Received 7 October 2022, accepted 4 November 2022, date of publication 10 November 2022, date of current version 16 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3221427

RESEARCH ARTICLE

A Heuristic Approach Using Template Miners for Error Prediction in Telecommunication Networks

PÉTER MARJAI[1], PÉTER LEHOTAY-KÉRY[1], AND ATTILA KISS[1,2]
[1]Department of Information Systems, ELTE Eötvös Loránd University, 1117 Budapest, Hungary
[2]Department of Informatics, J. Selye University, 945 01 Komárno, Slovakia

Corresponding author: Attila Kiss (kiss@inf.elte.hu)

This work was supported in part by the National Research, Development and Innovation Fund of Hungary through the Thematic Excellence Programme (National Challenges Subprogramme) for the Project Application Domain Specific Highly Reliable IT Solutions under Grant TKP2020-NKA-06; in part by the Research and Development Operational Programme for the Project ''Modernisation and Improvement of Technical Infrastructure for Research and Development of J. Selye University in the Fields of Nanotechnology and Intelligent Space'' under Grant ITMS 26210120042; in part by the European Regional Development Fund; in part by the National Research, Development and Innovation Fund through the New National Excellence Program of the Ministry for Innovation and Technology under Grant ÚNKP-21-3; and in part by the Ericsson-Eötvös Loránd University (ELTE) Software Technology Laboratory.

ABSTRACT With the appearance of large-scale systems, the size of the generated logs increased rapidly. Almost every software produces such files. Log files contain runtime information of the software, as well as indicate noteworthy events or suspicious behaviors like errors. To understand and monitor the operation of the system, log files are a valuable source of information, which can be used to predict upcoming anomalies. In recent years numerous techniques have been proposed for this purpose. There are supervised models like SVM or decision trees and also unsupervised ones like Isolation Forest, Log Clustering, or PCA. There are also methods that use deep learning, like Autoencoder, CNN, LSTM, or Transformer. Many of the above-mentioned methods take advantage of template miners, that extract the event types from the unstructured data. In our paper, we propose a method that uses these templates to predict upcoming anomalies. We use 80% of our data for training, and 20% for tests. First, we use half of the train data and sort the templates that have an occurrence that is followed by an error to create a list of candidate templates. In the second step, we use the other half, to check how often the ten upcoming lines after a candidate template actually contain an anomaly. If a given percentage is reached, we consider the template as an indicator for upcoming anomalies. We conduct various experiments to verify the capability of our method like measuring the precision, recall, f-score accuracy, and speed on various data sources. The proposed method slightly falls behind SVM and CNN with an average of 88.06% precision, 90.43% recall, and 89.11% f-score, however, it has better accuracy with 98.19%. In addition, our algorithm is two times faster than SVM and three and a half times faster than CNN.

INDEX TERMS Anomaly detection, autoencoder, CNN, IPLoM, isolation forest, log clustering, log parsing, LSTM, PCA, SVM, transformer.

I. INTRODUCTION

The creation of log files is a general task that every software performs. The developers insert print statements into the source code, to save the runtime information of the application. With the use of the generated log files, crucial information can be retrieved, which then can be used for several objectives, like business model mining [1], [2], user behavior identification [3], [4], office tracking [5], performance monitoring [6], [7], or reliability engineering [8].

Log files contain information about the preferences of users on various subjects. Business models can be created with the use of this information and be used to enhance the performance of the service, for example, with the use of targeted advertising. In [1] a conceptual model that uses hyper-personalization is introduced to escalate buyer interest. In [2] an e-customer behavioral graph is proposed that can be used to measure the movements of e-customers based on their behavior change.

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

User behavior identification can also be done by the use of information gathered from log files. Patterns of mental problems, addictions, or underperforming can come to light with inspection of such files. In [3] they use machine learning and linear regression to identify students that are likely to fail a subject. The authors of [4] also use machine learning in pair with Weighted Support-vector Machine (SVM), BernoulliNB, logistic regression, and MLPClassifier to detect ‘‘internet addiction’’ in the case of children.

In [5] the system logs that are logged in a distributed file system are used in a scalable online manner to be checked again policies that are formulated in an expressive temporal logic.

Due to the rapid development of information technology, more and more large-scale system has appeared. Since it is impossible to monitor the functionality of such systems by hand, software-aided performance monitoring has become a crucial task. The authors of [6] propose a clustering-based algorithm that uses system key performance indicators and log sequences to identify impactful system problems that result in performance loss. Since the troubleshooting of cloud services is especially hard, a new approach called megatables is represented in [7] that outputs millibottleneck predictions and supporting visualizations based on automatic interpretation of log data.

Reliability engineering can be done with the use of automated log analysis. In [8] a survey is provided regarding this subject.

Over the course of the last few years, numerous algorithms have been proposed to retrieve event types from unstructured logs. Each log line can be associated with an event template. These templates are made up of a constant part, that is identical at all occurrences and a parameter part, that could be different at each instance. The algorithm in [9] builds a fix-depth tree. Each layer of internal nodes sorts the log lines based on a heuristic. A node in the first layer represents log messages of the same length. At the second layer, each node contains messages that have the same constant token at the beginning of the message. The third layer filters the log entries into groups based on their token similarity. The authors of [10] make us of the assumption that entries that correspond to a template have words with equivalent length at the same token positions. In [11] Non-dominated Sorting Genetic Algorithm II (NSGA-II) [12] a multi-objective genetic algorithm is employed to randomly generate a group of chromosomes, which are possible solutions. These chromosomes are then evolved and reproduced with the use of crossover and mutation. In the end, out of all possible solutions, the knee point is chosen to be the final solution. With the use of Longest Common Subsequent (LCS), the algorithm proposed in [13] the log messages are converted into a sequence of tokens identified by a unique ID. With the use of backtracking, they retrieve the message types. The authors of [14] propose a new tree-based template mining technique, that in addition to the rate of matching tokens in two distinct log entries, also considers the tokens at which two log entries disagree.

Our method is built to be used in a system that automatically collects logs, configuration, and state description files from telecommunication network nodes on daily basis. After the collection of these files, they can be analyzed to support the maintenance of the network. Using our method, based on the vast amount of collected logs, possible networking failures can be predicted in advance, so that these failures can be prevented. The algorithm can also predict exact error codes that are going to be thrown on the networking nodes. Using the information provided by our system, customer support, and network maintenance groups can intervene effectively before the actual failure happens. Naturally, our method can also be used in other kinds of systems too, where logs are generated by the software.

II. RELATED WORKS

Until this day, numerous methods have been proposed to predict upcoming anomalies and errors. It can be done manually with the use of domain knowledge, however, it is usually a slow process. Because of this, detection is especially important in the case of large-scale services, where the precise and fast detection of an anomaly can aid the operators in quickly solving the problem. Anomaly detection has been employed in various fields, like performance monitoring, intrusion detection, fault detection in manufacturing, fraud detection, or even healthcare to spot potential medical problems or risks.

Many researchers are taking advantage of machine learning techniques, to make a prediction. In [15] they provide a framework that detects anomalies based on application log files. First, a correlation analysis is made on the data. The result of this analysis is the input for various machine learning techniques of which the one with the best precision and accuracy values. They state that the combination of machine learning and parameter approximation aided by time-series models yielded the best results.

The authors of [16] propose an approach that is capable of intrusion detection in industrial control systems with the use of anomaly detection. A hybrid model is used to take advantage of the difference between the communication patterns that occur between ground devices and the anticipated communication. They use several techniques like data preprocessing, dimensionality reduction, nearest-neighbor rule, or Bloom filters.

Cyber-physical systems are complex systems that contain both software and physical components. They don’t have an accurate model, due to their ever-changing nature, thus anomaly detection is a hard task in the case of such systems. With the use of the logs of such a system, the authors of [17] propose a method that is capable of finding outliers that indicate actual faults in the system. They first collect the related log entries. After that, each entry is converted into a real-value vector. Each vector is then normalized. A single vector is then created from the vectors within a given window. For every window, the outlier factor is calculated with the

Local outlier factor (LOF) [18] algorithm. Finally, every high outlier factor is identified as an anomaly.

There is also a growing interest in research on financial fraud detection. An in-depth survey is performed in [19] regarding unsupervised, clustering-based anomaly detection in the economic domain. There are also other graph-based techniques that inspect connectivity patterns to recognize fraud. The authors of [20] have created a framework of such methods that had been published between 2007 and 2018. There are very little labeled data (legal/fraud) in the field of tax fraud. In [21] unsupervised anomaly techniques are employed to detect fraud based on the labeled data.

Detecting anomalies in medical images like X-rays are a very laborious task. To meet this challenge, the authors of [22] develop a new method that uses a re-designed training pipeline as input for classic autoencoders. These pipelines are capable of handling complex images that have high resolution. A survey about the usage of machine learning methods in medical anomaly detection is concluded in [23].

In a previous paper [24] we investigated the prediction capability of cosine distance, Jaccard similarity, and Euclidean distance between the actual log entry and the previous entries. The methods were used in pair with Term Frequency-Inverse Document Frequency(TFIDF), Doc2Vec, and Locality Sensitive Hashing (LSH).

In this paper, we propose a method that predicts if an error is going to happen in the window of the next ten entries based on the template of the actual entry. With this method higher than 90% values of precision, recall, f-score, and accuracy can be reached within a short time. A more detailed explanation of the algorithm can be found in Section IV.

The paper has the following structure. Section III contains the introduction of the log parsing concept. It is followed by a brief summary of the used template miner, Iterative Partitioning Log Mining (IPLoM). A high-level overview of other anomaly detection methods such as Isolation Forest, Log Clustering, Principal Component Analysis (PCA), and SVM can also be seen here. Other algorithms, that use machine learning techniques like Autoencoder, Convolutional Neural Network (CNN), Long Short-term Memory (LSTM), and Transformers are also explained here. A brief explanation of the used evaluation metrics can also be found in this section. We provide a detailed description of the proposed algorithm in Section IV. In Section V, we present an explanation of the concluded experiments, that measure the achieved precision, recall, f-score, and accuracy values on log files that were created by real-work networking assets. The speed of the different methods is also evaluated in this section. The conclusion of the paper and the possible future works are listed in Section VI.

III. CONCEPTS AND PROBLEMS

A. ACQUIRING TEMPLATES

1) LOG PARSING

Since developers can insert free-text messages, the entries in a log file are usually raw and unstructured. An entry contains information about events that have occurred during the runtime of the software like system upgrades, restarts, error messages, and such. A log entry is a collection of information like timestamps, flags, sequence numbers, messages, and so on. The message part can be divided into two groups. The tokens (strings separated by spaces) can either be constants or parameters. A constant token is always the same at each occurrence of a log line corresponding to an event type. Parameter tokens can be different on each occasion. A fragment of our working data can be seen in Figure 1.

The template (event type) corresponding to the fifth line consists of a single constant token "XF_START". It can be seen that it is the same at each occurrence. The fourth-row "NPU Cold Restart" is made up of two constant tokens "NPU" and "Restart", while the "Cold" token is a parameter. In the line with sequence number 1088, the "NPU" and "Restart" tokens are the same, however instead of "Cold", a "Warm" parameter can be seen. During log parsing, each message m of all messages $m_1, m_2, \ldots, m_N$ is assigned to a group that correspond to a template T. Although log parsers are powerful tools, they have their limitations. Pre- and post-processing of the data can improve the performance. For example, the number of entries to be parsed can be reduced by deleting duplicates or regex can be used to eliminate unnecessary fields like timestamp. A list of suggestions for pre-and post-processing can be found in [25].

2) IPLoM

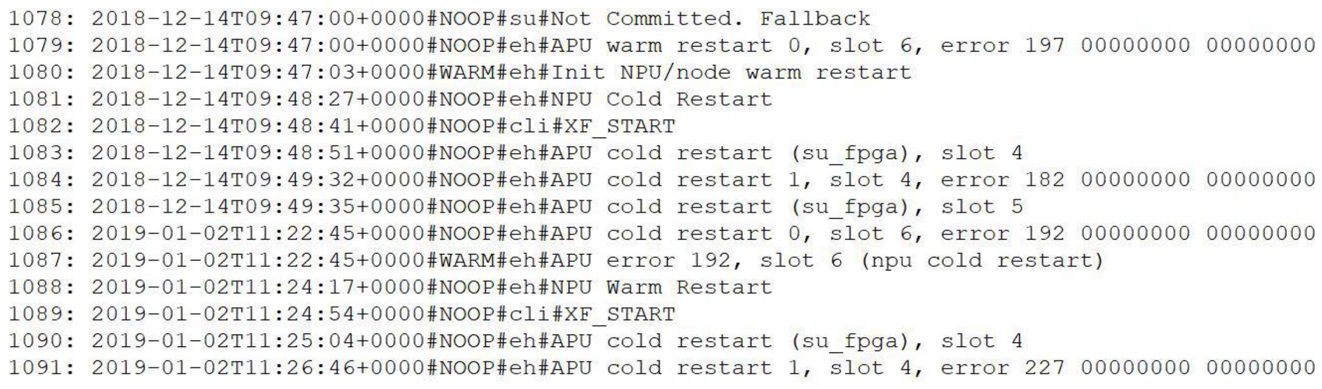
The authors of [26] propose a method that iteratively clusters the log entries based on different heuristics to retrieve the templates. In the first three steps, different heuristics are used, while the fourth step yields the event types.

The first step uses the assumption that log entries that are made of the same number of tokens are corresponding to the same template. Based on this, distinct groups are created that contain lines that are n long sequences of words.

The number of unique tokens can also be used to partition the log entries. Based on the assumption that the token position with the least unique values is a constant, the algorithm partitions the lines into groups. At the end of this step, each line in the group has the same constant token at the same position.

In the last partitioning step, bijection between the entries in a group is used as a heuristic. The most frequently appearing token count expresses the number of different templates in the group. The algorithm partitions the entries into different groups based on the first two token positions that have an equal number of unique tokens as the most frequent token count.

By this time, the groups only contain lines that correspond to the same event type, however, the distinction between constant tokens and parameters has to be done. To achieve this, the algorithm counts the number of different words at a position. If it has multiple words, then it is a parameter otherwise, it is a constant.



```
1078: 2018-12-14T09:47:00+0000#NOOP#su#Not Committed. Fallback
1079: 2018-12-14T09:47:00+0000#NOOP#eh#APU warm restart 0, slot 6, error 197 00000000 00000000
1080: 2018-12-14T09:47:03+0000#WARM#eh#Init NPU/node warm restart
1081: 2018-12-14T09:48:27+0000#NOOP#eh#NPU Cold Restart
1082: 2018-12-14T09:48:41+0000#NOOP#cli#XF_START
1083: 2018-12-14T09:48:51+0000#NOOP#eh#APU cold restart (su_fpga), slot 4
1084: 2018-12-14T09:49:32+0000#NOOP#eh#APU cold restart 1, slot 4, error 182 00000000 00000000
1085: 2018-12-14T09:49:35+0000#NOOP#eh#APU cold restart (su_fpga), slot 5
1086: 2019-01-02T11:22:45+0000#NOOP#eh#APU cold restart 0, slot 6, error 192 00000000 00000000
1087: 2019-01-02T11:22:45+0000#WARM#eh#APU error 192, slot 6 (npu cold restart)
1088: 2019-01-02T11:24:17+0000#NOOP#eh#NPU Warm Restart
1089: 2019-01-02T11:24:54+0000#NOOP#cli#XF_START
1090: 2019-01-02T11:25:04+0000#NOOP#eh#APU cold restart (su_fpga), slot 4
1091: 2019-01-02T11:26:46+0000#NOOP#eh#APU cold restart 1, slot 4, error 227 00000000 00000000
```

FIGURE 1. An extract of a log file generated by real-world networking device.

B. REGULAR ANOMALY DETECTION METHODS

1) ISOLATION FOREST

Isolation Forest was introduced in [27] and uses the idea that under random partitioning, anomalies are more likely to be isolated. They use data-induced random trees, that recursively repeat the partitioning of instances, until all of them are separated. In the paper, they first randomly select an attribute, then randomly select a spit value between the minimum and maximum values of that attribute.

Let T be a node of an isolation tree. T can either be an internal node with one test that has two child nodes (T_l, T_r), or an isolated node that has no child. During a test data points are split between T_l and T_r based on the test $q < p$, where q is an attribute, and p is the split value. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a data sample of n elements with d- variate distribution. The isolation tree is built as follows. X is recursively partitioned by q and p until one of the three followings is satisfied: either the height limit of the tree is reached, all elements in X have the same values, or $|X| = 1$.

The anomalies are detected with the use of the path length. The path length $h(x)$ of a node is the number of edges that are needed to reach it from the root node. Since anomalies result in a lesser number of partitions, shorter paths in the tree indicate the presence of an anomaly.

During the training phase, a bunch of isolation trees is built with sub-samples (this is the isolation forest). In the test phase, the test elements are passed through these trees and an anomaly score is calculated with the use of the path length. When such a forest produces outstandingly short paths for some distinct set of nodes, then they are considered to be anomalies.

2) LOG CLUSTERING

The authors of [28] introduce a method to identify problems in the case of online systems. The algorithm consists of two steps. The first one is the construction, where log sequences that were acquired from a test environment are turned into vectors and these vectors are then clustered. The representative vector of each sequence and the steps required to resolve the problem are saved into a knowledge base. The second phase is the same, except that the sequences are collected from a production setting. The representative vectors are compared with the ones in the knowledge database. Only the unseen sequences have to be examined by the engineers. A more detailed description of the steps follows.

The log sequences contain multiple events. These sequences are transformed into a vector where each element represents an event. Since not every event has the same informative power, the vectors are weighted with the use of IDF [29] and Contrast-based Event Weighting. IDF is calculated as:

$$w_{idf}(t) = log\left(\frac{N}{n_t}\right) \tag{1}$$

where t is a template, N is the total number of log sequences and n_t is the number of the appearances of t. With this, lower weights are assigned to frequent events, while uncommon events have higher weights. Based on the assumption that events in the production setting are more likely to express errors. Either 1 or 0 is assigned to an event:

$$w_{con}(t) = \begin{cases} 1 & \text{if } t \text{ appears in } \Delta S \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where the set of the log events that only appear in the production setting is denoted as ΔS. For every t event, the contrast-based weight and the IDF-based weight are combined:

$$w(t) = 0.5 \times w_{con}(t) + 0.5 \times Norm(w_{idf}(t)) \tag{3}$$

where $Norm$ is a Sigmoid function that is used to normalize the IDF-based weight to a value between 1 and 0. After this the cosine similarity between any two V_i and V_j vectors is calculated as follows:

$$\begin{aligned} Sim(V_i, V_j) &= \frac{S_i \cdot S_j}{\parallel S_i \parallel \parallel S_j \parallel} \\ &= \frac{\sum_{k=1}^{n} S_iE_k \times S_jE_k}{\sqrt{\sum_{k=1}^{n}(S_iE_k)^2} \times \sqrt{\sum_{k=1}^{n}(S_jE_k)^2}} \end{aligned} \tag{4}$$

where *SjEk* indicates the kth event in the jth vector. With the use of the *Sim* value, an Agglomerative Hierarchical clustering [30] takes place. In the end, the centroid of each cluster is used as a representative vector for the cluster.

3) PCA

A general methodology to detect system anomalies is proposed in [31]. The approach consists of three separate phases. The first one can be broken down into two further steps. First, they analyze the source code, to salvage all of the print statements and deduce variable types, which are related to the statement. This is followed by runtime log parsing to acquire all possible message templates.

The second phase is the feature extraction, during which correlating messages are grouped together, and represented with a vector per group, to create the message count vector [32]. Each element of the vector represents the number of occurrences of a given template.

In the third phase, Principal Component Analysis (PCA) [33], an unsupervised machine learning technique is applied to the feature vector to identify abnormal and normal messages. PCA creates new uncorrelated variables that successively maximize variance.

4) SVM

The authors of [34] investigate the anomaly prediction performance of different machine learning techniques like RIPPER, a rule-based classifier [35], Nearest Neighbor-based classification, and Support Vector Machines (SVM) [36]. Out of these three methods, for our paper, we use the SVM approach. SVMs are a set of generalized classifiers. Normally, a hyperplane or a set of hyperplanes is constructed by a support-vector machine. These are usually high- or infinite-dimensional spaces, that can be used for various purposes like regression, classification, or outlier detection. The hyperplane that is the farthest from any of the training-data points usually achieves good separations, since the bigger the margin, the lower the generalization error. Outlying points are usually identified as anomalies.

C. MACHINE LEARNING ANOMALY DETECTION METHODS

1) AUTOENCODER

An unsupervised method that combines two deep Autoencoder networks with an Isolation forest is introduced in [37]. An Autoencoder is a feed-forward multilayer neural network that has an equal number of input and output units [38]. During training, a compact representation with minimized error is created with the use of a loss function is used that ensures that the output is not far from the input. An Autoencoder that has more than one hidden layer is called a deep Autoencoder [39].

After pre-processing the text and padding the sentences the data is split into t_1, t_2, t_3, where t_1 and t_2 are small training sets, while t_3 is the test set. The Autoencoders are used for feature extraction in an unsupervised way. Lets denote the two Autoencoders as a_1 and a_2. As the first step, t_1 is used to train a_1. After this, a_1 is fed with t_2 and t_3, which results in the extraction of feature sets f_1 and f_2. To predict the data, f_1 is used as an input for an Isolation Forest with 100 Isolation Trees. The positive predicted data from the output of the Isolation forest is split into p_1 and p_2 and then is used as an input for a_2. Lastly, the output of a_2 with a threshold is used to detect anomalies.

2) CNN

The authors of [40] employ Convolutional Neural Networks to detect anomalies. Instead of global information, CNN is capable of seizing local semantic information and overcoming over-fitting difficulties that arise in a normal neural network. Features are extracted from local receptive fields that are part of feature maps of the previous field, with the use of a convolution operator. A non-linear transformation is executed with the use of an activation function such as Rectified Linear Units(ReLU), Tahn, or Sigmoid. The value of an unit position (m, n) in ith layers jth feature map is denoted as $v_{i,j}^{m,n}$:

$$v_{i,j}^{m,n} = \sigma\left(b_{i,j} + \sum_{N}\sum_{p=0}^{P_{i-1}}\sum_{q=0}^{Q_{i-1}} w_{ij}^{pq} v_{(i-1)N}^{(x+p)(y+q)}\right) \tag{5}$$

where P_i is the height of the kernel, Q_i is the width of the kernel, N indexes over the set of feature maps in the $(i-1)^{th}$ layer, w_{ij}^{pq} is the parameter and $b_{i,j}$ stands for a bias function of this feature map [41].

First, a codebook (a trainable matrix) is created to map each log key (a frequent constant) in a sequence into a vector. They call this approach logkey2vec. Since CNN require a 2-dimension matrix as its input, the codebook is used to map 1-dimension vectors into 2-dimension matrices. After this, CNN convolutes over the embedded log vectors with three one-layer filters at the same time. Leaky Rectified Linear Unit (ReLU), which solves the dead ReLU problem, is used as an activation function. After the three independent filterings, a max-pooling layer is employed to concatenate the output of the filters. In the end, a softmax function is applied to the concatenated output, which results in the labeling of normal and abnormal logs.

3) LSTM

The approach proposed in [42] introduces a new word representation model, template2vec. To create the template vector, first, the set of synonyms and antonyms are constructed. This is followed by the word vector generation, which is based on the distributional Lexical-Contrast Embedding (dLCE) [43] algorithm. In the end, the template vector, which is the weighted average of the word vectors of the words that make up the template.

Based on the assumption that during normal operation, log entries have some kind of sequential pattern, they assume that if no anomaly arises, the next template is predictable. Denote the set of all discrete template vectors as $\Omega = \{v_1, v_2, \ldots, v_n\}$. The detection sequence is a sliding window of w recent templates. Suppose that

$S_j = (s_j, s_{j+1}, \ldots, s_{j+w-1})$ is a subsequence of $S = (s_1, s_2, \ldots, s_m)$ log sequence. In this case, the template vector sequence for S_j is $V_j = (v_{(s_j)}, v_{(s_{j+1})}, \ldots, v_{(s_{j+w-1})})$, where the template vector of s_i is $v_{(s_i)}$ and $v_{(s_i)} \in \Omega$.

At this step, they employ a favored recurrent neural network architecture, that is capable of prediction for sequences, namely LSTM (Long Short Term Memory) [44]. For S_j in the training stage, V_j is the input for LSTM.

Apart from sequential patterns, a template sequence can also have quantitative relations that can be also used to detect abnormal behavior. Based on this, $C_i = (c_i(v_1), c_i(v_2), \ldots, c_i(v_n))$ denotes the count vector of the log sequence a log entry $s_i \in S_j$, where $c_i(v_k)$ indicates the number of v_k in the template vector sequence $(v_{(s_{i-w+1})}, v_{(s_{i-w+2})}, \ldots, v_{(s_i)})$ and $v_k \in V$. To learn the quantitative pattern of S_j, $C_j, C_{j+1}, \ldots, C_{j+w-1}$ is used as the input of the LSTM. Using LSTM, the probabilities of the possible upcoming template vectors after a log sequence is learned. In the end, if the actual template vector is considered to be normal, if it is included in the top k candidates provided by LSTM, otherwise it is considered to be abnormal.

4) TRANSFORMER

The algorithm introduced in [45] consists of two main parts, the tokenization of the log messages, and the use of a transformer, a simple network architecture that is based solely on an attention mechanism.

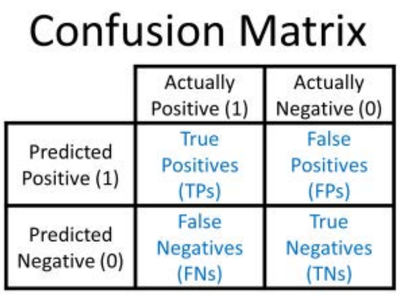
First, the log entries are converted into a token sequence with the use of the Natural Language Toolkit (NLTK) text processing module [46]. Special ASCII characters are removed, and the entries are transformed into lowercase and broken into word tokens. The numbers and stop words are eliminated, and a special '[EMBEDDING]' token is added as the first token, which can be used to summarize the context of the log entry as a vector representation.

The transformer architecture has an offline and online mode. The offline mode is used to tune parameters with the use of the backpropagation of the log entries. During this step, optimal hyperparameters are chosen. To retrieve the anomaly score and respective log vector representation z, the entries are passed through the saved model in the online phase. The positional encoding is achieved with the use of a multi-head, self-attention Transformer model [47], which takes the tokenized entries as its input. The model considers normal data, to have small distances, while anomalies to be distant. In the end, a Spherical Cross-Entropy Loss function is used to calculate the anomaly score for test entries.

D. EVALUATION METRICS

To indicate the performance of a classification model on a set of test data for which the true values are known, a confusion matrix is usually used. Figure 2 showcases the structure of such a matrix.

where T_p (True Positives) is the number of positive samples that the model correctly classified as positive, T_n (True Negatives) is the number of negative samples that are predicted to be negative, F_p (False Positives) is the number of negative samples which are predicted to be positive, F_n (False Negatives) is the number of positive samples that the model incorrectly classified as negative.


Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

FIGURE 2. Confusion matrix.

Accuracy indicates the ratio of the data that has been correctly predicted and is denoted as:

$$A = \frac{T_p + T_n}{T_p + T_n + F_p + F_n} \tag{6}$$

Precision is a metric that quantifies the number of correct positive predictions made:

$$P = \frac{T_p}{T_p + F_p} \tag{7}$$

Recall is the fraction of relevant instances that were retrieved:

$$P = \frac{T_p}{T_p + F_n} \tag{8}$$

F-Measure provides a way to combine both precision and recall into a single measure that captures both properties:

$$P = \frac{2 \times P \times R}{P + R} \tag{9}$$

IV. THE PROPOSED ALGORITHM

In this paper, we propose a method that uses templates to predict upcoming anomalies.

First, we use IPLoM, to retrieve the corresponding template for all of the log lines. It consists of four steps based on different heuristics. First, messages with the same word length are grouped together. Next, the positions that have the least different words are considered to be constants. The messages that have the same constants at the same position are put into the same collection together. After this, a bijection is made between the entries of a collection to further group them, resulting in groups that represent different message types. Finally, the algorithm decides if a word is constant or not based on the number of different words in that position. While there are other template miners, based on our experiments in [48] and the benchmarking results that were proposed in [49], we chose to use IPLoM for our algorithm. A more detailed description of IPLoM can be found in Section III-A2.

The base idea of our approach is that there could be specific templates that occur before an error. Error messages usually contain strings like "error", "fail", "fault" and so on. Based on the domain knowledge, the user of our algorithm can define the set words or substrings that may indicate anomalous log messages. Let this set be denoted as E. We split our data into three parts, 40-40% for training purposes and 20% for testing The two training sets are denoted as t_1 and t_2, while the testing set is denoted as t_3. The algorithm consists of three steps.

In the first step, we use the first 40% of our data, t_1. We first identify all the templates, that appear before errors or anomalies. A logline l_i is considered to be an anomaly if it contains any words or substrings from E.First, an empty candidate set is created, namely C. The corresponding template C_i of a log line, l_i is added to the candidate set C, if the following log line l_{i+1} contains any substrings from E.

In the second step, we iterate over t_2 and watch for candidate templates. If a candidate template is found, the upcoming ten lines are checked for errors and anomalies. Once again, the words and substrings in E are used to detect anomalies. We employ two counters for each candidate template C_i. The first one, c_{in} is increased at every occurrence of C_i while the second one, c_{ip} is only increased if an error has been found in the next ten lines after the candidate template C_i. Candidate templates, that do not reach a user-defined k ratio of $\frac{c_{ip}}{c_{in}}$ are removed from the candidate set C.

In the testing phase, the remaining candidate templates are used to predict upcoming errors on t_3 If a candidate template C_i is reached, we signal that the next ten-line is likely to have an error or anomaly.

A pseudo-code of the proposed algorithm is presented as Algorithm 1 and a flowchart explaining the proposed algorithm can be seen in Figure 3.

The time complexity of our algorithm is $\mathcal{O}(n)$, where n is the number of log messages that are used to train our algorithm. This would be a drawback in the case of large datasets with the size of TBs, however since there are a fixed number of templates that can occur during the runtime of an algorithm (no new print statements can be inserted into the code while running), the algorithm can be trained on small datasets that contain the templates. This way, our algorithm can be trained on a predetermined chunk of data, which makes the algorithm perform in $\mathcal{O}(1)$ time, since only the actual log line has to be checked, and no distribution of the algorithm is needed. For the same reason, once trained, the algorithm also can handle real-time data.

Algorithm 1 Acquiring the Candidate Templates

```
Input: A collection of log messages
Input: k
Input: E
Output: A set of log templates that indicate upcoming
  anomalies
  $t_1 \leftarrow$ 40% of the data
  $t_2 \leftarrow$ 40% of the data
  $C \leftarrow \{\}$
  $l_i \leftarrow$ first line of $t_1$
  while $l_i$ in $t_1$ do    ▷ Iterating over all of the log lines in $t_1$
      if any from $E$ is in $l_{i+1}$ then
          $C \leftarrow C_i$
      end if
      $l_i \leftarrow l_{i+1}$
  end while
  for $C_i$ in $C$ do
      $c_{in} \leftarrow 0$
      $c_{ip} \leftarrow 0$
  end for
  $l_i \leftarrow$ first line of $t_2$
  while $l_i$ in $t_2$ do    ▷ Iterating over all of the log lines in $t_2$
      if $C_i$ is in $C$ then
          $c_{in} \leftarrow c_{in} + 1$
          $Bool \leftarrow$False
          for $j = 1, j{+}{+}, j \leq 10$ do
              if any from $E$ is in $l_{i+j}$ then
                  $Bool \leftarrow$ True
              end if
          end for
          if $Bool$ then
              $c_{ip} \leftarrow c_{ip} + 1$
          end if
      end if
      $l_i \leftarrow l_{i+1}$
  end while
  for $C_i$ in $C$ do
      if $\frac{c_{ip}}{c_{in}} < k$ then
          remove $C_i$ from $C$
      end if
  end for
```

V. RESULTS

A. DATA

Four of our datasets, namely “Small”, “Mid”, “Large” and “Extra Large” were generated by real-life networking devices that operate at Ericsson-ELTE Software Technology Lab. All of the provided datasets are distinct. In the case of these datasets, based on domain knowledge, messages containing the word “error” were considered to be anomalies. To get a more detailed picture of the performance of the proposed algorithm compared to the other investigated methods, the experiments were also carried out on two other datasets “BGL” and “OpenSSH”, which are available at [50].In the case of “BGL” anomalies are indicated with anything other than “−” in the first column. In the case of “OpenSSH”, the “fail” string was used to classify the messages, based on domain knowledge. While log entries from our dataset can belong to 107 possible templates, there are 27 event types regarding “OpenSSH” and 377 in the “BGL” dataset. More detailed information about the datasets can be seen in Table 1.

B. EXPERIMENTAL ANALYSIS

We performed several experiments to measure the prediction performance of our proposed algorithm. The runtime of the

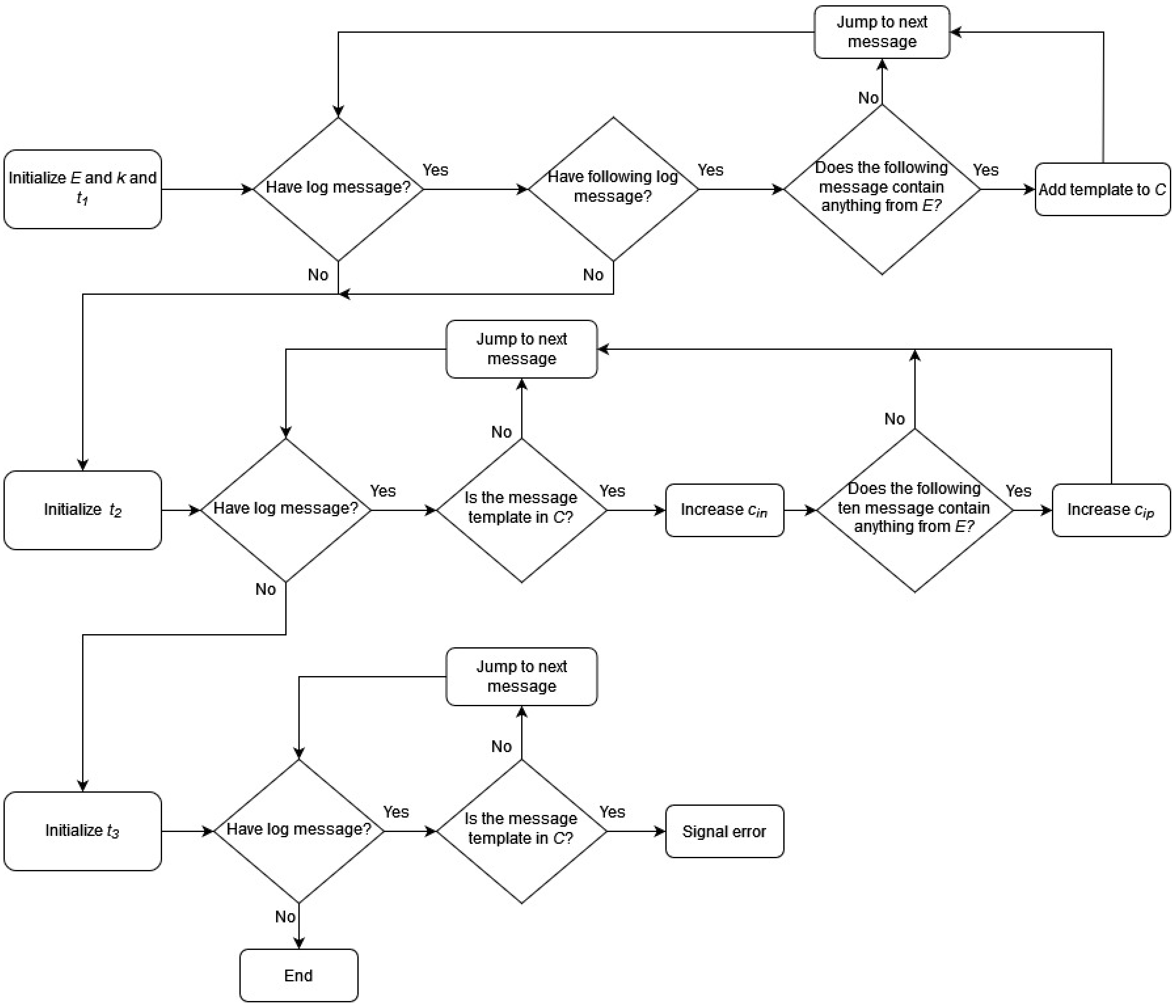

Initialize E and k and t1
Have log message?
Yes
No
Have following log message?
Does the following message contain anything from E?
Add template to C
Jump to next message
Initialize t2
Is the message template in C?
Increase c_in
Does the following ten message contain anything from E?
Increase c_ip
Initialize t3
Signal error
End

FIGURE 3. The flowchart of the proposed algorithm.

TABLE 1. Size of the datasets.

Name	Number of messages	Size in kilobytes	Number of templates
OpenSSH	2,000	220 KB	27
BGL	2,000	310 KB	377
Small	39,139	1,152 KB	107
Mid	124,433	4,607 KB	107
Large	280,002	10,198 KB	107
ExtraLarge	637,369	22,840 KB	107

method was also inspected. In the case of all the datasets, IPLoM [26] was used to retrieve the templates. Since the other algorithms required blocks (a set of lines after each other), we grouped the lines by tens to create blocks. As mentioned before, in the case of our datasets, blocks containing "error" were considered to be anomalies. The "fail" string was associated with anomalies in the case of "OpenSSH". In the case of the "BGL" dataset, every label other than "–" is an anomaly [50]. To acquire our user-defined parameter k, we ran the algorithm on a small subset of our data, for all possible parameters (1-100%), and the one yielding the best result was chosen. In the case of the examined algorithms, the default parameters were used [51], [52]. A summary of the parameters that have been used by the different machine learning algorithms can be seen in Table 2. The "OpenSSH" and "BGL" datasets were also used to evaluate performance in the papers of the investigated methods. The results on these datasets, as well as on our datasets can be seen in Tables 3, 4, 5, 6, 7.

1) EXPERIMENT 1: COMPARING THE PRECISION VALUES ACHIEVED BY THE DIFFERENT ALGORITHMS

First, we investigated the precision values achieved by the different algorithms on the used datasets, to measure the

TABLE 2. The parameters used for the machine learning algorithms in the experiments.

	Hidden size	Directions	Layers	Embedding dimension	Head number	Kernel size
Autoencoder	128	2	2	32	-	-
CNN	128	-	-	32	-	2
LSTM	128	2	2	32	-	-
Transformer	128	-	-	32	2	-

TABLE 3. The precision values achieved by the different algorithms.

	BGL	OpenSSH	Small	Mid	Large	ExtraLarge	Average
Proposed	52,00	100,00	92,06	85,63	100,00	98,74	88,07
Autoencoder	50,00	50,00	86,08	54,22	54,76	60,05	59,19
CNN	60,00	100,00	99,69	99,10	99,72	99,80	93,05
LSTM	58,33	80,00	53,82	49,67	41,33	44,65	54,63
Transformer	60,00	75,00	52,63	37,25	32,66	38,30	49,31
IsolationForest	0,00	0,00	84,50	68,60	79,10	90,90	53,85
LogClustering	27,50	100,00	96,50	89,50	97,80	98,80	85,02
PCA	41,70	0,00	49,10	45,60	55,40	59,30	41,85
SVM	52,20	100,00	98,20	99,80	100,00	100,00	91,70

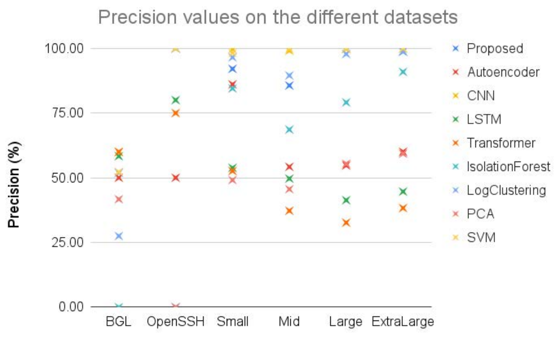


FIGURE 4. Precision values achieved by the different algorithms on the investigated datasets.

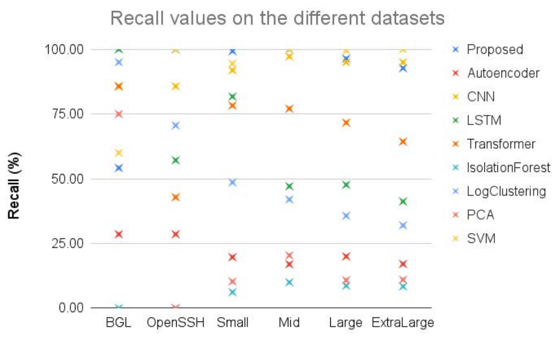


FIGURE 5. Recall values achieved by the different algorithms on the investigated datasets.

number of correct predictions. The results can be seen in Figure 4. The numerical results are shown in Table 3.

It can be seen that the proposed algorithm ranks in the top four in the case of every dataset. On average, the proposed method achieves 88.07% precision and is only outperformed by SVM (91.7%) and CNN (93.05%), while PCA has the worst performance with 41.85% on average.

2) EXPERIMENT 2: COMPARING THE RECALL VALUES ACHIEVED BY THE DIFFERENT ALGORITHMS

To evaluate the algorithm's capability to return most of the relevant results, we also measured the recall values achieved on the different datasets. The results are shown in Figure 5. The numerical results can be seen in Table 4.

In the case of almost all of the investigated datasets, our algorithm has the second-best value. It accomplishes 90.43% on average, slightly behind SVM and CNN that reaching 91.17% and 91, 76% respectively. In the case of recall, Isolation Forest comes the worst with 5.55%.

3) EXPERIMENT 3: COMPARING THE F-SCORE ACHIEVED BY THE DIFFERENT ALGORITHMS

In this experiment, we measure the F-score of the investigated methods, which provides a way to combine both precision and recall into a single measure that captures both properties.

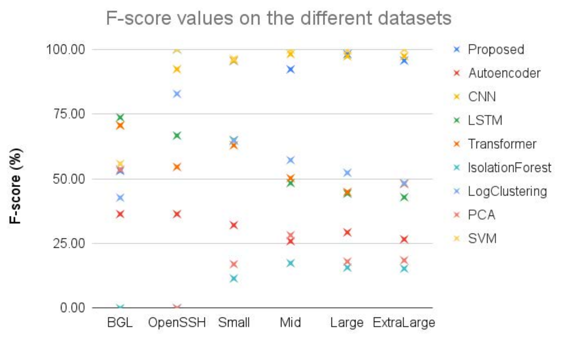


FIGURE 6. F-score values achieved by the different algorithms on the investigated datasets.

The results are shown in Figure 6 and the detailed results can be seen in Table 5.

Based on our experiments, the proposed algorithm is practically among the three best algorithms. It scores an average of 89.11%, somewhat falling behind CNN and SVN with 91.89% and 91.95%, while Isolation Forest comes last with 9.98%.

4) EXPERIMENT 4: COMPARING THE ACCURACY VALUES ACHIEVED BY THE DIFFERENT ALGORITHMS

Accuracy indicates how close or far off a given set of measurements (observations or readings) are to their true value.

TABLE 4. The recall values achieved by the different algorithms.

	BGL	OpenSSH	Small	Mid	Large	ExtraLarge	Average
Proposed	57,14	100,00	99,29	100,00	96,40	92,72	90,43
Autoencoder	28,57	28,57	19,71	17,00	20,00	17,09	21,82
CNN	85,71	85,71	91,88	97,23	95,02	94,99	91,76
LSTM	100,00	57,14	81,74	47,10	47,67	41,23	62,48
Transformer	85,71	42,86	78,26	77,08	71,67	64,35	69,98
IsolationForest	0,00	0,00	6,20	10,00	8,70	8,40	5,55
LogClustering	95,00	70,60	48,60	42,00	35,70	32,00	53,98
PCA	75,00	0,00	10,30	20,40	10,80	11,00	21,25
SVM	60,00	100,00	94,50	100,00	99,50	100,00	91,17

TABLE 5. The f-score values achieved by the different algorithms.

	BGL	OpenSSH	Small	Mid	Large	ExtraLarge	Average
Proposed	53,06	100,00	95,54	92,26	98,17	95,64	89,11
Autoencoder	36,36	36,36	32,08	25,89	29,30	26,61	31,1
CNN	70,59	92,31	95,63	98,16	97,31	97,34	91,89
LSTM	73,68	66,67	64,90	48,35	44,28	42,87	56,79
Transformer	70,59	54,55	62,94	50,23	44,87	48,02	55,20
IsolationForest	0,00	0,00	11,50	17,40	15,70	15,30	9,98
LogClustering	42,70	82,80	64,70	57,20	52,30	48,30	58,00
PCA	53,60	0,00	17,00	28,20	18,00	18,50	22,25
SVM	55,80	100,00	96,30	99,90	99,70	100,00	91,95

TABLE 6. The accuracy values achieved by the different algorithms.

	BGL	OpenSSH	Small	Mid	Large	ExtraLarge	Average
Proposed	92,81	100,00	98,72	98,48	99,82	99,30	98,19
Autoencoder	82,50	82,50	63,17	69,93	73,71	68,84	73,44
CNN	87,50	97,50	96,29	98,83	98,57	98,28	96,16
LSTM	82,50	82,50	55,88	68,09	72,77	66,95	71,44
Transformer	82,50	82,50	55,88	68,09	72,77	66,95	71,44
IsolationForest	80,00	83,20	61,40	70,40	74,00	69,50	73,08
LogClustering	49,00	95,00	78,40	80,30	81,90	77,50	77,01
PCA	74,00	83,20	59,20	67,40	72,80	68,20	70,80
SVM	81,00	100,00	97,00	99,90	99,90	100,00	96,30

TABLE 7. The speed values achieved by the different algorithms in seconds.

	BGL	OpenSSH	Small	Mid	Large	ExtraLarge	Average
Proposed	0,076	0,071	1,242	3,378	9,628	31,956	7,72
Autoencoder	1,345	1,365	11,114	28,048	62,930	265,978	61,79
CNN	0,330	0,700	6,878	11,256	25,824	74,548	19,92
LSTM	2,002	1,764	7,811	23,404	53,328	261,412	58,28
Transformer	1,413	0,750	2,297	17,884	19,832	65,812	17,99
IsolationForest	0,394	0,388	3,082	11,333	20,522	45,193	13,48
LogClustering	0,255	0,259	18,525	40,642	89,944	290,464	73,34
PCA	0,170	0,166	2,101	6,747	19,877	98,798	21,30
SVM	0,157	0,154	2,448	7,354	16,770	50,211	12,84

In this experiment, we measure the achieved accuracy values of the different algorithms. The results can be seen in Figure 7 as well as Table 6.

It can be seen that the proposed algorithm has the best or the second-best score achieved on the given datasets. On average, it has the best value with 98.19% and is followed by SVM which scored 96.30%, and CNN with 96.16%, while PCA is the worst with 70.8%.

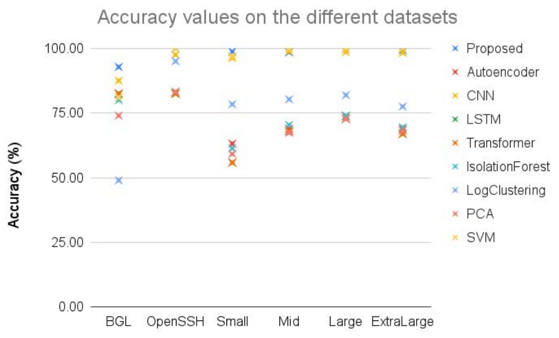


FIGURE 7. Accuracy values achieved by the different algorithms on the investigated datasets.

5) EXPERIMENT 5: COMPARING THE SPEED OF THE DIFFERENT ALGORITHMS

While the precision, recall, f-score, and accuracy values of an algorithm are important, the time an algorithm takes to predict or detect anomalies is also important, especially in the case of large-scale software or healthcare software, where immediate response is vitally important. In this experiment, we measured the run time of the different methods. The results are shown in Figure 8 and Table 7.

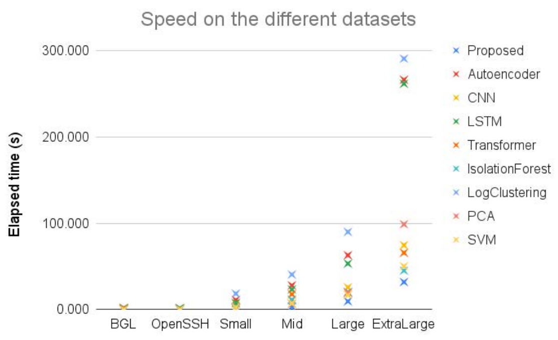


FIGURE 8. **The speed of the different algorithms on the investigated datasets.**

It can be seen that our algorithm is the fastest out of all of the investigated methods. It needs about two times less time to make the predictions/detections than SVM, and about three and a half times less time than CNN. Based on our experiments, the Autoencoder, LSTM, and LogClustering algorithms needed the most time to yield the results.

VI. DISCUSSION AND CONCLUSION

In this paper, we propose a new algorithm that takes the template of the actual log entry into account while predicting if either an anomaly/error will happen, or not. The data is first split into three pieces, 40-40% for the two training phases, and 20% for the testing phase. In the first training phase, the templates that are followed with an error or anomaly are collected into a set, these are called candidate templates. In the second training set, we check if there is an actual error in the next ten template after a candidate template. If a user-defined percentage is reached, we keep the template, otherwise, we discard it from the candidate set.

To evaluate the performance of the new method, we conducted various experiments like measuring the precision, recall, f-score, or accuracy values, as well as the time it takes to make a prediction. The results yielded that the proposed algorithm achieves high values, an average of 88.06% for precision, 90.43% for recall, and 89.11% for f-score. These values are slightly lower (about 1% to 4%) than the values that were achieved by SVM and CNN. In the case of accuracy, our algorithm had the best results with an average of 98.19%. Our algorithm is also faster than all of the other investigated methods, it is about two times faster than SVM and three and a half times faster than CNN.

Thanks to the speed and high accuracy of our method, it can be effectively used in the prevention of software failures in telecommunication networks, based on logs collected from the network nodes, but in various other environments too helping the maintenance of such systems.

We only evaluated the performance of a set of algorithms, in the future, it would be beneficial to compare the performance with other methods. It would be also interesting to investigate if machine learning algorithms used in pair with our algorithm could achieve better results.

REFERENCES

[1] I. W. R. Wijaya and Mudjahidin, "Development of conceptual model to increase customer interest using recommendation system in e-commerce," *Proc. Comput. Sci.*, vol. 197, pp. 727–733, Jan. 2022.

[2] I. S. Y. Kwan, J. Fong, and H. K. Wong, "An e-customer behavior model with online analytical mining for internet marketing planning," *Decis. Support Syst.*, vol. 41, no. 1, pp. 189–204, Nov. 2005.

[3] A. Nguyen, "Using machine learning to predict the low grade risk for students based on log file in moodle learning management system," *Int. J. Comput. Digit. Syst.*, vol. 11, no. 1, pp. 1133–1140, Mar. 2022.

[4] R. M. Alguliyev, F. J. Abdullayeva, and S. S. Ojagverdiyeva, "Log-file analysis to identify internet-addiction in children," *Int. J. Mod. Educ. Comput. Sci.*, vol. 13, no. 5, pp. 1–9, 2021.

[5] D. Basin, M. Gras, S. Krstic, and J. Schneider, "Scalable online monitoring of distributed systems," in *Proc. Int. Conf. Runtime Verification*. Cham, Switzerland: Springer, 2020, pp. 197–220.

[6] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Oct. 2018, pp. 60–70.

[7] J. Kimball, R. A. Lima, and C. Pu, "Finding performance patterns from logs with high confidence," in *Proc. Int. Conf. Web Services*. Cham, Switzerland: Springer, 2020, pp. 164–178.

[8] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surveys*, vol. 54, no. 6, pp. 1–37, Jul. 2021.

[9] 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)*, Honolulu, HI, USA, Jun. 2017, pp. 33–40, doi: 10.1109/ICWS.2017.13.

[10] K. Shima, "Length matters: Clustering system log messages using length of words," 2016, *arXiv:1611.03213*.

[11] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *Proc. 26th Conf. Program Comprehension*, Gothenburg, Sweden, May 2018, pp. 167–177.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002, doi: 10.1109/4235.996017.

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

[14] D. Plaisted and M. Xie, "DIP: A log parser based on 'disagreement index token' conditions," in *Proc. ACM Southeast Conf.*, 2022, pp. 113–122.

[15] I. Yagoub, M. A. Khan, and L. Jiyun, "IT equipment monitoring and analyzing system for forecasting and detecting anomalies in log files utilizing machine learning techniques," in *Proc. Int. Conf. Adv. Big Data, Comput. Data Commun. Syst. (icABCD)*, Aug. 2018, pp. 1–6.

[16] I. A. Khan, D. Pi, Z. U. Khan, Y. Hussain, and A. Nawaz, "HML-IDS: A hybrid-multilevel anomaly prediction approach for intrusion detection in SCADA systems," *IEEE Access*, vol. 7, pp. 89507–89521, 2019.

[17] Y. Harada, Y. Yamagata, O. Mizuno, and E.-H. Choi, "Log-based anomaly detection of CPS using a statistical method," in *Proc. 8th Int. Workshop Empirical Softw. Eng. Pract. (IWESEP)*, Mar. 2017, pp. 1–6.

[18] M. M. Breunig, H. P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2000, pp. 93–104.

[19] M. Ahmed, A. N. Mahmood, and M. R. Islam, "A survey of anomaly detection techniques in financial domain," *Future Gener. Comput. Syst.*, vol. 55, pp. 278–288, Feb. 2016.

[20] T. Pourhabibi, K.-L. Ong, B. H. Kam, and Y. L. Boo, "Fraud detection: A systematic literature review of graph-based anomaly detection approaches," *Decis. Support Syst.*, vol. 133, Jun. 2020, Art. no. 113303.

[21] J. Vanhoeyveld, D. Martens, and B. Peeters, "Value-added tax fraud detection with scalable anomaly detection techniques," *Appl. Soft Comput.*, vol. 86, Jan. 2020, Art. no. 105895.

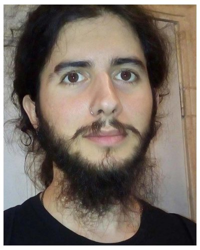
[22] N. Shvetsova, B. Bakker, I. Fedulova, H. Schulz, and D. V. Dylov, "Anomaly detection in medical imaging with deep perceptual autoencoders," *IEEE Access*, vol. 9, pp. 118571–118583, 2021.

[23] T. Fernando, H. Gammulle, S. Denman, S. Sridharan, and C. Fookes, "Deep learning for medical anomaly detection—A survey," *ACM Comput. Surv.*, vol. 54, no. 7, pp. 1–37, 2021.

[24] P. Marjai, P. Lehotay-Kéry, and A. Kiss, "Document similarity for error prediction," *J. Inf. Telecommun.*, vol. 5, no. 4, pp. 407–420, Oct. 2021.

[25] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Toulouse, France, Jun. 2016, pp. 654–661.
[26] 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, doi: 10.1109/TKDE.2011.138
[27] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 413–422.
[28] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proc. 38th Int. Conf. Softw. Eng. Companion*, May 2016, pp. 102–111.
[29] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to Information Retrieval*, vol. 39. Cambridge, U.K.: Cambridge Univ. Press, 2008, pp. 234–265.
[30] J. C. Gower and G. J. Ross, "Minimum spanning trees and single linkage cluster analysis," *J. Roy. Stat. Soc., C Appl. Statist.*, vol. 18, no. 1, pp. 54–64, 1969.
[31] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Largescale system problem detection by mining console logs," in *Proc. SOSP*, 2009, pp. 1–20.
[32] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ. (SOSP)*, 2009, pp. 117–132.
[33] R. Dunia and S. J. Qin, "Multi-dimensional fault diagnosis using a subspace approach," in *Proc. Amer. Control Conf.*, 1997, pp. 1–5.
[34] Y. Zhang and A. Sivasubramaniam, "Failure prediction in IBM BlueGene/L event logs," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Apr. 2008, pp. 583–588.
[35] M. V. Joshi, R. C. Agarwal, and V. Kumar, "Mining needle in a haystack: Classifying rare classes via two-phase rule induction," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2001, pp. 91–102.
[36] M. V. Joshi, R. C. Agarwal, and V. Kumar, "Predicting rare classes: Can boosting make any weak learner strong?" in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2002, pp. 297–306.
[37] A. Farzad and T. A. Gulliver, "Unsupervised log message anomaly detection," *ICT Exp.*, vol. 6, no. 3, pp. 229–237, 2020.
[38] J. L. McClelland and D. E. Rumelhart, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Psychological and Biological Models*, vol. 2. Cambridge, MA, USA: MIT Press, 1987.
[39] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[40] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *Proc. IEEE 16th Intl Conf Dependable, Autonomic Secure Comput., 16th Intl Conf Pervasive Intell. Comput., 4th Intl Conf Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech)*, Aug. 2018, pp. 151–158.
[41] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2012.
[42] 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, vol. 19, no. 7, pp. 4739–4745.
[43] K. A. Nguyen, S. S. I. Walde, and N. T. Vu, "Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction," 2016, *arXiv:1605.07766*.
[44] 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.
[45] 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.
[46] E. Loper and S. Bird, "NLTK: The natural language toolkit," 2002, *arXiv:cs/0205028*.
[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[48] P. Marjai, P. Lehotay-Kéry, and A. Kiss, "The use of template miners and encryption in log message compression," *Computers*, vol. 10, no. 7, p. 83, 2021.
[49] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, May 2019, pp. 121–130.
[50] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," 2020, *arXiv:2008.06448*.
[51] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *Proc. IEEE 27th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2016, pp. 207–218.
[52] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," 2021, *arXiv:2107.05908*.

PÉTER MARJAI received the M.Sc. degree in computer science from the Faculty of Informatics, Eötvös Loránd University, Budapest, in 2021. He is currently pursuing the Ph.D. degree with specialization in information systems. His scientific research interests include centrality measures, log processing, parsing, and compression.



PÉTER LEHOTAY-KÉRY received the M.Sc. degree in computer science from the Faculty of Informatics, Eötvös Loránd University, Budapest, in 2018. He is currently pursuing the Ph.D. degree with specialization in information systems. His scientific research interests include databases, big data, datamining, and bioinformatics.

ATTILA KISS received the Ph.D. degree database theory, in 1991. Seven students received their Ph.D. degrees under his supervision. Since 2010, he has been the Head of the Department of Information Systems, Eötvös Loránd University, Hungary. He is also teaching with J. Selye University, Slovakia. He has more than 190 scientific publications. His research interests include information systems, data mining, and artificial intelligence.