Robust Analysis of IT Infrastructure's Log Data with BERT Language Model

Deepali Arun Bhanage[1], Ambika Vishal Pawar[2]
Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune 412115, India[1, 2]
Dept. of Computer Engineering-Pimpri Chinchwad Education Trust's, Pimpri Chinchwad College of Engineering, Pune, India[1]

***Abstract*—Now-a-days, failure detection and prediction have become a significant research focus on enhancing the reliability and availability of IT infrastructure components. Log analysis is an emerging domain aimed at diminishing downtime caused by IT infrastructure components' failure. However, it can be challenging due to poor log quality and large data sizes. The proposed system automatically classifies logs based on log level and semantic analysis, allowing for a precise understanding of the meaning of log entries. Using the BERT pre-trained model, semantic vectors are generated for various IT infrastructures, such as Server Applications, Cloud Systems, Operating Systems, Supercomputers, and Mobile Systems. These vectors are then used to train machine learning (ML) classifiers for log categorization. The trained models are competent in classifying logs by comprehending the context of different types of logs. Additionally, semantic analysis outperforms sentiment analysis when dealing with unobserved log records. The proposed system significantly reduces engineers' day-to-day error-handling work by automating the log analysis process.**

Keywords*—System log; log analysis; BERT; classification; failure prediction; failure detection

I. Introduction

IT infrastructures, consisting of complex and interconnected systems, are vulnerable to various failures, such as hardware failures, software glitches, network outages, security breaches, and other unforeseen events that can disrupt critical business operations. With rapid development in size and functionality, IT infrastructures have become increasingly complex and agile. Enriched accessibility to IT infrastructure is vital as the usage of computer systems has penetrated all aspects of society. Moreover, a small failure in any of the infrastructure components gives rise to catastrophic failures accompanied by downtime [1]. Research [2] shows that these failures can lead to financial losses, reputational damage, and customer dissatisfaction. Thus, developing a system that can perform accurate and timely failure detection is paramount. Such a system will be helpful for organizations to proactively detect and resolve potential problems, minimize downtime, and improve the overall reliability and efficiency of IT operations.

System logs are one of the most worthwhile records that register important events, various services, and the state of operations. By analyzing system logs, IT teams can monitor for signs of anomalies or irregularities that may indicate potential failures. Accordingly, system logs have been widely used to understand the behavior of computer systems and monitor their health. Each computer system generates system logs on the execution of the event; thus, an ample amount of records are available. Even so, log analysis is troublesome due to the size of the data. As stated in a systematic literature review [3], many researchers have used logs in log analysis, anomaly and failure detection, troubleshooting, and prediction research.

The failure detection using log data framework comprises six steps, such as i) Log collection: Logs are obtainable in raw and unstructured formats. Different systems generate various types of logs; therefore, different types of logs ought to be collected for investigation. ii) Log parsing: In this step, unstructured logs are refined to be converted into a structured format. The primary objective of log parsing is to excerpt log templates from raw system logs. Log parsing substitutes the variable part of the log with special characters and preserves only the constant part. iii) Structured logs: Results acquired from the parsing are stored in the .csv file format; this data is used for further processing. iv) Feature extraction: Log templates and the contents produced in the course of log parsing are preferred as features for encoding. v) Vector representation: Log templates and contents are converted into vector representation in order to furnish them as input to machine learning models. vi) Anomaly / Failure Detection: Eventually, excavated vectors are served to the machine learning or deep learning models to classify logs in accordance with the allocated log level. Logs are classified into different categories, which include "fail," "Fatal," "error," etc. levels. These categories demand attention as they indicate the abnormal behavior of the system. The stated log levels are allocated to the logging statements on executing any exception in the system. Thus, the administrator gets anomalous data to emphasize and can take remedial action accordingly.

As per the literature, machine learning [4] and deep learning [5] have popular techniques 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 combination of text, numbers, and special symbols. The data is available in natural language format and cannot be directly used to build ML (Machine Learning) and DL (Deep Learning) models. Many researchers utilized various NLP techniques for embedding purposes in the existing literature. But considering the nature of the log data and challenges in handling system logs such as voluminous data, commonly used words, the occurrence of the same word with different meanings, etc., direct vector conversion is not significant. Thus, vectors are required to

generate based on the word's context. Therefore, it is necessary to follow the process of text data conversion to numerical vectors based on semantics.

Proposed feature extraction with semantic analysis conquers the challenges related to variation in log format and imbalanced data. The proposed systems comprehend the semantic analysis of log templates by practicing the BERT pre-trained model. The system employs the BERT to procreate sentence vectors, bearing in mind the log templates and contents acquired against log parsing. At last, machine learning techniques are employed to classify log entries contingent on earmarked levels.

The Contributions in this paper are summarized as follows.

1) The proposed system for classifying logs is based on analyzing the meaning of logs with the BERT model that has already been trained.

2) Different Infrastructures such as Apache, OpenStack, Windows, BGL, and Android logs are collected and parsed using the "Drain" parser to derive log templates.

3) Sentence embedding is done on the derived log templates to determine each entry's meaning.

4) Extracted features are provided to machine learning classifiers to analyze logs pertaining to levels. The main goal of the classification is to test the efficiency of the semantic analysis done by different NLP techniques.

The proposed system will significantly diminish manual errors by enabling automated and accurate solutions to failure prediction.

The paper has eight sections, including details: Section II discusses related work. Section III has the descriptive analysis of the datasets, including log data collection and preprocessing. Section IV includes NLP-based feature extraction techniques. Section V investigates the models and technical definitions of the methodology used to perform experimentations. Sections VI and VII emphasize the experimental setup, followed by derived results. Finally, the conclusion and future directions are stated in Section VIII.

II. Related Work

Failure prediction is a crucial aspect of IT infrastructure monitoring as it enables organizations to proactively detect and mitigate potential issues before they result in costly downtime or performance degradation. The system can identify patterns or anomalies that may indicate impending failures and take preventive measures to avoid or minimize the impact of such failures. System logs, which are records of events and activities generated by various components of an IT system, can be invaluable in failure detection and prediction in IT infrastructures. System logs capture essential real-time information about IT resources' behavior, performance, and status, such as servers, networks, applications, and databases. One of the primary uses of system logs in failure detection is to provide visibility into the operational state of IT systems. By monitoring system logs, IT teams can detect such anomalies early and take preventive actions to mitigate potential failures. System logs can also be used in failure prediction by leveraging machine learning and statistical techniques.

Wang et al. [6] propose that system downtime can be reduced by identifying the reason for failure, making anomaly and failure detection, prediction, and root cause analysis. Despite being an emerging domain, automated log analysis is complicated due to the manual evaluation of system logs by administrators, who track simple words like "kill," "exception," "dead," "fail," etc., to investigate defects [3]. In order to address the challenges of unavailability, reliability, and performance in IT infrastructure, it is vital to study machines as they are, understanding what they do instead of what is expected [7]. Various rule-based and classification-based approaches [8][9], including machine learning [10][11] and deep learning [12][13] techniques, have been proposed for automated system log analysis. Moreover, supervised [14] and unsupervised [15] learning techniques applied to massive, unstructured system logs have gained significant attention in recent years, with a substantial research corpus of similar work.

Recently, NLP-based analysis has been introduced to understand the meaning of logs for log analysis in complex IT infrastructures [16]. Word2Vec has been applied by authors [17] to perform word embedding of log contents, followed by finding log sequences using TF-IDF. Unsupervised learning has been utilized for the extracted features, resulting in a 67.25% improved F1 score compared to LogCluster [18]. Researchers [19] have calculated polarity scores to identify abnormal behaviors in HPC systems with a 96% F-score. In the recent past, many researchers have been concentrating on the use of BERT [20] re-BERT [21] pre-trained model as an embedding technique and LSTM [22][23], Bi-LSTM [24][25] attention base mechanism for classification purpose.

III. Illustrative Analysis of the Dataset

A. Dataset Collection

System logs are intended to be the primary source of information about the system; thus, the availability of a log dataset for research is a demanding obligation. Log data records every operational detail of each component of the IT infrastructure at run-time. The mishandling of such sensitive data may cause several issues. Therefore, system logs are not easily obtainable for research and experimentation. He, Zhu, He, & Lyu, in 2020 [26], collected sample logs and made them available on "loghub" [27] for study. An extra set of logs are produced in the labs and released for research determination. In the systematic literature review [3], we discussed details about availability of more datasets that are accessible for research purpose.

B. Dataset Preprocessing with Log Parser

Systems logs are the "print" statements scripted by engineers under software development and documented in the course of the carrying out of affiliated operations. The logs are composed of a constant log header (id, state, timestamp, level, etc.) and a dynamic part (updates on operation execution). The primary purpose of log parsing is to transform unstructured logs toward structured data by extracting the constant part from logs called log templates. The sample log parsing

technique is shown in Fig. 1. Log message or content is "Component State Change: Component \042SCSI-WWWID:01000010:6005-08b4-0001-00c6-0006-3000-003d-0000\042 is in the unavailable state (HWID=1973)" from which log template is extracted as "Component State Change: Component <*> is in the unavailable state (HWID=<*>)."

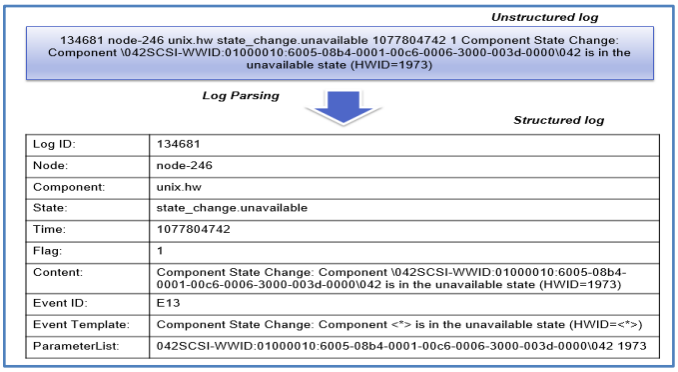

Fig. 1. Components of example HPC log.

Various researchers have discussed miscellaneous log parsers such as POP [28], Spell [29], SLCT [30], etc. Drain [31] parser has been adapted in this research work to parse log datasets. Drain elected for experimentation by examining execution time, availability, accuracy, and flexibility parameters. The drain parser employed the fixed-depth tree structure to perform and retrieve log templates. Table I presents the summary of findings on the performance of Drain on various types of logs. For the parsing, datasets are selected from multiple infrastructures, such as Apache as a server application, OpenStack as a cloud system, Windows as an operating system, BGL as a supercomputer, and Android as a mobile system. Table I contains the column number of log messages utilized for the parsing, the derived unique number of templates, and the maximum template length. Thus "Drain's" is a better parser for this research due to its parsing accuracy.

C. Feature Extraction

System logs are a combination of text, numbers, and special symbols. Natural language data cannot be used directly to build ML (Machine Learning) and DL (Deep Learning) models. For this reason, it is imperative to follow the action of text data conversion to numerical vectors, known as vectorization or word embedding. The mined vectors can be employed to train different Machine learning and Deep learning models for classification, detection, and prediction purposes; in this way, word embedding is imitated for feature extraction.

At present, different Natural Language Processing (NLP) models are available for feature extraction in view of sentiment and semantic analysis. TF-IDF, polarity score, word2vec, and doc2vec work based on word frequency or position occurrence in the given text and analyze word-related sentiments. Whereas BERT, GPT2, and XL [16] function contingent on the semantics of words regarding the position and meaning of words accompanying them. The BERT model is pre-trained on massive datasets like Wikipedia and proposed by Google to be fine-tuned on a particular dataset. Pre-trained word embedding models are applied for vector representation of log templates and to strengthen the prediction of unobserved log entries. Moreover, BERT supports domain-specific semantic information and can address out-of-vocabulary (OOV) words in novel kinds of logs during run-time [32].

This experimentation focuses on doc2vec and BERT sentence embedding techniques to get vectors of log templates. Whereas TF-IDF is unsuitable in log datasets as the TF-IDF work on the weighting methods, and weights are assigned considering the frequency of occurrence of words. In the case of system logs, common words represent the different meanings of the log messages, and frequently occurring words are unnecessary. Thus, TF-IDF is unsuitable, even if it archives good classification accuracy. Fig. 2 renders the process of feature extraction. First, the unstructured log is processed toward a structured format; then, log templates are excavated with the Drain parser. Then the log template is preprocessed to expel special symbols and stop words; further steaming is performed. This cleaned data will be available for tokenization, followed by vectorization.

IV. Feature Extraction Techniques

A. Doc2Vec

Doc2vec is a Natural Language Processing (NLP) technique for converting documents into vectors. Doc2vec's work is based on the conception of Word2vec. The direct encouragement for the development of doc2vec is to induce a vector illustration of a group of words collected together to be presented as a single unit, irrespective of the length of the document. The pivotal variance in the word and sentence representation is that words carry logical structure, but documents don't. Mikilov and Le [23] introduced an additional vector, Paragraph ID, along with the word2vec model to solve this issue. Thus, at the time of word vectors training, the document vector also gets trained, and eventually, the document is converted to numerical form. This model is the Distributed Memory version of the Paragraph Vector (PV-DM).

TABLE I. Experimental Results of Drain Parser on Various Datasets

Dataset	Source Type	Size of Data	Number of log Messages	Number of Unique Template	Template Max Length	Parsing Accuracy
Apache	Server Application	4.90 MB	56,481	44	42	1
OpenStack	OpenStack infrastructure log	5.4 MB	207,820	7,221	104	0.73
Windows	Windows event log	267.465 MB	611,103	176	173	0.99
BGL	Supercomputer	708.76 MB	4,747,963	619	376	0.99
Android	Android framework log	25.7 MB	1,555,005	14,899	124	0.91

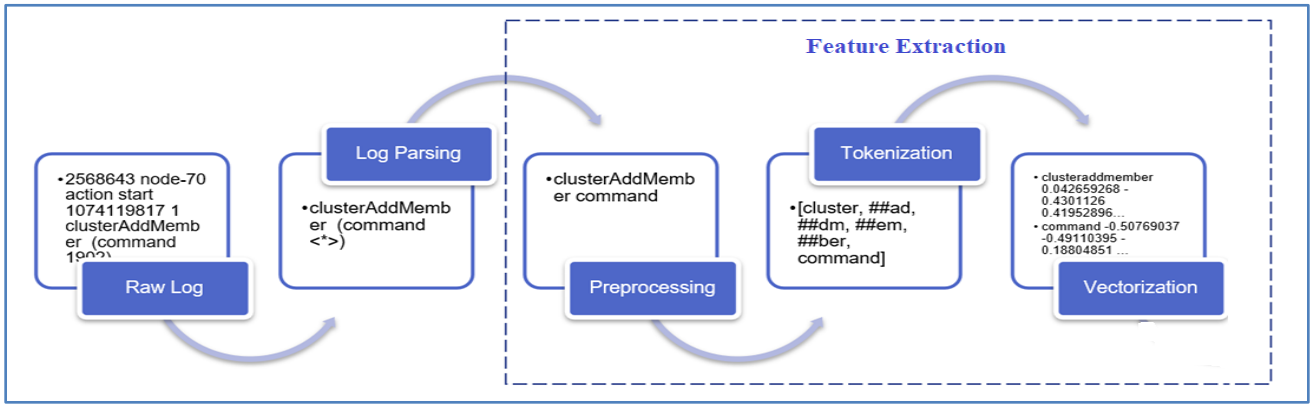

Fig. 2. Process of feature extraction.

To utilize the doc2vec model, the following steps need to be pursued. First, furnish the dataset, which is to be turned into vectors. A word vector is created for an individual word, and combining related word vectors creates a document vector for each document [33]. The softmax hidden layer is used to train the weights, and then all weights are set to find the sentence vector.

B. *Bidirectional Encoder Representations from Transformers (BERT)*

BERT models are the most acceptable preference for obtaining superior-quality language features from the offered text dataset. Furthermore, the model can be fine-tuned for specific tasks such as semantic analysis or question answering, relying on input datasets. BERT is a pre-trained language model that is directionally trained [34]. Devlin et al. demonstrated that a directionally trained language model could possess a more profound sense of language context and flow than single-direction language models [35].

The BERT pre-trained model outperforms word2vec as this approach assigns dynamic numerical vectors to each token, considering the context within which the word appears whereas in word2vec, each word has a fixed numerical vector allocated. Experimentation uses BERT to pull out features by generating word and sentence embedding vectors from log templates and contents. This research focuses on the feature extraction part of the BERT; thus, the remaining part is not considered for an explanation. Here, log templates and contents are elicited from the Drain parser and presented to the BERT model to extract the features. BERT is a pre-trained model taking input data in a specific format. BERT mainly adds an [SEP] as the split between consecutive sentences and an [CLS] at the start of the sentence. The BERT model offers intrinsic tokenizing. The supplied input is spitted into multiple tokens considering the corpus records. Following that, the embedding layer creates an embedding vector for each token, which includes [CLS] and [SEP]. Log template data desires to be converted into torch tensors and called the BERT model to evoke embedding. The BERT PyTorch interface demands that the data be in torch tensors rather than Python lists. The bert-base-uncased model contains 13 layers (1 for input embedding and +12 for output embedding) of the transformer encoder and 768-hidden units of all transformers.

Every token has 13 independent vectors, each of length 768 but necessary to get separate vectors for every token or single vector presentation of the entire sentence. Individual vectors are calculated by adding the last four layers together. Furthermore, a 768-length vector is calculated for each sentence by taking the average of the second to the final hidden layer.

Table II presents a comparative analysis of Doc2Vec and BERT embedding techniques. This summarized view is bestowed in reference to the critical points observed during the study of Doc2Vec and BERT techniques. These techniques are compared using a type of embedding suitable for which kind of data and the pros-cons of the method. Doc2Vec works on static sentence embedding, whereas BERT considers the context of the words for embedding.

Thus, Doc2Vec is the appropriate choice in a problem where semantic relations between the words are essential. To extract contextual ties between words, BERT works very efficiently.

TABLE II. COMPARATIVE ANALYSIS OF WORD EMBEDDING TECHNIQUES / MODELS

Technique/Model	Embedding Type	Suitable for	Pros	Cons
Doc2Vec	Static Sentence Embedding	Semantic Relation Between Word	Generate a vector representation of a group of words collected to present as a single unit.	The co-occurrence matrix of sentences occupies plenty of memory for storage.
BERT	Contextualized Word Embedding	Contextual Relation Between Word	Capable of gaining context-sensitive bi-directional feature representation.	Fine-tuning and pre-training are inconsistent. Long training time due to the immense size of model files

V. CLASSIFICATION MODEL

A. Model Definition

Machine learning classifiers are essential tools for a wide range of applications. They have the potential to revolutionize many industries by automating tasks, improving accuracy, and providing new insights into complex systems. These classifiers work by learning patterns and relationships within a given dataset and then using that knowledge to classify new data into pre-defined categories or classes. In recent research, the authors explored using multiple classifiers to group logs based on log level. The working of machine learning classifiers can vary depending on the algorithm used. This study experimented on five different infrastructure logs using k-Nearest Neighbors, Linear Regression, Support Vector Machines, Naïve Bayes, Gradient Boosting Decision Trees, and Random Forest machine learning classifiers.

1) K-Nearest Neighbors (KNN): IT is a machine learning classifier that can be used for regression and classification tasks. A non-parametric algorithm finds the K closest training examples (i.e., neighbors) to a new data point and uses their class labels to make a prediction [36].

2) Linear regression: Linear regression is a type of regression analysis that models the relationship between a dependent variable and one or more independent variables [37]. It is commonly used for predicting continuous values, such as sales revenue or stock prices.

3) Support Vector Machines (SVMs): SVMs are supervised learning algorithms that can be used for classification or regression tasks [38]. SVMs try to find the optimal hyperplane that separates the different classes in the dataset.

4) Naïve bayes: Naïve Bayes is a probabilistic algorithm that can be used for classification tasks [39]. It is based on Bayes' theorem and assumes that the features in the data are independent of each other.

5) Gradient boosting decision trees: Gradient boosting is an ensemble learning technique that combines multiple decision trees to improve prediction accuracy. It involves training a series of decision trees in sequence, with each subsequent tree trying to correct the errors of the previous one [40].

6) Random forests: Random forests are also an ensemble learning technique that uses multiple decision trees to improve prediction accuracy [41]. However, unlike gradient boosting, random forests train each decision tree independently and then aggregate their predictions to make the final prediction.

B. Evaluation Metrics

Classification of logs is based on the level earmarked for the log entry. In the different IT infrastructures, log entries hold numerous types of levels. Thus, a multi-class classification technique is favored to accomplish the classification. Generally, a multi-class classifier's performance is appraised by the Micro-F1 score and Macro-F1 Score [42]. Therefore, TP (True Positives), TN (True Negative), FP (False Positives), and FN (False Negatives) values were collected from each category of level and further utilized to calculate micro precision, macro precision, micro recall, macro recall, and macro-F1.

For a provided log category i, outcomes are labeled as TP_i, TN_i, FP_i, and FN_i. Where TP_i represents the number of true positives in logs belonging to the i category. TN_i represents the true negative in logs belonging to the i category. FP_i represents false positives, and FN_i means false negatives in logs belonging to the i category.

Considering values of TP_i, TN_i, FP_i, and FN_i, $precision_i$, and $recall_i$ are evaluated as:

$Precision_i$ can be calculated as the percentage of positively labeled predictions made out of all predictions under the i category of the level [43].

$$Precision_i = \frac{TP_i}{TP_i+FP_i} \quad (1)$$

The $Recall_i$ can be calculated as the number of correct predicted results divided by applicable instances. Recall provides the number of accurately predicted results divided by all relevant samples [43].

$$Recall_i = \frac{TP_i}{TP_i+FN_i} \quad (2)$$

Macro-F1: Employed to compute the F1- score in the instance of multi-class settings. Macro-F1 is known as the macro-averaged F1 score and is calculated as simple arithmetic means of the F1 scores of each class [44].

$$Macro\ Average\ Precision = \frac{\sum_{k=1}^{k} Precision\kappa}{k} \quad (3)$$

$$Macro\ Average\ Recall = \frac{\sum_{k=1}^{k} Recall\kappa}{k} \quad (4)$$

$$Macro\ F1\ Score = 2 * \left(\frac{MacroAveragePrecision * MacroAverageRecall}{MacroAveragePrecision^{-1} - MacroAverageRecall^{-1}}\right) \quad (5)$$

Specificity is calculated on the negatives that are detected accurately. Specificity is also known as True Negative Rate (TNR), which denotes the classifier's ability to enter negative entries in the actual class [45]. In the case of logs, the system administrator can select a log level with correct specificity to proctor the anomalies or failures.

$$Specificity_i = \frac{TN_i}{TN_i+FP_i} \quad (6)$$

The metrics used to measure model performance are training and testing splits accuracy. Accuracy is the rate of the absolutely classified data to all the data [46].

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (7)$$

VI. IMPLEMENTATION DETAILS

All models are implemented in Python and executed on the Symbiosis Institute of Technology (Pune, India) server. Various datasets such as Apache, OpenStack, Windows, BGL, and Android were utilized to conduct the experimentation. These datasets carry a vast number of log messages ranging

from around fifty-six thousand to 11 million (refer to Table III for dataset details). Because of the excellent server configuration made executing the BERT feature extraction technique and ML classifiers on a massive data size possible.

TABLE III. DATASET DESCRIPTION

Dataset	Description	Source Type	Period	Number of logs
Apache	Apache webserver error log	Server Application	263.9 days	56,481
OpenStack	OpenStack infrastructure log	Cloud System	NA.	207,820
Windows	Windows event log	Operating System	NA.	611,103
BGL	Blue Gene/L supercomputer log	Supercomputer	214.7 days	4,747,963
Android	Android framework log	Mobile System	NA.	1,555,005

The "Drain" parser is employed to parse the log messages into log templates. An environment was created to run the Drain parser by installing dependencies such as Python 2.7, Scipy, NumPy, sci-kit-learn, and pandas. The unstructured logs were converted into the structured format and preserved in the .csv file. The results of parsing using Drain are presented in Table I.

For feature extraction, we construct Doc2Vec and BERT sentence embedding models. Doc2Vec model was developed considering vector size 10, windows as 2, minimum count of records one, and assigned workers as 4. At the same time, the Bert-base-uncased model was employed to obtain sentence vectors of log templates and contents. Considering the time required for embedding and model training, optimizing the performance of the BERT pre-trained model was indispensable. According to [47] BERT model works effectively when max_seq_len is 25, pooling_layer is set as 12, priority batch size is 16, and prefetch_size is set as 10. The exact configuration was followed to improve the speed of the embedding process.

The classification models were trained over random training and testing data selection from the provided datasets. Records are selected using different seeds, as 70% and 80% of log entries as training data, and 30% and 20% remain as testing data. All classifiers were experimentally evaluated based on precision, recall, F1-score, specificity, and accuracy for each log level within our labeled dataset.

VII. RESULTS

Table IV demonstrates the accuracy of classification models where BERT is utilized as an embedding technique to apprehend the meaning of log templates and contents. The K-nearest neighbor model indicates lower accuracy among the seven implemented classifiers, whereas Random Forest offers higher classification accuracy for all datasets. As per the observation from Table IV, although the training and testing ratio changes yet there is an insignificant difference in the accuracy values.

Minimum accuracy was recorded as 81.47% for the BGL dataset using KNN, whereas 90.22% accuracy for the Apache dataset using the SVM model. The accuracy score greater than 90% is highlighted in Table IV. Higher accuracy was derived on an 80% training ratio for the Apache dataset using Linear Regression, SVM, and Random Forest and for the OpenStack dataset using Random Forest. Table IV observations show the KNN model returns lower accuracy, and the Random Forest model returns higher accuracy for almost all datasets. The difference between the minimum and maximum accuracy is 8.75%; thus, we can conclude that all implemented classification models have roughly comparable accuracy on Apache, OpenStack, Windows, BGL, and Android datasets. Based on this discussion, it is stated that semantic analysis using BERT helps classify various types of log records efficaciously. In addition, it is claimed that OpenStack and Android datasets are more suitable for the evaluation of the robustness of the classification model in the case of unseen log records. A more significant number of log templates are recorded for OpenStack and Android datasets in the parsing process (stated in Table I).

A. Results on Apache Dataset

Fig. 3 presents a metaphorical evaluation of seven classifiers over the Apache webserver error log dataset, considering a 30% and 20% testing data ratio. Prior to the classification, features were extracted with the help of the BERT model. Random Forest achieves the highest precision (96.08%) among the seven techniques and carries an F1 score of 92.71% in both cases, considering the 30% and 20% testing data ratio. This demonstrates that Random Forest provides the best classification results on semantic analysis of log templates and contents of log records. KNN, LinearRegression Support Vector Machines, Gradient Boosting Decision Trees, and Random Forests obtain consistent precision values on the Apache dataset, although the training-to-testing data ratio varies. Gradient Boosting Decision Trees and Random Forests show high precision but a low recall rate compared to other models. It is ascertained that all implemented models achieve consistent results on the Apache dataset, which implies that the semantics of the Apache log template and contents are derived correctly; thus, models can understand and perform classification operations. Also, unique templates (44) are derived during parsing 56,481 log records (refer to Table I) with 100% accuracy. The observation revealed the importance of log parsing in the log-based failure detection process.

TABLE IV. CLASSIFICATION ACCURACY IN PERCENTAGE ON VARIOUS DATASETS CONSIDERING 30% AND 20% TESTING DATA USING BERT EMBEDDING TECHNIQUE

IT Infra-structure	Training Ratio	k-Nearest Neighbors	Linear Regression	Support Vector Machines	Naïve Bayes	Gradient Boosting Decision Trees	Random Forest
Apache	70%	86.36	88.04	88.54	87.88	88.69	89.69
	80%	86.95	**90.01**	**90.22**	89.46	89.13	89.65
OpenStack	70%	84.17	86.11	86.34	85.27	86.07	88.17
	80%	85.17	88.02	88.14	87.94	87.81	89.34
Windows	70%	82.11	84.36	84.36	83.99	84.39	84.45
	80%	83.35	86.77	86.77	85.23	85.73	85.39
BGL	70%	81.47	83.39	83.39	83.79	83.09	85.21
	80%	82.98	85.79	85.70	85.74	85.01	86.89
Android	70%	81.89	83.89	83.89	81.87	82.46	85.72
	80%	83.67	85.79	85.79	84.72	84.72	87.56

Fig. 3. Precision, recall, f1-score, and specificity in percentage on Apache datasets considering 70% and 80% of training data using BERT embedding technique.

B. Results on OpenStack Dataset

Fig. 4 presents a metaphorical evaluation of seven classifiers over the OpenStack infrastructure log dataset, considering a 70% and 80% training data ratio, respectively. Before the classification, features were extracted with the help of the BERT model. KNN, Linear Regression, Support Vector Machines, Naïve Bayes, Gradient Boosting Decision Trees, and Random Forests achieved more than 90% precision when experiments were conducted on 30% of testing records and 20% of testing records, respectively. The precision, recall, F1-Score, and specificity improved by increasing the training-to-testing ratio. Among the seven implemented classifiers, Random Forest has *the* highest precision (95.13%), recall (89.31%), and F1 Score (92.13%) over 80% of the training data. The OpenStack dataset is preferred to check the classification efficiency for unobserved log records as it records a higher number (7,221) of log templates in 207,820 total log entries (refer to Table I), 3.47% of the whole dataset. In contrast, other datasets retrieve less than 1% of log templates. More variations in the log template promote checking the capability of semantic analysis to extract rigorous meaning that imparts to accurate classification.

C. Results on Windows Dataset

Fig. 5 presents a metaphorical evaluation of seven classifiers over the Windows event log dataset, considering a 70% and 80% training data ratio. Before the classification, features were extracted with the help of the BERT model. K-Nearest Neighbors records minimum precision as 85.67% and maximum precision by Random Forest as 87.99%, which means the difference in precision is significantly less for seven classification models. Although the precision, recall, and F1-Score values are less than 90%, they are consistent for all implemented classifiers. The Windows dataset results are decreasing compared to Apache and OpenStack datasets due to the size of the data and the number of unique templates. In the Windows dataset, 176 unique templates were extracted from 611,103 (refer to Table I) event records, which is only 0.02%. Here, unique templates are fewer, but the contents of the individual events fluctuate in compliance with the recorded message.

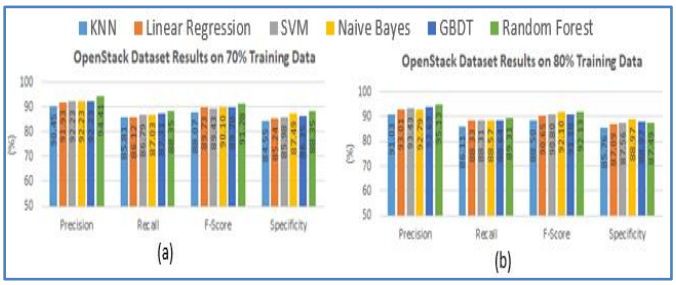

Fig. 4. Precision, recall, f1-score, and specificity in percentage on Openstack datasets considering 70% and 80% of training data using the BERT embedding technique.

Fig. 5. Precision, recall, f1-score, and specificity in percentage on Windows datasets considering 70% and 80% of training data using BERT embedding technique.

D. Results on BGL Dataset

Fig. 6 presents an illustrative evaluation of seven classifiers over the Blue Gene/L supercomputer log dataset, considering a 70% and 80% training data ratio. Before the classification, features were extracted with the help of the BERT model. KNN records minimum precision as 82.89% and maximum by Random Forest as 86.77%, which means the

difference in precision is significantly less for seven classification models. Although the precision, recall, and F1-Score values are less than 90%, they are consistent for all implemented classifiers. The BGL dataset results are decreasing compared to Apache and OpenStack due to the size of the data and the number of unique templates. In the Windows dataset, 619 unique templates were extracted from 4,747,963 (refer to Table I) log records, which is only 0.01%. Here unique templates are lesser, but the contents in the individual log fluctuate in compliance with the recorded message. According to observation, 1-2 % change in the precision, recall, and F1-Score values on different testing ratios, such as lower results recorded on the 30% testing ratio, whereas improved results by 1-2% recorded on 20% testing data.

Fig. 6. Precision, recall, f1-score, and specificity in percentage on BGL datasets considering 70% and 80% of training data using the BERT embedding technique.

E. Results on Android Dataset

Fig. 7 presents a metaphorical evaluation of seven classifiers over the Android framework log dataset, considering a 70% and 80% training data ratio. Before the classification, features were extracted with the help of the BERT model. KNN records the minimum precision as 78.34%. The difference in precision is significantly less for the seven classification models. The Android dataset results are decreasing compared to Apache and OpenStack due to the size of the data and the number of unique templates. In the Android dataset, 14,899 unique templates were extracted from 1,555,005 (refer to Table I) log records, which is only 0.09% of the whole dataset. Thus, the Android dataset is preferred to check the classification efficiency for unseen log records. More variations in the template help check the capability of semantic analysis to extract exact meaning that contributes to accurate classification. As a bottom line, it is stated that the greater the number of log records and the greater the number of unique templates, the more they help to train the model effectively.

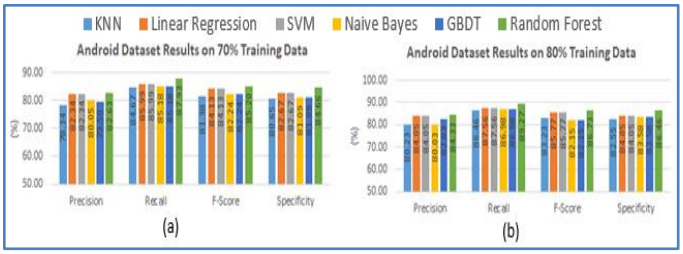

Fig. 7. Precision, recall, f1-score, and specificity in percentage on Android datasets considering 70% and 80% of training data using the BERT embedding technique.

VIII. Conclusion and Future Work

This paper describes an automatic and accurate classification of logs to facilitate system administrators during cause analysis of failures using system logs generated by various massive-scale IT infrastructures. The implemented models are able to understand the meaning of records and then classify them based on their level for log entries from multiple infrastructures such as Apache, OpenStack, Windows, BGL, and Android. The system admin can pay more attention to bizarre records and adopt remedial measures on the Anomalous records pointed out in the classification results,

The proposed system works efficiently on different types of log entries irrespective of changes in the format and imbalanced data. Thus, this work indicates how semantic analysis using BERT and classification using Linear Regression, Support Vector Machines, Naïve Bayes, Gradient Boosting Decision Trees, and Random Forests models furnish robust classification of new log entries. Considering the results and discussion points, K-Nearest Neighbors does not work well due to the imbalanced nature of log records. It is observed that, as compared with Doc2Vec, the semantic analysis achieved by the BERT pre-trained model is better while working with different classifiers. In addition, BERT influences the classification of any log record type with all classifiers and precisely processes the unseen or new log entries.

Experimentation using BERT as an embedding technique and machine learning models as classifiers derived precision, recall, F1 scores, and specificity in the range of 80% to 90%. In the extension to this work, we will try to improve the results to reduce false alerts with the help of applying deep learning techniques such as LSTM. Future work put forward the enforcement of LSTM models and propounding modified LSTM models to secure better results. Also, the system implemented with feature extraction and classification is semi-automated. In the future, the proposed system will be enhanced to implement a fully automated classification system to reduce human intervention.

References

[1] D. A. Bhanage, "DigitalCommons @ University of Nebraska - Lincoln Review and Analysis of Failure Detection and Prevention Techniques in IT Infrastructure Monitoring," 2021.

[2] D. A. Bhanage and A. V. Pawar, "Bibliometric survey of IT Infrastructure Management to Avoid Failure Conditions," *Inf. Discov. Deliv.*, vol. 49, no. 1, pp. 45–56, Nov. 2020, doi: 10.1108/IDD-06-2020-0060.

[3] 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, doi: 10.1109/access.2021.3128283.

[4] N. Aussel, Y. Petetin, and S. Chabridon, "Improving performances of log mining for anomaly prediction through nlp-based log parsing," *Proc. - 26th IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst. MASCOTS 2018*, pp. 237–243, 2018, doi: 10.1109/MASCOTS.2018.00031.

[5] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed, "DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series," *IEEE Access*, vol. 7, no. December 2018, pp. 1991–2005, 2019, doi: 10.1109/ACCESS.2018.2886457.

[6] 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, doi: 10.32604/csse.2022.022365.

[7] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," *Proc. Int. Conf. Dependable Syst. Networks*, pp. 575–584, 2007, doi: 10.1109/DSN.2007.103.

[8] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," *Proc. - IEEE Int. Conf. Data Mining, ICDM*, vol. 2020-Novem, pp. 1196–1201, 2020, doi: 10.1109/ICDM50108.2020.00148.

[9] J. Wang, C. Li, S. Han, S. Sarkar, and X. Zhou, "Predictive maintenance based on event-log analysis: A case study," *IBM J. Res. Dev.*, vol. 61, no. 1, pp. 121–132, 2017, doi: 10.1147/JRD.2017.2648298.

[10] X. Liu *et al.*, "Smart Server Crash Prediction in Cloud Service Data Center," *Intersoc. Conf. Therm. Thermomechanical Phenom. Electron. Syst. ITHERM*, vol. 2020-July, pp. 1350–1355, 2020, doi: 10.1109/ITherm45881.2020.9190321.

[11] S. Huang *et al.*, "HitAnomaly: Hierarchical Transformers for Anomaly Detection in System Log," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, no. 4, pp. 2064–2076, 2020, doi: 10.1109/TNSM.2020.3034647.

[12] M. A. Elsayed and M. Zulkernine, "PredictDeep: Security Analytics as a Service for Anomaly Detection and Prediction," *IEEE Access*, vol. 8, pp. 45184–45197, 2020, doi: 10.1109/ACCESS.2020.2977325.

[13] X. Zhang *et al.*, "Robust log-based anomaly detection on unstable log data," *ESEC/FSE 2019 - Proc. 2019 27th ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, pp. 807–817, 2019, doi: 10.1145/3338906.3338931.

[14] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, "Proactive failure detection learning generation patterns of large-scale network logs," *IEICE Trans. Commun.*, no. 2, pp. 306–316, 2019, doi: 10.1587/transcom.2018EBP3103.

[15] M. Pettinato, J. P. Gil, P. Galeas, and B. Russo, "Log mining to re-construct system behavior: An exploratory study on a large telescope system," *Inf. Softw. Technol.*, vol. 114, no. June, pp. 121–136, 2019, doi: 10.1016/j.infsof.2019.06.011.

[16] H. Ott, J. Bogatinovski, A. Acker, S. Nedelkoski, and O. Kao, "Robust and Transferable Anomaly Detection in Log Data using Pre-Trained Language Models," 2021, [Online]. Available: http://arxiv.org/abs/2102.11570.

[17] J. Wang *et al.*, "LogEvent2vec: LogEvent-to-vector based anomaly detection for large-scale logs in internet of things," *Sensors (Switzerland)*, vol. 20, no. 9, pp. 1–19, 2020, doi: 10.3390/s20092451.

[18] R. Vaarandi and M. Pihelgas, "LogCluster - A data clustering and pattern mining algorithm for event logs," *Proc. 11th Int. Conf. Netw. Serv. Manag. CNSM 2015*, pp. 1–7, 2015, doi: 10.1109/CNSM.2015.7367331.

[19] K. A. Alharthi, A. Jhumka, S. Di, F. Cappello, and E. Chuah, "Sentiment Analysis based Error Detection for Large-Scale Systems," *Proc. - 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, DSN 2021*, no. i, pp. 237–249, 2021, doi: 10.1109/DSN48987.2021.00037.

[20] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log Anomaly Detection via BERT," *Proc. Int. Jt. Conf. Neural Networks*, vol. 2021-July, 2021, doi: 10.1109/IJCNN52387.2021.9534113.

[21] H. Yang, X. Zhao, D. Sun, Y. Wang, and W. Huang, *Sprelog: Log-Based Anomaly Detection with Self-matching Networks and Pre-trained Models*, vol. 2. Springer International Publishing, 2021. doi: 10.1007/978-3-030-91431-8_50.

[22] E. Elbasani and J. D. Kim, "LLAD: Life-Log Anomaly Detection Based on Recurrent Neural Network LSTM," *J. Healthc. Eng.*, vol. 2021, 2021, doi: 10.1155/2021/8829403.

[23] X. Duan, S. Ying, H. Cheng, W. Yuan, and X. Yin, "OILog: An online incremental log keyword extraction approach based on MDP-LSTM neural network," *Inf. Syst.*, vol. 95, p. 101618, 2021, doi: 10.1016/j.is.2020.101618.

[24] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, vol. 2020-Octob, pp. 92–103, 2020, doi: 10.1109/ISSRE5003.2020.00018.

[25] Y. Xie, K. Yang, and P. Luo, "LogM: Log Analysis for Multiple Components of Hadoop Platform," *IEEE Access*, vol. 9, pp. 73522–73532, 2021, doi: 10.1109/ACCESS.2021.3076897.

[26] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," *arXiv*. arXiv, Aug. 14, 2020.

[27] "GitHub - logpai/loghub: A large collection of system log datasets for AI-powered log analytics." https://github.com/logpai/loghub (accessed Jun. 27, 2021).

[28] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards Automated Log Parsing for Large-Scale Log Data Analysis," *IEEE Trans. Dependable Secur. Comput.*, vol. 15, no. 6, pp. 931–944, 2018, doi: 10.1109/TDSC.2017.2762673.

[29] M. Du and F. Li, "Spell: Streaming parsing of system event logs," *Proc. - IEEE Int. Conf. Data Mining, ICDM*, pp. 859–864, 2017, doi: 10.1109/ICDM.2016.160.

[30] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," *Proc. 3rd IEEE Work. IP Oper. Manag. IPOM 2003*, pp. 119–126, 2003, doi: 10.1109/IPOM.2003.1251233.

[31] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," *Proc. - 2017 IEEE 24th Int. Conf. Web Serv. ICWS 2017*, pp. 33–40, 2017, doi: 10.1109/ICWS.2017.13.

[32] W. Meng *et al.*, "A Semantic-aware Representation Framework for Online Log Analysis," *Proc. - Int. Conf. Comput. Commun. Networks, ICCCN*, vol. 2020-Augus, pp. 1–7, 2020, doi: 10.1109/ICCCN49398.2020.9209707.

[33] "A gentle introduction to Doc2Vec. TL;DR | by Gidi Shperber | Wisio | Medium." https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e (accessed Jan. 31, 2022).

[34] "BERT Explained: A Complete Guide with Theory and Tutorial – Towards Machine Learning." https://towardsml.com/2019/09/17/bert-explained-a-complete-guide-with-theory-and-tutorial/ (accessed Jul. 21, 2021).

[35] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.*, vol. 1, no. Mlm, pp. 4171–4186, 2019.

[36] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "KNN model-based approach in classification," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 2888, no. November 2012, pp. 986–996, 2003, doi: 10.1007/978-3-540-39964-3_62.

[37] 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, 2002, doi: 10.1080/00220670209598786.

[38] 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, 2020, doi: 10.1016/j.neucom.2019.10.118.

[39] 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, 2021, doi: 10.1186/s13634-021-00742-6.

[40] H. Seto *et al.*, "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, 2022, doi: 10.1038/s41598-022-20149-z.

[41] L. E. O. Breiman, "Random Forests," pp. 5–32, 2001.

[42] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "RCV1: A new benchmark collection for text categorization research," *J. Mach. Learn. Res.*, vol. 5, pp. 361–397, 2004.

[43] H. M and S. MN, "A Review on Evaluation Metrics for Data Classification Evaluations," *Int. J. Data Min. Knowl. Manag. Process*, vol. 5, no. 2, pp. 01–11, 2015, doi: 10.5121/ijdkp.2015.5201.

[44] M. Grandini, E. Bagli, and G. Visani, "Metrics for Multi-Class Classification: an Overview," pp. 1–17, 2020, [Online]. Available: http://arxiv.org/abs/2008.05756.

[45] R. Trevethan, "Sensitivity, Specificity, and Predictive Values: Foundations, Pliabilities, and Pitfalls in Research and Practice," *Front. Public Heal.*, vol. 5, no. November, pp. 1–7, 2017, doi: 10.3389/fpubh.2017.00307.

[46] "Different metrics to evaluate the performance of a Machine Learning model | by Swapnil Vishwakarma | Analytics Vidhya | Medium." https://medium.com/analytics-vidhya/different-metrics-to-evaluate-the-performance-of-a-machine-learning-model-90acec9e8726 (accessed Jul. 09, 2021).

[47] "Benchmark — bert-as-service 1.6.1 documentation." https://bert-as-service.readthedocs.io/en/latest/section/benchmark.html#speed-wrt-max-batch-size (accessed Jun. 22, 2022).