IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 September 2023, accepted 18 September 2023, date of publication 26 September 2023,
date of current version 6 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3319438

RESEARCH ARTICLE

Failure Detection Using Semantic Analysis and Attention-Based Classifier Model for IT Infrastructure Log Data

DEEPALI ARUN BHANAGE[1,2], AMBIKA VISHAL PAWAR[1], KETAN KOTECHA[3,4], AND AJITH ABRAHAM[4,5], (Senior Member, IEEE)

[1]Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune 412115, India
[2]PCET's, Pimpri Chinchwad College of Engineering, Pune 411044, India
[3]Symbiosis Centre for Applied Artificial Intelligence, Symbiosis International University, Pune 412115, India
[4]School of Computer Science, Engineering and Technology, Bennett University, Greater Noida, Uttar Pradesh 201310, India
[5]Center for Artificial Intelligence, Innopolis University, 420500 Innopolis, Russia

Corresponding authors: Ambika Vishal Pawar (ambikadshelke@gmail.com) and Deepali Arun Bhanage (phdgrad.deepali.bhanage@siu.edu.in)

This work was supported by the Analytical Center for the Government of the Russian Federation under Agreement 70-2021-00143 dd.01.11.2021 and Agreement IGK 000000D730321P5Q0002.

ABSTRACT The improvement in the reliability, availability, and maintenance of the IT infrastructure components is paramount to ensure uninterrupted services in large-scale IT Infrastructures. The massive system logs generated by infrastructures have proved to be advantageous to pursue the runtime circumstances and behavior of the system. Existing literature has log-based failure detection techniques carrying semantic analysis but on limited log features, reflecting ineffectiveness in anomaly detection for unstable and unseen log records. We have proposed in this paper a semantic log analysis model with three log features to apprehend the gist of the log message. BERT pre-trained model is employed to adapt the feature embedding. The generated numerical vectors are further furnished to train an attention-based OLSTM (Optimized Long Short-Term Memory Networks) classifier to detect failures in diverse infrastructures. The proposed model is evaluated on five different infrastructures: Apache from a server application, OpenStack from the Distributed Systems, Windows from the Operating System, BGL from a Supercomputer, and Android from the Mobile System. The findings illustrate that the proposed system delivers improved and stable results, considering the varied IT infrastructures.

INDEX TERMS Log analysis, system log, IT infrastructure, deep learning, BERT.

I. INTRODUCTION

The usage of IT infrastructure has been growing expeditiously over the past few years. Due to unavoidable shortcomings in operating software and hardware, IT infrastructures are prone to failures that result in system outages. Any minor unavailability of services gives rise to catastrophic failures and results in financial [1] together with productivity losses [2]. Large-scale IT infrastructures such as Supercomputers, Distributed systems, Cloud Infrastructures, etc., are tough to control as failures grow with their volume and complexity. Furthermore, failure mitigation takes more than billions of dollars of investments; developers consume most of their hours debugging problems [3]. Prompt and precise identification of such failures is pivotal for the improved reliability, stability, and mitigation of casualties in complex IT Infrastructures [4].

System logs register every detail of the executed operation and provide a lot of dimensional information about it. The system logs document the cause of the problem of IT infrastructure components. The system log is the first place where the system administrator investigates issues on failure alerts. Consequently, system logs have been widely used in anomaly and failure detection or prediction because of their

The associate editor coordinating the review of this manuscript and approving it for publication was Prakasam Periasamy.

directness and usefulness [5], [6], [7], [8]. Complex computer systems record a massive collection of logs that can make useful information available; on the other hand, analyzing colossal data is challenging.

Moreover, system logs present communication between data, files, services, operations, applications, etc. They are occupied with analyzing system behavior and resolving the problems that may emerge [9]. According to a systematic literature survey [10], lots of work has been done in anomaly and failure identification and prediction. The study retrieves a substantial number of articles focusing on system logs.

Although ample system logs are available, which are rich in information to conduct analysis, it imposes significant challenges. System logs are unstructured, unstable, and in enormous formats [11]. Thus, designing a generalized tool to analyze system logs with various formats is tricky. Moreover, manual evaluation of voluminous log data is error-prone and infeasible. The log parsing process is introduced to overcome the stated issues; first of all, unstructured logs are converted into a structured format. The unstructured log messages are presented in the constant and variable part against the parsing operation. The constant part is from the logging statement (log template), and the variable part is the parameters recorded on the execution of the event. In the existing literature, various log parsing tools are available [12].

In the log-based anomaly or failure detection literature, Machine Learning [13], [14] and Deep Learning [15], [16], [17], [18] are popular techniques that are effectively applied to classify logs. This classification can save time on log analysis and assist system administrators in concentrating on doubtful log entries. System logs are a fusion of text, numbers, and special symbols. The data are available in natural language format and cannot be used directly to build Machine Learning and Deep Learning models. Therefore, it is required to follow the action of text data conversion into numerical vectors. In the existing literature, researchers applied indexed-based methods [19], [20], [21] or semantic-based methods [22], [23], [24] to extract the features of logs. In the index-based extraction, log data is converted to log template indexes, and afterward, sequential or quantitative features are extracted against the generated indexes. Such indexes can perform adequately on stable log data but fail to handle unstable log data [18]. The direct conversion of log entries to numerical representation does not furnish reliable results due to the following challenges in log data:

1) System-dependent data formats: Currently, there is no standard style or template to be followed to write logging statements. Due to this, developers write logging statements in different forms. Logs are recorded in various formats depending on the utility of the components in different infrastructures.
2) Imbalanced data: IT infrastructures usually run regularly. Hence, anomalous records are lesser than regular execution logs in the historical data compared to non-anomalies records. As a result of the imbalanced dataset, generated models are unable to analyse newly introduced logs accurately.
3) Use of common words for different purposes: In the case of system logs, common words represent different meanings, and frequently occurring words are unnecessary. Therefore, word-based vector generation does not contribute advantageously to the case of recently developed logging statements.

Semantic-based indexing is the preferred approach to resolve the abovementioned challenges [25]. Many researchers have employed Natural Language Processing (NLP) techniques for semantic-based indexing on log/event templates [13], [22], [26]. The majority of the researchers utilized only a log template as a feature in the process of anomaly or failure detection. Although it accomplished a better outcome, they ignored the other parameters in the log records that may contribute significantly to understand the current behavior of the system. As event templates are the constant part of the log message, it is standard for similar types of logging statements.

In contrast, the system's actual behavior can be tracked based on the runtime detail of the logging statement. Such runtime details get recorded under the parameters of the variable part. Therefore, the variable part (Content of the log message, parameter list, temporal information, etc.) can be used as features, which can identify heterogeneous failures accompanying the unseen log records.

This paper proposed a semantic-based log analysis technique for failure detection to handle various infrastructures' diverse nature log records and overcome the limitations of the existing methods. The various infrastructure logs are studied to summarize the common log format. Furthermore, the BERT (Bidirectional Encoder Representations from Transformers) pre-trained [27] model is employed to extract the semantic embedding of the selected features. Then, extracted vectors of a combination of log Content, EventTemplate, and ParameterList are provided as input to the optimized Long-Short-Term Memory classifier. The proposed system can spontaneously understand the eminence of different logs and context-dependent knowledge in the log message. Subsequently, an alert generated on abnormal behavior detection will be sent to the system admin, along with the list of probable solutions. This will help the system admin quickly mitigate the failure to avoid subsequent losses. Five different IT infrastructure logs are utilized to perform the experiments. The experimental findings demonstrate that the proposed system efficiently contributes to different IT infrastructures. It helps to diminish manual errors significantly by enabling automated and accurate solutions to failure detection.

The contributions in this paper are summarized in the following way:

1) We extracted common and essential log features of various IT infrastructures by studying and analyzing different structured log entries.
2) We proposed failure detection techniques by taking advantage of additional log features (log Content,

EventTemplate, and ParameterList) to improve performance in unseen log records.
3) We built the semantic-based encoding method to precisely procure the core meaning of the combined log features such as log Content, EventTemplate and ParameterList and generate embedding.
4) We implemented Optimized LSTM classification models and evaluated on five infrastructure logs from various categories: Apache from a server application, OpenStack from the distributed system, Windows from the operating system, BGL from a supercomputer, and Android from the mobile system. The analysis demonstrates improved and persistent performance in identifying various failures.

The remaining part of the paper is structured in the following way: Section II discusses the background. Section III provides the details of the designs and methodology used in the proposed system. Section IV emphasized the experiment settings and derived results. Section V states brief related work, and eventually, the concluding remarks and future directions are expressed in Section VI.

II. BACKGROUND

A. STUDY OF LOG FEATURES

Software programmers define system logs by virtue of logging statements (e.g., printf(), logger.info()) while developing software [28]. Consequently, system logs are substantially diverse in the case of various IT Infrastructures. Table 1 presents the unstructured and structured log formats of 16 systems from 6 different IT Infrastructure categories [29]. Although the systems fall under the same Infrastructure category, the log format varies. The unstructured logs are available in the regular text sentence. It is segregated under specific columns (structured format) with the help of log parsing. However, observations say all the logs carry standard features (highlighted in bold). These features symbolize the log header, which includes the date, time, and timestamp as temporal information, and the log message, which includes contents, level, parameters, etc. Log messages are critical in the case of event details extraction as they record dynamically at run time. Eventually, six commonly occurring features are selected as essential parameters to get noteworthy information about the system. Where <**LineId**>: Sequential record, <**time**>**:** temporal information, <**Content**>: recorded message during event execution, <**EventId**>: unique event id under a specific set of events, <**EventTemplate**>: static part of the log message, and <**ParameterList**>: dynamic part of the log message. Our research mainly focuses on log Content, EventTemplate, and ParameterList out of commonly occurring features.

B. USEFULNESS OF SYSTEM LOGS

System logs originated from free-from-text structure logging instructions are written by software developers. They are predominantly designed to monitor and document runtime system status and critical events. System logs are a rich origin of evidence and extensively occur in each and every software system. Moreover, system behavior can be understood after analyzing the historical logs. Thus, log analysis can furnish additional dimensional evidence to identify the failure. Due to forthrightness and efficacy, system logs are the first choice by the system administrator for troubleshooting problems. As a result, system logs play a valuable role in maintaining the IT infrastructure's health.

Despite the usefulness of the rich system logs, there are a few challenges, such as 1) unavailability of logs due to sensitivity, 2) colossal data size, as it records each event in the system, and 3) imbalanced data due to continuous smooth operational characteristics. System logs hold details of each event happening in all IT Infrastructure components. It carries crucial and confidential data. Due to strict business rules, logs are not readily available. The constantly operating infrastructures produce massive logs (around 50GB/hour), which makes their analysis challenging. Generally, IT services work continuously; thus, anomalous records are lesser than successful events.

Figure 1 presents the elements of the sample Windows log. Here, the log is initially present in a single sentence in a natural language format, but it comprises of log header and log message. The log header comprehends temporal data, severity level, and event source information. A log message is a combination of scripted statements and parameters updated at runtime. The log level plays a crucial role in failure detection out of these elements. Log4j [30] states that logs carry six levels: defaults, info, debug, trace, warn, fatal, and error [11]. The log entry indicates "warn," "fatal," and "error" log levels that can be considered anomalous records and need more attention by the system administrator. Table 2 describes elements of the Windows log. These elements commonly occurring in almost all types of infrastructure logs.

C. COMPARISON WITH EXISTING TECHNIQUES

Many researchers recently adapted NLP techniques for vector generation by using single or multiple log features. The vector generation was conducted based on word or sentence embedding techniques. In this section, a comparative analysis of existing research is carried out based on the applied embedding techniques and the log features utilized in the experimentation. Table 3 furnishes the list of state-of-art anomaly detection tools, vector generation techniques, and log features.

LogAnomaly [22], Logtransfer [32], LogFlow [34], Sprelog [17], and LogUAD [13] tools performed word embedding on LogTemplate for anomaly detection. Word2 Vec, Glove, and Template2Vec (inspired by Word2Vec) methods were employed for word embedding. Word embedding techniques cannot embed Out-Of-Vocabulary (OOV) words as they are typically trained on the vocabulary of historical logs. Although these tools detect abnormal log sequences

TABLE 1. Different IT infrastructure logs and their structured format.

System	Unstructured log	Structured log format
	Distributed System	
Hadoop	"2015-10-18 18:01:53,447 INFO [main] org.apache.hadoop.mapreduce.v2.app.rm.RMContainerRequestor: nodeBlacklistingEnabled:true"	<**LineId**><Date><**Time**><Level><Process><Component><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
HDFS	"081109 203519 147 INFO dfs.DataNode$PacketResponder: PacketResponder 0 for block blk_-1608999687919862906 terminating"	<**LineId**><Date><**Time**><Pid><Level><Component><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
Spark	"17/06/09 20:10:54 INFO storage.BlockManager: Found block rdd_2_1 locally"	<**LineId**><Date><**Time**><Level><Component><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
Open Stack	"nova-compute.log.1.2017-05-16_13:55:31 2017-05-16 00:01:55.266 2931 INFO nova.compute.manager [-] [instance: b562ef10-ba2d-48ae-bf4a-18666cba4a51] VM Stopped (Lifecycle Event)"	<**LineId**><Logrecord><Date><**Time**><Pid><Level><Component><ADDR><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
Zookeeper	"2015-07-29 19:36:49,735 - INFO [/10.10.34.11:3888:QuorumCnxManager$Listener@493] - Received connection request /10.10.34.13:33255"	<**LineId**><Date><**Time**><Level><Node><Component><Id><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
	Supercomputer	
BGL	"KERNSTOR 1118765205 2005.06.14 R15-M1-NF-C:J11-U11 2005-06-14-09.06.45.752792 R15-M1-NF-C:J11-U11 RAS KERNEL FATAL data storage interrupt"	<**LineId**><Label><Timestamp><Date><Node><**Time**><NodeRepeat><Type><Component><Level><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
HPC	"2572286 node-17 action start 1074126278 1 bootGenvmunix (command 1903)"	<**LineId**><LogId><Node><Component><State><**Time**><Flag><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
Thunderbird	"- 1131567060 2005.11.09 tbird-admin1 Nov 9 12:11:00 local@tbird-admin1 xinetd[1798]: removing echo"	<**LineId**><Label><Timestamp><Date><User><Month><Day><**Time**><Location><Component><PID><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
	Operating System	
Windows	"2016-09-28 04:30:31, Info CBS Warning: Unrecognised packageExtended attribute."	<**LineId**><Date><**Time**><Level><Component><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
Linux	"Jun 15 04:06:18 combo su(pam_unix)[21416]: session opened for user cyrus by (uid=0)"	<**LineId**><Month><Date><**Time**><Level><Component><PID><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
Mac	"Jul 1 09:02:26 calvisitor-10-105-160-95 kernel[0]: en0: channel changed to 1"	<**LineId**><Month><Date><**Time**><User><Component><PID><Address><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
	Mobile System	
Andriod	"03-17 16:13:38.955 2227 2227 I PhoneStatusBar: cancelAutohide"	<**LineId**><Date><**Time**><Pid><Tid><Level><Component><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
HealthApp	"20171223-22:15:30:335|Step_LSC|30002312|onExtend:1514038531000 1 0 4"	<**LineId**><**Time**><Component><Pid><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
	Server Application	
Apache	"[Sun Dec 04 04:54:18 2005] [error] mod_jk child workerEnv in error state 6"	<**LineId**><**Time**><Level><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
OpenSSH	"Dec 10 07:28:25 LabSZ sshd[24263]: Received disconnect from 112.95.230.3: 11: Bye Bye [preauth]"	<**LineId**><Date><Day><**Time**><Component><Pid><**Content**><**EventId**><**EventTemplate**><**ParameterList**>
	Standard Software	
Proxifier	"[10.30 18:03:15] Skype.exe - 91.190.216.125:443 close, 5 bytes sent, 0 bytes received, lifetime 00:04"	<**LineId**><**Time**><Program><**Content**><**EventId**><**EventTemplate**><**ParameterList**>

TABLE 2. Description of elements of windows log.

ELEMENT	DESCRIPTION
Time Created	The time stamp that identifies when the event was logged.
Level	Contains the severity level of the event.
Component/ Source	Identifies the provider that logged the event.
Message	It contains the event message that is rendered for the event.

effectively, they fail to analyze unstable and newly generated log records. The use of a single feature (LogTemplate) may result in the loss of information such as IP address, component number, error message, etc. Thus, there was a need to add more log features in the analysis so that valuable information would not be discarded. Swisslog [31], HitAnomaly [33], DeepSyslog [26], and BERT-Log [35] performed sentence embedding with the help of BERT or Re-BERT pre-trained models. Moreover, these tools put additional features of logs along with LogTemplate for the analysis. Swisslog combined LogTemplate and Time features but could not detect event parameter-based anomalies. HitAnomlay combined LogTemplate and Parameter for research but was unprepared to detect time interval-based anomalies. Our proposed system selected three essential features concerning the study and observation. Selected features like: Log Content, EventTemplate, and ParameterList, which will give details about the structure of the logging statement, list of resources, and error messages.

III. DESIGN OF THE PROPOSED SYSTEM

A. OVERALL FRAMEWORK

The system is proposed to leverage the three features, such as Log Content, EventTemplate, and ParameterList of logs,

TABLE 3. Comparative analysis of state of art vector generation-based anomaly detection techniques and proposed system.

Technique	Year	Vectors Generation		Log features Used				
		Word	Sentence	Template	Time	Parameter	Content	Other
LogAnomaly[22]	2019	✓		✓				
Swisslog[31]	2020		✓	✓	✓			
Logtransfer[32]	2020	✓		✓				
HitAnomaly[33]	2020		✓	✓		✓		
LogFlow[34]	2021	✓		✓				
Sprelog[17]	2021	✓		✓				
LogUAD [13]	2022	✓		✓				
DeepSyslog[26]	2022	✓	✓					✓(Event Metadata
BERT-Log[35]	2022		✓	✓				
Proposed System			✓	✓		✓	✓	

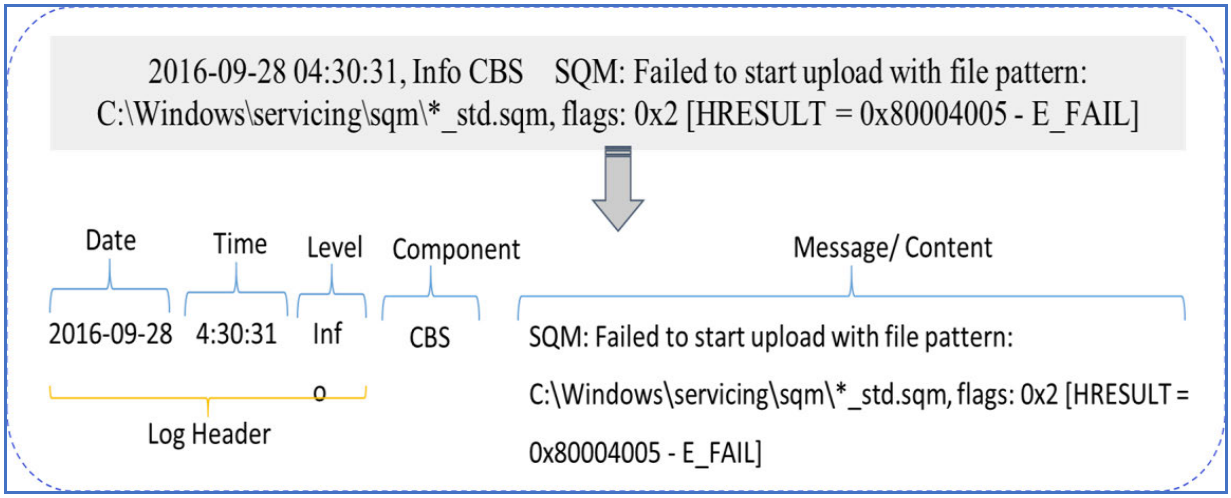

FIGURE 1. Log parsing of sample windows log entry.

to detect log failures in various IT Infrastructures. The historical logs are used for model training, whereas new logs are used for failure detection. The overall architecture of the proposed system is depicted in Figure 2 with three major phases: The first phase is Data Acquisition (Section III-B). The collected raw logs are converted to a structured format using the Log Parsing Technique in this stage. The second phase is Log Semantic Embedding (Section III-C):

In this phase, Log Content, EventTemplate, and ParameterList as log features are leveraged to excerpt the semantic embedding by applying a BERT pre-trained model. To identify the manifested failures in the extracted log features, the third phase (Section III-D) is presented. In this phase, an attention-based classification technique comprised of LSTM plus other layers has appertained to capture the importance of log sequences based on the context. Thus, the significant log sequences will contribute to the detection of failures.

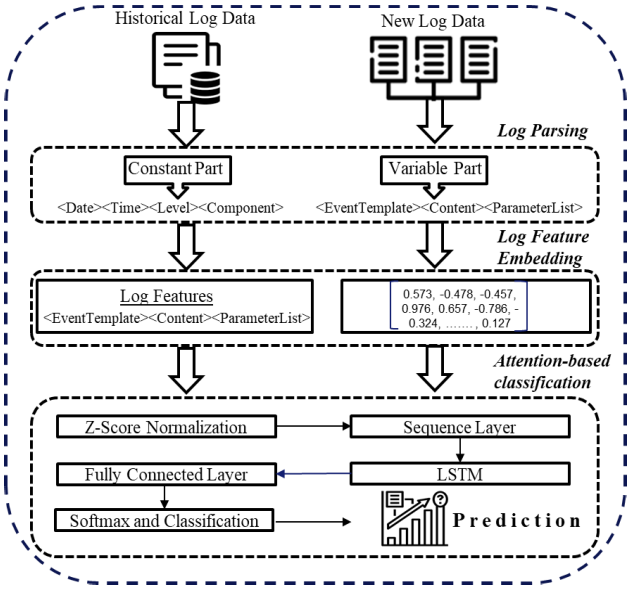

FIGURE 2. The overall architecture of the proposed system.

B. DATA ACQUISITION

Developers' script, logging statements with the help of the "print" command while developing software. Furthermore, these statements get executed on the occurrence of an event and result in system logs. There is no defined format present

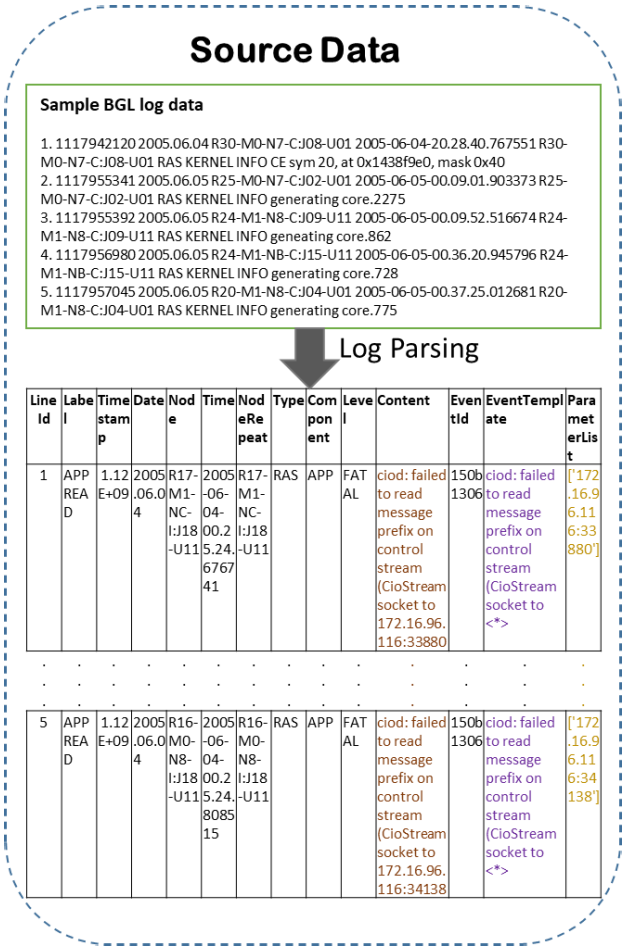

FIGURE 3. Illustration of parsing BGL logs data from unstructured to structured logs.

for writing logging statements. Thus, developers follow their style while scripting. In addition, various types of logging statements are present due to the increase in the use of open-source software [19]. The logs generated through such logging statements are in a raw or unstructured format that needs to be converted to a structured format, with the help of log parsing, before they are used in log analysis. Figure 3 represents the conversion of the BGL raw log into a structured log.

Log statements are disunited during parsing into the label, timestamp, date, level, Content, EventTemplate, and parameter list, as commonly used features.

The system logs are composed of constant and dynamic parts. The constant part is scripted in the logging statements, whereas the dynamic part updates on the execution of the logging statement. The primary task of the parser is to identify the header (timestamp, level, component, etc.) part and process the content part to extract the EventTemplate and ParameterList. Miscellaneous log parsers [36], [37], [38], [39] have been developed by many researchers in the existing literature. Accuracy and efficiency are vital parameters in the selection of log parsers. According to the comparative analysis conducted in our research [10], the Drain [40] parser achieves the optimum performance.

Drain [41] parser is adopted in this research. Drain works on the concept of the steady-depth tree structure. In the tree structure, the leaf nodes in the tree preserve log clusters; likewise, the in-between nodes of the tree implant various Heuristic rules. Log records are present in the structured format on parsing, as demonstrated in Figure 3. However, the parsing procedure may inject certain noise because of the variations in the log format. The proposed system handles the noise by performing a semantic analysis of 3 log features.

C. LOG SEMANTIC EMBEDDING

The research mainly aims to identify the failures in the various IT infrastructures. We focused on analyzing 3 log features in the log semantic embedding. As stated in Section III-B, system logs are diverse in nature, and noise gets imported during the parsing process. However, the original meaning of the log statement remains intact. Therefore, semantic embedding is the appropriate solution to produce vectors. This research uses the BERT pre-trained model to withdraw semantic vectors of logs.

In the recent past, ample researchers applied NLP techniques to analyse system Logs, considering that logs are present in the natural language. In the literature, TF-IDF [14], Word2Vec [42], [43], and Glove [32] techniques were used to conduct log analysis, but these techniques failed to handle homophones, homonyms, and out-of-vocabulary words. Due to these limitations, these techniques are unsuitable for unstable and newly generated log records. The cutting-edge approaches are considered for the study to tackle the limitations of stated NLP techniques. In the experimentation process comprises of, pre-trained word embedding models are applied for vector representation of Log Content, EventTemplate, and ParameterList and to strengthen the prediction of unobserved log entries. The BERT model is pre-trained on massive datasets like Wikipedia and proposed by Google to be fine-tuned on a particular dataset. Moreover, BERT supports domain-specific semantic information and can address Out-Of-Vocabulary (OOV) words in novel kinds of logs during runtime [23].

Algorithm 1 BERT-Based Features Extraction

Input
Input_Dataset: Pre-processed log features
Output
F: Numerical feature vectors for each input log entry

1. Start
2. md = pretrained BERT Model
3. hiddenunits = 128
4. T(i) = tokenization (md, Input_Dataset (i))
5. B(i) = encoding (md, T(i), hiddenunits)
6. F(i) = doc2sequence(B (i))
7. Return (F)
8. Stop

Let us consider the dataset D, which contains m a number of log parameters and n a number of total recorded logs. Each $i^{th} \in n$ log contains various parameters like lineid, date, Time, Level (labels), Content, EventId, EventTemplate, ParameterList, etc. Each IT Infrastructure log dataset has these standard vital parameters and a few corresponding ones. The research performs pre-processing to extract the essential parameters from each log, which is responsible for reasonably predicting failures. We have extracted and combined three fields (Content, EventTemplate, and ParameterList) from each log during the pre-processing phase.

The pre-processing steps are:

$$d1 = D(i) \cdot Content \tag{1}$$

$$d2 = D(i) \cdot EventTemplate \tag{2}$$

$$d3 = D(i) \cdot ParameterList \tag{3}$$

As given below, the combined feature vector is built, and all three parameters in string format are joined.

$$d(i) = \{d1, d\ 2, d3\} \tag{4}$$

$$Input\_Dataset(i) = strjoin(d(i)) \tag{5}$$

After pre-processing, we performed the BERT-based features extraction with the help of the pre-trained BERT model of tiny size. The steps of the BERT model are given mathematically in Algorithm 1. The process of semantic embedding is depicted diagrammatically in Figure 4.

D. CLASSIFICATION MODEL

After the sentence embedding step, each combination [d(i) = {d1, d2, d3}] of log content (d1), EventTemplate (d2), and ParameterList (d3) is converted into log feature embedding 'B' as well as individual log sequence is consequently denoted as 'F(i) = doc2sequence(B(i))'. The generated log embedding sequence is supplied to the proposed Optimized LSTM to detect failures, as shown in Figure 5.

The LSTM network is an Artificial Neural Network designed to find and excerpt long-range correlation in sequential data [44] that can capture The contextual information of the sequence. Several existing researchers [14], [22], [32], [34], [45] exercised LSTM and exhibited its productiveness in log-based failure identification. However, the existing LSTM model for the text classification problem domain suffers from serious challenges: 1) longer training time, 2) more memory for model training, 3) Dropout is harder to implement, and 4) LSTMs are sensitive to different random weight initializations.

The first two problems are mainly caused due to the use of the word embedding layer in LSTM. A word embedding layer maps a sequence of word indices to embedding vectors and learns the word embedding during training. This causes other subsequent problems, such as it is hard to apply the dropout layer in existing LSTMs. Due to the lack of a dropout layer, it is accessible to overfitting.

The Optimized LSTM (OLSTM) model is proposed to perform an accurate prediction with minimum computational requirements. The dropout layer replaces the existing LSTM model word embedding layer. The Fully Connected Layer (FCL) is introduced after the LSTM layer proposition to mitigate the overfitting problem. The removal of the word embedding layer accelerates a reduction in training time and memory requirements. Furthermore, the Z-score normalization layer is introduced as the first layer of the OLSTM model, where a feature vector of size 1×128 of each i^{th} log is normalized to the particular standard pattern. This results in a reduction in error rates and an improvement in accuracy.

The sequential layer reads the sequential features for each log. The LSTM layer is applied, which is followed by the Sequential layer. The LSTM layer comprises the input, hidden neurons, and output layers. The remaining layers in the OLSTM are the Dropout layer, FCL layer, and Softmax and Classification Layer (SCL). The OLSTM consists of 6 layers to train the input dataset. After training, the classification is performed to predict the failure in the selected IT infrastructures.

The design of the OLSTM layers is further explained mathematically.

In the z-score normalization layer, each log feature vector is normalized using the z-score. The formula stated in Equation 6 is as follows:

$$N(i) = \left(\frac{(F(i) - \mu)}{\sigma} \right) \tag{6}$$

where, $F(i)$ is the i^{th} log feature vector, μ is the mean set to 0, and σ is the standard deviation set to 1 by default. The output of the normalized feature vector for each log of size 1×128 is stored in N.

The LSTM input layer accommodates N at the current time interval t. The LSTM layer comprises an input gate i, output gate o, forget gate f, and a memory cell c. For every time t LSTM computes its gate's activations $\{i_t, f_t\}$ and updates its memory cell from c_{t-1} to c_t, it then computes the output gate activation o_t and outputs a hidden representation h_t. The hidden representation from the previous time step is h_{t-1}.

The following equations are applied in LSTM for update functions:

$$i_t = \sigma\ (N \cdot W_{xi} + h_{t-1} W_{hi} + c_{t-1} W_{ci} + b_i) \tag{7}$$

$$f_t = \sigma\ \left(N \cdot W_{xf} + h_{t-1} W_{hf} + c_{t-1} W_{cf} + b_f\right) \tag{8}$$

$$o_t = \sigma\ (N \cdot W_{xo} + h_{t-1} W_{ho} + c_{t-1} W_{co} + b_o) \tag{9}$$

$$c_t = f_t \diamond c_{t-1} + i_t \diamond tanh\ tanh\ (N \cdot W_{xc} + h_{t-1} W_{hc} + b_c) \tag{10}$$

$$h_t = o_t \diamond tanh\ (c_t) \tag{11}$$

where N represents the input sequential features vector, i_t, f_t, o_t, c_t, h_t represents input gate, forget gate, output gate, current cell state, and output of LSTM layer, respectively, at the current sequential step t.

W_{xi}, W_{xf}, W_{xo} Represents the weights among the input-input gate, input-forget gate, and input-output gate.

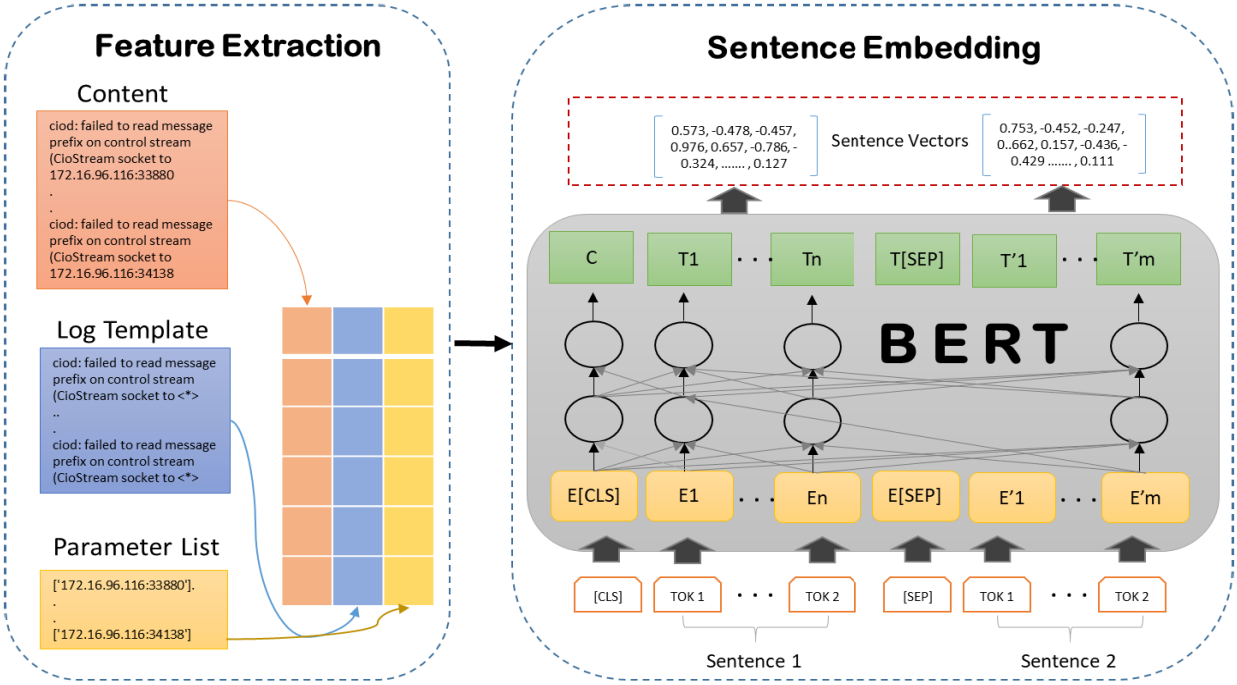

FIGURE 4. Feature extraction and semantic embedding using BERT pre-trained model.

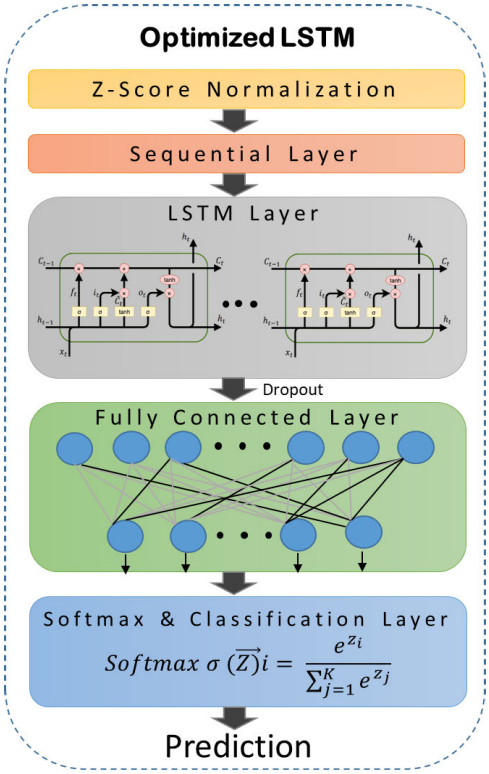

FIGURE 5. The design of attention-based optimized LSTM classifier for failure detection.

W_{hi}, W_{hf}, W_{ho} Represents the weights among hidden recurrent layer-input gate, hidden recurrent layer-forget gate, and hidden recurrent layer-output gate, respectively.

W_{ci}, W_{cf}, W_{co} Represents weights among the cell state-input gate, cell state-forget gate, and cell state-output gate, respectively.

b_i, b_f, b_o, b_c Represents the additive bias functions for the input gate, forget gate, output gate, and cell state, respectively.

The attention technique is commenced to handle the impact of unimportant logs. A Fully Connected (FC) layer is included as an attention mechanism at OLSTM. The different weights are assigned to the distinct log statements to justify the individual's significance. A Fully Connected (FC) layer affirms hidden state h_t as input and outputs the weight of attention α that indicates the relevance of the log message. The activation function is calculated using Equation 12, which is the weight matrix of the attention layer at time t, and tanh $(\cdot)$ is an activation function α_t.

$$\alpha_t = tanh\left(W_t^{\alpha} \cdot h_t\right) \tag{12}$$

Finally, the softmax and classification layer are added to provide the final classification result 'pred' calculated using Equation 13. Where W denotes the softmax layer weight, T represents the total length of the log embedding sequence and $\sum_{t=0}^{T} \alpha_t \cdot h_t$ Is the sum of the results that all hidden states h_t multiply corresponding attention weights αt

$$Pred = Softmax(W' \cdot (\sum_{t=0}^{T} \alpha_t \cdot h_t) \tag{13}$$

IV. EXPERIMENT

Section IV describes the experiment settings in IV A, including dataset description, evaluation metrics, baselines, and implementation. Section IV-B presents experimental results

TABLE 4. Dataset descriptions.

Dataset	Size of Data	No. of logs	Anomalies
Apache	4.90 MB	56,481	38,081
OpenStack	5.4 MB	207,820	3,314
Windows	267.465 MB	611,103	16,372
BGL	708.76 MB	4,747,963	348,460
Android	25.7 MB	1,555,005	159,254

on five different Infrastructure logs. The performance analysis is exhibited with the help of ROC curves.

A. EXPERIMENT SETTINGS

1) DATASET

The system is proposed as a generalized solution for the different types of IT Infrastructures. Thus, to assess the proposed system, we have performed experiments on logs from various IT Infrastructures, such as Apache from a server application, OpenStack from the distributed system, Windows from the operating system, BGL from a supercomputer, and Android from the mobile system. Table 4 lists the statistics summary for log datasets.

a: APACHE

Apache HTTP Server [46] is one of the most famous web servers. The Apache dataset provides an access log (56,481 Number of records) and an error log (38,081 Number of records). This dataset was composed out of a Linux system running an Apache Web server, which consisted of the Public Security Log Sharing Site project [47] for the research on anomaly detection.

b: WINDOWS

The Windows dataset was collected by aggregating several logs from a lab computer running Windows 7. The original logs were located at C:/Windows/Logs/CBS. CBS (Component Based Servicing) is a componentization architecture in Windows that works at the package/update level [29]. There are 2.6% anomalous log entries out of collected Windows logs.

c: OPENSTACK

OpenStack [48] is a cloud operating system that leads to enormous pools of computing, storage, and networking resources around a data center. This dataset was generated on CloudLab [49], a flexible, scientific infrastructure for research on cloud computing. There are 3,314 anomalous records in the collection, which are injected manually to generate a dataset for anomaly detection research.

d: BGL

The BGL dataset collected from BGL is an open logs dataset gathered from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California, Comprising 131,072 processors and 32,768 GB memory [50]. This dataset contains 4,747,963 log records; out of these, 348,460 are labelled as anomalous entries. The label information is convenient for system administrators to distinguish abnormal conditions and accomplish remedial measures.

e: ANDROID

Many smart mobiles work with the Android operating system. Google has developed this operating system, which is becoming very popular [51]. As mobile data carries personal details in the logs, Android logs are not freely available. The Android log used in this paper is generated during the testing of Android smartphones and is released for research purposes.

We leveraged different rations of training and testing data for experimentation. Moreover, these datasets are collected from the existing Repositories. Thus, we take release data as the ground truth in favor of the assessment. All the models are trained individually for each dataset as the log formats differ.

2) EVALUATION METRICS

To determine the efficacy of the proposed system in log classification, we leverage extensively utilized Precision, Recall, F1-Score, and Accuracy as metrics. To calculate these metrics, a confusion matrix is created using True Positive (TP), the number of logs classified accurately by Machine and Deep Learning models. False Positive (FP) refers to a number of logs categorized under the wrong class level. True Negative (TN) represents the number of log records with the different levels categorized under other classes. False Negative (FN) stands for a number of Log records that are not correctly classified.

A true positive (TP) occurs when log entries with identical combinations of Content, EventTemplate, and ParameterList are correctly classified into the same category by manual categorization. True negatives (TN) are log entries with the same mix of Content, EventTemplate, and ParameterList but manually categorized into separate groups. In the context of log entries, a false positive (FP) occurs when log entries with identical combinations of Content, EventTemplate, and ParameterList are manually classified into distinct groups. False negatives (FNs) occur when log entries with the same combination of Content, EventTemplate, and ParameterList are manually classified into the same groups.

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

Precision gives the number of correct classified results divided by the number of classifieds derived from the classifier:

$$Recall = \frac{TP}{TP + FN} \tag{15}$$

Recall provides the number of correct classified results divided by the number of all application instances:

$$F1Score = \frac{2Recall * Precision}{Recall + Precision} \tag{16}$$

F1-Score calculates the harmonic mean by combining the precision and recall values:

$$Accuracy = \frac{TP + FN}{TP + FN + TN + FP} \tag{17}$$

Accuracy shows the percentage of accurately classified logs.

3) BASELINE

Researchers make Machine or Deep learning classifiers a popular choice for automatic data analysis, collecting valuable insights, and simplifying processes. Machine Learning (ML) and Deep Learning (DL) classifiers are advantageous for automating the prior manual work. Classifiers learn the pattern from the provided dataset and then classify the new records into several classes. This research utilizes various classifiers to organize log entries concerning levels specified in the log entry. The output of the classification is a class or group but not any specific value such as ''error,'' ''info,'' ''fatal,'' ''warning,'' etc. It is tough to select a single classifier that would constantly and effectively operate. Thus, multiple models were tested to record the classification accuracy. This study includes k-Nearest Neighbors, Linear Regression, Support Vector Machines, Naïve Bayes, Gradient Boosting Decision Trees, Random Forests, LSTM, and modified LSTM as OLSTM (Optimized LSTM). We compared the achievement of the proposed OLSTM with the other seven models.

K-Nearest Neighbor (KNN) is a machine-learning technique that detects outliers in log records based on their distance [52]. It estimates the similarity between log entries and adds a new log to the most similar category. Logistic Regression classifies records using a sigmoid curve [53], while Support Vector Machines (SVM) classify data and train models using super finite degrees of polarity [54]. Naive Bayes classifies unstructured data using a posterior probability, but this assumption may not be accurate [55]. Gradient Boosting Decision Trees (GBDT) is a classifier that builds decision trees in succession to enhance performance [56]. Random Forest is a popular classifier due to its ensemble of multiple decision trees, which reduces variance and is suitable for classifying unseen log records [57]. LSTM is a Recurrent Neural Network (RNN) modification that has gained popularity for sequential learning. It addresses vanishing gradient and gradient explosion issues during long sequence training, improving performance in longer sequences. LSTM uses an individual memory cell to memorize long-term dependencies, possibly refreshed based on input. System logs are a chronological order of log entries, and abnormal behaviors can be tracked based on generated log sequences. LSTM is particularly useful for representing intricate sequential dependencies in various logs and capturing prospective non-linear and high-dimensional dependencies among keywords similar to log records during model training [16].

4) IMPLEMENTATION

All models are implemented in Python and executed on the server of Symbiosis Institute of Technology (Pune, India), which is configured with NVIDIA DGX Station with 251GiB System memory, Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz, 64bit, and 4 GPUs: Tesla V100-DGXS-32GB. Various datasets such as Apache, OpenStack, Windows, BGL, and Android were utilized to conduct the experimentation. Table 4 states that these datasets carry a vast number of log messages ranging from around Fifty-six thousand to Eleven million. The excellent server configuration made possible the execution of the BERT feature extraction technique and classifiers on a massive data size.

Few substantial hyperparameters are selected to make model training efficacious as concerns both time and fit. The proposed method has been implemented using the following hyperparameters for BERT Pre-trained and OLSTM Models. To improve the computational speed of the BERT model, the values of the hyperparameters are referred to as suggested in [58]. Where max_seq_len is considered 25, pooling_layer is equal to -12, priority_batch_size is set as 16, and prefetc_size is 10. In addition, in the OLSTM model, the hyperparameters were utilized as follows. The number of epochs for model training is 50 with a minimum batch size of 32, and the number of hidden layers is fixed at 20. Experimentation was conducted on the different splits of all datasets to check the model's performance. In the first iteration of model training, the split ratio was considered 50%-50%, and then the training dataset proportion increased to 60%, 70%, and 80%. At the same time, the testing dataset proportion decreases to 40%, 30%, and 20%, respectively. Further specifications regarding the dense layer, dropout layer, and Activation function are discussed in section III-D.

B. RESULTS AND ANALYSIS

The effectiveness of the proposed system is evaluated on different IT infrastructure logs by applying eight classifiers. The comparative analysis of baseline models and the proposed system is demonstrated as follows:

1) EXPERIMENTS ON APACHE DATASET

Table 5 demonstrates the proposed system's precision, Recall, F1-score, and accuracy compared to seven other classifiers founded on the Apache dataset. The proposed system has achieved the highest performance across implemented methods, with 98.99% precision, 98.19% recall, 98.58% F1-Score, and 98.09% accuracy. Machine Learning-based classifiers performed failure detection with F1-Score from 90% to 92%,

TABLE 5. Experimental results of the apache dataset.

Model	Precision	Recall	F1-Score	Accuracy
KNN	94.51	87.25	90.74	86.95
Linear Regression	95.00	89.83	93.13	90.01
SVM	95.88	90.53	93.13	90.22
Naive Bayes	91.78	89.89	91.31	89.46
GBDT	95.78	89.09	92.31	89.13
Random Forest	95.79	89.82	92.71	89.65
LSTM	97.98	95.56	96.75	95.78
OLSTM	98.99	98.19	98.58	98.09

TABLE 6. Experimental results of the openstack dataset.

Model	Precision	Recall	F1-Score	Accuracy
KNN	91.03	86.11	88.50	85.17
Linear Regression	93.01	88.33	90.65	88.02
SVM	93.43	88.31	90.80	88.14
Naive Bayes	92.79	88.57	92.10	87.94
GBDT	93.69	88.64	91.10	87.81
Random Forest	95.13	89.31	92.13	89.34
LSTM	96.98	94.09	95.51	94.13
OLSTM	98.02	97.12	97.57	97.02

such as KNN, Linear Regression, SVM, Naive Bayes, BDT, and Random Forest. In comparison to above stated models, Neural Network-based classifiers detect failures with improved F1-scores from 96% to 98%, such as LSTM and OLSTM. The results suggest that LSTM and OLSTM classification on semantic analysis conducted using the BERT pre-trained model are superior for acquiring the semantic information of log features and improved recall, the correct percentage of failure detection from all the present failure records. After failure detection, notifications are needed to be sent to the system admin to take corrective actions.

Moreover, it is essential to identify an accurate failure without a false alarm. The proposed system records a Recall of 98.19%, which is significantly better than other methods. It proves that the precise meaning of log messages is extracted with the help of semantic embedding, carried out by the BERT pre-trained model. The authors train all the models and record the results on randomly selected training and testing records.

2) EXPERIMENTS ON OPENSTACK DATASET

Table 6 demonstrates the Precision, Recall, F1-score, and accuracy of the proposed system compared to seven other classifiers founded on the OpenStack dataset. The proposed system has achieved the highest performance across implemented methods, with 98.02% precision, 97.12% recall, 97.57% F1-Score, and 97.02% accuracy. Machine Learning-based classifiers performed failure detection with F1-Score from 88% to 92%, such as KNN, Linear Regression, SVM, Naive Bayes, BDT, and Random Forest. In comparison to above stated models, Neural Network-based classifiers detect failures with improved F1-scores from 95% to 97%, such as LSTM and OLSTM. The results suggest that LSTM and OLSTM Classification conducted on the semantic analysis, using the BERT pre-trained model, are superior for acquiring the semantic information of log feature. Also, it delivers improved recall of the correct percentage of failure detection from all the present failure records. After failure detection, notifications are needed to be sent to the system admin to take corrective actions.

Moreover, it is essential to identify an accurate failure without a false alarm. The proposed system records a Recall of 97.12%, which is significantly better than the other methods. It proves that the precise meaning of log messages is extracted with the help of semantic embedding, which the BERT pre-trained model carries out. The authors train all the models, and results are recorded on randomly selected training and testing records.

3) EXPERIMENTS ON WINDOWS DATASET

Table 7 demonstrates the precision of the proposed system, Recall, F1-score, and accuracy compared to seven other classifiers founded on the Windows dataset. The proposed system has achieved the highest performance across implemented methods, with 98.23% precision, 97.23% recall, 97.73% F1-Score, and 97.89% accuracy. Machine Learning-based classifiers performed failure detection with F1-Score from 84% to 86%, such as KNN, Linear Regression, SVM, Naive Bayes, BDT, and Random Forest. In comparison to above stated models, Neural Network-based classifiers detect failures with improved F1-scores from 96% to 98%, such as LSTM and OLSTM. The results suggest that LSTM and OLSTM classification conducted on semantic analysis using a BERT pre-trained model are superior for acquiring the semantic information of log features. After failure detection, notifications are needed to be sent to the system admin to take corrective actions.

Moreover, it is essential to identify an accurate failure without a false alarm. The proposed system records that a Recall of 97.23% is significantly better than other methods. It proves that the precise meaning of log messages is extracted with the help of semantic embedding, which is carried out by the BERT pre-trained model. The authors train all the models, and the results are recorded on randomly selected training and testing records.

4) EXPERIMENTS ON BGL DATASET

Table 8 demonstrates the precision, Recall, F1-score, and accuracy of the proposed system compared to seven other classifiers founded on the BGL dataset. The proposed system

TABLE 7. Experimental results of windows dataset.

Model	Precision	Recall	F1-Score	Accuracy
KNN	85.67	83.78	84.71	83.35
Linear Regression	88.45	87.23	87.84	86.77
SVM	88.45	87.23	87.84	86.77
Naive Bayes	85.53	83.99	84.07	85.23
GBDT	85.53	83.89	84.68	85.73
Random Forest	87.85	85.82	86.82	85.39
LSTM	96.98	95.34	96.15	95.23
OLSTM	98.23	97.23	97.73	97.89

TABLE 8. Experimental results of the BGL dataset.

Model	Precision	Recall	F1-Score	Accuracy
KNN	84.71	81.89	83.28	82.98
Linear Regression	87.22	84.82	86.00	85.79
SVM	87.22	84.82	86.00	85.79
Naive Bayes	85.00	80.00	81.36	85.74
GBDT	84.59	79.49	81.96	85.01
Random Forest	87.57	85.02	86.28	86.89
LSTM	94.89	94.89	95.05	93.23
OLSTM	96.07	96.07	96.59	96.02

has achieved the highest performance across the implemented methods, with 96.07% precision, 96.07% recall, 96.59% F1-Score, and 96.02% accuracy. Machine Learning-based classifiers performed failure detection with F1-Score from 83% to 86%, such as KNN, Linear Regression, SVM, Naive Bayes, BDT, and Random Forest. In comparison to above stated models, Neural Network-based classifiers detect failures with improved F1-scores from 95% to 97%, such as LSTM and OLSTM. The results suggest that LSTM and OLSTM classification conducted on semantic analysis using a BERT pre-trained model are superior for acquiring the semantic information of log features. After failure detection, notifications are needed to be sent to the system admin to take corrective actions.

Moreover, it is essential to identify an accurate failure without a false alarm. The proposed system records Recall of 96.07% is significantly better than other methods. It proves that the precise meaning of log messages is extracted with the help of semantic embedding carried out by the BERT pre-trained model. The authors train the models, and the outcomes are documented based on random training and testing data selection. The utilization of the Randomization approach is contingent upon the dataset reaching a sufficient size and containing relevant log entries. Therefore, a random train-test split can yield a decent approximation of the performance of a model.

TABLE 9. Experimental results of the android dataset.

Model	Precision	Recall	F1-Score	Accuracy
KNN	80.23	86.46	83.23	83.67
Linear Regression	84.05	87.56	85.77	85.79
SVM	84.05	87.56	85.77	85.79
Naive Bayes	80.03	86.98	82.15	84.72
GBDT	82.83	86.98	82.15	84.72
Random Forest	84.33	89.27	86.73	87.56
LSTM	93.49	96.68	95.06	94.99
OLSTM	94.06	98.47	96.21	96.19

5) EXPERIMENTS ON ANDROID DATASET

Table 9 demonstrates the precision, recall, and F1-score of the proposed system compared to seven other classifiers founded on the Android dataset. The proposed system has achieved the highest performance across the implemented methods, with 94.06% precision, 98.47% recall, and 96.21% F1-Score. Machine Learning-based classifiers performed failure detection with F1-Score from 83% to 86%, such as KNN, Linear Regression, SVM, Naive Bayes, BDT, and Random Forest. In comparison to above stated models, Neural Network-based classifiers detect failures with improved F1-scores from 95% to 97%, such as LSTM and OLSTM. The results suggest that LSTM and OLSTM classification conducted on semantic analysis using a BERT pre-trained model are superior for acquiring the semantic information of log features. After failure detection, notifications are needed to be sent to the system admin to take corrective actions.

Moreover, it is essential to identify an accurate failure without a false alarm. The proposed system, which records recall of 98.47%, is significantly better than other methods. It proves that the precise meaning of log messages is extracted with the help of semantic embedding, which is carried out by the BERT pre-trained model. The authors train all the models, and the results are recorded on randomly selected training and testing records.

After thoroughly analyzing the results of various classifiers for the Apache, OpenStack, Windows, BGL, and Android datasets, the authors have claimed that the results are consistent for all IT infrastructure log records. Although the number of log records, the proportion of log templates, and the format of the logging statement vary, the results are unfluctuating. This proves that semantic analysis using the BERT pre-trained model delivers the literal meaning of log messages. Thus, it brings out the meticulous outcomes for different classifiers.

6) ACCURACY EVALUATION

Figure 6 presents the accuracy evaluation of seven classifiers in different IT infrastructures: Apache, OpenStack, Windows, BGL, and Android datasets. The proposed classifier OLSTM

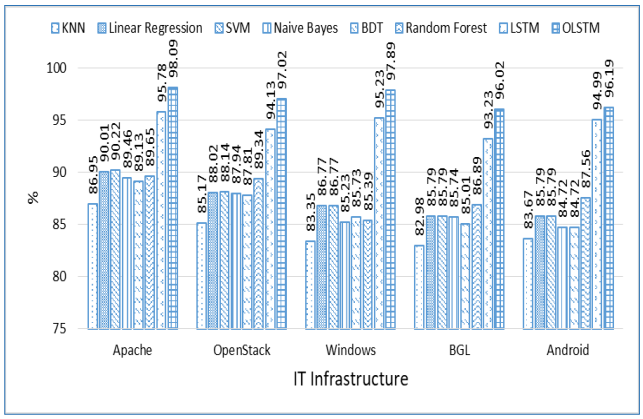

FIGURE 6. Accuracy evaluation and comparative analysis of different classifiers.

delivers the highest accuracy in all the types of IT infrastructures. The accuracy of all the infrastructures does not fluctuate considerably. Thus, it confirms that BERT pretrained model-based semantic analysis is the most suitable for all types of IT infrastructure log records. Although there is not much divergence in the accuracy of LSTM and OLSTM, a significant reduction in the execution time is observed during experimentation. In the case of the OLSTM model, training time declined due to the introduction of new layers (as shown in Figure 5) along with the LSTM layer.

The recorded model training time for both LSTM and OLSTM has been documented across all five datasets pertaining to IT Infrastructures. In the context of an Apache server infrastructure containing 56,481 logs, the OLSTM model demonstrated a 2.93% reduction in execution time compared to the LSTM model. The OpenStack Infrastructure, which falls under the distributed system category, comprises a total of 207,820 log records. In terms of training time, the OLSTM model demonstrates a reduction of 6.83% compared to the LSTM model. A total of 6,11,103 logs from the Windows operating system were trained within a duration of 2009.3 seconds using the OLSTM model, exhibiting a decrease of 5.35% compared to the LSTM model. The dataset used for training consisted of 15,55,005 logs in the Android domain. The training process took a total of 2317.1 seconds. The results obtained from this training indicate an improvement of 8.94% compared to the performance achieved by the LSTM model. The BGL supercomputer, which consisted of 47,47,963 logs, was utilized for training the OLSTM model, resulting in a training time of 3108.89 seconds, making it the most substantial dataset among the options considered. Based on the provided information, it can be noted that there is an improvement in the training time difference between the LSTM and OLSTM models as the dataset size grows.

7) COMPARATIVE PERFORMANCE ANALYSIS WITH STATE-OF-ART TOOLS FOR BGL INFRASTRUCTURE

The proposed system's performance is compared with other state-of-the-art techniques such as PCA, DeepLog, LogAnomaly, and LogBert. The optimal parameters are considered to get the result of all the techniques on the BGL dataset. The implementation of PCA can be found at loglizer [59]. For DeepLog, the code is open source at GitHub [60]. The LogAnomaly code was taken from open source [61], and Guo et al. [62] released code for LogBert to use as a baseline for other researchers.

TABLE 10. Comparative analysis of the proposed system with State-of-art techniques using BGL dataset.

Model	Precision	Recall	F1-Score	Accuracy
PCA	9.07	68.12	14.60	20.24
DeepLog	84.55	87.83	81.22	87.66
LogAnamaly	95.00	67.09	78.64	97.05
LogBERT	87.68	90.45	87.83	91.26
OLSTM	96.07	96.07	96.59	96.02

PCA [19]: Principal Component Analysis anomaly detection is a technique used to identify outliers or anomalies in high-dimensional datasets. It transforms the data into a lower-dimensional space while preserving the most significant variance. Anomalies are detected by measuring the Euclidean distance between data points and their projections onto the lower-dimensional subspace.

DeepLog [45]: DeepLog is a research framework for anomaly detection in system logs, developed to identify unusual patterns in log data. It utilizes deep learning techniques, specifically recurrent neural networks (RNNs), to model sequential dependencies in logs and flag anomalies. By capturing the temporal relationships in log entries, DeepLog can effectively distinguish between normal and anomalous behavior in complex IT systems, making it a valuable tool for cybersecurity and system monitoring.

LogAnomaly [22]: LogAnomaly is a research approach for anomaly detection in system logs designed explicitly for complex IT environments. It employs a combination of techniques, including sequence-to-sequence modelling and attention mechanisms, to capture the intricate relationships in log data and identify unusual patterns or deviations. LogAnomaly can effectively distinguish between normal and anomalous log entries by focusing on contextual information and temporal dependencies, enhancing system monitoring capabilities.

LogBERT [15]: LogBERT is a novel research approach for anomaly detection in system logs, leveraging the power of the BERT (Bidirectional Encoder Representations from Transformers) language model. It fine-tunes BERT on log data to capture the semantic meaning and contextual information in log messages. This enables LogBERT to effectively identify anomalies by recognizing log entries that deviate from the expected linguistic patterns, improving log-based anomaly detection in IT systems.

The statistical data of evaluation parameters is shown in Table 10. The models' most exceptional outcomes are

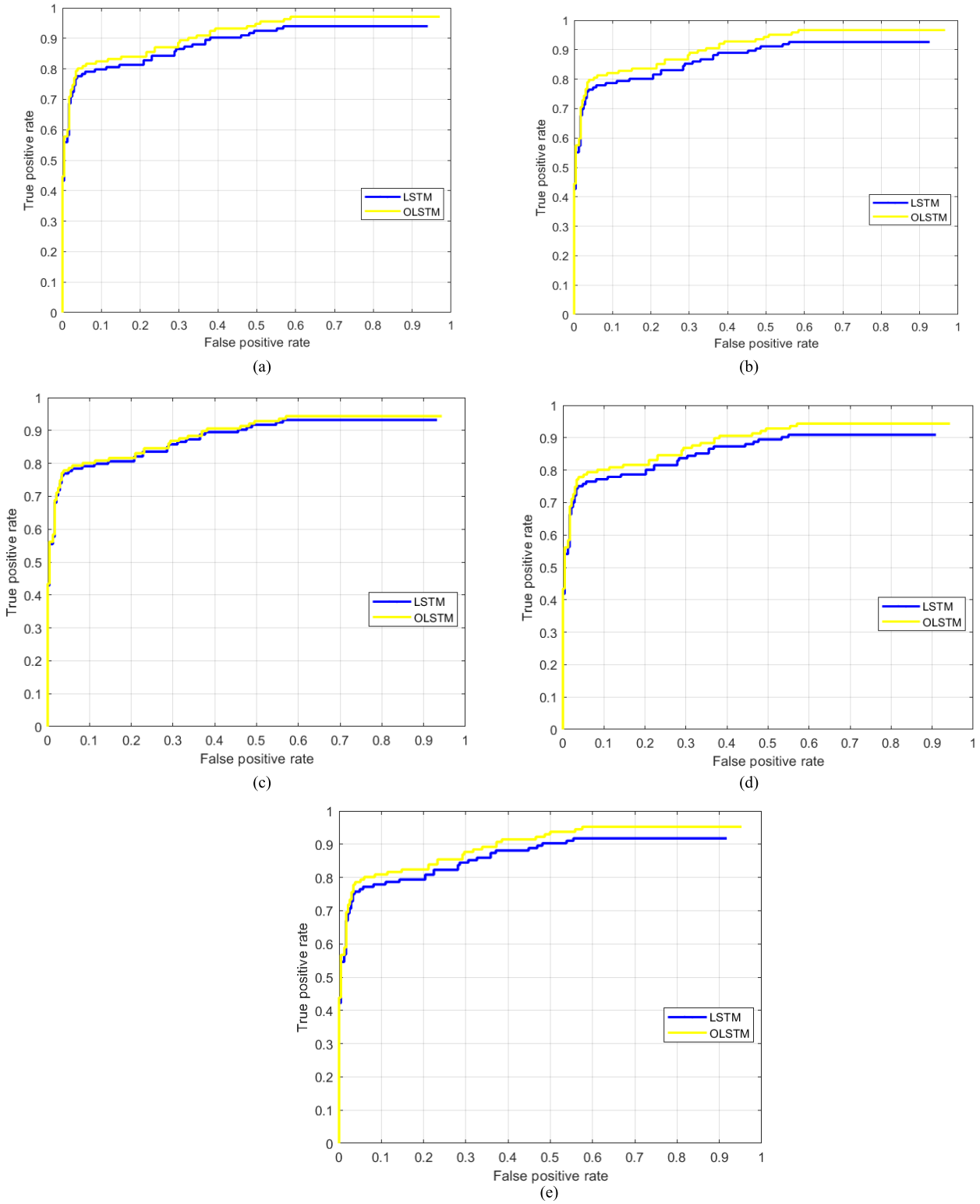

FIGURE 7. a. ROC curve comparison with apache (56,481 logs). b. ROC curve comparison with openstack (2, 07,820 logs). c. ROC curve comparison with windows (6, 11,103 logs). d. ROC curve comparison with bgl (15, 55,005 logs). e. ROC curve comparison with android (47, 47,963 logs).

highlighted in Table 10. The empirical findings provide compelling evidence that the sentence embedding performance of the proposed system using BERT and the attention-based OLSTM model outperformed the state-of-the-art approaches.

Case	Label	Content	EventTemplate	ParameterList	Level
I	Normal	**NtpClient** succeeds in resolving manual peer time.windows.com,0x9 after a previous failure.	NtpClient succeeds in resolving manual peer <*> <*> after a previous failure.	**['time.windows.com,0x9]**	Information
	Anomalous	**NtpClient** was unable to set a manual peer to use as a time source because of DNS resolution error on 'time.windows.com,0x9'. NtpClient will try again in 15 minutes and double the reattempt interval thereafter. The error was: No such host is known. (0x80072AF9)	NtpClient was unable to set a manual peer to use as a time source because of DNS resolution error on <*> <*>. NtpClient will try again in 15 minutes and double the reattempt interval thereafter. The error was: No such host is known. (0x80072AF9)	**['time.windows.com,0x9']**	Warning
II	Normal	svchost (4636,D,0) $RUJet: **A request to write to the file** ""C:\WINDOWS\system32\$RU\$RUtmp.log"" at offset 0 (0x0000000000000000) for 65536 (0x00010000) bytes succeeded, but took an abnormally long time (44 seconds) to be serviced by the OS. This problem is likely due to faulty hardware. Please contact your hardware vendor for further assistance diagnosing the problem.	**svchost <*> SRUJet:** A request to write to the file <*> **at offset** <*> <*> **for** <*> <*> **bytes succeeded, but took an abnormally long time** <*> **seconds) to be serviced by the** OS. This problem is likely due to faulty hardware. Please contact your hardware vendor for further assistance diagnosing the problem.	['(4636,D,0)', '""C:\\WINDOWS\\system32\\$RU\\$RUtmp.log""', '0 (0x0000000000000000)', '65536 (0x00010000)',	Warning
	Anomalous	svchost (4084,D,29) $RUJet: **The database page read from the file** ""C:\WINDOWS\system32\$RU\$RUDB.dat"" at offset 8417280 (0x0000000000807000) (database page 2054 (0x806)) for 4096 (0x00001000) bytes failed verification due to a persisted lost flush detection timestamp mismatch. The read operation will fail with error -1119 (0xfffffba1). This problem is likely due to faulty hardware. Please contact your hardware vendor for further assistance diagnosing the problem.	**svchost <*> SRUJet:** The database page read from the file <*> **at offset** <*> <*> **for** <*> <*> **bytes** failed verification due to a persisted lost flush detection timestamp mismatch. The read operation will fail with error <*> <*>. This problem is likely due to faulty hardware. Please contact your hardware vendor for further assistance diagnosing the problem.	['(4084,D,29)',""C:\WINDOWS\system32\$RU\$RUDB.dat"", '8417280 (0x0000000000807000) (database page 2054 (0x806))', '4096 (0x00001000)','-1119 (0xfffffba1)',]	Error
III	Normal	The description for Event ID 1 from source googledrivefs3758 cannot be found. Either the component that raises this event is not installed on your local computer or the installation is corrupted. You can install or repair the component on the local computer. If the event originated on another computer, the display information had to be saved with the event. The following information was included with the event: Issuing a clean mount manager delete for device ""\Device\Volume{0946a5b5-f5e2-3949-be9f-060c193 The message resource is present but the message was not found in the message table	**The description for Event ID <*> from source <*> cannot be found. Either the component that raises this event is not installed on your local computer or the installation is corrupted. You can install or repair the component on the local computer. If the event originated on another computer, the display information had to be saved with the event. The following information was included with the event:** <*> <*> <*> <*> <*> <*> <*> <*> <*> **The message resource is present but the message was not found in the message table**	['1','googledrivefs3758','Issuing a clean mount manager delete for device ""\Device\Volume{0946a5b5-f5e2-3949-be9f-060c193 ']	Information
	Anomalous	The description for Event ID 2 from source googledrivefs3525 cannot be found. Either the component that raises this event is not installed on your local computer or the installation is corrupted. You can install or repair the component on the local computer. If the event originated on another computer, the display information had to be saved with the event. The following information was included with the event: The message resource is present but the message was not found in the message table	**The description for Event ID <*> from source <*> cannot be found. Either the component that raises this event is not installed on your local computer or the installation is corrupted. You can install or repair the component on the local computer. If the event originated on another computer, the display information had to be saved with the event. The following information was included with the event:** <*> <*> <*> <*> <*> <*> <*> <*> <*> **The message resource is present but the message was not found in the message table**	['2','googledrivefs3525','The driver version of the disk does not match.']	Error

FIGURE 8. Sample log records from windows system logs.

8) CLASSIFICATION EFFECT EVALUATION

The classification quality is analyzed with the help of the generating of the Receiver Operating Characteristic (ROC) curve, which provides the relation between False Positive Rate (FPR) at the X-axis and True Positive Rate (TPR) at the Y-axis for varying thresholds. The area under the ROC curve (AUC) measures the area within the ROC curve and the X-axis. AUC is always present between TPR as one and FPR as 0. The curve nearer to (0, 1) implies a better classification impact. Figure 7a to Figure 7e demonstrates the ROC curve of LSTM and OLSTM models on five datasets. Figure 7a to e shows that the OLTM classifier achieves enhanced AUC values than the LSTM model. It indicates that the proposed classifier OLSTM has an improved classification result than the traditional LSTM model. The addition of new layers in the LSTM model enhanced the performed in the case of classification as well as a reduction in training time.

9) METICULOUS ANALYSIS OF PROPOSED SYSTEM

As a part of the results, meticulous analysis is exercised to endorse the proposed system's originality regarding the list of benefits and limitations. Table 11 briefly discusses the proposed system's benefits and limitations. Five key benefits are summarized based on the vital contribution of the research work, resulting in improved results. However, three limitations of the proposed system are mentioned in Table 11. The accommodation of the identified limitations will possibly enhance the system in the future. This summarized study will be valuable for aspirants working in the domain of IT Infrastructure monitoring.

C. CASE STUDY

The sample normal and anomalous log records are presented in figure 8 to evidence the competence of the proposed system in case of detection of different types of failures in IT infrastructures. The Windows system logs are collected from the personal computer, and a few records are shortlisted to defend the results. Three distinct cases are considered according to the similarity or the differences in the log features. In the log records, a similar part is highlighted in bold, whereas the decision-making contents are highlighted in green. On semantic analysis of the combination of three log features (Content, EventTemplate, and ParameterList) using BERT and Classification with attention-based OLSTM, log records are labeled as either ''Normal'' or ''Anomalous.''

Case I: In case I, both log entries are from the identical event source; they carry the common ParameterList, whereas EventTemplates are different. Thus, they are classified considering the Level of Information as ''Normal'' and Warning as ''Anomalous.'' Records retention dissimilar EventTemplate is a relatively ordinary circumstance. Any state-of-art tools that utilize EventTemplate for analysis can perform such operations.

Case II: In case II, EventTemplates are approximately alike, although they are from different event sources. But, the allocated level to the log record is different. Due to common parts in the EventTemplate, analysis tools are influenced to release false alarms. To upgrade the failure detection accuracy and reduce false detection, we have added more log features along with EvenTemplate. In log records I and II under case II, decision-making particulars are highlighted in green under the content log feature.

Case III: In case III, EventTemplates are identical, although log records are from different event sources. Here, the additional log feature ParameterList assists in classifying and labeling abnormal log entries. The decision-making particulars are in the ParameterList feature and highlighted in green.

Based on the case studies discussed, there are variations in the log entries in terms of Event Source, EventTemplate, Log Message, ParameterList, etc. Thus, making use of multiple features for analysis is significant. Moreover, these selected features are analyzed based on the context and the occurrence of the words using BERT; thus, the challenges in handling homophones and homonyms are addressed. Furthermore, semantic vectors provide attention-based OLSTM to classify log entries in view of the belonging level. The OLSTM is modified LSTM algorithm that shows improved classification results in terms of accuracy, precision, recall, and F1Score, and computational time (Stated in B. Result and Analysis point under Section IV).

V. RELATED WORK

Some companies have developed a monitoring system to collect the runtime operation properties of each component in order to prevent failure in IT infrastructure. It helps to speculate on the health of the system [57]. The system logs are extraordinarily informative and are automatically generated on every computer system. The system logs are primarily intended to note system status and valuable events happening inside the system components. Network administrators can inspect the system log data to comprehend the system status, analyze system functioning, and conduct root cause analysis. Thus, system logs have become an immensely desirable resource for monitoring System operations and a pivotal part of maintaining system health. According to Wang et al. [42], system downtime is controllable and can be reduced after the identification of the reason for failure. Consequently, anomaly and failure detection, prediction, and root cause analysis have become vital research areas. Though the automated log analysis is an emerging research domain, yet administrators manually evaluate the system logs to investigate defects by tracking simple words like ''kill,'' ''exception,'' ''dead,'' ''fail,'' etc.

Furthermore, automated log analysis is intricate due to the low quality and the massive data size of logs. As a matter of fact, complex IT infrastructure components daily generate terabytes of logs and millions of metrics. Thus, electing valuable features to train and test detection or prediction models is crucial.

During figuring out the unavailability, reliability, and performance challenges confronted in IT infrastructure, it is essential to study machines as artefacts and understand what they do instead of what we expect them to do [63]. Log analysis, rule-based, method-based, and classification-based approaches have been proposed owing to numerous solutions to the automated system. Machine Learning [64], [65] and

TABLE 11. Discussion on benefits and limitations of the proposed system.

Key Point	Description
	Benefits
Use of Multiple Features	• The majority of the researchers utilized only a log template. They ignored the other parameters in the log records that contribute significantly to understanding the system's current behavior. The system's actual behavior can be tracked based on the runtime detail of the logging statement. Such runtime details get recorded under the parameters of the variable part. Therefore, the variable part (Content of the log message, parameter list, temporal information, etc.) can be used as features. • Sometimes, the log parsing process may inject inevitable noise because of the variations in the log format. Using a single feature (LogTemplate) may result in losing information such as IP address, component number, error message, etc. Thus, there was a need to add more log features in the analysis so that valuable information would not be discarded. • Also, the selection of all the features results in unnecessary computational time. Thus, we have selected the most relevant log features based on the study conducted in "Study of log features" (section II.A). • Our proposed system selected three essential features: Log Content, EventTemplate, and ParameterList, which will give details about the structure of the logging statement, list of resources, and error messages.
Efficient in analyzing unseen or new records	• System logs records on the execution of the logging statements on the occurrence of the specific event. Thus, new log records can be introduced at runtime depending on the utilization of the IT infrastructure components. • System logs are comprised of many commonly used words and similar structures. • Simple embedding techniques failed to handle homophones, homonyms, and out-of-vocabulary words. Due to these limitations, these techniques are unsuitable for unstable and newly generated log records. • The accurate analysis of unseen or new log records is possible with the help of context-based analysis using the BERT pre-trained model.
Sentence-based semantic vector generation	• In the case of system logs, common words represent different meanings, and frequently occurring words are unnecessary. Therefore, word-based vector generation does not contribute advantageously. • System logs are diverse in nature, and noise gets imported during the parsing process. However, the original meaning of the log statement remains intact. Therefore, semantic embedding is the appropriate solution to produce vectors.
Generalized solution	• The crucial challenge in the analysis of system logs is System-dependent data formats. Currently, there is no standard style or template to be followed to write logging statements. Due to this, developers write logging statements in different forms. System logs are substantially diverse in the case of various IT Infrastructures. • System logs are unstructured, unstable, and present in enormous formats. Therefore, the development of a common analysis model is tricky. • The proposed model is evaluated on five different infrastructures: Apache from a server application, OpenStack from the Distributed Systems, Windows from the Operating System, BGL from a Supercomputer, and Android from the Mobile System. Thus, this is treated as a generalized solution for different types of IT Infrastructure.
Declined training time	• Conducting semantic embedding on a vast number of log messages and training classification models on generated vectors is labour-intensive. • This difficulty was fingered out by adjusting the

TABLE 11. *(Continued.)* **Discussion on benefits and limitations of the proposed system.**

	essential hyperparameters as stated in the "Implementation" point under section IV. • Also, the proposed OLSTM algorithm is the principal element in the decrease in the training time. The OLSTM is developed with the help of making a few changes in the existing LSTM. The dropout layer replaces the current LSTM model word embedding layer. The Fully Connected Layer (FCL) is introduced after the LSTM layer proposition to mitigate the overfitting problem. Removing the word embedding layer accelerates a reduction in training time and memory requirements. Furthermore, the Z-score normalization layer is introduced as the first layer of the OLSTM model, where a feature vector of size 1x128 of each. the log is normalized to the particular standard pattern. This results in a reduction in error rates and an improvement in the accuracy.
	Limitations
Challenging to collect real-time data	• System logs hold details of each event happening in all IT Infrastructure components. It carries crucial and confidential data. Due to strict business rules, real-time logs are not readily available.
High-configuration machine required	• Due to the increase in the complexity, utilization, and execution of IT Infrastructure components, huge log records are created. Utilization of massive volume data for experimentation is challenging. • The datasets used for experimentation in this research are many log messages ranging from around Fifty-six thousand to Eleven million. The execution of the BERT feature extraction technique and classifiers on massive data size is required excellent machine configuration.
Semi-Automated System	• The proposed system is capable of identifying the anomalous log record and notifying the failure condition to the system administrator. At present, the action of mitigation is manual. Human intervention is required to handle failure conditions. Thus, failure detection and prediction are automatic, but the system admin manually manages failure.

Deep Learning [66] techniques are popularly employed for anomaly or failure identification, prediction, and root cause analysis. Applying supervised and unsupervised learning techniques to massive, unstructured system logs has attracted plenty of attention over recent years and has an outstanding research corpus of similar work.

The sentiment and semantic analysis-based approaches in the literature have been initiated to leverage the intrinsic meaning behind the logs for failure detection in complex systems [65], [67]. The authors [42] applied word2vec to perform word embedding of log contents and then found the log sequence using TF-IDF. The researchers applied unsupervised anomaly detection on extracted features, which offers a 67.25% improved F1 score over LogCluster [38]. Research has been done [14] to calculate polarity scores and identify the erroneous behaviors in the HPC system with a 96% F-score. The researchers [15] developed a system using the BERT pre-trained model as a transformer encoder to design log sequences. Deep Learning models are trained to identify the typical log sequence pattern. At the same time, the authors [34] have developed an automatic tool to identify correlations between logs. The authors employed a Drain log parser to retrieve structured logs, further word2vec embedding for feature extraction, and an LSTM plus temporal attention-based model to find a correlation in the experimentation.

The present automated system logs analysis techniques are mainly considered log templates for analysis. Moreover, in recent research, few authors examined multiple log parameters or features to understand the meaning of the log record. A combination of Log Template, Content, and parameter list was not used by them. The log template is the generalized (static part) format, and the Content and Parameter List updates at runtime (dynamic part). We can understand the current state of the infrastructure component, taking the listed three features into consideration. Thus, we proposed a system that will consider three log features for semantic analysis and an optimized LSTM model to improve the classification result and reduce execution time on unseen log records.

VI. CONCLUSION

System logs are a vital information source for mitigating IT infrastructure failure. In the existing literature, the numerous existing methods used only log templates for the analysis out of various log features. Although these tools detect abnormal log sequences effectively, they fail to analyse unstable and newly generated log records. We designed a semantic encoder on three log features to procure their numerical vectors and also trained an attention-based classifier to detect a potential failure in different IT infrastructures. We evaluated our proposed model on five IT infrastructures from different categories: Apache from a server application, OpenStack from the distributed system, Windows from the operating system, BGL from a supercomputer, and Android from the mobile system. Our experimental results indicated that semantic analysis of multiple log features applying the BERT pre-trained model delivers the literal meaning of log messages. Therefore, it produced precise results for different classifiers. Moreover, model training time is decreased in the case of OLSTM due to the introduction of new layers along with the LSTM layer.

In the future, we will gather more real-time log records of different IT infrastructures comprising a balanced dataset to assess the proposed model. In the current system, we have provided probable solutions and failure notifications. At present, the action of mitigation is manual. In the future, we will try to construct a system that can automatically select an appropriate solution and can handle anomalous conditions. This will help to reduce the human mediation in the automatic failure detection and handling of failure conditions for IT infrastructure monitoring.

GLOSSARY

- AUC - Area Under the Curve
- BERT - Bidirectional Encoder Representations from Transformers
- BGL - Blue Gene/L
- Bi-LSTM - Bidirectional Long Short-Term Memory
- CNN - Convolutional Neural Network

- DL - Deep Learning
- FCL - Fully Connected Layer
- FN - False Negative
- FP - False Positive
- FPR - False Alarm Rate
- GB - gigabyte
- GBDT - Gradient Boosting Decision Tree
- Glove - Global Vectors for Word Representation
- GPU - Graphics Processing Unit
- HDFC - Hadoop Distributed File System
- HPC - High-Performance Cluster
- IT - Information Technology
- KNN - k-Nearest Neighbor
- LSTM - Long Short-Term Memory
- ML - Machine Learning
- NLP - Natural Language Processing
- NN - Neural network
- OLSTM - Optimized Long Short-Term Memory Networks
- RNN - Recurrent Neural Network
- ROC - Receiver Operating Characteristic
- SVM - Support Vector Machine
- TF-IDF - Term Frequency-Inverse Document Frequency
- TN - True Negative
- TP - True Positive
- TPR - True Positive Rate
- UI - User Interface

REFERENCES

[1] *The Cost of Downtime—Andrew Lerner*. Accessed: Apr. 30, 2020. [Online]. Available: https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/

[2] D. A. Bhanage and A. V. Pawar, "Bibliometric survey of IT infrastructure management to avoid failure conditions," *Inf. Discovery Del.*, vol. 49, no. 1, pp. 45–56, Feb. 2021.

[3] Y. Zhang, K. Rodrigues, Y. Luo, M. Stumm, and D. Yuan, "The inflection point hypothesis: A principled debugging approach for locating the root cause of a failure," in *Proc. 27th ACM Symp. Oper. Syst. Princ.*, Oct. 2019, pp. 131–146.

[4] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, Z. Zang, X. Jing, and M. Feng, "FUNNEL: Assessing software changes in web-based services," *IEEE Trans. Services Comput.*, vol. 11, no. 1, pp. 34–48, Jan. 2018.

[5] S. Khatuya, N. Ganguly, J. Basak, M. Bharde, and B. Mitra, "ADELE: Anomaly detection from event log empiricism," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 2114–2122.

[6] W. Meng, Y. Liu, S. Zhang, D. Pei, H. Dong, L. Song, and X. Luo, "Device-agnostic log anomaly classification with partial labels," in *Proc. IEEE/ACM 26th Int. Symp. Quality Service (IWQoS)*, Jun. 2018, pp. 1–6.

[7] S. Zhang, "PreFix: Switch failure prediction in datacenter networks," in *Proc. ACM Meas. Anal. Comput. Syst.*, USA, Jun. 2018, doi: 10.1145/3219617.3219643.

[8] S. Satpathi, S. Deb, R. Srikant, and H. Yan, "Learning latent events from network message logs," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1728–1741, Aug. 2019.

[9] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2020, pp. 1196–1201.

[10] D. A. Bhanage, A. V. Pawar, and K. Kotecha, "IT infrastructure anomaly detection and failure handling: A systematic literature review focusing on datasets, log preprocessing, machine & deep learning approaches and automated tool," *IEEE Access*, vol. 9, pp. 156392–156421, 2021.

[11] D. A. Bhanage, "DigitalCommons @ University of Nebraska—Lincoln review and analysis of failure detection and prevention techniques in IT infrastructure monitoring," *Library Philosophy Pract.*, Apr. 2021.

[12] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "Paddy: An event log parsing approach using dynamic dictionary," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2020, pp. 1–8.

[13] J. Wang, C. Zhao, S. He, Y. Gu, O. Alfarraj, and A. Abugabah, "LogUAD: Log unsupervised anomaly detection based on Word2Vec," *Comput. Syst. Sci. Eng.*, vol. 41, no. 3, pp. 1207–1222, 2022.

[14] K. A. Alharthi, A. Jhumka, S. Di, F. Cappello, and E. Chuah, "Sentiment analysis based error detection for large-scale systems," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, no. 1, Jun. 2021, pp. 237–249.

[15] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8.

[16] Y. Xie, K. Yang, and P. Luo, "LogM: Log analysis for multiple components of Hadoop platform," *IEEE Access*, vol. 9, pp. 73522–73532, 2021.

[17] H. Yang, X. Zhao, D. Sun, Y. Wang, and W. Huang, "Sprelog: Log-based anomaly detection with self-matching networks and pre-trained models," *Service-Oriented Computing*, vol. 2. Cham, Switzerland: Springer, 2021.

[18] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Aug. 2019, pp. 807–817.

[19] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 37–44.

[20] J. Lou, "Mining invariants from console logs for system problem detection," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, Jun. 2010.

[21] E. Chuah, A. Jhumka, S. Alt, T. Damoulas, N. Gurumdimma, M.-C. Sawley, W. L. Barth, T. Minyard, and J. C. Browne, "Enabling dependability-driven resource use and message log-analysis for cluster system diagnosis," in *Proc. IEEE 24th Int. Conf. High Perform. Comput. (HiPC)*, Dec. 2017, pp. 317–327.

[22] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 4739–4745.

[23] W. Meng, Y. Liu, Y. Huang, S. Zhang, F. Zaiter, B. Chen, and D. Pei, "A semantic-aware representation framework for online log analysis," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2020, pp. 1–7.

[24] D. A. Bhanage and A. V. Pawar, "Robust analysis of IT infrastructure's log data with BERT language model," *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 6, pp. 705–714, 2023.

[25] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proc. 33rd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2018, pp. 178–189.

[26] J. Zhou, Y. Qian, Q. Zou, P. Liu, and J. Xiang, "DeepSyslog: Deep anomaly detection on syslog using sentence embedding and metadata," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 3051–3061, 2022.

[27] *BERT Explained: A Complete Guide with Theory and Tutorial—Towards Machine Learning*. Accessed: Jul. 21, 2021. [Online]. Available: https://towardsml.com/2019/09/17/bert-explained-a-complete-guide-with-theory-and-tutorial/

[28] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? An empirical study on logging practices in industry," in *Proc. 36th Int. Conf. Softw. Eng.*, May 2014, pp. 24–33.

[29] S. He, J. Zhu, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," Aug. 2020, *arXiv:2008.06448*.

[30] *Log4j—Apache Log4jTM 2*. Accessed: Mar. 24, 2023. [Online]. Available: https://logging.apache.org/log4j/2.x/

[31] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "SwissLog: Robust and unified deep learning based log anomaly detection for diverse faults," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2020, pp. 92–103.

[32] R. Chen, S. Zhang, D. Li, Y. Zhang, F. Guo, W. Meng, D. Pei, Y. Zhang, X. Chen, and Y. Liu, "LogTransfer: Cross-system log anomaly detection for software systems with transfer learning," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2020, pp. 37–47.

[33] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "HitAnomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 4, pp. 2064–2076, Dec. 2020.

[34] M. Platini, T. Ropars, B. Pelletier, and N. De Palma, "LogFlow: Simplified log analysis for large scale systems," in *Proc. 22nd Int. Conf. Distrib. Comput. Netw.*, Jan. 2021, pp. 116–125.

[35] S. Chen and H. Liao, "BERT-log: Anomaly detection for system logs based on pre-trained language model," *Appl. Artif. Intell.*, vol. 36, no. 1, pp. e2145642-1–e2145642-23, Dec. 2022.

[36] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *Proc. IEEE 16th Int. Conf. Data Mining (ICDM)*, Dec. 2016, pp. 859–864.

[37] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 11, pp. 1921–1936, Nov. 2012.

[38] R. Vaarandi and M. Pihelgas, "LogCluster—A data clustering and pattern mining algorithm for event logs," in *Proc. 11th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2015, pp. 1–7.

[39] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *Proc. Int. Conf. Softw. Eng.*, 2018, pp. 167–177.

[40] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 33–40.

[41] P. He, J. Zhu, P. Xu, Z. Zheng, and M. R. Lyu, "A directed acyclic graph approach to online log parsing," 2018, pp. 1–14, *arXiv:1806.04356*.

[42] J. Wang, C. Zhao, S. He, Y. Gu, O. Alfarraj, and A. Abugabah, "LogUAD: Log unsupervised anomaly detection based on Word2Vec," *Comput. Syst. Sci. Eng.*, vol. 41, no. 3, pp. 1207–1222, 2022.

[43] D. A. Bhanage and A. V. Pawar, "Improving classification-based log analysis using vectorization techniques," in *Proc. 3rd Int. Conf. Adv. Comput. Eng. Commun. Syst.*, 2023, pp. 271–282.

[44] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering—A systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, 2009.

[45] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.

[46] *Welcome!—The Apache HTTP Server Project*. Accessed: Mar. 13, 2023. [Online]. Available: https://httpd.apache.org/

[47] *Public Security Log Sharing*. Accessed: Mar. 13, 2023. [Online]. Available: https://log-sharing.dreamhosters.com/

[48] *Open Source Cloud Computing Infrastructure—OpenStack*. Accessed: Mar. 13, 2023. [Online]. Available: https://www.openstack.org/

[49] *CloudLab*. Accessed: Mar. 13, 2023. [Online]. Available: https://cloudlab.us/

[50] Y. Liang and Y. Zhang, "Filtering failure logs for a BlueGene/L prototype," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN)*, 2005.

[51] *Android | The Platform Pushing What's Possible*. Accessed: Mar. 13, 2023. [Online]. Available: https://www.android.com/

[52] C.-Y.-J. Peng, K. L. Lee, and G. M. Ingersoll, "An introduction to logistic regression analysis and reporting," *J. Educ. Res.*, vol. 96, no. 1, pp. 3–14, Sep. 2002.

[53] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," *Neurocomputing*, vol. 408, pp. 189–215, Sep. 2020.

[54] H. Chen, S. Hu, R. Hua, and X. Zhao, "Improved naive Bayes classification algorithm for traffic risk management," *EURASIP J. Adv. Signal Process.*, vol. 2021, no. 1, pp. 1–12, Dec. 2021.

[55] H. Seto, A. Oyama, S. Kitora, H. Toki, R. Yamamoto, J. Kotoku, A. Haga, M. Shinzawa, M. Yamakawa, S. Fukui, and T. Moriyama, "Gradient boosting decision tree becomes more reliable than logistic regression in predicting probability for diabetes with big data," *Sci. Rep.*, vol. 12, no. 1, pp. 1–10, Oct. 2022.

[56] L. E. O. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.

[57] X. Wang, F. Liu, Y. Feng, and J. Zhao, "A two-layer architecture for failure prediction based on high-dimension monitoring sequences," *Complexity*, vol. 2021, pp. 1–9, Mar. 2021.

[58] *Benchmark—Bert-As-Service 1.6.1 Documentation*. Accessed: Jun. 22, 2022. [Online]. Available: https://bert-as-service.readthedocs.io/en/latest/section/benchmark.html#speed-wrt-max-batch-size

[59] *GitHub—Logpai/Loglizer: A Machine Learning Toolkit for Log-Based Anomaly Detection*. Accessed: Sep. 2, 2023. [Online]. Available: https://github.com/logpai/loglizer

[60] *GitHub—Nailo2c/Deeplog: PyTorch Implements DeepLog: Anomaly Detection and Diagnosis From System Logs Through Deep Learning*. Accessed: Sep. 2, 2023. [Online]. Available: https://github.com/nailo2c/deeplog

[61] *GitHub—Donglee-Afar/Logdeep: Log Anomaly Detection Toolkit Including DeepLog*. Accessed: Sep. 2, 2023. [Online]. Available: https://github.com/donglee-afar/logdeep

[62] *GitHub—HelenGuohx/Logbert: Log Anomaly Detection Via BERT*. Accessed: Sep. 2, 2023. [Online]. Available: https://github.com/HelenGuohx/logbert

[63] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2007, pp. 575–584.

[64] Y. Li, Z. M. Jiang, H. Li, A. E. Hassan, C. He, R. Huang, Z. Zeng, M. Wang, and P. Chen, "Predicting node failures in an ultra-large-scale cloud computing platform," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 2, pp. 1–24, Apr. 2020.

[65] J. Wang, Y. Tang, S. He, C. Zhao, P. K. Sharma, O. Alfarraj, and A. Tolba, "LogEvent2vec: LogEvent-to-vector based anomaly detection for large-scale logs in Internet of Things," *Sensors*, vol. 20, no. 9, pp. 1–19, 2020.

[66] R. Ren, J. Cheng, Y. Yin, J. Zhan, L. Wang, J. Li, and C. Luo, "Deep convolutional neural networks for log event classification on distributed cluster systems," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 1639–1646.

[67] H. Ott, J. Bogatinovski, A. Acker, S. Nedelkoski, and O. Kao, "Robust and transferable anomaly detection in log data using pre-trained language models," in *Proc. IEEE/ACM Int. Workshop Cloud Intell. (CloudIntelligence)*, May 2021, pp. 1–6.

DEEPALI ARUN BHANAGE received the master's degree in computer engineering from the Sinhgad Institute of Technology, University of Pune. She is currently pursuing the Ph.D. degree with the Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune. She is currently an Assistant Professor with the Pimpri Chinchwad College of Engineering, PCET, Pune. Her current research interests include IT infrastructure monitoring, machine learning, deep learning, and natural language processing.

AMBIKA VISHAL PAWAR received the Ph.D. degree from Symbiosis International (Deemed University), Pune, India. She is currently a senior manager of learning and development. She is also heading the Higher Education and University Tie-Up, Persistent University, Persistent Systems, Pune. She is associated with Symbiosis International (Deemed) University as a Ph.D. Supervisor. She has more than 20 years of experience as an academician and more than 11 years as a researcher. She has published 50 research paper publications in international journals/conferences and one book published by Taylor and Francis and CRC Press. According to Google Scholar, her articles have 140 citations, with an H-index of Six and an I10-index of four. Her current research interests include security and privacy solutions using blockchain and AIMLDL Technologies.

KETAN KOTECHA was an Administrator with Parul University and Nirma University and has several achievements in these roles to his credit. He currently heads the Symbiosis Centre for Applied Artificial Intelligence (SCAAI). He is considered a foremost expert in AI and aligned technologies. He has vast and varied experience in administrative roles. He has expertise and experience in cutting-edge research and AI and deep learning projects for more than the last 25 years. He has published widely in several excellent peer-reviewed journals on various topics ranging from education policies and teaching-learning practices and AI for all. He also has pioneered education technology. He is a Team Member for the nationwide initiative on AI and deep learning skilling and research named Leadingindia.ai initiative sponsored by the Royal Academy of Engineering, U.K., under the Newton Bhabha Fund.

AJITH ABRAHAM (Senior Member, IEEE) received the B.Tech. degree in electrical and electronic engineering from the University of Calicut in1990, the M.S. degree from Nanyang Technological University, Singapore, in 1998, and the Ph.D. degree in computer science from Monash University, Melbourne, Australia, in 2001. He is currently the Pro-Vice Chancellor of Bennett University, India. Prior to this, he was the Dean of the Faculty of Computing and Mathematical Sciences, FLAME University, Pune, and the Founding Director of the Machine Intelligence Research Laboratories (MIR Labs), USA, a Not-for-Profit Scientific Network for Innovation and Research Excellence, connecting industry and academia. During the last three years, he also held two University Professorial appointments, including a Professor of Artificial Intelligence with Innopolis University, Russia, and the Yayasan Tun Ismail Mohamed Ali Professorial Chair of Artificial Intelligence, UCSI, Malaysia. He works in a multi-disciplinary environment and has authored/coauthored more than 1,400+ research publications. He has more than 54,000 academic citations (H-index is more than 110 as per Google Scholar). He has given more than 200 plenary lectures and conference tutorials (in more than 20 countries). He was the Chair of the IEEE Systems, Man and Cybernetics Society Technical Committee on Soft Computing (which has over more than 200 members), from 2008 to 2021. He served as a Distinguished Lecturer for the IEEE Computer Society Representing Europe, from 2011 to 2013. He was the Editor-in-Chief of *Engineering Applications of Artificial Intelligence* (EAAI), from 2016 to 2021. He serves/served on the editorial board for over 15 international journals indexed by Thomson ISI.