IEEE Access
Multidisciplinary Rapid Review Open Access Journal

Received 7 August 2023, accepted 25 August 2023, date of publication 1 September 2023, date of current version 8 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3311146

RESEARCH ARTICLE

TransSentLog: Interpretable Anomaly Detection Using Transformer and Sentiment Analysis on Individual Log Event

TUAN-ANH PHAM AND JONG-HOON LEE
Department of AI Laboratory, MOADATA, Global Convergence Center, Seongnam-si 13449, South Korea
Corresponding author: Jong-Hoon Lee (jhlee@moadata.ai)

ABSTRACT Event logs play a crucial role in monitoring the status of IT systems. These logs contain text that describes how a system operates using natural language, which can be associated with sentiment polarity. When a system is functioning correctly, event logs generally convey positive sentiment. However, if unexpected behaviors like errors or failures occur, negative sentiment can be detected. In order to identify anomalies in individual log messages without the need for log parsing, we propose TranSentLog. This method combines Transformer and sentiment analysis, leveraging the sentiment polarity of event logs. To gain a better understanding of the model predictions, we employ Integrated Gradients, an attribution method that extracts important features from the model inputs. Through extensive experimentation on public system log datasets, we demonstrate that our proposed method overcomes the limitations of existing approaches and achieves F1 scores of 99.73% on trained datasets and 94.99% on untrained datasets.

INDEX TERMS Log anomaly detection, transformer, sentiment analysis, system log, integrated gradients.

I. INTRODUCTION

In IT system monitoring, logs serve as records that are generated by different components, including hardware, databases, networks, and applications. These logs are crucial for monitoring and understanding the behaviors of the system. However, as systems evolve, the types of logs and the volume of data generated from them increase significantly, making it impractical to manually examine and identify system issues. Furthermore, system failures can lead to substantial losses in terms of revenue, time, and overall performance. Therefore, it is imperative to design an automated method for detecting anomalies in logs, enabling early detection of issues in IT systems.

Anomalies refer to patterns that deviate from well-defined norms or expected behaviors. The determination of normal and anomalous behaviors is typically based on various characteristics specific to the application domain, as defined by the analyst's interests. Anomalies are commonly classified into three types: point, contextual, and collective [1]. A point anomaly is an individual observation that stands out from the rest of the data, making it suspicious. On the other hand, contextual anomalies are more challenging to detect as they depend on specific contexts. An observation may be considered anomalous in one context but normal in another. In contrast to the first two anomaly types, a collective anomaly refers to a group of related observations that are considered anomalous when they occur together but are deemed normal when each observation is evaluated independently.

In log anomaly detection, log messages exhibit various abnormal patterns during system incidents. Two common abnormal patterns: keywords (point anomaly) and template sequence (collective anomaly) are presented in the study [2]. In the case of keywords, a log event is considered abnormal if it contains negative keywords such as "fatal", "disk failure", or "failed to connect", etc. Although manually defining keywords is traditional and labor-intensive, it is still necessary due to its interpretability. Unlike the keyword-based method that mainly identifies anomalies in individual event logs, the template sequence [3], [4] examines consecutive logs to identify abnormalities in task

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

execution when the log order deviates from the expected sequence.

The primary focus of this study is to enhance the performance of log anomaly detection specifically for point anomalies, which are represented by keywords. Based on our observations, we have identified several key points that highlight the importance of exploring the detection of anomalies in individual log events. Firstly, applying keywords without considering the context of the surrounding words within a log event can result in false positives. Secondly, as data volumes increase, certain keywords may be missed or system operators may not be aware of which keywords to define, leading to false negatives. Thirdly, through the analysis of various types of system log data in [5], we have discovered that the majority of system incidents can be detected just by examining single log events. Lastly, incident diagnosis for each individual log is relatively straightforward, especially when it comes to revising false alarm log messages.

Basically, log events use textual and natural language to describe the activities within a system. Therefore, it is possible to examine logs by examining the sentiment expressed. For system incidents such as warnings, errors, and failures, log entries typically include negative words alongside contextual elements (system, device, router, etc.). Otherwise, log events are categorized as positive when the systems are functioning normally. Figure 1 demonstrates the difference between normal and abnormal log messages based on their positive and negative sentiment polarity, respectively.

Inspired by Pylogsentiment [6], the state-of-the-art approach utilizes a deep learning model for log anomaly detection through sentiment analysis. Although the method performs well, it still has some limitations that hinder its practical deployment. Primarily, Pylogsentiment heavily relies on Nerlogparser [7] during the log preprocessing stage. Figure 2 illustrates an example of event log parsing from a syslog file, the "message" is further used for sentiment classfication. However, this log event parser, based on a bidirectional LSTM network, is time-consuming during inference and cannot effectively handle a large volume of log events in real-world scenarios. Additionally, retraining the log parser is needed when the systems encounter new log structures to avoid parsing errors. Basically, Pylogsentiment employs GRU [8] for training and classifying log events as normal or abnormal based on sentiment analysis. Because GRU is a variant of the recurrent neural network, it requires substantial time for the learning process due to sequential dependencies [9]. Lastly, returning only classification results make it difficult to understand the reasoning behind the model predictions.

To solve the above-mentioned limitations, our proposed approach introduces a log anomaly detection method that relies on sentiment analysis. Applying sentiment analysis to log data can offer several benefits for anomaly detection. First, logs expressing errors, issues, unauthorized access, or system failures can be triggered by negative polarity. Second, it is able to detect unknown anomalies when combining sentiment analysis with word embedding such as GloVe. For instance, the word "error" has a close meaning to "failed", and "error" appears in the training dataset while "failed" is not. After training the model, the model can still detect "failed" as anomalous because of the semantic similarity between "error" and "failed".

By eliminating the need for the log parser, we redesign the log preprocessing step and ensure that the processed logs contain sufficient details for subsequent training and detection. To enhance training speed and detection performance, we derive the Transformer architecture, a powerful method for natural language processing, for classifying log events. The foundation of the Transformer is the self-attention mechanism, which allows the model to weigh the importance of different parts in the input. Furthermore, it also enables the model to capture relevant context regardless of the distance between positions. Based on that, we expect this mechanism can capture the relationship between negative words and contextual elements in log data. In order to interpret the model predictions, we adopt the Integrated Gradients attribution method, which is applied to alarm log events, thus filtering out false positives effectively.

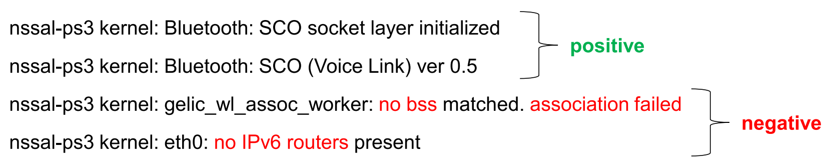
The contributions of the paper are listed as follows:

- We propose a practical approach called TransSentLog for log anomaly detection, which is derived from the Transformer architecture. Unlike the existing method, this approach does not rely on the log parser, yet it achieves comparable performance results
- The proposed approach is extensively evaluated using public system log datasets. The evaluation provides insights into the model performance, allowing for a better understanding of its effectiveness in log anomaly detection
- We uncover the black box of the model by applying the Integrated Gradients (IG) method. This may reduce investigation time and alleviate alarm fatigue experienced by system operators. Moreover, applying this technique enables a clearer understanding of the reasons behind the model predictions.
- To facilitate reproducibility and further research, we publish the code implementation and details of evaluation results on Github repository.

The structure of the remaining sections in this paper is organized as follows. Section II introduces existing studies that employ sentiment analysis for anomaly detection. Section III represents the overall architecture of the proposed method and the details of its components. The experimental setup is described in Section IV. Then, the extensive evaluation of model performance is discussed in Section V. Finally, we summarize the key points of the paper, main findings, and discuss future works in Section VII.

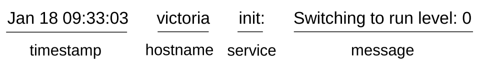
II. RELATED WORKS

Anomaly detection based on sentiment analysis has gained significant attention, particularly in the context of social media networks and product reviews. A study [10] proposes an enhanced lexicon-based text classifier that incorporates



```
nssal-ps3 kernel: Bluetooth: SCO socket layer initialized
nssal-ps3 kernel: Bluetooth: SCO (Voice Link) ver 0.5
nssal-ps3 kernel: gelic_wl_assoc_worker: no bss matched. association failed
nssal-ps3 kernel: eth0: no IPv6 routers present
```

FIGURE 1. Illustration of sentiment polarity in OS kernel log messages.



```
Jan 18 09:33:03 victoria init: Switching to run level: 0
```

FIGURE 2. Example of event log parsing from syslog file.

target objects to identify unusual opinions or sudden changes in the sentiment expressed in Twitter tweets. Similarly, [11] employs a support vector machine (SVM) to classify tweet sentiment after aggregating tweets within specific time windows. The resulting counts of positive, neutral, and negative tweets in each window are then transformed into time-series data for detecting anomalies. To detect potential insider threats within a company, [12] analyzes the sentiment expressed in employee emails based on specific aspects. Initially, various aspects such as business, contract, work, etc., are extracted from the emails. Subsequently, the email contents and their corresponding aspects are inputted into BiGRU (Bidirectional Gated Recurrent Unit) for sentiment classification.

Log anomaly detection based on individual log events and sentiment analysis has not received much attention. However, with some benefits of detecting anomalies in individual events, several studies have aimed to enhance detection performance. In an effort to combine sentiment and aspect terms within a log event, [13] introduces two attention mechanisms: context attention and content attention, along with a GRU network, for sentiment classification of log events. In order to develop a generic model, SentiLog [14] extracts logging statements obtained from various source codes of parallel file systems for training purposes. Although this approach demonstrates promising results, accessing source codes to collect logging statements is not feasible in practice. Similar to SentiLog, which uses multiple log sources, ADLILog collects source codes from over 1000 Github projects and extracts event descriptions from logging statements. It is assumed that normal logs are categorized under the log level "INFO", while abnormal logs fall under "ERROR", "CRITICAL", and "FATAL" log levels. Then, ADLILog applied a Transformer network for log anomaly detection.

Pylogsentiment [6] uses sentiment analysis and individual log events for anomaly detection. Initially, a log parser called Nerlogparser is applied to extract structured information from log events. Subsequently, the message part which conveys sentiment information is converted into numerical embedding vectors to enhance the representation of log events using GloVe word embedding [15]. Since Pylogsentiment is a supervised learning model, an imbalanced dataset can significantly impact its performance. To address this, an under-sampling method known as Tomek-link is employed to eliminate data points from the majority class that are close to the borderline with the minority class. Finally, a GRU network is employed to classify log events as either normal or abnormal.

III. PROPOSED METHOD

The overall architecture of TransSentLog is presented in Figure 3, the architecture consists of training and inference phases. During the training phase, the model is trained using various system log data to create a more generalized model capable of effectively detecting new types of log data. The log events are then preprocessed to eliminate noise, and GloVe word embeddings are utilized to represent each log event as a numeric embedding vector. To address the impact of padding and unknown tokens on model training, a masking layer is applied to instruct the attention layers to disregard these irrelevant tokens. Additionally, SpatialDropout1D is employed to drop entire one-dimensional feature maps, aiding the model in learning meaningful features. The embedding vectors are combined with positional encoding vectors and passed through Transformer blocks. Positional encoding ensures that the order of words in a log event is preserved. Furthermore, an average pool layer with masked inputs aggregates and computes the context embedding vector of a log event using word encoding vectors retrieved from the last Transformer block. Finally, a log event is classified as either positive or negative, indicating normal or abnormal, respectively.

In the inference phase, a raw log event is preprocessed and transformed into an embedding vector same as in the training process. Later, if the log event is classified as an anomaly, the integrated gradient method is applied directly to interpret the reason why the model gives that prediction. Based on the prediction, system operators can decide to update the trained model if the prediction is a false positive.

A. LOG PREPROCESSING

Unlike Pylogsentiment, our approach does not apply the log parser for log event preprocessing due to its time-consuming. To demonstrate the inefficiency of Pylogsentiment, Table 1 presents a comparison of preprocessing times between Pylogsentiment and TransSentLog. The table reveals that Pylogsentiment can only process an average of 150 logs per second, which is exceptionally slow and impractical for real-world system development. Consequently, we design a series of steps to process raw logs and ensure the processed logs are of sufficient quality for subsequent detection.

The proposed steps are as follows. Firstly, we address the issue of negative contractions by expanding them into their regular forms. This expansion enables the model to

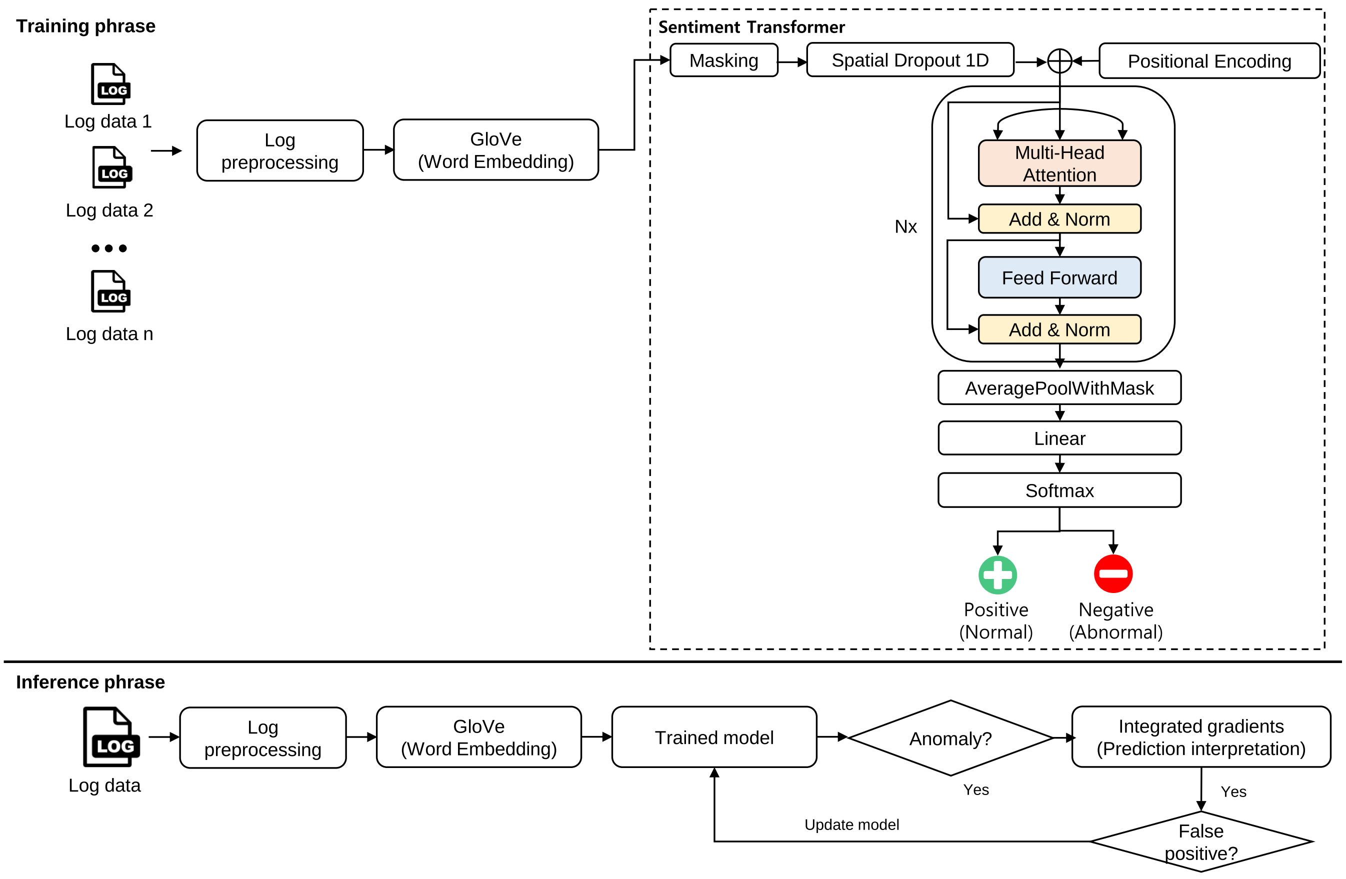


FIGURE 3. **Overall architecture of TransSentLog.**

capture negative verbs that are not contained in the GloVe dictionary. For example, "isn't" becomes "is not" and "hasn't" becomes "has not".

Next, we truncate words surrounded by special characters. By removing these characters, we extract the essential words from log events. For instance, "driver:" becomes "driver" and "?terminate" becomes "terminate".

We exclude log levels such as "info" and "warning" from the log events while retaining levels such as "error", "fatal", and "critical". This step prevents the model from focusing on "info" and "warning" levels during training and forces it to learn from other words within the log events. Even if the log level is "info", the log event still might be an abnormal log event [16].

Furthermore, we eliminate months of the year and days of the week using regular expressions, as these words are not useful to describe the sentiment of log events.

Lastly, any numbers or words containing numbers and special characters are removed. The scope of our study is system logs and these steps are proposed by analyzing various types of logs from Loghub repository [5].

B. SENTIMENT TRANSFORMER

Initially, we define a fixed log message length, denoted as L. After preprocessing the raw log messages, a processed log message is either truncated if its length exceeds L or padded

TABLE 1. **Comparision between the preprocessing time of Pylogsentiment and TransSentLog.**

Dataset	No. logs	Pylogsentiment	TransSentLog
Casper	11,086	62.9s	5.23s
Windows	24,991	211.79s	5.91s
Honey5	124,386	803.21s	13.74s

with zeros if its length is smaller than L. A set of processed log messages is defined as $\mathbf{M} = \{M_1, M_2, \ldots, M_{|\mathbf{M}|}\}$ and $M_i = \{w_{i1}, w_{i2}, \ldots, w_{iL}\}$ where $\mathbf{M}$ is total number of logs and w is a word in a processed log message M_i.

To capture the semantic meaning of log events, we employ GloVE word embedding to represent log events as embedding vectors. GloVe performs statistical analysis of word co-occurrence, resulting in similar words having similar embedding vectors. This similarity can be measured using techniques such as cosine similarity. For our study, we utilize a pre-trained word embedding called Glove version glove.6B.50d.txt. This version of GloVe is trained on a corpus comprising 6 billion tokens from Wikipedia 2014 and Gigaword 5. It contains a vocabulary of 400,000 words, with each word vector having a dimension of 50.

To train the network, we load all the word embedding vectors from the provided.txt file into an embedding layer. This layer acts as a fixed dictionary, where the word indices serve as keys and their corresponding word embeddings as values. Therefore, the input format of the embedding layer should be a vector of integer numbers. We define $M'_i = \{\mathrm{wi}_1, \mathrm{wi}_2, \ldots, \mathrm{wi}_L\}, 1 \leq \mathrm{wi} \leq 400000$ as a word index vector for a log message M_i. Similar to Pylogsentiment, we address the class imbalance issue by applying the under-sampling method Tomek link [17] to the word index matrix $\mathbf{M}' = \{M'_1, M'_2, \ldots, M_{|\mathbf{M}'|}\}$.

Subsequently, we employ a masking layer to ignore padding and unknown tokens. Additionally, we use the SpartialDropout1D layer, which drops entire one-dimensional features, promoting feature independence and facilitating the detection of new types of log data.

The position encoding layer and Transformer encoder are derived from the original Transformer model [18]. To encode the information of word order of a log event in the model, the position encoding matrix $\mathrm{PE} \in \mathbb{R}^{L\times d}$ is calculated as follows:

$$\begin{aligned}\mathrm{PE}[\mathrm{pos}, 2i] &= \sin\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d}}}\right)\\ \mathrm{PE}[\mathrm{pos}, 2i+1] &= \cos\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d}}}\right)\end{aligned} \tag{1}$$

where pos is the position index of a word in a log event ($1 \leq \mathrm{pos} \leq L$), $2i$ denotes the even position in the embedding dimension ($0 \leq i < \frac{d}{2}$), d represents the embedding dimension ($d = 50$ in this case). The matrix PE is then added to the input embeddings and fed into Transformer blocks.

The model consists of N identical blocks, each containing two sublayers: multi-head attention and position-wise fully connected feed-forward network. Both sublayers are augmented with a residual connection followed by layer normalization. The multi-head attention mechanism incorporates self-attention layers called heads, which run in parallel. The self-attention layer can be defined as a scaled dot product function:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

where $Q, K, V \in \mathbb{R}^{L\times d_k}$ are query, key and value matrices, respectively. In this case, these inputs are the same because of self-attention. The scaling factor $\sqrt{d_k}$ normalizes the dot product and d_k is the feature dimension of the input matrix.

Multi-head attention helps the model explore the information from different representation subspaces at different positions. The output of the multi-head attention layer is obtained by concatenating the outputs of each head. Denote h is the number of heads and multi-head attention is:

$$\begin{aligned}&\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)W^O\\ &\qquad \text{where } \mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned} \tag{3}$$

where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d\times d_k}$ and $W^O \in {}^{d\times d}$ are parameter matrices. Note that $d = d_k \times h$ and the output of the multi-head layer will have the same shape as the input embedding ($L \times d$).

The second sublayer, the position-wise feed-forward network, consists of two linear transformations with a ReLU activation function in between:

$$\mathrm{FeedForward}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{4}$$

To capture the contextual information of each log event as a whole, we employ an average pooling layer with masked values propagated from the upstream layers. The contextual vector is calculated as follows:

$$\bar{x} = \frac{1}{L' + \epsilon}\sum_{i=1}^{L'} x_i \tag{5}$$

where ϵ is a very small number to avoid division by zero, $x_i \in \mathbb{R}^d$ is a word encoding vector retrieved from the last Transformer block, L' is the number of masked words in a log event.

Lastly, a linear layer followed by a softmax activation function is applied to classify the sentiment type of the log events. Normal log messages are classified as positive, while abnormal log messages are classified as negative.

C. INTEGRATED GRADIENT

Predicting the sentiment of a log event as either positive or negative is insufficient in practical scenarios, particularly when negative predictions are made. It is crucial for the model to have a mechanism that clarifies the factors influencing its decisions based on the input. This is beneficial for system operators as it allows them to quickly assess anomalous logs and verify the model functionality. To meet this requirement, we have employed an Explainable AI technique called Integrated Gradients (IG) [19], which enables the interpretation of feature importance on the model predictions. In the context of sentiment classification for a log message, IG helps system operators identify the words within the message having the greatest impact on the predicted sentiment.

The underlying concept of IG involves computing the integral of the model's output gradients with respect to the input features along a path from a baseline input to the actual input. This path is a series of points called interpolated inputs and these interpolated inputs are obtained by calculating small increments with the difference between the actual and baseline input.

Suppose that a function $\mathrm{F} : \mathbb{R}^{L\times d} \rightarrow \{0, 1\}$ represents a deep network. In our study, the deep network is the TransSentLog model, 0 and 1 are abnormal and normal predictions, respectively. Because the IG method cannot process a text message as input, the message should first be mapped to an embedded matrix $x \in \mathbb{R}^{L\times d}$. For sentiment analysis, the baseline input is defined as a zero embedding matrix.

To avoid numerical integration issues, we approximate the numerical value of IG rather than relying on the integral itself. The approximation is calculated by the following equation:

$$\begin{aligned}&\mathrm{IntegratedGrads}_i^{\mathrm{approx}}(x)\\ &\quad\approx (x_i - x'_i) \times \frac{1}{m} \times \sum_{k=1}^{m} \frac{\partial \mathrm{F}(x' + \frac{k}{m} \times (x - x'))}{\partial x_i}\end{aligned} \tag{6}$$

where x, x' are actual and baseline inputs, respectively. m denotes the number of steps in the sum approximation, x_i

corresponds to the i^{th} feature of the input, and $x' + \frac{k}{m} \times (x - x')$ is an interpolated input. The final score vector $s \in \mathbb{R}^L$ which implies the importance of each word in a log message is summarized along the embedding dimension.

IV. EXPERIMENTAL SETTINGS

A. DATASETS

To assess the detection performance of TransSentLog, we utilized 9 publicly available system log datasets, which were identical to those used in Pylogsentiment. A comprehensive overview of these datasets is presented in Table 2.

TABLE 2. Log dataset details.

Dataset	Logs	Positives	Negatives	Unique logs
Casper	11,086	9,874	1,212	843
Jhuisi	14,360	11,126	3,234	501
Nssal	107,081	91,349	15,732	2,475
Honey7	8,712	8,162	550	529
Zookeeper	74,380	25,873	48,507	85
Hadoop	180,896	169,458	11,438	344
Spark	33,236,604	31,511,707	1,724,897	634
Honey5	124,386	67,798	56,588	8,315
Windows	24,991	18,534	6,457	181

The Casper dataset [20] consists of operating system logs extracted from a disk image collected by Digital Corporal. These logs originated from the ext3 file system of Ubuntu 8.10. The Jhuisi and Nssal datasets [21] were obtained from the Digital Forensic Research Workshop challenge. This challenge involved investigating the traces of attackers using various Linux OS log files. Additionally, we also used datasets from forensic challenges, namely Honey5 Marty et al. [22] and Honey7 Arcas et al. [23], which were collected from compromised Linux servers where unauthorized access to server resources and data had been obtained by attackers.

The log data from the Zookeeper service, an open-source distributed coordination service, was aggregated by the laboratory of the Chinese University of Hong Kong over a span of 26 days [24]. As for Hadoop, a framework for processing large datasets across computer clusters, the logs were generated from a Hadoop cluster comprising five machines with a total of 46 cores [25]. Spark is a managerial tool designed for processing large volumes of data. The logs were collected from the Spark system which comprises 32 machines [24]. Lastly, the Windows dataset consists of logs obtained from Component-Based Servicing (CBS) in Windows 7.

To verify the ability of the proposed model on detecting new types of log data, we only train the first six datasets, reserving the remaining ones for inference. Since our approach operates as a supervised learning method, labeling the data poses a challenge for system operators. Therefore, we aggregate the count of unique processed logs and present them in the table. The number of unique logs is relatively small in comparison to the total number of logs, which significantly reduces the effort required for reviewing log labels.

B. EXPERIMENTAL SETUP

The experiments are evaluated in a PC with Windows 10, Intel(R) Core(TM) i7-10700(16 CPUs), NVIDIA RTX 2060, and 16GB of RAM. The environment is set up with Anaconda 4.11.0, Python 3.8.16, and Tensorflow 2.8.4. Our code and details of evaluation results are released at https://github.com/tuananhphamds/TransSentLog.

The details of hyperparameters are described in Table 3. The learning rate warmup has proved to achieve success in stabilizing Transformer training [26], we utilized AdamW [27] as the optimization algorithm for the model parameters, setting an initial learning rate of 0.0001 and allocating 20% of the training steps for the warmup phase.

TABLE 3. Hyperparameter values used in experiments.

Hyperparameter	Value
Log length L	10
Number of Transfomer blocks N	2
Number of heads in each Transformer block h	2
Word embedding dimension d	50
Dropout	0.1
Number of hidden units in the Feed Forward layer	2048
Batch size	1024
Number of epochs	20
Train:Validation:Test split	60:20:20
Optimizer	AdamW
Number of steps in Integrated Gradients m	50

For evaluating the model, we employ the F1 Score metric, which is derived from precision and recall. Precision quantifies the ratio of true positives to the total positives predicted by the model, while recall assesses the model ability to correctly classify positive cases. The F1 Score represents the harmonic mean of precision and recall, providing a balanced measure of the model performance. To test the stability of the proposed model, we conduct 20 trials of experiments and calculate the average results.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{8}$$

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{9}$$

To calculate these metrics, we utilize the "macro average" option from the sci-kit-learn library, which computes the unweighted mean per class.

V. EXPERIMENT RESULTS AND ANALYSIS

A. EVALUATION OF THE DETECTION PERFORMANCE OF PYLOGSENTIMENT AND TRANSSENTLOG WITH AND WITHOUT THE LOG PARSER

To examine the impact of the proposed log preprocessing step, we compare the detection performance of Pylogsentiment and TransSentLog with and without the log parser. The comparison results of the F1 score among different datasets are presented in Table 4.

TABLE 4. Performance comparison F1 Score between Pylogsentiment and TranSentLog with and without the log parser.

Dataset	With log parser		Without log parser	
	Pylogsentiment	TransSentLog	Pylogsentiment	TransSentLog
Casper	99.52	**99.62**	99.47	99.41
Jhuisi	99.85	99.86	99.87	**99.91**
Nssal	99.83	99.83	99.84	**99.85**
Honey7	**99.37**	99.22	**99.37**	99.27
Zookeeper	**99.99**	99.98	**99.99**	99.98
Hadoop	99.97	99.97	**99.99**	99.92
Spark	62.67	68.13	78.56	**88.47**
Honey5	**99.44**	99.37	99.08	99.10
Windows	94.30	97.15	71.17	**97.41**
Average	94.99	95.90	94.15	**98.15**

TABLE 5. Performance detection of machine learning and deep learning approaches.

Method	Trained datasets	Untrained datasets	Overall
SVM	99.75	86.86	95.45
Decision Tree	99.75	67.34	88.95
CNN	99.76	79.96	93.16
Pylogsentiment (GRU)	99.76	82.95	94.15
GRU + Masking + Attention layer	**99.86**	86.56	95.42
TransSentLog	99.73	**94.99**	**98.15**

With the log parser, we expect to verify whether the proposed architecture is able to improve the F1 score compared to Pylogsentiment. It is shown that TranSentLog is not overperformed Pylogsentiment in the trained datasets but it achieved better results in the untrained datasets, specifically 5.46% for Spark and 2.85% for Windows.

The log parser is beneficial to ignore several unnecessary fields in a log event such as timestamp, hostname, service, etc. Thus, it is very crucial to design the log preprocessing step to not degrade the model performance. Without the log parser, TransSentLog still outperforms in detection performance compared to it with the log parser. The largest improvement is the Spark dataset with 20.34%, and 2.25% on average (95.90% → 98.15%). While the TransSentLog can overcome the limitations of not using the log parser, the performance of Pylogsentiment is much affected by that. Specifically, the F1 score of the Windows dataset decreased by 23.13%, and 0.84% on average (94.99% → 94.15%).

B. COMPARISION OF VARIOUS MACHINE LEARNING AND DEEP LEARNING APPROACHES

To assess the effectiveness of the proposed work, we compare TransSentLog to machine learning methods such as SVM and Decision Tree, as well as deep learning approaches like CNN [28], Pylogsentiment, and a variant of Pylogsentiment (GRU + Masking + Attention layer). The comparison results in terms of trained datasets, untrained datasets, and overall are presented in Table 5.

For the SVM and Decision Tree methods, we first flatten the embedding matrix of a log event and then use the embedded vector for classification. We also created a modified version of Pylogsentiment by adding a masking layer and dot-product attention layer to learn important words in log events.

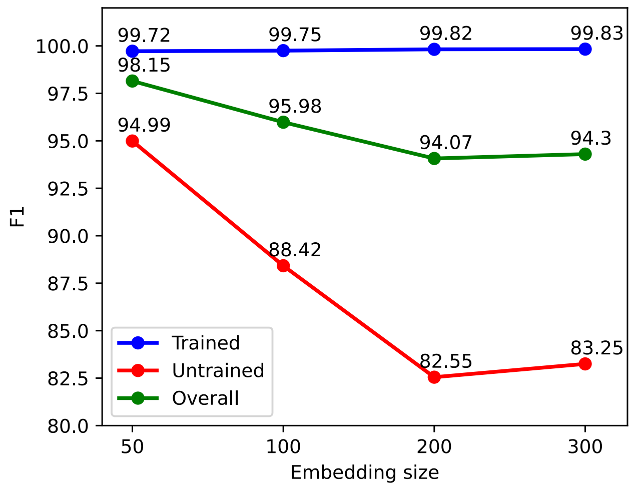


FIGURE 4. Effect of different GloVe word embedding sizes on model performance.

It can be seen that the F1 score of various methods on the trained datasets is comparable, the highest F1 score is 99.86% by the variant of Pylogsentiment. That demonstrates the effectiveness of the masking and attention layers. On the other hand, TransSentLog outperforms other methods on the untrained datasets, with 8.13% improvement compared to 86.86% of the SVM method.

We observe that the F1 score of TransLogSent on trained datasets can be improved by setting a larger initial learning rate, removing the SpatialDropout1D layer, or adding more TransformerBlock. However, there is a trade-off between the F1 Score of trained and untrained datasets. Increasing the F1 score of trained datasets leads to a decrease in untrained datasets, and vice versa.

C. EFFECT OF DIFFERENT WORD EMBEDDING VECTORS ON MODEL PERFORMANCE

The GloVe word embedding 6B tokens has four dimension versions: 50d, 100d, 200d, and 300d. Therefore, we make a comparison between them in order to choose the best version. The effect of different GloVe embedding sizes on the proposed model is illustrated in Figure 4.

As the word embedding size increases, the F1 Score of TransSentLog slightly improves on the trained datasets, while it significantly decreases on the untrained datasets. This can be explained by two reasons. Firstly, as the embedding size increases, the number of model parameters also increases, enabling the model to learn more data and resulting in a higher F1 score on the trained datasets. For example, the number of parameters is 400,000 for 50d, but 4 million for 300d. Secondly, since the importance of each feature in word embedding vectors learned by GloVe is different, the attention heads from TransSentLog may not extract effective features when the number of features is large, particularly in 200d and 300d. This leads to a degradation of detection performance on the new types of datasets (untrained).

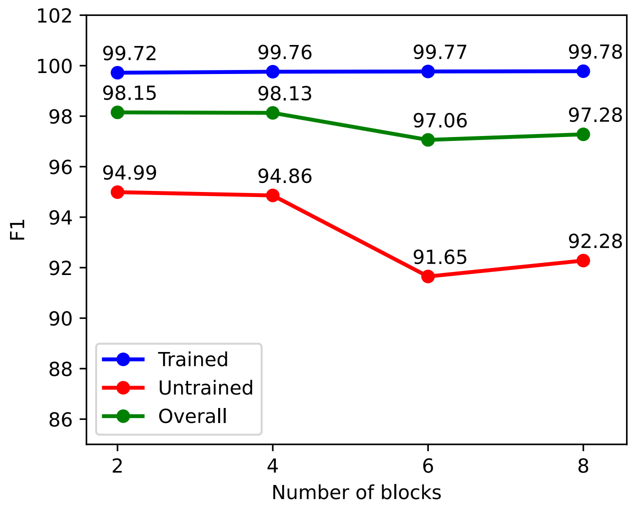


FIGURE 5. **Effect of different number of transformer blocks on model performance.**

D. EFFECT OF DIFFERENT NUMBER OF TRANSFORMER BLOCKS ON MODEL PERFORMANCE

Generally, increasing the number of Transformer blocks may enhance the model capacity. Therefore, we examined how different numbers of Transformer blocks affect the model performance. The evaluation results in terms of F1 scores for various numbers of blocks are depicted in Figure 5.

Although the increase in Transformer blocks helps improve the F1 Score on the trained datasets (99.72% → 99.78%), the ability of the model to detect anomalies on new types of log data decreases from 94.99% of 2 blocks to 92.28% of 8 blocks. This might be attributed to overfitting the training data when the model becomes too complex and start to memorize the training examples instead of learning general patterns.

E. EFFECT OF THE LENGTH OF LOG EVENTS ON MODEL PERFORMANCE

To evaluate the effect of the length of log events on the model performance, we counted the lengths of all processed log events in the training data. The distribution of log lengths from the training data is presented in Figure 6.

Most log events have lengths below 20 words, which takes 98.22% of all log events. In the training data, we observed that log events that have lengths larger than 30 words were "ERROR" or "Exception" messages. Additionally, the negative keywords and main words of these logs can be determined within the first 20 words, while the remaining words are just details of log events. Based on that, we only choose the lengths of 5, 10, 15, and 20 for the evaluation.

Figure 7 presents the detection performance of TransSentLog with various log lengths. The log length of 5 is insufficient to capture meaningful information for model classification. It can be easily noticed that TransSentLog achieved the lowest result 95.42%, among the four log lengths. The model performance becomes more stable when the log length is larger than 10, and there is not much difference in the F1 score among the last three log lengths.

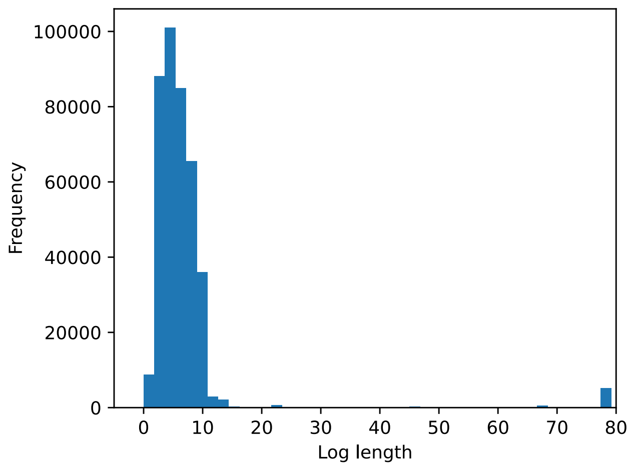


FIGURE 6. **Log lengths counted from training data.**

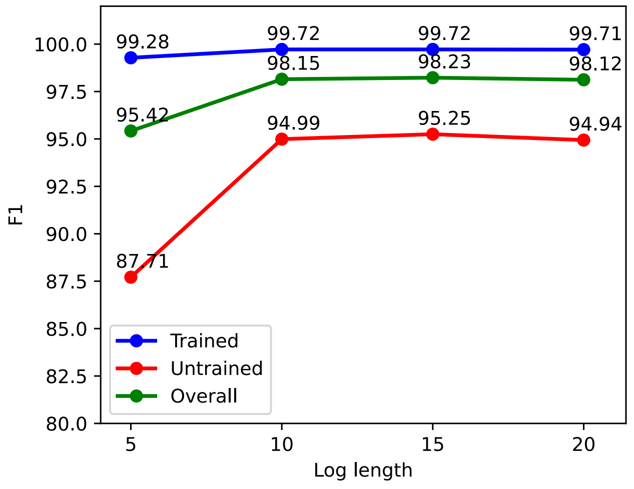


FIGURE 7. **Effect of different log lengths on model performance.**

F. EFFECT OF TOMEK-LINK, SPATIALDROPOUT, AND MASKING LAYER ON MODEL PERFORMANCE

To examine the effectiveness of three components in TransSentLog (Tomek link, Spatial Dropout, and Masking layer), we sequentially remove them from the proposed architecture and compute the F1 Score. Table 6 presents the F1 scores of different variants of TransSentLog among nine log datasets.

Tomek link is an undersampling method that has been shown as an effective technique to solve imbalanced log datasets compared to other methods such as oversampling or random sampling [3]. The table shows that TransSentLog performs slightly better than the variant without the Tomek link.

For the Spatial Dropout component, it is obvious that the F1 scores of the second variant mostly perform better than the full version on the trained datasets. Meanwhile, for the untrained datasets, However, the model shows signs of overfitting on the training data and lacks generalization ability on new data types. For the Spark dataset, TransSentLog achieves a significant improvement of 5.82% (82.65% → 88.47%) when applying the Spatial Dropout.

TABLE 6. Effect of Tomek link, SpatialDropout, and masking layer on model performance.

Dataset	No Tomek link	No Spatial Dropout	No masking layer	All
Casper	99.38	99.31	**99.59**	99.41
Jhuisi	99.91	**99.98**	99.81	99.91
Nssal	99.85	**99.86**	99.83	99.85
Honey7	99.23	99.47	**99.62**	99.27
Zookeeper	**99.98**	**99.98**	99.92	**99.98**
Hadoop	99.92	**99.97**	99.89	99.92
Spark	84.13	82.65	70.73	**88.47**
Honey5	99.06	99.03	99.08	**99.10**
Windows	**98.61**	95.22	77.82	97.41
Average	97.78	97.27	94.03	**98.15**

TABLE 7. Training and inference time (second).

Method	Training time	Inference time	Number of params
Pylogsentiment (128)	2615.29	94.78	237,058
Pylogsentiment (1024)	356.31	93.75	237,058
TransSentLog	139.74	71.62	454,998

Lastly, the masking layer is deployed to avoid the impact of unknown and padding tokens on model performance. Without the masking layer, the model performance degrades significantly. Specifically, the F1 score on the Spark dataset is just 70.73% compared to 88.47% of the original TransSentLog. Similarly, the F1 score on the Windows dataset reduces from 97.41% to 77.82%. The effect of the masking layer is clearly stated in Table 4 as well, where the F1 scores of Spark and Windows datasets are 78.56% and 71.17%, respectively, because Pylogsentiment does not apply the masking layer in its method.

G. ASSESSMENT OF TRAINING AND INFERENCE TIME

The training and inference time between Pylogsentiment and TransSentLog is presented in Table 7. Pylogsentiment utilizes a batch size of 128 for training, thus it is unfair to use a batch size of 1024 in training TransSentLog for comparison. We use the same batch size of 1024 for both Pylogsentiment and TransSentLog. The results clearly show that the proposed approach required significantly less time for both training (approximately 2.5 times faster) and inference (approximately 0.24 times faster) compared to the existing approach.

H. PREDICTION INTERPRETATION USING INTEGRATED GRADIENTS

The Integrated Gradients (IG) method is adopted for interpreting the model predictions, specifically abnormal outcomes. The anomalies classified by the model can be true positives or false alarms. To support system operators in making decisions, the importance weights of each word in log events are plotted to indicate which words are the most contributions to the model outputs. The top three words which have the highest weights are extracted to quickly determine negative words and their contextual counterparts. Moreover, these three words also help system operators determine whether the output is a false positive. Since the weights retrieved from IG can be negative or positive, larger weights indicate greater contributions of the corresponding features to the model predictions. For the sake of readability, we normalize all the weights of each log event within a range of 0 to 1. Figure 8 demonstrates three examples of logs and their prediction interpretations, all of which have been classified as anomalies by the model.

The first log event indicates an issue when the system is unable to reserve a specific memory range. All timestamps, numbers, and hex codes (0×0, 0×9ffff) are removed after preprocessing the log. The negative word "not" and its context word "range" and "could" can be interpreted as the ones that mostly contribute to the classification result.

The second log event indicates an interruption in the processing of a message on the queue. In the processed message, the "WARN" log level was ignored in the preprocessing step. Consequently, the model learned to pay attention to other words in the log message such as the negative word "interrupted", and context words ("message" and "queue"). We can see that the top 3 words ("interrupted", "message", and "queue") sufficiently explain the abnormal behavior of the log event.

The last example states that the system encountered an error in which the given ID of a container could not be found in the system. The model assigned the highest weight to the "error" log level and the other two words "for" and "id".

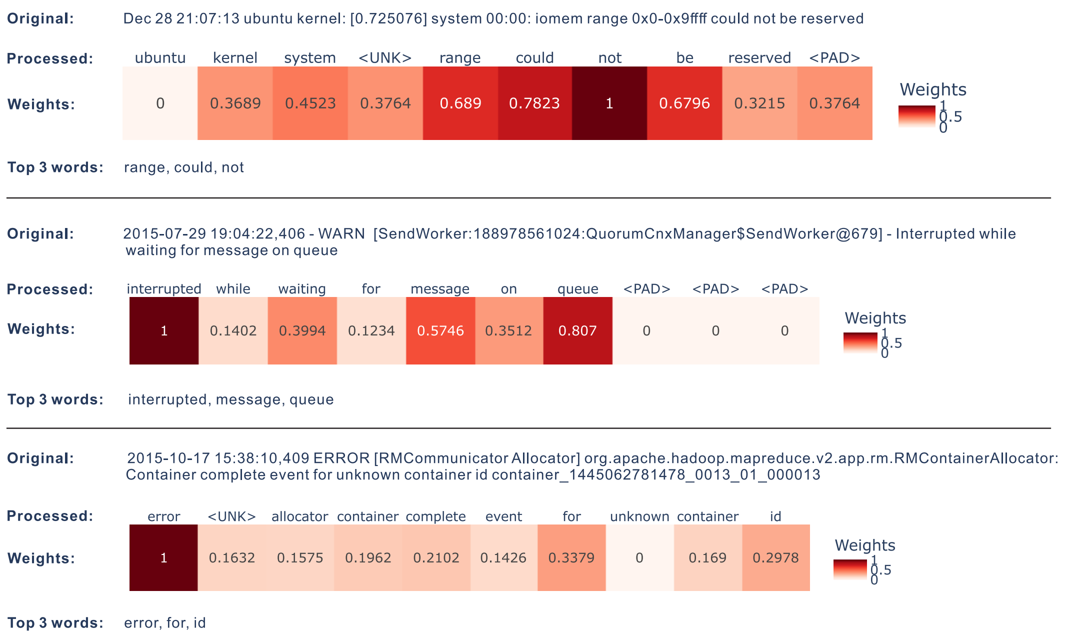
Based on the analysis of the above-mentioned examples, TransSentLog effectively shows the ability to extract negative keywords as well as words related to entities or components within log events. Therefore, the proposed approach can be considered a better alternative method for detecting log anomalies through keyword analysis.

VI. LOG ANOMALY DETECTION IN REAL-WORLD SCENARIO

To demonstrate the applicability of TransSentLog, we present a disk failure scenario in Figure 9. In the scenario, TransSentLog was applied to detect anomalous log events in our system. The upper part of the figure shows our service architecture while the lower part presents the detected log messages when the disk failure occurred.

Once a physical disk has encountered a failure, the OSD (Object Storage Device), a component in the ceph cluster, could not perform its functions such as reading or writing from that disk. In this scenario, we did not replicate the data stored on the failed disk. As a result, the OSD was no longer available and cannot be monitored by the Ceph cluster. Hence, TransSentLog detected the first log message and raised it to the user from AnomalyDetector. Moreover, when the OSD in the ceph cluster was interrupted, MySQL service was not able to write its data to the physical disk. This led to the MySQL service hanging for more than 600 seconds, as shown in the second detected log event. Finally, the user cannot access the web page running on Tomcat because of the unavailable data.

Without help from the model, it is really hard for the user to detect manually what happened in the system, particularly when numerous services are running. Besides the detected results, the keywords also show information about detected



Original: Dec 28 21:07:13 ubuntu kernel: [0.725076] system 00:00: iomem range 0x0-0x9ffff could not be reserved

Processed:	ubuntu	kernel	system	<UNK>	range	could	not	be	reserved	<PAD>
Weights:	0	0.3689	0.4523	0.3764	0.689	0.7823	1	0.6796	0.3215	0.3764

Top 3 words: range, could, not

Original: 2015-07-29 19:04:22,406 - WARN [SendWorker:188978561024:QuorumCnxManager$SendWorker@679] - Interrupted while waiting for message on queue

Processed:	interrupted	while	waiting	for	message	on	queue	<PAD>	<PAD>	<PAD>
Weights:	1	0.1402	0.3994	0.1234	0.5746	0.3512	0.807	0	0	0

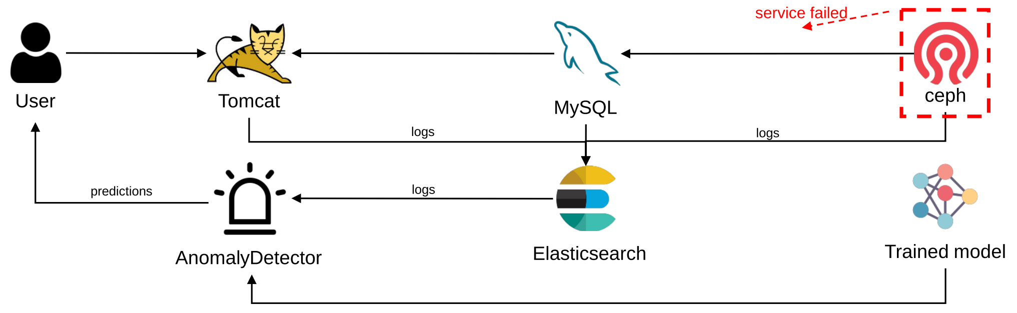
Top 3 words: interrupted, message, queue

Original: 2015-10-17 15:38:10,409 ERROR [RMCommunicator Allocator] org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: Container complete event for unknown container id container_1445062781478_0013_01_000013

Processed:	error	<UNK>	allocator	container	complete	event	for	unknown	container	id
Weights:	1	0.1632	0.1575	0.1962	0.2102	0.1426	0.3379	0	0.169	0.2978

Top 3 words: error, for, id

FIGURE 8. Examples of interpreting model predictions.



Detected logs

Log message	Service name	Keywords
2023-04-06 02:41:49.006 7f8fd4f17700 0 log_channel(cluster) log [WRN] : Health check failed: 2 hosts (1 osds) down (OSD_HOST_DOWN)	ceph	heath, check, wrn, down
2023-04-16 02:52:53 0 [ERROR] [FATAL] InnoDB: Semaphore wait has lasted > 600 seconds. We intentionally crash the server because it appears to be hung	mysql	fatal, server, crash, wait

FIGURE 9. Log anomaly detection in disk failure scenario.

log events and whether these logs are false alarms or not. It is important to note that these detected log events were not seen in the training dataset.

VII. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a supervised learning method TransSentLog with sentiment analysis for log anomaly detection. Through extensive evaluation exploring different scenarios, TransSentLog demonstrates its practicality and potential for real-world application. The main contributions of the model are four key points. Firstly, despite not having the log parser in the preprocessing step, TransSentLog still achieves better performance compared to the existing work. Secondly, we conducted various experiments extensively in order to gain in-depth insights from training and inferencing the model. Thirdly, it is obvious that the adoption of the Integrated Gradient method significantly enhances the interpretation of model predictions, which somehow eliminates the black box of deep learning methods. Lastly, we released the code implementation and details of evaluation results on GitHub repository for reproducibility.

There are several limitations of our work that would be considered in future works. We did not take into account the varying number of logs across different log types during model training. For example, the Hadoop dataset has more than 10 times the number of logs compared to the Casper. We expect to collect more types of log data that may help the model learn better general patterns. Also, we may explore log events and their sentiment from other domains apart from system logs to enhance the generalizability of TransSentLog. In the last example of interpreting model predictions using Integrated Gradients, the model should pay attention to "error", "unknown", and "id" rather than "error", "for", and "id". Although system operators can label the log events with unique logs, it would be better to develop an automatic labeling mechanism without losing the data reliability. Considering transferring the knowledge from large language models such as ChatGPT, and LLaMa or applying them for labeling data might be effective alternatives. As the scope of this work is anomaly detection on individual log events which does not explore the abnormal behaviors of template sequences, we plan to combine both types of anomalies and process mining techniques in future work.

ACKNOWLEDGMENT

The authors would like to thank H. Studiawan for providing the labels of public log datasets whose paper is Pylogsentiment.

REFERENCES

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009.

[2] N. Zhao, H. Wang, Z. Li, X. Peng, G. Wang, Z. Pan, Y. Wu, Z. Feng, X. Wen, W. Zhang, K. Sui, and D. Pei, "An empirical investigation of practical log anomaly detection for online service system," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021, pp. 1404–1415, doi: 10.1145/3468264.3473933.

[3] 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.*, Texas, TX, USA, Oct. 2017, pp. 1285–1298.

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

[5] 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*.

[6] H. Studiawan, F. Sohel, and C. Payne, "Anomaly detection in operating system logs with deep learning-based sentiment analysis," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2136–2148, Sep. 2021.

[7] H. Studiawan, F. Sohel, and C. Payne, "Automatic log parser to support forensic analysis," in *Proc. 16th Austral. Digit. Forensics Conf.*, 2018, pp. 1–10.

[8] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1–15.

[9] T. Lei, Y. Zhang, S. I. Wang, H. Dai, and Y. Artzi, "Simple recurrent units for highly parallelizable recurrence," 2017, *arXiv:1709.02755*.

[10] Z. Wang, V. Joo, C. Tong, X. Xin, and H. C. Chin, "Anomaly detection through enhanced sentiment analysis on social media data," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2014, pp. 917–922.

[11] K. Patel, O. Hoeber, and H. Haminton, "Real-time sentiment-based anomaly detection in Twitter data streams," in *Proc. Canadian Conf. Artif. Intell.*, 2015, pp. 196–203.

[12] C. Soh, S. Yu, A. Narayanan, S. Duraisamy, and L. Chen, "Employee profiling via aspect-based sentiment and network for insider threats detection," *Exp. Syst. Appl.*, vol. 135, pp. 351–361, Nov. 2019.

[13] H. Studiawan, F. Sohel, and C. Payne, "Sentiment analysis in a forensic timeline with deep learning," *IEEE Access*, vol. 8, pp. 60664–60675, 2020.

[14] D. Zhang, D. Dai, R. Han, and M. Zheng, "SentiLog: Anomaly detecting on parallel file systems via log-based sentiment analysis," in *Proc. 13th ACM Workshop Hot Topics Storage File Syst.*, Jul. 2021, pp. 86–93, doi: 10.1145/3465332.3470873.

[15] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.

[16] X. Duan, S. Ying, W. Yuan, H. Cheng, and X. Yin, "QLLog: A log anomaly detection method based on Q-learning algorithm," *Inf. Process. Manag.*, vol. 58, no. 3, May 2021, Art. no. 102540.

[17] I. Tomek, "Two modifications of CNN," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, no. 11, pp. 769–772, Nov. 1976.

[18] 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. NIPS*, 2017, pp. 5998–6008.

[19] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3319–3328.

[20] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora," *Digit. Invest.*, vol. 6, pp. 2–11, Sep. 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287609000346

[21] E. Casey and G. G. Richard III. (2009). *DFRWS Forensic Challenge 2009*. [Online]. Available: http://old.dfrws.org/2009/challenge/

[22] R. Marty, A. Chuvakin, and S. Tricaud. (2010). *Challenge 5 of the Honeynet Project Forensic Challenge 2010—Log Mysteries*. [Online]. Available: https://www.honeynet.org/challenges/forensic-challenge-5-2010-log-mysteries/

[23] G. Arcas, H. Gonzales, and J. Cheng. (2011). *Challenge 7 of the Honeynet Project Forensic Challenge 2010—Log Mysteries*. [Online]. Available: https://www.honeynet.org/challenges/forensic-challenge-7-analysis-of-a-compromised-server/

[24] 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.

[25] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2016, pp. 102–111.

[26] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," 2019, *arXiv:1908.03265*.

[27] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proc. ICLR*, New Orleans, LA, USA, 2019, pp. 1–19.

[28] A. Severyn and A. Moschitti, "UNITN: Training deep convolutional neural network for Twitter sentiment classification," in *Proc. 9th Int. Workshop Semantic Eval. (SemEval)*, 2015, pp. 464–469.

TUAN-ANH PHAM received the B.Eng. degree from the University of Information Technology, Vietnam National University at Ho Chi Minh City, Ho Chi Minh City, Vietnam, in 2018, and the M.S. degree from Soongsil University, Seoul, South Korea, in 2020. He is currently a Senior Research Engineer with the AI Laboratory, MOADATA, Seongnam-si, South Korea. His research interests include time series analysis, log processing, natural language processing, and anomaly detection.

JONG-HOON LEE received the B.Eng. degree and the M.S. degree in electronic engineering from Soongsil University, Seoul, South Korea, in 2005 and 2014, respectively. He is currently a AI Research Engineer and a CTO with the AI Laboratory, MOADATA, Seongnam-si, South Korea. He is studying on biological age and survival analysis model in healthcare. His research interests include anomaly detection to prevent lots of anomalies from time-series, text, audio, and vision data in the real world. He is also interested in synthetic data by generative models and LLMs.