pp. 559–574.
14. Gocardless Service Outage on October 10th 2017. 2017. Available online: <https://gocardless.com/blog/incident-review-api-and-dashboard-outage-on-10th-october> (accessed on 25 September 2022).
15. Office 365 Update on Recent Customer Issues. 2017. Available online: <https://blogs.office.com/2012/11/13/update-on-recent-customer-issues/> (accessed on 8 October 2022).
16. Google Compute Engine Incident 17008. 2017. Available online: <https://status.cloud.google.com/incident/compute/17008> (accessed on 12 November 2022).
17. Twilio Billing Incident Post-Mortem: Breakdown, Analysis and Root Cause. 2013. Available online: <https://bit.ly/2V8rurP> (accessed on 24 December 2022).
18. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
19. Zhao, X.; Rodrigues, K.; Luo, Y.; Yuan, D.; Stumm, M. Non-Intrusive Performance Profiling for Entire Software Stacks Based on the Flow Reconstruction Principle. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; USENIX Association: Savannah, GA, USA, 2016; pp. 603–618.
20. Nagappan, M.; Wu, K.; Vouk, M.A. Efficiently extracting operational profiles from execution logs using suffix arrays. In Proceedings of the 20th International Symposium on Software Reliability Engineering (ISSRE'09), Mysuru, India, 16–19 November 2009; pp. 41–50. [\[CrossRef\]](#)
21. Li, X.; Chen, P.; Jing, L.; He, Z.; Yu, G. Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults. In Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Coimbra, Portugal, 12–15 October 2020; IEEE: Washington, DC, USA, 2020; pp. 92–103.
22. Fu, Q.; Lou, J.-G.; Wang, Y.; Li, J. Execution anomaly detection in distributed systems through unstructured log analysis. In Proceedings of the 9th IEEE International Conference on Data Mining (ICDM'09), Miami Beach, FL, USA, 6–9 December 2009; pp. 149–158. [\[CrossRef\]](#)
23. Mizutani, M. Incremental mining of system log format. In Proceedings of the 2013 IEEE International Conference on Services Computing (SCC'13), Santa Clara, CA, USA, 28 June–3 July 2013; pp. 595–602. [\[CrossRef\]](#)
24. Hamooni, H.; Debnath, B.; Xu, J.; Zhang, H.; Jiang, G.; Mueen, A. Logmine: Fast pattern recognition for log analytics. In Proceedings of the 25th ACM International Conference on Information and Knowledge Management (CIKM'16), Indianapolis, IN, USA, 24–28 October 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 1573–1582. [\[CrossRef\]](#)
25. Zhu, K.Q.; Fisher, K.; Walker, D. Incremental learning of system log formats. *ACM SIGOPS Oper. Syst. Rev.* **2010**, *44*, 85–90. [\[CrossRef\]](#)
26. Shima, K. Length matters: Clustering system log messages using length of words. *arXiv* **2016**, arXiv:1611.03213.
27. Vaarandi, R. A data clustering algorithm for mining patterns from event logs. In Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM'03), Kansas City, MO, USA, 1–3 October 2003; pp. 119–126. [\[CrossRef\]](#)
28. Nagappan, M.; Vouk, M.A. Abstracting log lines to log event types for mining software system logs. In Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR'10), Cape Town, South Africa, 2–3 May 2010; pp. 114–117. [\[CrossRef\]](#)
29. Vaarandi, R.; Pihelgas, M. Logcluster-a data clustering and pattern mining algorithm for event logs. In Proceedings of the CNSM'15: 11th International Conference on Network and Service Management, Barcelona, Spain, 9–13 November 2015; pp. 1–7. [\[CrossRef\]](#)
30. He, P.; Zhu, J.; Zheng, Z.; Lyu, M.R. Drain: An online log parsing approach with fixed depth tree. In Proceedings of the 2017 IEEE International Conference on Web Services (ICWS'17), Honolulu, HI, USA, 25–30 June 2017; pp. 33–40.
31. Du, M.; Li, F. Spell: Streaming parsing of system event logs. In Proceedings of the ICDM'16: 16th International Conference on Data Mining, Barcelona, Spain, 12–15 December 2016; pp. 859–864. [\[CrossRef\]](#)
32. Dai, H.; Li, H.; Chen, C.S.; Shang, W.; Chen, T.-H. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Trans. Softw. Eng.* **2020**, *43*, 879–892. [\[CrossRef\]](#)
33. Meng, W.; Liu, Y.; Zaiter, F.; Zhang, S.; Chen, Y.; Zhang, Y.; Zhu, Y.; Wang, E.; Zhang, R.; Tao, S.; et al. Logparse: Making log parsing adaptive through word classification. In Proceedings of the 2020 29th International Conference on Computer Communications and Networks (ICCCN), Honolulu, HI, USA, 3–6 August 2020.
34. Nedelkoski, S.; Bogatinovski, J.; Acker, A.; Cardoso, J.; Kao, O. Self-supervised log parsing. *arXiv* **2020**, arXiv:2003.07905.

35. Altaf, A.; Abbass, H.; Iqbal, F.; Khan, M.M.Z.M.; Rauf, A.; Kanwal, T. Mitigating Service-Oriented Attacks using Context-Based Trust for Smart Cities in IoT Networks. *J. Syst. Archit.* **2021**, *115*, 102028. [\[CrossRef\]](#)
36. Altaf, A.; Abbass, H.; Iqbal, F.; Daneshmand, M. Robust, Secure and Adaptive Trust-Based Service Selection in Smart Buildings. *IEEE Internet Things J.* **2020**, *8*, 7497–7509. [\[CrossRef\]](#)
37. Altaf, A.; Abbass, H.; Iqbal, F.; Derhab, A. Trust Models of Internet of Smart Things: A Survey, Open Issues and Future Directions. *J. Netw. Comput. Appl.* **2019**, *137*, 93–111. [\[CrossRef\]](#)
38. Altaf, A.; Abbass, H.; Iqbal, F.; Khan, F.A.; Rubab, S. Context-Oriented Trust Computational Model for Malicious Node Avoidance using Edge Intelligence in Industrial Internet of Things. *J. Comput. Electr. Eng.* **2021**, *9*, 1–13. [\[CrossRef\]](#)
39. Altaf, A.; Iqbal, F.; Latif, R.; Yakubu, B.M.; Latif, S.; Samiullah, H. A Survey of Blockchain Technology: Architecture, Applied Domains, Platforms, and Security Threats. *Soc. Sci. Comput. Rev.* **2022**, 08944393221110148. [\[CrossRef\]](#)
40. Chen, M.; Zheng, A.X.; Lloyd, J.; Jordan, M.I.; Brewer, E. Failure diagnosis using decision trees. In Proceedings of the First International Conference on Autonomic Computing (ICAC'04), New York, NY, USA, 17–19 May 2004; pp. 36–43. [\[CrossRef\]](#)
41. Liang, Y.; Zhang, Y.; Xiong, H.; Sahoo, R. Failure prediction in ibm bluegene/levent logs. In Proceedings of the ICDM'07: 7th IEEE International Conference on Data Mining, Omaha, NH, USA, 28–31 October 2007; pp. 583–588. [\[CrossRef\]](#)
42. Farshchi, M.; Schneider, J.-G.; Weber, I.; Grundy, J. Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis. In Proceedings of the 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), Gaithersbury, MD, USA, 2–5 November 2015; pp. 24–34. [\[CrossRef\]](#)
43. Vinayakumar, R.; Soman, K.; Poornachandran, P. Long short-term memory based operation log anomaly detection. In Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI'17), Udupi, India, 13–16 September 2017; pp. 236–242. [\[CrossRef\]](#)
44. Hu, Z.; Chen, P.; Yu, G.; He, Z.; Li, X. TS-InvarNet: Anomaly Detection and Localization based on Tempo-spatial KPI Invariants in Distributed Services. In Proceedings of the 2022 IEEE International Conference on Web Services (ICWS), Barcelona, Spain, 10–16 July 2022; pp. 109–119. [\[CrossRef\]](#)
45. Sadanandan, S.K.; Ahmed, A.; Pandey, S.; Srivastava, A.K. Synchrophasor Data Analytics for Anomaly and Event Detection, Classification, and Localization. In *Intelligent Data Mining and Analysis in Power and Energy Systems: Models and Applications for Smarter Efficient Power Systems*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2023; pp. 105–127. [\[CrossRef\]](#)
46. Mäntylä, M.; Varela, M.; Hashemi, S. Pinpointing Anomaly Events in Logs from Stability Testing—N-Grams vs. Deep-Learning. In Proceedings of the 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Valencia, Spain, 4–13 April 2022; pp. 285–292. [\[CrossRef\]](#)
47. Nyssälä, J.; Mäntylä, M.; Varela, M. How to Configure Masked Event Anomaly Detection on Software Logs? In Proceedings of the 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), Limassol, Cyprus, 3–7 October 2022; pp. 414–418. [\[CrossRef\]](#)
48. Leao, B.P.; Vempati, J.; Muenz, U.; Shekhar, S.; Pandey, A.; Hingos, D.; Bhela, S.; Wang, J.; Bilby, C. Machine Learning-based False Data Injection Attack Detection and Localization in Power Grids. In Proceedings of the 2022 IEEE Conference on Communications and Network Security (CNS), Austin, TX, USA, 3–5 October 2022; pp. 1–8. [\[CrossRef\]](#)
49. Liu, M.; Zhao, C.; Xia, J.; Deng, R.; Cheng, P.; Chen, J. PDDL: Proactive Distributed Detection and Localization Against Stealthy Deception Attacks in DC Microgrids. *IEEE Trans. Smart Grid* **2023**, *14*, 714–731. [\[CrossRef\]](#)
50. Sana, L.; Nazir, M.M.; Iqbal, M.; Hussain, L.; Ali, A. Anomaly Detection for Cyber Internet of Things Attacks: A Systematic Review. *Appl. Artif. Intell.* **2022**, *36*, 2137639. [\[CrossRef\]](#)
51. Saba, T.; Rehman, A.; Sadad, T.; Kolivand, H.; Bahaj, S.A. Anomaly-based intrusion detection system for IoT networks through deep learning model. *Comput. Electr. Eng.* **2022**, *99*, 107810. [\[CrossRef\]](#)
52. Muneer, A.; Taib, S.M.; Fati, S.M.; Balogun, A.O.; Aziz, I.A. A Hybrid Deep Learning-Based Unsupervised Anomaly Detection in High Dimensional Data. *Comput. Mater. Contin.* **2021**, *70*, 6073–6088. [\[CrossRef\]](#)
53. Saleh, R.A.; Driss, M.; Almomani, I. CBiLSTM: A Hybrid Deep Learning Model for Efficient Reputation Assessment of Cloud Services. *IEEE Access* **2022**, *10*, 35321–35335. [\[CrossRef\]](#)
54. Zhu, J.; He, S.; Liu, J.; He, P.; Xie, Q.; Zheng, Z.; Lyu, M.R. Tools and benchmarks for automated log parsing. In Proceedings of the ICSE(SEIP)'19: Software Engineering in Practice, Montreal, QC, Canada, 25–31 May 2019; pp. 121–130. [\[CrossRef\]](#)
55. He, S.; Zhu, J.; He, P.; Lyu, M.R. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics. *arXiv* **2020**, arXiv:2008.06448.
56. Oliner, A.; Stearley, J. What supercomputers say: A study of five system logs. In Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), Edinburgh, UK, 25–28 June 2007; pp. 575–584. [\[CrossRef\]](#)

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.