ContrastBERT: Supervised Contrastive Learning of BERT-Encoded IT logs for Anomaly Classification

Raziuddin Mahmood
University of California, Berkeley
Berkeley, CA, USA
razi_mahmood@berkeley.edu

Xiaotong Liu
IBM Watson AI
San Jose, CA, USA
Xiaotong.Liu@ibm.com

Anbang Xu
LinkedIn, Inc.
San Francisco, CA, USA
xabang@gmail.com

Rama Akkiraju
NVIDIA
San Jose, CA, USA
rama.akkiraju@gmail.com

***Abstract*—Maintaining up time for cloud systems is critically important. Many of the systems output their statuses in logs that record all major transactions and events encountered. These have become a valuable resource for understanding system status and performance issues. Often, the IT logs came in the form of free text intermixed with identifiers such as block ids. While some anomalies may form distinct event patterns, learning such patterns itself may be difficult. In this paper, we present an approach that directly uses the textual content of the logs and derives a discriminative embedding from supervised contrastive learning of Sentence BERT-encoded IT logs. The contrastive embedding is then used to train a contrastive classifier and combined with a one class SVM to increase the accuracy with which both anomalies and normal logs are recognized. Results are shown on several benchmark datasets and compared to state of the art methods.**

***Index Terms*—BERT, neural networks, IT logs, anomaly detection, supervised contrastive learning**

I. Introduction

Maintaining up time for enterprise cloud systems is critically important. These include servers, storage, and network systems, many of which output their status on a continuous basis in logs [6]. Detecting anomalous events through analysis of such logs is vitally important to maintain performance and honor service level agreements. The logs record all major transactions and events encountered which become a valuable resource for understanding system status and performance issues. They are usually small textual documents of a few hundred sentences consisting of language text (English mostly), interspersed with codes, identifiers, specifications of time, etc. Figure 1 shows examples of HDFS logs in two different blocks. Figure 1a shows a normal block while Figure 1c shows a block that is labeled as anomalous. As can be seen, the two pieces of text are apparently very similar and the anomaly is usually due to a small change such as a small textual fragment being present as extra or being missing, or the order of events being disturbed. This can also be seen by reducing these pieces of text to event patterns using popular log parsers [20] as shown in Figure 1b and Figure 1d. Here the normal and abnormal patterns differ by a length difference of 1, and out of order placement of at most two events. In addition, since anomalies are rare events, training classifiers for such patterns with severe class imbalance is difficult. Thus discriminating between anomaly and normal logs is a challenging problem.

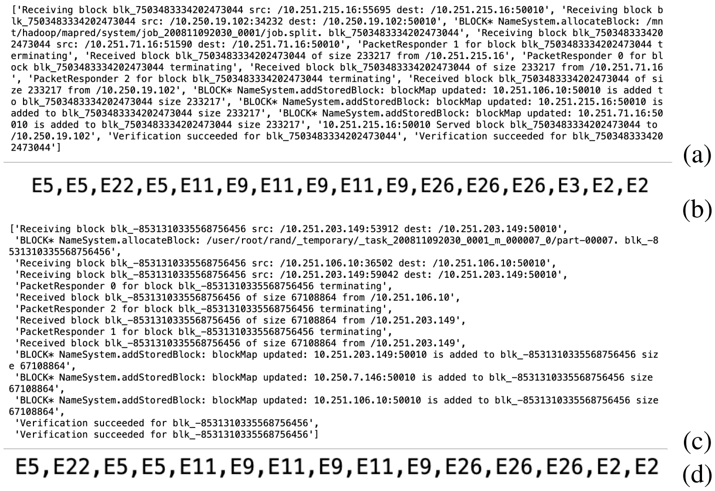
The predominant approaches to log anomaly detection involve extracting event patterns such as those in Figure 1b,d using tools such as log parsers [20]. Typically, the end-to-end processing includes parsing logs into structured data, and creating log sequences to begin the modeling. Once the event sequences are obtained, they are further analyzed by either custom feature extractors, such as TF/IDF features and then sent to classifiers [12] or a deep learning model is trained on the normal patterns to detect deviations [8]. Both the detection of event patterns, as well as the reliable classification of anomalies remain as challenging problems. It is difficult to infer event patterns reliably and in a general way for the large variety of text in logs being the output of many different system components. Similarly, finding anomalies suffers from the large class imbalance problem making it a challenge for developing a discriminable classifier.

In this paper, we present two novel enhancements to address both issues in anomaly classification. Specifically, we work directly with raw textual logs and produce embeddings that capture the context better both within and across sentences. We then find a representation that embeds the log encodings in a contrastive space that helps differentiate anomalies from normal logs. The classifiers are then built using the contrastive embeddings. The result is a better separation of anomalies from logs leading to higher accuracies and F-scores for the overall anomaly classification problem.

To our knowledge, ContrastBERT is the first formulation in which a supervised contrastive embedding is learned for BERT-encoded text. Previous approaches have used unsupervised contrastive learning as a pre-training step to the construction of BERT model [17]. Our approach addresses a major limitation of existing anomaly classification methods many of which rely on the availability of an exhaustive list of prior-specified anomaly patterns extractable from textual logs. The generalized contextual modeling of SBERT allows easy capture of regular expression and other sequence patterns that are unique to IT logs without requiring explicit prior cataloging.

II. Related Work

There are a number of approaches taken by researchers to analyze logs for anomalies, all of which work with regular expression patterns captured from raw text messages as event



```
['Receiving block blk_7503483334202473044 src: /10.251.215.16:55695 dest: /10.251.215.16:50010', 'Receiving block b
lk_7503483334202473044 src: /10.250.19.102:34232 dest: /10.250.19.102:50010', 'BLOCK* NameSystem.allocateBlock: /mn
t/hadoop/mapred/system/job_200811092030_0001/job.split. blk_7503483334202473044', 'Receiving block blk_750348333420
2473044 src: /10.251.71.16:51590 dest: /10.251.71.16:50010', 'PacketResponder 1 for block blk_7503483334202473044 t
erminating', 'Received block blk_7503483334202473044 of size 233217 from /10.251.215.16', 'PacketResponder 0 for bl
ock blk_7503483334202473044 terminating', 'Received block blk_7503483334202473044 of size 233217 from /10.251.71.16
', 'PacketResponder 2 for block blk_7503483334202473044 terminating', 'Received block blk_7503483334202473044 of si
ze 233217 from /10.250.19.102', 'BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.106.10:50010 is added t
o blk_7503483334202473044 size 233217', 'BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.215.16:50010 is
added to blk_7503483334202473044 size 233217', 'BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.71.16:50
010 is added to blk_7503483334202473044 size 233217', '10.251.215.16:50010 Served block blk_7503483334202473044 to
/10.250.19.102', 'Verification succeeded for blk_7503483334202473044', 'Verification succeeded for blk_750348333420
2473044']
```

```
E5,E5,E22,E5,E11,E9,E11,E9,E11,E9,E26,E26,E26,E3,E2,E2
```

```
['Receiving block blk_-8531310335568756456 src: /10.251.203.149:53912 dest: /10.251.203.149:50010',
 'BLOCK* NameSystem.allocateBlock: /user/root/rand/_temporary/_task_200811092030_0001_m_000007_0/part-00007. blk_-8
531310335568756456',
 'Receiving block blk_-8531310335568756456 src: /10.251.106.10:36502 dest: /10.251.106.10:50010',
 'Receiving block blk_-8531310335568756456 src: /10.251.203.149:59042 dest: /10.251.203.149:50010',
 'PacketResponder 0 for block blk_-8531310335568756456 terminating',
 'Received block blk_-8531310335568756456 of size 67108864 from /10.251.106.10',
 'PacketResponder 2 for block blk_-8531310335568756456 terminating',
 'Received block blk_-8531310335568756456 of size 67108864 from /10.251.203.149',
 'PacketResponder 1 for block blk_-8531310335568756456 terminating',
 'Received block blk_-8531310335568756456 of size 67108864 from /10.251.203.149',
 'BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.203.149:50010 is added to blk_-8531310335568756456 siz
e 67108864',
 'BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.7.146:50010 is added to blk_-8531310335568756456 size
67108864',
 'BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.106.10:50010 is added to blk_-8531310335568756456 size
67108864',
 'Verification succeeded for blk_-8531310335568756456',
 'Verification succeeded for blk_-8531310335568756456']
```

```
E5,E22,E5,E5,E11,E9,E11,E9,E11,E9,E26,E26,E26,E2,E2
```

Fig. 1. Illustration of the difficulty of anomaly classification in IT logs. The top log (a)-(b) is normal while the lower log (c)-(d) is anomalous.

patterns through log parsers [20]. Many feature extraction methods are applied to such event patterns including PCA [19], methods to capture co-occurrence patterns between different log keys [13], and TF/IDF analysis [12]. The classification of anomalies then uses statistical machine learning methods on extracted features such as logistic regression or support vector machines (SVM) [12], and one-class SVM [11]. With the advent of deep learning approaches, the log anomaly detection problem is being addressed through deep learning networks [8]. Specifically, the event sequence pattern is treated as a text string that follows certain patterns and grammar rules, and the sequence modeled through an LSTM formalism to encode the pattern. The normal execution patterns are learned through the model and deviations from normal system execution are flagged as anomalies. Similarly other recurrent neural networks (RNNs) are also used for log anomaly detection [14], [18]. These networks model the context in one direction only aiming to predict the next log event sequence pattern given previous messages.

With the advent of transformer methods such as the bidirectional encoder representations from transformers (BERT), recent work has tried to model the log event sequences through BERT [9]. However, the representations used to drive BERT models are still based on event patterns to be extracted and the full power of raw textual context is not exploited. Further, the anomaly detection method is through masked log key prediction and seeing the deviations from the expected keys for normal logs limiting the types of anomalies that can be detected.

III. Our Approach

Our work addresses two of the key limitations of the current approaches, namely, (a) the need to extract log event patterns, and (b) designing an embedding that is targeted for separating the normal from abnormal logs.

A. Encoding raw log text using SBERT

In our approach, we model the log text directly using a variant of BERT called SBERT [16] which is suitable for predicting at the sentence level rather than the word level. Let $S_1, S_2, ..S_k$ be the sentences in a textual log. Ordinarily, we could encode each sentence S_i using BERT and average to produce a single sentence vector encoding. However, this type of averaging has been known to yield a bad encoding [16]. We instead form a single string by concatenating individual sentences $S = S_1.S_2....S_K$. The resulting string is encoded using SBERT which uses BERT underneath and is trained using a Siamese network architecture to enable variable length strings to be encoded into uniform size encoding vectors. Even so, since BERT pre-trained models have a maximum word length of 512, the input text of a log may be broken into chunks of 512 each for the above modeling. By using a single string approach to logs, and producing a uniform length encoding, we normalize for differences in the size of the textual logs in terms of sentences. Also, by treating such large chunks of text as one unit, we are able to model the context over larger distances in text spanning beyond a single sentence.

B. Generating a contrastive embedding

The SBERT embedding by itself is not a very discriminative embedding for the purpose of distinguishing between normal and abnormal blocks. For example, the cosine distance between a 768-length SBERT embedding of the block of text in Figure 1a and c is 0.93 still indicating a high degree of similarity.

To produce a more discriminable embedding therefore, we construct a supervised contrastive embedding that is designed to move the SBERT vectors of abnormal and normal logs away from each other following the contrastive encoding paradigm. It models all members of the anomaly logs as positive samples (label 1) and normal logs as negative samples (label 0). The contrastive embedding is designed to pull together SBERT encodings of positive samples while pushing apart the negative samples of the normal class. In order to do this effectively, the class imbalance must be addressed in the training stage. Using the approach of oversampling the abnormals and under-sampling the normals, we select training data that is roughly in equal ratios for training such an encoder. In particular, following the multi-class supervised contrastive learning framework outlined in [10], we generate a new encoder-decoder network consisting of an encoder and a decoder/projection head network. The encoder is a 3 layer network with one input layer, a hidden dense layer and a dense fully connected layer with ReLU activation as shown in Figure 2. The projection network is another 2 layer network with a fully connected layer with ReLU, followed by an output layer with ReLU for binary classification as shown in Figure 2. The encoder maps incoming SBERT vectors I_i to a representation vector R_i normalized to unit hypersphere, and the projection network renders the output z_i to match the expected output Y_i. The similarity between two SBERT log vectors W_i and $W_j \in S_i$ be captured by the contrastive loss as

$$L_{contrast}(S_i) = \sum_{W_j \in S_i} \log \frac{\exp(z_i \cdot z_j/\tau)}{\displaystyle\sum_{a \in A(i)} \exp(z_i \cdot z_a/\tau)} \quad (1)$$

Here z_i is the projected vector for SBERT input vector W_i and z_j is the projected vector similarly for $W_j \in S_i$ where S_i are the logs that belong to the same class as W_i. Finally, z_a is the projected vector for any W_a either inside or outside the class. The contribution between the two classes is weighted by temperature τ. Also, considering both the anomaly and normal classes together, the cumulative contrastive loss is given as:

$$L_{contrast} = \sum_{j}^{|V|} L_{contrast}(S_j) \quad (2)$$

Thus in the above formulation, the design of the contrastive encoder deliberately deviates from the incidence distribution of the logs in order to produce an embedding that can separate the two classes. With this learned embedding, we train two sets of classifiers. One of the classifiers is a neural net classifier consisting of 2 dense layers alternating with 2 drop-out layers with the first layer being a Relu and the second being a Softmax classifier. Using the balanced data for training ensures that both the recall for the abnormal class and precision for the normal class will be high. However, the precision for the abnormal class tends to be low implying a number of normal logs may be mis-classified as abnormal. In anomaly detection, it is desirable to achieve high recall for both anomalies and normal logs. For this, we augment the classification with a one-class SVM to learn the larger class (i.e. normal logs). Such a classifier will have high recall for the normal class while the recall for the abnormal class may be lower. We fuse the output of the two classifiers using the following rule $L(i) =$ Abnormal iff $L_1(i) = L_2(i) =$ Abnormal and normal otherwise, where L_1 and L_2 are the contrastive and One Class SVM classifiers respectively.

C. ContrastBERT model - Training

The overall architecture of the proposed ContrastBERT model in training mode is shown in Figure 2a. The textual logs from both abnormal and normal textual logs are encoded in 512-word chunks using a 768-dimensional SBERT model available from Huggingface (paraphrase-distilroberta-base-v1) [5]. The 512 word limitation comes from the original BERT model built into sentence BERT. Each chunk is then encoded using a 300-dimensional contrastive encoder with a 64-dimensional projection head for training the encoder. In the supervised contrastive loss function for training the encoder, we set temperature=0.05, and a batch size of 30, and trained over 100 epochs or until the network error convergence was reached. We used the Adam optimizer for fast convergence with the learning rate as 0.001. Two NVIDIA P100 GPUs with 16 GB were used for training and training took less a few hours. The contrastive encoder had over 1 million parameters. The choice of the hyper-parameters for temperature and learning rate was derived from cross-validation experiments. We implemented SBERT encoding of logs in PyTorch, and supervised contrastive learning in Tensorflow with tfa-addons libraries.

A contrastive classifier was then trained using the learned encoder as shown in Figure 2b. It consists of 4 layers (2 dense, 2 dropouts of progressively decreasing size 64,32,32,16)and an output layer which uses a Softmax classifier. The intermediate layers use RELU for the nonlinearity. During the training of the contrastive classifier, the weights of the encoder are frozen as they have already been trained using the projection head. For the contrastive classifier we used sparse categorical cross entropy, while the supervised contrastive loss of Equation 2 was used to train the contrastive encoder using the project head. We used the One Class SVM (OSVM) provided in sklearn with a radial basis function (RBF)as the kernel, and gamma='scale' and $nu = 0.01$ to place an upper bound on the training errors. The embeddings from normal logs were used to train the one-class SVM. Note that unlike the contrastive classifier, the OSVM classifier looks only at normals, and anomalies are not present during training.

D. ContrastBERT model - Inference

To classify incoming IT logs, the inference mode of ContrastBERT is as illustrated in Figure 3. All logs (normal or abnormal) go through SBERT encoding followed by Embedding using the contrastive encoder. The embedding vector is fed to both classifiers (One Class SVM and Contrastive classifier) and the results fused to produce the final label.

Our fusion method differs from ensemble learning methods as it is based on a Boolean logic that accepts or rejects the abnormal hypothesis from the contrastive classifier based on the inverse evidence seen in the one-class SVM trained on normals.

E. Time complexity

Time complexity of SBERT encoding (inference) is O(kn) where k is the number of word tokens in a log message and n is the number of logs. This stage can be easily parallelized as a pre-processing step. The contrastive learning complexity is O(mp) where m is the number of batches and p is the number of epochs. The number of batches is a function of the dataset which ranged from 26,000 (HDFS) to 1.2 million logs (Thunderbird-mini). The overall training time is much smaller compared to the original BERT model which was trained on 4 cloud TPUs for 4 days.

IV. Results

A. Datasets

We now present results of using our approach for anomaly classification in IT logs on 3 benchmark datasets summarized in Table I. The HDFS-1 dataset [19] consists of free text sentences generated by the Hadoop file system in a cloud environment while running Map-Reduce jobs. The anomalies were manually identified using a set of handcrafted rules. The full HDFS dataset was provided as a single json file consisting of nearly 11,172,157 messages from 558,223 blocks. Since the anomalies were marked at the level of a block, the log messages were grouped by blocks yielding 16,838 anomalous blocks or nearly 3% of the total blocks were anomalies. The average length of the normal and abnormal blocks was 13.3 and 10.45 sentences respectively.

The BGL dataset [15] was collected from a BlueGene/L supercomputer system at Lawrence Livermore Labs and recorded the performance logs consisting of alert (anomalies) and non-alert messages (indicated by -). Each row was treated as an individual log message for our analysis yielding 4,747,963 log messages, of which 348,460 or 7% of the data were anomalies. Similarly, Thunderbird [15] is another large log dataset in a format similar to BGL with each row signaling a normal or abnormal log. The dataset has 20,000,000 log messages of which 758,562 are anomalous (also 3%) of the full data.

B. Creation of train-test datasets

Using the philosophy of creating a balanced dataset for the contrastive encoder and classifier, we under-sampled the normal logs to maintain a appropriate ratio for the abnormal and normal logs. This allowed the encoder to learn the characteristics of the two classes without a large bias towards one class. The one class SVM, however, was trained with only the normal class to supply the necessary incidence bias during inference. Specifically, we retained 80% of the *abnormal* logs through random sampling. An equal number of *normal* logs were retained through random sampling to create the overall training dataset for the contrastive encoder/classifier. The dataset used for testing the models, however, followed the incidence distribution. For this, we used the remaining 20% of the anomalous logs and retained sufficient randomly sampled normal logs such that the overall abnormal/normal log ratio remained the same as in the original dataset. The total number of logs retained in training and test dataset for the contrastive encoder and classifier are shown in Table I in Columns 7 and 8. For the One Class SVM, the normal class in the training dataset was used, while the same test dataset was available for testing both the contrastive classifier and the One Class SVM.

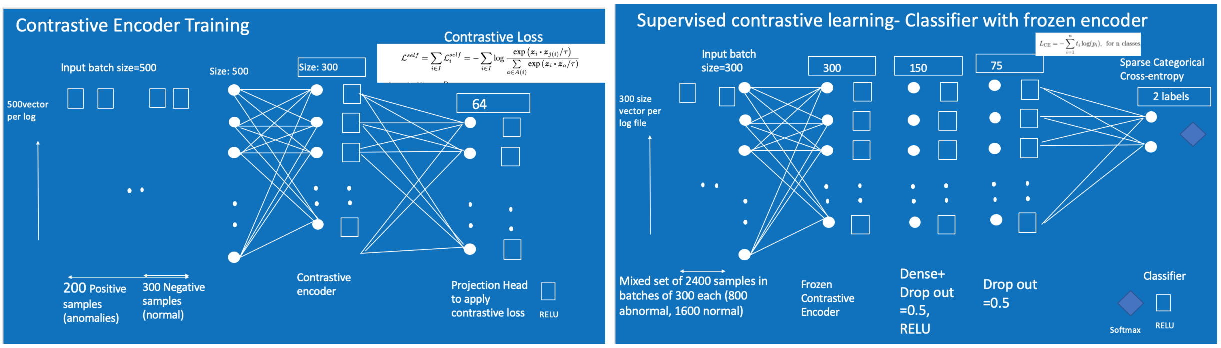
C. Evaluation metrics

In IT logs, since it is critical to not miss a single abnormal, the recall of the abnormal cases is the most important metric, while the precision adds to the burden of verification possibly increasing costs. The F1-score is the goto-score for binary classification [1]. However, when there is class imbalance, it is important to report an F1-score for the two classes separately as combining the two classes, or using other metrics such as accuracy or AUC may give an artificial sense of better performance due to the extreme dominance of the normal class. Our goal in designing the anomaly classifier is to maximize the recall of the anomalous and normal classes. In doing so, we aim to miss as few anomalies and minimize the number of false alarms when the normal logs get mislabeled as anomalies. Thus rather than using a combined F-score, we evaluate it separately per class using the usual formula of $F_1 = \frac{2*precision*recall}{precision+recall}$ per label.

D. Comparison of Performance

We compare ContrastBERT to four major types of approaches whose code was publicly available. All comparative algorithms were used as is from open source and their libraries/parameter details were as specified in their implementations posted on github. These include SVMLog, a statistical machine learning binary classifier with linear kernel and fed with TF/IDF features derived from log events [3], One-Class SVM [11], a variant of SVM trained on normal data only using the same TF/IDF features, DeepLog, a deep learning classifier modeling the sequence information through LSTM models [8], and finally, logBERT a recent approach that uses BERT to encode event sequences derived from logs [9]. The code for One Class SVM came from Sklearn package, while SVMLog, and DeepLog were taken from the open source Loglizer GitHub repository [4]. Finally, logBERT was adopted from logBERT GitHub repository [2]. Since all these methods are based on event sequences rather than the raw text, we adopted the log parser available in logpai [20] to parse the log messages into log keys. To keep the comparison fair, we allowed all comparable methods to learn from 80% of the training data (both normal and abnormal logs randomly sampled). Note that our method of grouping the individual messages into logs differs from the approaches used earlier where chunks were formed based on time duration [14]. Table II lists the performance of the various algorithms (Rows 1-8) in comparison to our approach (Rows 9-10). We observe from this table that all methods appear to do well on normal logs. However, the recall performance is worse in non-deep learning approaches. Secondly, We observe from this table, that our approach maintains a high recall for both normal and abnormal classes implying a more accurate and discriminative anomaly classification using the contrastively learned features.

Comparative methods all showed sensitivity to class imbalance as also previously described [7]. Our approach addresses


Contrastive Encoder Training
Contrastive Loss
$\mathcal{L}^{self} = \sum_{i \in I} \mathcal{L}_i^{self} = -\sum_{i \in I} \log \frac{\exp(z_i \cdot z_{j(i)}/\tau)}{\sum_{a \in A(i)} \exp(z_i \cdot z_a/\tau)}$
Input batch size=500
Size: 500
Size: 300
500vector per log
64
200 Positive samples (anomalies)
300 Negative samples (normal)
Contrastive encoder
Projection Head to apply contrastive loss
RELU
Supervised contrastive learning- Classifier with frozen encoder
$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i)$, for n classes
Input batch size=300
300
150
75
Sparse Categorical Cross-entropy
2 labels
300 size vector per log file
Mixed set of 2400 samples in batches of 300 each (800 abnormal, 1600 normal)
Frozen Contrastive Encoder
Dense+ Drop out =0.5, RELU
Drop out =0.5
Classifier
Softmax
RELU

(a) (b)

Fig. 2. ContrastBERT Log Anomaly Classification architecture. (a) Contrastive encoder training. (b) Classification using frozen contrastive encoder. The projection head is used to train the contrastive encoder.

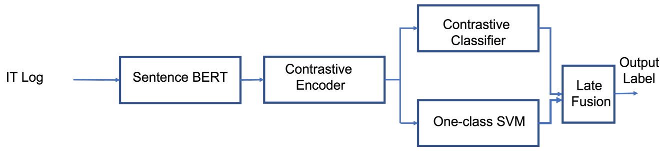


Fig. 3. ContrastBERT log anomaly classification - Overall workflow.

Dataset	Total Messages	Anomaly Messages	%age Log units	Blocks	Anomaly blocks	Train Anomaly ratio	Test Anomaly ratio
HDFS	11,172,157	284,818	3.01%	558,223	16838	26,940(1.0)	112,013(0.031)
BGL	4,747,963	348,460	7.3%	4,747,963	348460	557,536 (1.0)	1,024376 (0.073)
Thunderbird-mini	20,000,000	758,562	3.8%	20,000,000	758,562	1,213,698(1.0)	3,992,421(0.038)

TABLE I

DESCRIPTION OF THE DATASETS USED FOR EXPERIMENTS. THE LAST TWO COLUMNS SHOW THE BREAKDOWN OF THE TRAIN AND TEST DATASETS.

Method	Class	HDFS			BGL			Thunderbird		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
LogSVM	Normal	0.97	0.98	0.95	0.19	0.97	0.57	0.23	0.95	0.36
LogSVM	Anomaly	0.94	0.51	0.66	0.82	0.31	0.45	0.92	0.37	0.53
One Class SVM	Normal	0.98	0.99	0.99	0.94	0.92	0.91	0.91	0.89	0.90
One Class SVM	Anomaly	0.78	0.62	0.69	0.61	0.56	0.58	0.71	0.67	0.68
DeepLog	Normal	0.93	0.87	0.89	0.92	0.88	0.90	0.90	0.99	0.94
DeepLog	Anomaly	0.66	0.35	0.46	0.78	0.56	0.65	0.82	0.76	0.79
LogBERT	Normal	0.92	0.85	0.88	0.94	0.96	0.95	0.98	0.98	0.98
LogBERT	Anomaly	0.65	0.71	0.68	0.76	0.85	0.80	0.83	0.87	0.85
ContrastBERT	Normal	1.0	**0.93**	0.96	0.97	**0.98**	0.98	0.99	**1.0**	0.99
ContrastBERT	Anomaly	0.94	**1.0**	0.97	0.96	**0.96**	0.96	0.98	**0.98**	0.98
Contrastive Only	Normal	0.98	0.24	0.38	0.97	0.35	0.51	0.99	0.45	0.62
Contrastive Only	Anomaly	0.05	0.90	0.09	0.13	0.92	0.22	0.37	0.96	0.53
Contrastive SVM	Normal	0.98	0.99	0.99	0.97	0.98	0.98	0.99	0.98	0.99
Contrastive SVM	Anomaly	0.82	0.78	0.80	0.85	0.67	0.75	0.89	0.76	0.82

TABLE II

ILLUSTRATION OF COMPARATIVE PERFORMANCE ACROSS DATASETS OF VARIOUS LOG ANOMALY DETECTION/CLASSIFICATION METHODS. ALL METHODS EXCEPT OURS (ROW 9 ONWARD) ARE BASED ON LEARNING FROM LOG EVENT PATTERNS. THE LAST 6 ROWS SHOW RESULT OF ABLATION STUDIES ON THE RELATIVE BENEFIT OF EACH OF THE CLASSIFIERS USED INTERNALLY IN OUR APPROACH.

class imbalance with undersampling of normals during the contrastive encoding setup which leads to overall better performance as shown in Table II.

E. Ablation studies

In order to further understand the role of each classification within the ContrastBERT formulation, we performed ablation studies in which we recorded the performance using the contrastive classifier alone, and the One-Class contrastive SVM alone. The result is shown in Rows (11-14). As can be seen by comparing to rows 9-10 from Table II, combining the two classifiers led to the best performance in terms of maximizing the recall for both classes. By optimizing on recall for normal and abnormal and by fusion, we actually get better overall precision as well as seen from Table II.

F. Limitations

While the language context modeling of SBERT can help capture anomalies described by other regular expression pattern-based methods, more complex patterns requiring complex structural relationships or purely numerical measurements may be harder to capture by our approach. The majority of cloud logs, however, are textual in nature, making our methods still suitable. The generalizability of our approach across different log types still needs to be explored.

V. Conclusions

In this paper, we have presented a novel approach to anomaly classification. By working directly with text logs, no log parsing or custom feature extraction is needed. By using Sentence BERT, we are able to better model the sequential context both within and across sentences in IT logs. Using a contrastive encoder-decoder network and classifier combination, a discriminative embedding is learned from a balanced dataset created for the normal and abnormal logs. Finally, by fusing the output of contrastive classifier with a One Class SVM we are able to maximize the recall for both normal and abnormal logs leading to better overall anomaly classification.

References

[1] F1 score vs roc auc vs accuracy vs pr auc: Which evaluation metric should you choose? - neptune.ai.

[2] Github - helenguohx/logbert: log anomaly detection via bert.

[3] loghub/hdfs at master · logpai/loghub · github.

[4] loghub/hdfs at master · logpai/loghub · github.

[5] sentence-transformers/paraphrase-distilroberta-base-v1 · hugging face.

[6] Watson aiops: Bringing ai to it operations management — ibm.

[7] Log-based anomaly detection with deep learning: How far are we? 2022.

[8] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. 2017.

[9] Haixuan Guo, Shuhan Yuan, and Xintao Wu. Logbert: Log anomaly detection via bert.

[10] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised Contrastive Learning. apr 2020.

[11] K.L. Li, H.K. Huang, and W. Tian S.F.and Xu. Improving one-class svm for anomaly detection. *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics*, 2003.

[12] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. Failure prediction in ibm bluegene/l event logs. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 583–588, 2007.

[13] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Jiang Li, and Bin Wu. Mining program workflow from interleaved traces. *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010.

[14] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, and Rong Zhou. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. 2019.

[15] Adam Oliner and Jon Stearley. What supercomputers say: A study of five system logs. *Proceedings of the International Conference on Dependable Systems and Networks*, pages 575–584, 2007.

[16] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, pages 3982–3992, 8 2019.

[17] Peng Su, Yifan Peng, and K Vijay-Shanker. Improving bert model using contrastive learning for biomedical relation extraction. pages 1–10, 2021.

[18] Z. Wang, Z. Chen, J. Ni, H. Liu, H. Chen, and J. Tang. One-class recurrent neural networks for discrete event sequence anomaly detection. *Proc. ACM International Conference on Web Search and Data Mining (WSDM)*, 2021.

[19] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, page 117–132, 2009.

[20] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. Tools and benchmarks for automated log parsing.