Semi-supervised Log-based Anomaly Detection via Probabilistic Label Estimation

Lin Yang
College of Intelligence and Computing
Tianjin University
Tianjin, China
linyang@tju.edu.cn

Junjie Chen†
College of Intelligence and Computing
Tianjin University
Tianjin, China
junjiechen@tju.edu.cn

Zan Wang†
College of Intelligence and Computing
Tianjin University
Tianjin, China
wangzan@tju.edu.cn

Weijing Wang
College of Intelligence and Computing
Tianjin University
Tianjin, China
wangweijing@tju.edu.cn

Jiajun Jiang
College of Intelligence and Computing
Tianjin University
Tianjin, China
jiangjiajun@tju.edu.cn

Xuyuan Dong
Information and Network Center
Tianjin University
Tianjin, China
dongxuyuan@tju.edu.cn

Wenbin Zhang
Information and Network Center
Tianjin University
Tianjin, China
zhangwenbin@tju.edu.cn

2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) | 978-1-6654-0296-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICSE43902.2021.00130

***Abstract*—With the growth of software systems, logs have become an important data to aid system maintenance. Log-based anomaly detection is one of the most important methods for such purpose, which aims to automatically detect system anomalies via log analysis. However, existing log-based anomaly detection approaches still suffer from practical issues due to either depending on a large amount of manually labeled training data (supervised approaches) or unsatisfactory performance without learning the knowledge on historical anomalies (unsupervised and semi-supervised approaches).**

In this paper, we propose a novel practical log-based anomaly detection approach, PLELog, which is semi-supervised to get rid of time-consuming manual labeling and incorporates the knowledge on historical anomalies via probabilistic label estimation to bring supervised approaches' superiority into play. In addition, PLELog is able to stay immune to unstable log data via semantic embedding and detect anomalies efficiently and effectively by designing an attention-based GRU neural network. We evaluated PLELog on two most widely-used public datasets, and the results demonstrate the effectiveness of PLELog, significantly outperforming the compared approaches with an average of 181.6% improvement in terms of F1-score. In particular, PLELog has been applied to two real-world systems from our university and a large corporation, further demonstrating its practicability.

***Index Terms*—Log Analysis, Anomaly Detection, Deep Learning, Probabilistic Estimation, Label**

I. Introduction

Over the years, software systems become much larger and more complex, which largely aggravates the difficulty of maintaining them [1]–[7]. As presented in the existing work [8]–[17], logs have become important data for maintaining large-scale software systems (e.g., online service systems [18]), which are produced during the running of systems in order to record events and states of interest. Through examining recorded logs, developers can check system status, detect anomalies, and diagnose root causes. However, due to the large scale of systems, log data are massive, and thus manual examination for logs is very difficult, even infeasible. Therefore, there are a large amount of work focusing on automated log analysis in the literature [11], [19]–[25].

Log-based anomaly detection is one of the most important aspects in automated log analysis, which aims to automatically detect system anomalies based on logs [26]–[34]. Almost all the existing log-based anomaly detection approaches share the same high-level steps: 1) extracting log events (i.e., the templates of log messages) and log sequences (i.e., series of log events that record specific execution flows) from log messages, and 2) building an anomaly detection model through a machine learning or data mining technique based on log sequences. According to the used machine learning or data mining techniques, these existing approaches include supervised approaches (e.g., LogRobust [18]), unsupervised approaches (e.g., LogCluster [30]), and semi-supervised approaches (e.g., LogAnomaly [32]). Although they have been demonstrated to be effective in their corresponding studies, these existing approaches still suffer from the following practical issues:

- Supervised approaches are the most effective but rely on a large amount of training data, which contain both pos-

†Corresponding author.

978-1-6654-0296-5/21/$31.00 ©2021 IEEE
DOI 10.1109/ICSE43902.2021.00130

itive instances (i.e., anomalous log sequences) and negative instances (i.e., normal log sequences). In practice, it is easy to obtain normal log sequences, since when a system is normally running without any alerts, all the produced logs could be regarded as normal ones. However, identifying anomalous log sequence is very difficult, since when anomalies occur in a system, both anomalous and normal log sequences can be produced and mixed together. Since log data are massive and hard-to-understand, manual labeling is very time-consuming and expensive, and thus such supervised approaches are actually not practical.

- Unsupervised and semi-supervised (only knowing the labels of a set of normal log sequences) approaches get rid of the limitation of supervised approaches by using only part of the normal log sequences for training. As a result, their effectiveness tends to be worse than the latter due to lack of the knowledge on historical anomalies. Moreover, as presented in the existing work [18], [35], log data are unstable due to frequent modification of log statements in source code in practice, causing that some incoming log events or log sequences do not appear in the training data. Due to the neglect of unstable log data, the effectiveness of existing unsupervised and semi-supervised approaches can drop largely when coming across unseen log events or log sequences in practice.

Therefore, it is still desired to propose a more practical log-based anomaly detection approach.

Although both supervised approaches and unsupervised and semi-supervised approaches suffer from practical issues, both of them actually have complementary strengths. More specifically, the former has better effectiveness due to incorporating the knowledge on historical anomalies while the latter gets rid of time-consuming manual labeling. In particular, in this paper we propose a more practical log-based anomaly detection approach **PLELog**, which combines the above strengths via **P**robability **L**abel **E**stimation. To get rid of time-consuming manual labeling, PLELog is a semi-supervised approach only knowing the labels of a set of normal log sequences, which are obtained from systems when they normally run without any alerts. To incorporate the knowledge on historical anomalies, PLELog uses the idea of PU learning (positive-unlabeled learning) [36] for reference, which utilizes known positive instances to estimate the labels of unlabeled instances for subsequent model building. In our problem, a set of negative instances (normal log sequences) are known and PLELog aims to estimate the labels of a set of mixed anomalous and normal log sequences.

In PLELog, there are two major challenges. The first one is how to estimate the labels of unlabeled log sequences based on known normal log sequences. To overcome this challenge, PLELog adopts the clustering method (i.e., HDBSCAN [37]) to divide all log sequences in the training set into several groups, and the log sequences in the same group tend to share the same label. Therefore, according to whether a group contains known normal log sequences, PLELog preliminarily estimates the labels of unlabeled log sequences. However, it is hard to produce perfect clustering results, and thus the second challenge is how to reduce the influence of noise incurred by clustering. To overcome this challenge, instead of assigning a deterministic label, PLELog measures the probability that an unlabeled log sequence belongs to each label based on the clustering results (i.e., the uncertainty that the unlabeled log sequence is divided into the corresponding group) and then assigns a probabilistic label to it. In this way, noise can be assigned less confident labels to reduce their influence to some degree. Based on probabilistic label estimation, a labeled training set is obtained, and then PLELog builds an anomaly detection model through supervised machine learning, which can bring supervised approaches' superiority into play. Besides, to make PLELog more practical, it is required to perform well for unstable log data and detect anomalies efficiently and effectively. Here, PLELog incorporates semantic embedding for log sequences to satisfy the first criterion following the existing work [18]. To satisfy the second criterion, we design an attention-based GRU (Gate Recurrent Unit) [38] neural network for anomaly detection model building in PLELog.

To evaluate the performance of PLELog, we conducted an empirical study based on two most widely-used public datasets (i.e., HDFS [27] and BGL [39]) following the existing work [19]. In particular, to make our study closer to the practical scenario, we guaranteed that all log data in the training set are produced before those in the test set, which is ignored by almost all the existing studies and thus leads to the absence of unstable log data in their studies. Our experimental results demonstrate that PLELog significantly outperforms the state-of-the-art unsupervised and semi-supervised log-based anomaly detection approaches with the improvements of 3.8%$\sim$690.9% on HDFS and 30.2%$\sim$332.6% on BGL in terms of F1-score. Also, our results confirm the contribution of each main component in PLELog. In particular, PLELog has been successfully applied to two large-scale real-world systems from two different organizations, i.e., the network center of our university and one influential motor corporation throughout the world (we hide its name due to the company policy). In our work, we call them $\mathcal{A}$ and $\mathcal{B}$. During the practical evaluation, PLELog achieves 0.947 and 0.984 F1-score for $\mathcal{A}$ and $\mathcal{B}$ respectively, significantly outperforming the state-of-the-art unsupervised and semi-supervised approaches with the improvements of 8.1%$\sim$234.6% on $\mathcal{A}$ and 7.0%$\sim$18.6% on $\mathcal{B}$. The results further demonstrate the performance of PLELog in practice.

Our work makes the following major contributions:

- We propose a practical and robust log-based anomaly detection approach PLELog, which is semi-supervised and incorporates the knowledge on historical anomalies via probabilistic label estimation. Also, PLELog is able to stay immune to unstable log data via semantic embedding and detect anomalies efficiently and effectively by designing an attention-based GRU neural network.
- We propose an effective method to estimate labels of unlabeled log sequences based on known normal log sequences through clustering. In particular, we design probabilistic

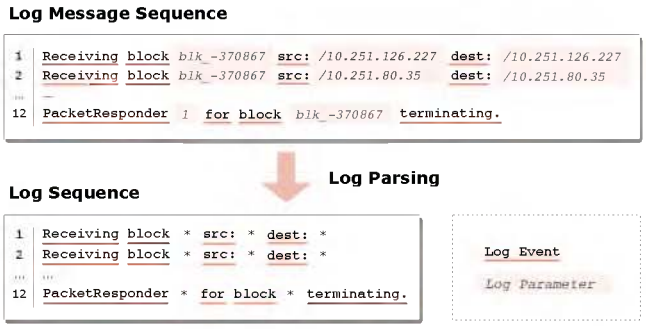


Fig. 1: An illustrating example for log terminology

label estimation to reduce the influence of noise incurred by clustering.

- We conduct an empirical study based on two most widely-used public datasets, demonstrating the high performance of PLELog. In particular, PLELog has been applied to two real-world systems in our university and a large motor corporation, further confirming the practicability of PLELog.

II. Log Terminology

In this section, we introduce log terminology used in our paper based on an illustrating example shown in Figure 1. The example of log data is selected from the public HDFS dataset [27]. From this figure, a **log message** is a raw unstructured sentence generated during system running, which records system status of the time. A log message (e.g., *PacketResponder 1 for block blk_-370867 terminating*) consists of a **log event** (e.g., *PacketResponder * for block * terminating*) and **log parameters** (e.g., *1*, *blk_-370867*). The former is a constant part in a log message, which is the template of a log message designed by developers. The latter is a variable part, which records some system attributes (e.g., IP address and block id). Log events can be extracted from log messages via **log parsing**, which is the first step of log-based anomaly detection and has been widely studied in the literature [9], [18], [19], [29]–[32], [40], [41]. A series of log messages form a **log message sequence**. A **log sequence** is a series of log events that record a specific execution flow, which can be obtained by the task ID of each log message or some strategies like sliding window. If a log sequence indicates system anomalies, it is an **anomalous log sequence**; otherwise, it is a **normal log sequence**.

III. Approach

A. Overview

In this paper, we propose a more practical log-based anomaly detection approach, called **PLELog**, by combining the strengths of unsupervised/semi-supervised approaches (getting rid of time-consuming manual labeling) and supervised approaches (incorporating the knowledge on historical anomalies). More specifically, PLELog is a semi-supervised approach only knowing the labels of a part of normal log sequences in a training set, which are easy-to-obtain as presented in Section I. Also, PLELog uses the idea of PU learning [36] for reference to incorporate the knowledge on historical anomalies, where PLELog estimates the labels of unlabeled log sequences, mixing both normal and anomalous ones in the training set, based on known normal log sequences via probabilistic estimation. Finally, the estimated labeled data will be used for subsequent supervised anomaly detection model building.

Figure 2 shows the overview of PLELog. PLELog consists of three stages, i.e., semantic embedding, probabilistic label estimation, and anomaly detection model building. As presented in Section I, log evolution is frequent in practice, and thus it is required for a practical log-based anomaly detection approach to be immune to such unstable log data. To achieve this goal, following the existing work [18], PLELog incorporates the semantic information of each log event during vector representation, which is called semantic embedding and is the first stage in PLELog (Section III-B). The second stage in PLELog is probabilistic label estimation (Section III-C), which is the core of PLELog. It solves a major challenge in our problem (i.e., how to estimate the labels of unlabeled log sequences in a training set based on known normal log sequences) through clustering. Further, it solves a follow-up challenge (i.e., how to reduce the influence of noise incurred by clustering) through measuring the probability that an unlabeled log sequence belongs to each label as the estimated label of the log sequence instead of assigning a deterministic label. Based on the training set including known normal log sequences and estimated normal and anomalous log sequences, PLELog builds an anomaly detection model through supervised deep learning, which is the third stage in PLELog (Section III-D). Besides the prediction effectiveness, it is required for a practical log-based anomaly detection approach to predict an incoming log sequence efficiently, since log-based anomaly detection is expected to monitor a system in real time. Therefore, we design an attention-based GRU neural network for anomaly detection model building by considering both prediction effectiveness and efficiency. In particular, our attention-based GRU neural network could further reduce the influence of noise incurred by clustering by well handling temporal relations in log sequences. Finally, we present the usage of PLELog in Section III-E.

B. Semantic Embedding

Instead of the widely-used one-hot representation [31], PLELog transforms a log event to a vector (called *semantic vector*) by extracting its semantic information in order to deal with unstable log data well. It consists of three steps: log parsing, word embedding, and TF-IDF based aggregation.

1) Log Parsing: As shown in Figure 1, raw log messages are unstructured data and contain variable log parameters, which could hinder automated log analysis [42], [43]. Therefore, following the practice of log-based anomaly detection [18], PLELog extracts log events from log messages via log parsing since log events are structured, which facilitates

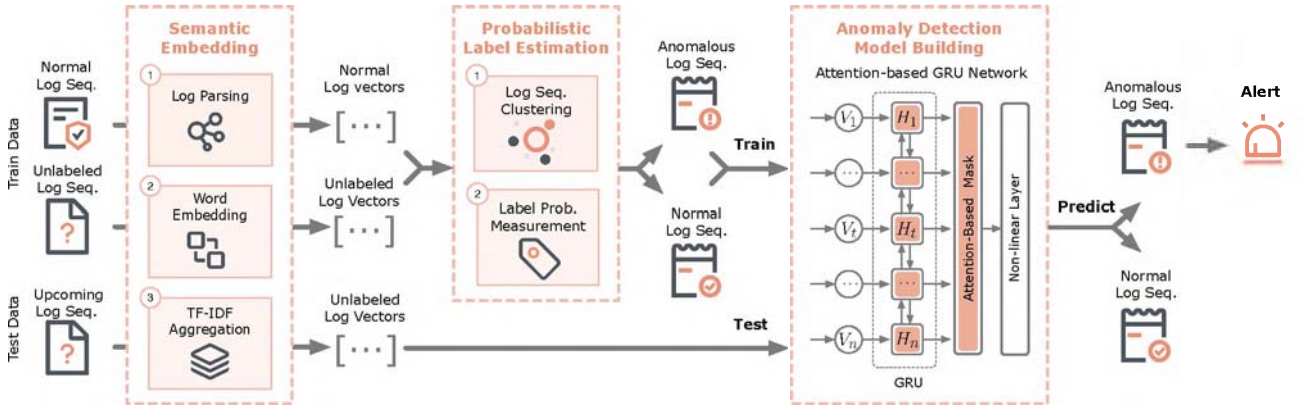


Fig. 2: Overview of PLELog

log analysis. In particular, PLELog adopts the state-of-the-art method Drain [42] for log parsing since it has been demonstrated to be very effective and efficient in the existing study [44].

2) Word Embedding: To extract semantic information of log events, PLELog regards each log event as a sentence in natural language. Since log events are designed by developers to record the running status of a system, most tokens in a log event are English words, which have their own semantics. Also, there are non-character tokens (e.g., delimiters, operators, punctuation marks, and number digits) and composite tokens that are concatenations of words (e.g., *NullPointerException*) due to the programming practice. Therefore, following the existing work [18], PLELog first pre-processes log events by removing non-character tokens and stop words and splitting composite tokens into individual words according to Camel Case [45]. Then, PLELog adopts the pre-trained word vectors based on Common Crawl Corpus using the FastText algorithm [46] (which can effectively capture the intrinsic relationship among words in natural language), to extract the semantic information from each word in a processed log event. That is, PLELog adopts the pre-trained word2vec model to transform each word in a processed log event to a d-dimension vector (denoted as v), where d is 300 in FastText word vectors.

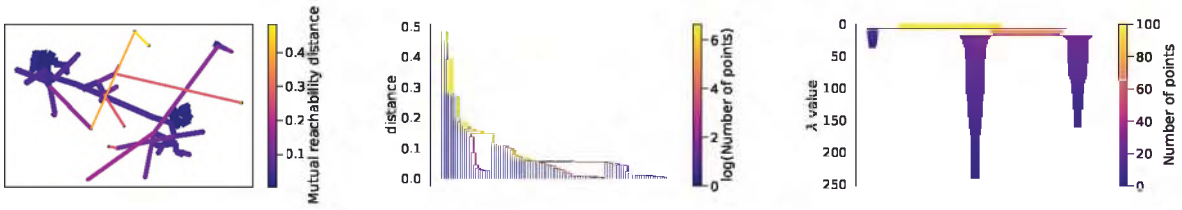
3) TF-IDF based Aggregation: After transforming a word to a d-dimension vector via word embedding, PLELog further transforms a log event to a semantic vector by aggregating all word vectors in the log event. Here, PLELog adopts *TF-IDF* [47], a widely-used method in information retrieval, for aggregation with considering the importance of each word, which can be effectively measured by *TF-IDF*. *TF* (Term Frequency) measures how frequently a word w occurs in a log event, which is calculated by $TF(w) = \frac{\#w}{\#N}$ where $\#w$ is the number of the occurrence of w in the log event and $\#N$ is the total number of words in the log event. *IDF* (Inverse Document Frequency) measures how common or rare a word w is in all log events, which is calculated by $IDF(w) = log(\frac{\#L}{\#L_w})$ where $\#L$ is the total number of log events and $\#L_w$ is the number of log events containing w. The weight (denoted as ω) of a word can be calculated by $TF \times IDF$. Finally, the semantic vector (denoted V) of a log event can be produced by summing up all word vectors in the log event with *TF-IDF* weights as $V = \frac{1}{N}\sum_{i=1}^{N} \omega_i \cdot v_i$.

In this way, PLELog represents log events as semantic vectors, which effectively incorporates their semantic information, facilitating to identify semantically similar log events and distinguish different log events. Therefore, PLELog is able to be immune to unstable log data to some degree.

C. Probabilistic Label Estimation

After semantic embedding, PLELog further estimates the labels of unlabeled log sequences in the training set based on known normal log sequences by using the idea of PU learning for reference, so that the strength of supervised approaches can be incorporated. Intuitively, log sequences with similar semantics are more likely to share the same label. With this intuition, PLELog first adopts advanced clustering to identify the log sequences with similar semantics to the same group. However, it is hard to produce perfect clustering results, and thus instead of assigning a deterministic label, PLELog assigns a probabilistic label for each unlabeled log sequence by measuring the probability that an unlabeled log sequence belongs to each label in order to reduce the influence of noise incurred by clustering. In the following, we present log sequence clustering in Section III-C1 and label probability measurement in Section III-C2 in details.

1) Log Sequence Clustering: As a log sequence contains a series of semantic vectors, to enable the clustering of log sequences, PLELog produces a vector for each log sequence (called *log-sequence semantic vector*) by aggregating all semantic vectors in the log sequence via summation. Here, PLELog adopts the HDBSCAN algorithm (Hierarchical Density-Based Spatial Clustering of Application with Noise) [48] to cluster all log sequences (including known normal log sequences and unlabeled log sequences) in the training set to different groups, each of which is more likely to contain the log sequences with similar semantics. The reasons why choosing HDBSCAN are threefold: (1) It is hard to know



(a) Minimum spanning tree (b) Connected component hierarchy (c) Condensed tree

Fig. 3: Visualization of main stages in HDBSCAN clustering by sampling 100 log sequences in the HDFS dataset [27]

the number of clusters in advance, and thus the clustering algorithms required to pre-define the number of clusters cannot be applicable, e.g., the widely-used K-means algorithm [49]. HDBSCAN is a density-based clustering algorithm, which groups the data points that are closely packed together, and thus it does not need to pre-define the number of clusters. (2) HDBSCAN has been demonstrated to be effective and efficient and has been widely used in many domains [50]–[53]. (3) HDBSCAN has a small number of parameters and is robust to parameter selection [37], and thus it is more easy-to-use in practice.

To clearly illustrate the clustering process for log sequences via HDBSCAN, we made visualization for main steps by sampling 100 log sequences in the HDFS dataset [27], which is shown in Figure 3. First, it builds a weighted graph, in which vertices are the data points (log-sequence semantic vectors) and weights of edges between vertices are the corresponding mutual reachability distances [48]. Second, it builds the minimum spanning tree of the weighted graph via Prim's algorithm [54]. According to the minimum spanning tree, a hierarchy of connected components is constructed by ranking tree edges in the ascending order of distances and iterating via creating a new merged cluster for each edge. Then, it condenses down the large cluster hierarchy into a smaller tree based on the defined minimum cluster size (a parameter in HDBSCAN). Finally, it extracts the stable clusters from the condensed tree by calculating the stability score of each cluster. In this way, log sequences in the training set are divided into several groups in order to identify the log sequences with similar semantics.

In particular, with the number of data points and the vector dimension increasing, HDBSCAN could become less efficient and thus it tends to be used together with dimension reduction methods in the literature [50]. Therefore, PLELog also conducts dimension reduction before clustering in order to make the clustering process efficient. Here, PLELog adopts the FastICA algorithm [55], which aims to find independent components and facilitate to find underlying factors by maximizing negative entropy [56] for dimension reduction following the existing work using HDBSCAN [50].

2) Label Probability Measurement: Through clustering, log sequences with similar semantics are divided to the same group, indicating they share the same label. Thus, according to whether there exist known normal log sequences in a group or not, PLELog preliminarily estimates the labels of the log sequences in the group. That is, if a group contains known normal log sequences, the log sequences in the group are more likely to have the same label with them (i.e., normal log sequences); Otherwise, they are more likely to be anomalous. However, it is hard to produce perfect clustering results, thus if we directly label each unlabeled log sequence as normal or anomalous based on preliminary estimation, the noise incurred by clustering could undermine the effectiveness of PLELog.

Instead of assigning a deterministic label (1 representing "anomalous", 0 representing "normal"), PLELog assigns a probabilistic label to each unlabeled log sequence by measuring the probability that the unlabeled log sequence belongs to each label to reduce the influence of noise. In this way, even though PLELog has preliminarily estimated a wrong label for a noisy log sequence in a group, PLELog can make the preliminarily estimated label less confident through such probabilistic labeling. Therefore, during the model building process with our attention-based GRU neural network (Section III-D), the importance of noisy log sequences can be effectively reduced so that the influence of noise can be also effectively reduced.

More specifically, PLELog measures the probability that an unlabeled log sequence belongs to each label based on clustering results. With HDBSCAN, each log sequence in a group is assigned a score, which reflects the uncertainty that the log sequence belongs to the group. The score ranges from 0 to 1, the closer to 0 the score is, the more confidence can be gained to cluster a log sequence to the corresponding group. Even though the uncertainty may be large for a log sequence in a group, it is still more confident for dividing it to this group compared with dividing it to other groups, and thus making a label probabilistic is required to meet the major premise of the preliminarily estimated label. Based on this major premise and the scores produced by HDBSCAN clustering, we transform each preliminarily estimated label to a probabilistic label as $P(anomalous) = 1 - \frac{score}{2}$ and $P(normal) = \frac{score}{2}$.

D. Anomaly Detection Model Building

Based on the set of labeled training data through probabilistic label estimation, we design an attention-based GRU neural network (shown in Figure 2) to build an effective and efficient anomaly detection model.

1) GRU Neural Network: GRU, a kind of Recurrent Neural Network (RNN), has been demonstrated to be effective to

handle temporal relations in sequential data [57], [58]. In our problem, a log sequence consists of a series of log events that are continuously produced during system running and closely related within a short period. Therefore, we employ GRU to build an anomaly detection model in PLELog. A typical GRU includes an *update gate* to decide how much information from a previous state passing to a new state, and a *reset gate* to decide how much information from the past should be forgotten. That is, both gates together determine the influence of past log events in a log sequence on the future.

For a log sequence $S = \{e_1, e_2, \ldots, e_n\}$, where e_t $(1 \leq t \leq n)$ is the t^{th} log event and n is the number of log events in S, the input of GRU at the t^{th} time step is the semantic vector of e_t denoted as V_t. According to the hidden state H_{t-1} at the $(t-1)^{th}$ time step, the *update gate* z_t and *reset gate* r_t at the t^{th} time step can be calculated by Formula 1.

$$\begin{aligned} z_t &= \sigma(W_z V_t + U_z H_{t-1}) \\ r_t &= \sigma(W_r V_t + U_r H_{t-1}) \end{aligned} \tag{1}$$

where σ is the logistic sigmoid function, and W_z, W_r, U_z and U_r are network parameters. Then, the hidden state H_t at the t^{th} time step can be calculated based on H_{t-1} by Formula 2.

$$\begin{aligned} H_t &= z_t H_{t-1} + (1 - z_t)\tilde{H}_t \\ \tilde{H}_t &= tanh(W V_t + U(r_t \odot H_{t-1})) \end{aligned} \tag{2}$$

where W and U are also network parameters, and $\odot$ is an element-wise multiplication. In particular, before the startup time step (i.e., $t = 0$), no log event comes and thus the hidden state H_0 is a zero vector. All network parameters can be learned during training.

2) Attention-Based Mask Strategy: The output of GRU at the t^{th} time step is mainly decided by the current log event (i.e., V_t) and the last hidden state H_{t-1}. When a log sequence comprises too many log events, it could suffer from the long dependency issue, which could be harmful to the anomaly detection effectiveness. Moreover, the noisy log events in a log sequence can also undermine the anomaly detection effectiveness. To tackle them, PLELog further incorporates an attention-based mask strategy in our GRU neural network.

Intuitively, a log event that has larger correlation with the anomaly detection result, is more important in a log sequence. That is, the larger the correlation between the hidden state H_i (corresponding to a log event e_i at the i^{th} time step) and the anomaly detection result, the larger the weight of H_i, indicating that e_i is possibly an indicator of anomalies. In this way, important log events can be highlighted by assigning them larger weights while noisy log events can be masked by assigning them smaller weights. Here, we assemble the attention vector H_t^A at the t^{th} time step by $H_t^A = \sum_{i=1}^{t} \lambda_{ti} H_i$, where λ_{ti} is the learned weight of H_i at the t^{th} time step and λ_{ti} is calculated by *softmax* function [18].

Finally, PLELog takes H_n^A as the input of GRU and leverages a non-linear layer (Formula 3) to predict the final result for a log sequence, which employs tanh as the activation function. W_n is a weight matrix learned during training.

$$P(normal, anomalous) = \tanh(W_n H_n^A) \tag{3}$$

E. Usage of PLELog

After building an anomaly detection model through PLELog, the model can be deployed to monitor the system in real time. When there is an incoming log message sequence, PLELog first conducts log parsing and semantic embedding to obtain semantic vectors, and then the model can predict whether it is anomalous or normal. If an anomaly is detected, an alert would be timely produced and sent to the operators in order to start the process of mitigation and diagnosis earlier.

IV. Evaluation

In our study, we address the following research questions:

- **RQ1**: How does PLELog perform in terms of effectiveness?
- **RQ2**: Does each main component contribute to PLELog?
- **RQ3**: How does different PLELog configurations impact the effectiveness of PLELog?
- **RQ4**: How does PLELog perform in terms of efficiency?

A. Datasets

In our study, we used two most widely-used public datasets to evaluate PLELog, i.e., HDFS (Hadoop Distributed File System) dataset [27] and BGL (Blue Gene/L supercomputer) dataset [39], which have been widely used in the existing work on log-based anomaly detection [18], [26], [31], [32], [59]–[61]. For ease of presentation, we call them *HDFS* and *BGL* directly.

HDFS contains 11,175,629 log messages produced through running Hadoop-based MapReduce jobs on more than 2,000 Amazon's EC2 nodes for 38.7 hours. According to *block_id* in its log messages, log sequences can be directly extracted. In total, there are 575,062 log sequences in HDFS, among which about 2.9% log sequences indicate system anomalies that were manually labeled by Hadoop domain experts.

BGL contains 4,747,963 log messages that were produced by the Blue Gene/L supercomputer, which consists of 128K processors and was deployed at the Lawrence Livermore National Laboratory, with a time span of 7 months. Each log message in BGL was manually labeled to be either anomalous or normal by BGL domain experts. In total, 348,460 log messages are anomalous. Unlike HDFS, BGL does not have obvious tags such as *block_id* to help extract log sequences, and thus similar to the existing work [31], [32] we extracted log sequences through the strategy of splitting windows with the size of 120. In particular, during log sequence extraction, we ranked all log messages according to their generation timestamps and considered the node on which each log message was produced. If a log sequence contains at least one anomalous log message, it is regarded as an anomalous log sequence. Finally, we obtained 49,274 normal log sequences and 36,303 anomalous log sequences in BGL.

Similar to the existing work [18], [23], [26], [30]–[32], we also split each dataset into a training set, a validation set for parameter tuning, and a test set with the ratio of 6:1:3, to evaluate the performance of a log-based anomaly detection approach. However, the splitting methods used in the existing work [18], [31], [32] shuffle all log sequences before splitting,

which can avoid the occurrence of unstable log data. Different from them, we split a dataset *in chronological order of log sequences* so as to guarantee that all log sequences in the training set are produced before the log sequences in the test set, which is much closer to the practical scenario. Also, the influence of dataset splitting methods has been investigated in the area of predicting software development practices [62] and it suggests to split a dataset in chronological order to avoid data leakage. In particular, with our splitting method, unstable log data indeed occur especially on BGL, since its time span of log data is longer. To evaluate our semi-supervised approach PLELog, we sampled 50% of training data as *known normal log sequences* and the remaining log sequences in the training data as *unlabeled log sequences*, to simulate the semi-supervised scenario.

B. Compared Approaches

1) Existing Log-based Anomaly Detection Approaches: As PLELog is a semi-supervised approach, we compared PLELog with the state-of-the-art semi-supervised and unsupervised log-based anomaly detection approaches:

- **Deeplog** [31] treats a log sequence as a natural language sequence and adopts a deep neural network (LSTM) to learn normal log patterns from normal log sequences. During anomaly detection, Deeplog predicts the next log event, and if the real log event is not included in the top prediction results, an anomaly is regarded to be detected.
- **LogAnomaly** [32] is similar to Deeplog to learn normal log patterns from normal log sequences. The main difference is that the former represents a log sequence by considering semantic information instead of one-hot representation used in the latter in order to improve prediction effectiveness. Moreover, LogAnomaly also counts log events during representation in order to detect the anomalies reflected by anomalous log event numbers.
- **LogCluster** [30] first represents a log sequence by considering the weights of log events, and then adopts the Agglomerative Hierarchical clustering algorithm [63] to cluster log sequences. In each cluster, it selects its centroid as the representative log sequence. For an incoming log sequence, it identifies whether it is normal or anomalous by measuring the distances between the incoming log sequence and all the representative log sequences.
- **PCA** (Principal Component Analysis) is a popular algorithm for dimension reduction [64]. For its application on log-based anomaly detection, it first represents a log sequence by counting log events, and then projects log sequences into two spaces, i.e. normal space and anomalous space, by considering the first k principal components and remaining principal components. Then, for an incoming log sequence, it can be identified to be normal or anomalous according to the space the log sequence belonging to after projection.

Besides, PLELog tries to incorporate the strength of supervised approaches via probabilistic label estimation, and thus it is interesting to investigate how much the gap between PLELog and the state-of-the-art supervised log-based anomaly detection approach, i.e., **LogRobust** [18]. Thus, we also compared PLELog with LogRobust, which builds a classifier via LSTM based on the manually *labeled* training set after representing a log sequence by considering semantic information.

2) Variants of PLELog: In RQ2, we aim to investigate the contributions of three main components to PLELog, including making label estimation probabilistic, incorporating an attention mechanism, and reducing semantic-vector dimension. Thus, we constructed three variants of PLELog accordingly:

- **PLELog**$_{noP}$ removes the component of making label estimation probabilistic from PLELog. That is, it directly assigns a deterministic label to each unlabeled log sequence based on the clustered groups.
- **PLELog**$_{noA}$ removes the component of incorporating an attention mechanism from PLELog. That is, it uses a GRU neural network without the attention mechanism to build an anomaly detection model based on the training set with our estimated probabilistic labels.
- **PLELog**$_{noR}$ removes the component of reducing semantic-vector dimension from PLELog. That is, it directly uses the original semantic vectors for the subsequent steps after semantic embedding.

3) Different PLELog Configurations: To answer RQ3, we investigated the impact of three main parameters in PLELog, including the number of GRU layers, the size of GRU hidden states, and the number of components in FastICA. Regarding the number of GRU layers, we studied 1, 2, 3, 4, 5. Regarding the size of GRU hidden states, we studied 50, 100, 150, 200, 250, 300. Regarding the number of components in FastICA, we studied 50, 100, 150, 200, 250.

C. Implementations and Environments

We implemented PLELog based on Python 3.8.3 and Pytorch 1.5.1 [65]. We adopted the implementations of HDBSCAN and FastICA provided by hdbscan 0.8.26 [66] and sklearn [67] respectively. For parameters in PLELog, we determined them through grid search based on validation sets. More specifically, we set the size of GRU hidden states to be 100, the number of GRU layers to be 1, learning rate to be 0.002, the number of components in FastICA to be 100, the number of epochs to be 20. In particular, we investigated the impact of parameter settings on PLELog in RQ3. Also, we set *min_cluster_size* to be 100 and *min_samples* to be 100 in HDBSCAN. As demonstrated by the existing work [37], [50], HDBSCAN is robust to parameter selection.

Regarding the existing log-based anomaly detection approaches, we adopts their public implementations [68], [69]. In particular, we determined their parameters by first reproducing the results in their corresponding studies [18], [31], [32] and further conducting grid search based on validation sets, in order to obtain the best parameter settings for them.

We conducted all the experiments on a Linux server with Intel(R) Xeon(R) Silver 4214 2.20GHz CPU, 128GB memory, RTX2080Ti with 11GB GPU memory and operating system

version is Ubuntu 18.04. In particular, our tool and experimental data are available in our project homepage[1].

D. Measurements

Log-based anomaly detection is actually a binary classification problem, and thus following the existing work [18], [31], [32], we adopted *Precision*, *Recall*, and *F1-score* to measure the effectiveness of log-based anomaly detection approaches. *Precision* is compuited by $\frac{TP}{TP+FP}$ while *Recall* is computed by $\frac{TP}{TP+FN}$, where *TP*, *FP*, and *FN* refer to the number of true positives (a log sequence is predicted to be *anomalous* and its ground truth is also *anomalous*), false positives (a log sequence is predicted to be *anomalous* but its ground truth is *normal*), and false negatives (a log sequence is predicted to be *normal* but its ground truth is *anomalous*), respectively. *F1-score* considers both *Precision* and *Recall*, which is calculated by $\frac{2\cdot(Precision\cdot Recall)}{Precision+Recall}$. Besides, for LogRobust and PLELog, the effectiveness of their built classifiers could be affected by the threshold used to distinguish the two classes, and thus it is also necessary to measure their effectiveness under different thresholds. Therefore, we drew the Receiver Operating Characteristic (*ROC*) [70] curve with thresholds from 0 to 1 with the step of 0.1 and then calculated its Area Under Curve (*AUC*) value. The larger the *AUC* value is, the better effectiveness the classifier has.

Besides, a log-based anomaly detection approach is deployed to monitor systems in real time, and thus the time spent on online predicting a log sequence is also important. Therefore, we recorded the time spent on *online* predicting a log sequence to measure the efficiency of a log-based anomaly detection approach. We also recorded the time spent on *offline* building an anomaly detection model based on a training set for each approach. We call the former *prediction time* and the latter *training time*.

E. Results and Analysis

1) Effectiveness of PLELog: Table I shows the effectiveness comparison results among all the studied approaches in terms of Precision, Recall, and F1-score. We found that PLELog is able to perform well on both HDFS and BGL in terms of all the three metrics. For example, F1-score of PLELog is 0.957 and 0.982 on HDFS and BGL, respectively. The results demonstrate the effectiveness of PLELog.

Comparison with existing semi-supervised and unsupervised approaches. From Table I, PLELog largely outperforms all the compared semi-supervised and unsupervised approaches on both HDFS and BGL in terms of all the three metrics (except LogCluster in Precision). Even though LogCluster achieves better Precision, its Recall and F1-score are significantly worse than those of PLELog. In particular, for anomaly detection, Recall is a much more important metric than Precision since missing to detect anomalies could lead to huge economic loss and other serious consequences. In terms of F1-score, the improvements of PLELog over all the compared semi-supervised and unsupervised approaches range from 3.8% to 690.9% on HDFS and from 30.2% to 332.6% on BGL.

[1]https://github.com/YangLin-George/PLELog

TABLE I: Experiment results of studied approaches on HDFS and BGL

Dataset	Method	Precision	Recall	F1-score
HDFS	DeepLog	0.945	0.900	0.922
	LogAnomaly	0.860	0.898	0.878
	LogClustering	**1.000**	0.836	0.911
	PCA	0.347	0.073	0.121
	PLELog	0.950	**0.963**	**0.957**
	LogRobust	0.999	0.995	0.997
BGL	DeepLog	0.138	0.63	0.227
	LogAnomaly	0.179	0.998	0.303
	LogClustering	0.914	0.642	0.754
	PCA	0.448	0.333	0.382
	PLELog	**0.965**	**0.999**	**0.982**
	LogRobust	0.999	0.998	0.999

Besides, we found that almost all the compared semi-supervised and unsupervised approaches perform worse on BGL than HDFS. For example, F1-score of the state-of-the-art semi-supervised approach Deeplog largely drops from 0.922 on HDFS to 0.227 on BGL. This is because there are more unstable data in BGL due to its longer time span compared with HDFS. More specifically, about 7.4% log events in the test set of BGL do not appear in its training set, while there is no such new log events on HDFS. As the state-of-the-art semi-supervised approaches Deeplog and LogAnomaly aim to predict the next log event in a log sequence, they can only predict the log events appearing in their training sets and are easy to treat unseen log events as anomalies, leading to bad effectiveness. Regarding LogCluster, since it adopts one-hot representation for log sequences and then clusters vectors, when coming across unseen log events it directly ignores them during representation. Moreover, the log sequences containing unseen log events are usually normal in BGL, and thus its effectiveness is better than Deeplog and LogAnomaly. Faced with such unstable log data, PLELog still performs well due to its semantic embedding and the capability of incorporating the knowledge on historical anomalies. Please note that the effectiveness of these compared semi-supervised and unsupervised approaches is worse than that reported in their corresponding studies, since our data splitting method avoids data leakage and incurs practical unstable log data.

Comparison with the state-of-the-art supervised approach. Even though PLELog is a semi-supervised approach, it tries to incorporate the strength of supervised approaches through probabilistic label estimation, and thus it is interesting to explore how much gap between PLELog and the state-of-the-art supervised approach LogRobust. From Table I, although PLELog performs worse than LogRobust, their gap in terms of various metrics is actually quite small, indicating the success of incorporating the strength of supervised approaches via probabilistic label estimation. Moreover, as LogRobust depends on a large amount of manually labeled training data,

PLELog has greater usability in practice. As presented in Section IV-D, AUC is also an important metric for such classifiers of LogRobust and PLELog, and thus we further compared them in terms of this metric. More specifically, AUC of PLELog is 0.981 on both HDFS and BGL while that of LogRobust is 0.998 on both HDFS and BGL. We found that AUC of PLELog is very high, even competitive with that of LogRobust, further demonstrating its stable effectiveness.

2) RQ2: Contribution of Main Components in PLELog: Table II shows the effectiveness comparison results between PLELog and its variants. From this table, PLELog outperforms PLELog$_{noP}$ and PLELog$_{noA}$ in terms of effectiveness, confirming the contribution of making label estimation probabilistic and incorporating an attention mechanism to the overall effectiveness of PLELog. More specifically, the improvement of PLELog over PLELog$_{noP}$ in terms of F1-score is 66.7% on HDFS and 19.0% on BGL while that of PLELog over PLELog$_{noA}$ is 11.5% on HDFS and 10.8% on BGL, demonstrating that the component of making label estimation probabilistic contributes more. Also, we found that PLELog$_{noP}$ performs much worse on HDFS (0.574) than BGL (0.825) in terms of F1-score. This is because the clustering effectiveness on HDFS is worse than that on BGL. Even so, the effectiveness of PLELog is similar on both HDFS and BGL, indicating that our probabilistic label estimation does effectively reduce the influence of noise incurred by clustering (no matter how much noise is incurred). Besides, even though the clustering effectiveness in PLELog may be not very high (e.g., 0.808 F1-score on HDFS), its final classification effectiveness can be excellent (e.g., 0.957 F1-score on HDFS), further confirming the importance of probabilistic label estimation and also our attention-based GRU network.

From Table II, PLELog cannot guarantee to outperform PLELog$_{noR}$ stably in terms of effectiveness. For example, with dimension reduction, F1-score of PLELog on BGL is largely improved from 0.888 to 0.982 but that on HDFS is slightly decreased from 0.988 to 0.957. On average, this component is still helpful to the overall effectiveness of PLELog since HDBSCAN has been demonstrated to be not very effective for high-dimensional data [48], [50]. In particular, through dimension reduction, the clustering and training efficiency of PLELog can be largely improved, which is its another important contribution to PLELog.

In summary, both making label estimation probabilistic and incorporating an attention mechanism contribute to the overall effectiveness of PLELog, and the component of reducing semantic-vector dimension contributes to both effectiveness and efficiency of PLELog.

3) RQ3: Impact of Different PLELog Configurations: Figure 4 shows the effectiveness of different PLELog configurations (introduced in Section IV-B3) in terms of AUC. We found that although there exists small perturbation under different configurations, the overall effectiveness is stable, i.e., the deviation of *AUC* is smaller than 0.05 across all the configurations, indicating PLELog is insensitive to different configurations and thus robust in practice.

TABLE II: Experiment results between variants of PLELog

Dataset	Method	Precision	Recall	F1-score
HDFS	PLELog	0.950	0.963	0.957
	PLELog$_{noP}$	0.997	0.402	0.574
	PLELog$_{noA}$	**0.998**	0.753	0.858
	PLELog$_{noR}$	0.981	**0.995**	**0.988**
BGL	PLELog	**0.965**	**0.999**	**0.982**
	PLELog$_{noP}$	0.702	0.998	0.825
	PLELog$_{noA}$	0.878	0.895	0.886
	PLELog$_{noR}$	0.799	0.998	0.888

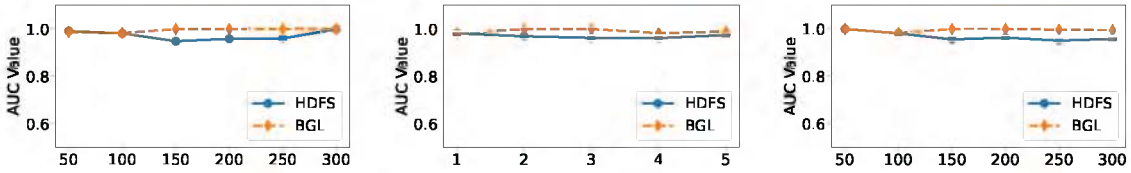
TABLE III: Time cost of studied approaches

Method	HDFS		BGL	
	Training	Testing	Training	Testing
PLELog	43m	42s	24m	10s
LogAnomaly	4h 40m	47m	4h 20m	39m
LogRobust	1h 20m	49s	30m	15s
DeepLog	1h 50m	20m	44m	7m
LogCluster	19m	23s	41s	40s
PCA	18m	1s	8s	1s

4) RQ4: Efficiency of PLELog: Table III presents the *training time* and *prediction time* of PLELog on HDFS and BGL, respectively. In general, all the approaches are efficient with very short prediction time (e.g., taking at most 17ms to predict one log sequence in HDFS). In particular, our GRU-based classifier is more efficient than the LSTM-based classifier of LogRobust, demonstrating that our GRU network is more helpful to support timely anomaly detection than the widely-used LSTM network. In terms of training time, PLELog performs better than the state-of-the-art semi-supervised approaches (i.e., Deeplog and LogAnomaly) and the state-of-the-art supervised approach (i.e., LogRobust), but performs worse than LogCluster and PCA, which only utilize simple clustering or projection. Since the training process is *offline*, the training time of PLELog (i.e., less than one hour) is still acceptable. To sum up, PLELog is efficient with very short prediction time and acceptable training time.

V. Practical Evaluation

We have successfully applied PLELog to two large-scale real-world distributed online systems from two different organizations, i.e., the network center of our university and one influential motor corporation throughout the world (we hide its name due to the company policy). Here, we call them $\mathcal{A}$ and $\mathcal{B}$ respectively. Since both systems are large-scale and complex and in the meanwhile the number of operators is limited, it is hard for them to manually analyze and label massive logs. PLELog is designed to conduct *automated* log-based anomaly detection without manually labeling anomalous logs, admirably satisfying their demands. Indeed, they largely appreciated the performance of PLELog on their systems. Here, we report the practical evaluation results based on two datasets from the two real-world systems to show the practical effectiveness of PLELog. For the dataset $\mathcal{A}$ there are 6,148,033



(a) Number of ICA components (b) Number of GRU Layers (c) Size of GRU Hidden States

Fig. 4: AUC value of various PLELog configurations.

TABLE IV: Practical evaluation results of studied approaches on real-world datasets

Dataset	Approach	Precision	Recall	F1-score
$\mathcal{A}$	DeepLog	0.883	0.833	0.858
	LogAnomaly	0.791	0.981	0.876
	LogCluster	0.165	0.998	0.283
	PCA	0.882	0.829	0.854
	PLELog	**0.900**	**1.000**	**0.947**
$\mathcal{B}$	DeepLog	0.794	0.982	0.878
	LogAnomaly	0.789	0.983	0.875
	LogCluster	0.897	0.945	0.920
	PCA	0.743	0.939	0.830
	PLELog	**0.978**	**0.991**	**0.984**

log messages, among which 4,833 are anomalous, while for the dataset $\mathcal{B}$ there are 334,567 log messages, among which 15,394 are anomalous. The number of log events are 25 and 243 for $\mathcal{A}$ and $\mathcal{B}$ respectively.

Table IV shows the effectiveness of all the studied semi-supervised and unsupervised approaches on the two real-world datasets. From this table, PLELog performs stably well on both datasets in terms of Precision, Recall, and F1-score. For example, F1-score of PLELog is 0.947 and 0.984 for $\mathcal{A}$ and $\mathcal{B}$ respectively, and in the meanwhile its Recall is 1.000 and 0.991. The results demonstrate that PLELog hardly misses to detect anomalies with very few false positives in the two real-world systems. Besides, PLELog still outperforms all the compared semi-supervised and unsupervised approaches on both datasets. For example, the improvements of PLELog over all the compared approaches range from 8.1% to 234.6% on $\mathcal{A}$ and from 7.0% to 18.6% on $\mathcal{B}$ in terms of F1-score. By combining the results in Tables I and IV, all the compared approaches perform sensitively on different datasets (i.e., the standard deviation of these compared approaches across all the four datasets ranges from 0.248 to 0.309 in terms of F1-score), while PLELog performs very stably regardless of the used datasets (i.e., its standard deviation across all the four datasets is only 0.016 in terms of F1-score). The results further demonstrate the generality and practicability of PLELog. Through an informal interview, developers indeed confirmed the usefulness of PLELog in practice.

VI. Discussion

A. Lessons Learned

Logs Evolution. In practice, logs evolve frequently especially for the companies adopting Continuous Delivery or Deployment [71], [72]. For example, as observed in the existing study [73], in their studied projects there are 20%∼45% log statements evolved throughout their lifetime. Also, there are hundreds of new log statements that are added to the source code every month for four systems in Google [74]. Log evolution could bring many new log events and log sequences. Therefore, handling such unstable log data is a capability essential to an log-based anomaly detection approach in practice and PLELog indeed possesses it as demonstrated in our study.

Anomaly Interpretability. In practice, accurate anomaly detection is helpful to mitigate and diagnose anomalies earlier, which could reduce the influence of anomalies. Moreover, recent studies have presented that making software analytics models interpretable to software practitioners is as important as achieving accurate prediction [75]–[77]. Thus, to further facilitate the mitigation and diagnosis of anomalies, operators expect to receive interpretable anomaly detection results. Otherwise, they still have to spend much time to identify root causes of anomalies. In particular, PLELog, besides effectively and efficiently detecting anomalies, also provides interpretable results to some extent through its attention mechanism. More specifically, PLELog provides the importance of each log event in an anomalous log sequence, which makes operators more clearly identify relevant log events to the detected anomaly so as to provide hints for subsequent root cause diagnosis. In the future, incorporating more advanced interpretability analysis in PLELog is necessary.

B. Extensions of PLELog

We plan to further improve PLELog from two aspects in the future. First, the anomaly detection effectiveness of PLELog relies on the effectiveness of our clustering to some degree. However, our used HDBSCAN could be costly for high-dimensional data and in the meanwhile dimension reduction could lose accuracy to some degree, and thus we plan to incorporate a more effective and efficient clustering method for high-dimensional data to improve PLELog. Second, as demonstrated in our study, PLELog still has a little gap with the state-of-the-art supervised approach (i.e., LogRobust) in terms of effectiveness. In the future, we can add a feedback process after operators mitigate/diagnose an anomaly, in order to incrementally update our model with the newly confirmed anomalous log data by operators. In this way, our model could perform stably and well over time to some degree.

C. Threats to Validity

The *internal* threat to validity mainly lies in the implementations of PLELog and compared approaches. To reduce this

threat, we implemented PLELog based on popular libraries (presented in Section IV-C) and two authors have carefully checked the source code. Regarding compared approaches, we adopted their open-source implementations directly.

The *external* threat to validity mainly lies in the used datasets. In our study, we used two widely-used public datasets, i.e., HDFS and BGL, following the existing work [26], [29], [30], [32], [39]. To further investigate the generality and practicability of PLELog, we applied PLELog to two industrial datasets from the network center of our university and an influential motor corporations respectively. In the future, we will evaluate PLELog on more datasets.

VII. Related Work

Supervised Log-based Anomaly Detection. With the help of *labeled* training data, supervised approaches perform relatively well for detecting anomalies [20], [28], [78]–[81]. For example, Liang et al. [80] explored four classifiers, i.e., RIPPER (a rule-based classifier), Support Vector Machines, a traditional Nearest Neighbor, and a customized Nearest Neighbor, for predicting failure events via logs. Bodik et al. [81] used a logistic regression model to identify previously seen performance crises in a datacenter. To further improve the performance of log-based anomaly detection, advanced deep learning techniques have been incorporated [18], [21], [31], [32], [41]. For example, besides the state-of-the-art supervised approach LogRobust (introduced in Section IV-B1), Vinayakumar et al. [41] proposed a stacked-LSTM model for accurate log-based anomaly detection.

Different from them, our work proposes a semi-supervised log-based anomaly detection approach PLELog, only requiring to know the labels of a set of easy-to-obtain normal log sequences, to get rid of time-consuming manual labeling, which is the practical limitation of supervised approaches. In particular, the effectiveness of PLELog is even close to that of the state-of-the-art supervised approach LogRobust as demonstrated in our study.

Unsupervised and Semi-supervised Log-based Anomaly Detection. Due to less depending on labeled data for model training, unsupervised and semi-supervised approaches tend to be more practical [16], [40], [59], [82]–[85]. Beside Deeplog, LogAnomaly, LogCluster, and PCA that have been introduced in Section IV-B1, Lou et al. [29] proposed Invariant Mining (IM) that detects anomalies by checking the violations against a set of *execution flow invariants* mined from previous log sequences. He et al. [59] proposed Log3C that utilizes the Hierarchical Agglomerative Clustering algorithm [86] to group similar log sequences, and identifies the correlations between logs and system KPIs (Key Performance Indicators).

Our proposed approach PLELog is also a semi-supervised approach only knowing the labels of a part of normal log sequences that are easy-to-obtain as presented in Section I. Different from them, PLELog further incorporates the strength of supervised approaches (i.e., learning the knowledge on historical anomalies) via probabilistic label estimation instead of manual labeling in order to largely improve the effectiveness of unsupervised and semi-supervised approaches.

VIII. Conclusion

Over the years, many log-based anomaly detection approaches have been proposed, but these existing approaches still suffer from practical issues due to either relying on a large amount of manually labeled training data (supervised approaches) or unsatisfactory performance without learning the knowledge on historical anomalies (unsupervised and semi-supervised approaches). In this paper, we proposed a practical semi-supervised (without time-consuming manual labeling) approach PLELog, which learns the knowledge on historical anomalies via probabilistic label estimation so that the strength of supervised approaches can be incorporated. Also, PLELog can stay immune to unstable log data via semantic embedding and detect anomalies efficiently and effectively with an attention-based GRU neural network. Our experimental results on two most widely-used public datasets demonstrate the effectiveness of PLELog. In particular, PLELog has been applied to two real-world systems from our university and an influential motor corporation, further demonstrating its practicability.

Acknowledgment

This work has been partially supported by the National Natural Science Foundation of China 61872263 and 62002256, Intelligent Manufacturing Special Fund of Tianjin 201193155 and Innovation Research Project of Tianjin University under Grant No. 2020XZC-0042.

References

[1] D. D. Yao, X. Shu, L. Cheng, and S. J. Stolfo, *Anomaly Detection as a Service: Challenges, Advances, and Opportunities*, ser. Synthesis Lectures on Information Security, Privacy, and Trust. Morgan & Claypool Publishers, 2017.

[2] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "An empirical investigation of incident triage for online service systems," in *ICSE (SEIP)*. IEEE / ACM, 2019, pp. 111–120.

[3] J. Chen, S. Zhang, X. He, Q. Lin, H. Zhang, D. Hao, Y. Kang, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems," in *ASE*. IEEE, 2020, pp. 373–384.

[4] N. Zhao, J. Chen, Z. Wang, X. Peng, G. Wang, Y. Wu, F. Zhou, Z. Feng, X. Nie, W. Zhang, K. Sui, and D. Pei, "Real-time incident prediction for online service systems," in *ESEC/SIGSOFT FSE*. ACM, 2020, pp. 315–326.

[5] Y. Chen, X. Yang, H. Dong, X. He, H. Zhang, Q. Lin, J. Chen, P. Zhao, Y. Kang, F. Gao, Z. Xu, and D. Zhang, "Identifying linked incidents in large-scale online service systems," in *ESEC/SIGSOFT FSE*. ACM, 2020, pp. 304–314.

[6] J. Jiang, W. Lu, J. Chen, Q. Lin, P. Zhao, Y. Kang, H. Zhang, Y. Xiong, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems," in *ESEC/SIGSOFT FSE*. ACM, 2020, pp. 1410–1420.

[7] N. Zhao, J. Chen, X. Peng, H. Wang, X. Wu, Y. Zhang, Z. Chen, X. Zheng, X. Nie, G. Wang, Y. Wu, F. Zhou, W. Zhang, K. Sui, and D. Pei, "Understanding and handling alert storm for online service systems," in *ICSE (SEIP)*. ACM, 2020, pp. 162–171.

[8] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *ICSE (1)*. IEEE Computer Society, 2015, pp. 415–425.

[9] D. Schipper, M. F. Aniche, and A. van Deursen, "Tracing back log data to its log statement: from research to practice," in *MSR*. IEEE / ACM, 2019, pp. 545–549.

[10] R. Ding, H. Zhou, J. Lou, H. Zhang, Q. Lin, Q. Fu, D. Zhang, and T. Xie, "Log2: A cost-aware logging mechanism for performance diagnosis," in *USENIX Annual Technical Conference*. USENIX Association, 2015, pp. 139–150.

[11] Q. Fu, J. Lou, Q. Lin, R. Ding, D. Zhang, and T. Xie, "Contextual analysis of program logs for understanding system behaviors," in *MSR*. IEEE Computer Society, 2013, pp. 397–400.

[12] Q. Fu, J. Zhu, W. Hu, J. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," in *ICSE Companion*. ACM, 2014, pp. 24–33.

[13] C. Zhi, J. Yin, S. Deng, M. Ye, M. Fu, and T. Xie, "An exploratory study of logging configuration practice in java," in *ICSME*. IEEE, 2019, pp. 459–469.

[14] J. Candido, M. F. Aniche, and A. van Deursen, "Contemporary software monitoring: A systematic literature review," *CoRR*, vol. abs/1912.05878, 2019.

[15] X. Shu, D. Yao, and N. Ramakrishnan, "Unearthing stealthy program attacks buried in extremely long execution paths," in *CCS*. ACM, 2015, pp. 401–413.

[16] H. Jiang, X. Li, Z. Yang, and J. Xuan, "What causes my test alarm?: automatic cause analysis for test alarms in system and integration testing," in *ICSE*. IEEE / ACM, 2017, pp. 712–723.

[17] L. Wang, N. Zhao, J. Chen, P. Li, W. Zhang, and K. Sui, "Root-cause metric location for microservice systems via log anomaly detection," in *ICWS*. IEEE, 2020, pp. 142–150.

[18] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *ESEC/SIGSOFT FSE*. ACM, 2019, pp. 807–817.

[19] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *ICSE (SEIP)*. IEEE / ACM, 2019, pp. 121–130.

[20] M. Farshchi, J. Schneider, I. Weber, and J. C. Grundy, "Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis," in *ISSRE*. IEEE Computer Society, 2015, pp. 24–34.

[21] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, "Automated IT system failure prediction: A deep learning approach," in *BigData*. IEEE Computer Society, 2016, pp. 1291–1300.

[22] H. Amar, L. Bao, N. Busany, D. Lo, and S. Maoz, "Using finite-state models for log differencing," in *ESEC/SIGSOFT FSE*. ACM, 2018, pp. 49–59.

[23] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Long short-term memory based operation log anomaly detection," in *ICACCI*. IEEE, 2017, pp. 236–242.

[24] X. Shu, J. Smiy, D. D. Yao, and H. Lin, "Massive distributed and parallel log analysis for organizational security," in *GLOBECOM Workshops*. IEEE, 2013, pp. 194–199.

[25] M. Nagappan, "Analysis of execution log files," in *ICSE (2)*. ACM, 2010, pp. 409–412.

[26] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *ISSRE*. IEEE Computer Society, 2016, pp. 207–218.

[27] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *ICML*. Omnipress, 2010, pp. 37–46.

[28] M. Y. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. A. Brewer, "Failure diagnosis using decision trees," in *ICAC*. IEEE Computer Society, 2004, pp. 36–43.

[29] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *ATC*. USENIX Association, 2010.

[30] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *ICSE (Companion Volume)*. ACM, 2016, pp. 102–111.

[31] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*. ACM, 2017, pp. 1285–1298.

[32] 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 *IJCAI*. ijcai.org, 2019, pp. 4739–4745.

[33] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *ESEC/SIGSOFT FSE*. ACM, 2019, pp. 683–694.

[34] B. Russo, G. Succi, and W. Pedrycz, "Mining system logs to learn error predictors: a case study of a telemetry system," *Empir. Softw. Eng.*, vol. 20, no. 4, pp. 879–927, 2015.

[35] S. Kabinna, C. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan, "Examining the stability of logging statements," *Empir. Softw. Eng.*, vol. 23, no. 1, pp. 290–333, 2018.

[36] Y. Zhang, L. Li, J. Zhou, X. Li, Y. Liu, Y. Zhang, and Z. Zhou, "POSTER: A PU learning based system for potential malicious URL detection," in *CSS*. ACM, 2017, pp. 2599–2601.

[37] L. McInnes, J. Healy, and S. Astels, "hdbscan: Hierarchical density based clustering," *J. Open Source Softw.*, vol. 2, no. 11, p. 205, 2017.

[38] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *SSST@EMNLP*. Association for Computational Linguistics, 2014, pp. 103–111.

[39] A. J. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *DSN*. IEEE Computer Society, 2007, pp. 575–584.

[40] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *CIKM*. ACM, 2016, pp. 1573–1582.

[41] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Long short-term memory based operation log anomaly detection," in *ICACCI*. IEEE, 2017, pp. 236–242.

[42] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *ICWS*. IEEE, 2017, pp. 33–40.

[43] 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.

[44] ——, "An evaluation study on log parsing and its use in log mining," in *DSN*. IEEE Computer Society, 2016, pp. 654–661.

[45] B. Dit, L. Guerrouj, D. Poshyvanyk, and G. Antoniol, "Can better identifier splitting techniques help feature location?" in *ICPC*. IEEE Computer Society, 2011, pp. 11–20.

[46] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*. ACL, 2014, pp. 1532–1543.

[47] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manag.*, vol. 24, no. 5, pp. 513–523, 1988.

[48] L. McInnes and J. Healy, "Accelerated hierarchical density based clustering," in *ICDM Workshops*. IEEE Computer Society, 2017, pp. 33–42.

[49] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[50] J. Chen, Z. Wu, Z. Wang, H. You, L. Zhang, and M. Yan, "Practical accuracy estimation for efficient deep neural network testing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*.

[51] P. Cifariello, P. Ferragina, and M. Ponza, "Wiser: A semantic approach for expert finding in academia based on entity linking," *Inf. Syst.*, vol. 82, pp. 1–16, 2019.

[52] A. Faroughi, R. Javidan, M. Mellia, A. Morichetta, F. Soro, and M. Trevisan, "Achieving horizontal scalability in density-based clustering for urls," in *BigData*. IEEE, 2018, pp. 3841–3846.

[53] N. Scheiner, N. Appenrodt, J. Dickmann, and B. Sick, "A multi-stage clustering framework for automotive radar data," in *ITSC*. IEEE, 2019, pp. 2060–2067.

[54] B. M. Moret and H. D. Shapiro, "An empirical assessment of algorithms for constructing a minimum spanning tree." *Computational Support for Discrete Mathematics*, vol. 15, pp. 99–117, 1992.

[55] E. Oja and Z. Yuan, "The fastica algorithm revisited: Convergence analysis," *IEEE Trans. Neural Networks*, vol. 17, no. 6, pp. 1370–1381, 2006.

[56] F. Biondi, A. Legay, B. F. Nielsen, and A. Wasowski, "Maximizing entropy over markov processes," *J. Log. Algebraic Methods Program.*, vol. 83, no. 5-6, pp. 384–399, 2014.

[57] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.

[58] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "Continuous incident triage for large-scale online service systems," in *ASE*. IEEE, 2019, pp. 364–375.

[59] S. He, Q. Lin, J. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *ESEC/SIGSOFT FSE*. ACM, 2018, pp. 60–70.
[60] K. Nagaraj, C. E. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *NSDI*. USENIX Association, 2012, pp. 353–366.
[61] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Trans. Software Eng.*, vol. 39, no. 6, pp. 806–821, 2013.
[62] F. Tu, J. Zhu, Q. Zheng, and M. Zhou, "Be careful of when: an empirical study on time-related misuse of issue tracking data," in *ESEC/SIGSOFT FSE*. ACM, 2018, pp. 307–318.
[63] J. C. Gower and G. J. Ross, "Minimum spanning trees and single linkage cluster analysis," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 18, no. 1, pp. 54–64, 1969.
[64] R. Dunia and S. J. Qin, "Multi-dimensional fault diagnosis using a subspace approach," in *American Control Conference*, 1997.
[65] "Pytorch official website," Accessed: 2020, https://pytorch.org/.
[66] "Hdbscan," Accessed: 2020, https://hdbscan.readthedocs.io/en/latest/basic_hdbscan.html.
[67] "Scikit-learn," Accessed: 2020, https://scikit-learn.org/stable/index.html.
[68] "Logdeep," Accessed: 2020, https://github.com/donglee-afar/logdeep.
[69] "Loghub," Accessed: 2020, https://github.com/logpai/loghub.
[70] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.
[71] X. Bai, M. Li, D. Pei, S. Li, and D. Ye, "Continuous delivery of personalized assessment and feedback in agile software engineering projects," in *ICSE (SEET)*. ACM, 2018, pp. 58–67.
[72] W. Stocker, "From agile to continuous development in the healthcare domain: lessons learned," in *ICSE (SEIP)*. ACM, 2018, pp. 211–212.
[73] S. Kabinna, C. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan, "Examining the stability of logging statements," *Empir. Softw. Eng.*, vol. 23, no. 1, pp. 290–333, 2018.
[74] W. Xu, "System problem detection by mining console logs," Ph.D. dissertation, University of California, Berkeley, USA, 2010.
[75] H. K. Dam, T. Tran, and A. Ghose, "Explainable software analytics," *CoRR*, vol. abs/1802.00603, 2018.
[76] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," *TSE*, 2020.
[77] S. Wattanakriengkrai, P. Thongtanunam, C. Tantithamthavorn, H. Hata, and K. Matsumoto, "Predicting defective lines using a model-agnostic technique," *CoRR*, vol. abs/2009.03612, 2020.
[78] J. Breier and J. Branisová, "A dynamic rule creation based anomaly detection method for identifying security breaches in log records," *Wirel. Pers. Commun.*, vol. 94, no. 3, pp. 497–511, 2017.
[79] Q. Fu, J. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *ICDM*. IEEE Computer Society, 2009, pp. 149–158.
[80] Y. Zhang and A. Sivasubramaniam, "Failure prediction in IBM bluegene/l event logs," in *IPDPS*. IEEE, 2008, pp. 1–5.
[81] P. Bodík, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *EuroSys*. ACM, 2010, pp. 111–124.
[82] M. Hassani, W. Shang, E. Shihab, and N. Tsantalis, "Studying and detecting log-related issues," *ESE*, vol. 23, no. 6, pp. 3248–3280, 2018.
[83] G. Vigna, F. Valeur, D. Balzarotti, W. K. Robertson, C. Kruegel, and E. Kirda, "Reducing errors in the anomaly-based detection of web-based attacks through the combined analysis of web requests and SQL queries," *J. Comput. Secur.*, vol. 17, no. 3, pp. 305–329, 2009.
[84] T. Li, Y. Jiang, C. Zeng, B. Xia, Z. Liu, W. Zhou, X. Zhu, W. Wang, L. Zhang, J. Wu, L. Xue, and D. Bao, "FLAP: an end-to-end event log analysis platform for system management," in *KDD*. ACM, 2017, pp. 1547–1556.
[85] A. Amar and P. C. Rigby, "Mining historical test logs to predict bugs and localize faults in the test logs," in *ICSE*. IEEE / ACM, 2019, pp. 140–151.
[86] D. Z. Chen and B. Xu, "Geometric algorithms for agglomerative hierarchical clustering," in *COCOON*, ser. Lecture Notes in Computer Science, vol. 2697. Springer, 2003, pp. 30–39.